UNIVERSITY OF OSLO
Department of Informatics

Læring av objektorientert programmering

Arbeidsprosesser i løsning av oppgaver

Masteroppgave

Ine Lyche Sigernes

24. mai 2005

Short summery.

In this thesis I have tried to find out what students have problems with when learning object oriented programming languages. I have studied how they solve their assignments and how well they succeed. In order to find differences between strong and weak students and in the end suggest advices for the teaching.

I have looked at 4 different video recordings of students programming at a computer. The video shows the pairs students, their code and their conversation. I have studied these video recordings and found examples of the questions I want to figure out. I have found examples of types of methods they use and what they struggle with. I have seen the difference between the different students and seen what the strong students do, compared to the weak students.

I found out that the students have problems with parameters, pointers and the understanding of objects. Some of the students forget to use the parameters and doesn't know how to use pointers. They misunderstand what an object is, and they think that an attribute name is the same as the object it self. These problems hamper the progress of the students' learning and I will suggest that parameters, pointers and objects need to be taught more.

I also found out that the students don't follow any specific method in their assignment solving. They use what ever method they feel useful at the time, the strong students use a method that works well for them. The method the strong students use consists of the steps understanding, writing and testing. From this thesis it looks like the less difference in time use at the different activities they use to solve the assignment the more successful they are. The teachers should encourage the students to make time to understand the assignment and the code, if parts of the code are given, before they try to solve the assignment.

The strong students have a broader perspective and they are able to regard their code as well as the program execution and the real world is to be represented in the program when developing their solution. The weaker students concentrated mostly on the code and the assignment, and didn't show any sign that they knew that they should think of the real world when they program. In this thesis it looks like the weaker students make use of the teachers' code, to get examples on the problem they have. In the start of the semester the students have to concentrate on fewer of the domains to succeed. They should be encouraged to take more of the domains in mind as they learn more trough the semester.

I advice that I give is that the learning should focus more on pointers, parameters and understanding of objects and that the students should be encouraged to make their perspective broader as they learn more through the semester. The teachers should also give the students examples of methods for solving programming assignments, so the students know where to start and what to do. I also give the advice I give for the teaching is that they should use the tool BlueJ, but I advice that they should have one assignment at the end of the semester where the students have to write code from the beginning like we do to day in Emacs. They should do that to ensure that the students really learn Java and object oriented thinking.

(The rest of the thesis is written in Norwegian).

Innholdsfortegnelse

1. Introduksjon.

I løpet av de siste årene er objektorientert programmering og modellering blitt en vanlig læringsmetode. Det er lagt merke til at det er store stryk prosenter og mange avhoppere på universitetsnivå som introduserer studentene for programmering, dette kan indikere at objektorientering og programmering er vanskelige å lære. Objektorientert programmering kan deles inn i to emner, disse to emnene er læring av objektorienterte begreper og læring av å programmere med objektorienterte begreper. I kapittel 5 tar jeg for meg læring av objektorienterte begreper og i kapittel 6 – 8 tar jeg for meg læring av å programmere med objektorienterte begreper. I denne oppgaven fokuserer jeg mest på emnet læring av å programmere med objektorienterte begreper, men læring av objektorienterte begreper er en nødvendig basis og derfor må vi kjenne til det også.

I dette studiet ser jeg etter hvilke problemer studentene har med de objektorienterte begrepene i Java språket. Finnes det noen problemer som alle studentene sliter med. I (Granerud et. al 2005) ser han på hvordan 6 – 9 klasse elever lærer seg objektorienterte begreper. Hva gir dem mest problemer og hva får de til. Jeg vil sammenlikne likheter mellom disse 6 – 9 klasse elevene og studentene i min studie.

Jeg ser også på hvordan studentene programmerer med objektorienterte begreper, det vil si at jeg ser etter måter de løser oppgavene sine på. Hva slags metoder bruker de for å løse oppgavene. Jeg vil prøve å finne ut hva slags ”problemløsning i programmeringslæring” studentene bruker, som jeg deler opp i to analyser. Den ene basert på arbeidsmetoder, som er aktivitetene ”lese oppgave”, ”lese kode”, ”taste kode”, ”kompilere” og ”kjøre program”, hvordan disse aktivitetene brukes kan indikere hva slags arbeidsmetode de bruker. Den andre er basert på domener (områder innen programmeringsomgivelsene) og relasjoner (de delene som kobler disse områdene sammen), dette beskrives mer i kapittel 2. I (S. Booth 1992) har hun gjort en grundig studie av læring av problemløsning i funksjonell programmering og jeg vil prøve å undersøke om noe av det hun kommer fram til i forhold til problemløsningen kan passe på de studentene jeg har sett på. I (Kaasbøll et. al 1998) er det gjort et grundig studie av objektorientert problemløsning for nybegynner studenter, jeg vil også studere hvordan det de kommer fram til kan overføres til de studentene som jeg har sett på.

Jeg vil analysere forskjellen mellom sterke og svake studenter finne ut hva som er forskjellen mellom de forskjellige studentene. Det å finne ut hva sterke studenter gjør i forhold til svakere studenter kan gi oss en pekepinn på hva som bør undervises grundigere for at de svakere studentene skal lære mest mulig. Det å forstå hvordan svake og sterke studenter jobber, hva de har problemer med og hva de ikke har problemer med kan gi oss nyttige tips på hvordan man bør undervise objektorientert programmering.

Jeg vil se om jeg kan finne noen tips for hvordan lærere bør undervise objektorientert programmering. Dette vil jeg basere på de funnene jeg finner i denne oppgaven og av annen litteratur som er skrevet om forskjellige undervisningsmetoder.

Min personlige motivasjon.

Når jeg som ny student tok begynner kurset i programmering på Universitetet i Oslo var det første gang dette kurset skulle undervise Java programmering. Den gang var det nytt for både studenter og forelesere, og det var en periode med litt prøving og feiling for å finne en

undervisningsmetode som fungerte for alle. Siden den gang har jeg vært interessert i å få satt lys på studentenes behov i undervisningen slik at man kommer seg best mulig gjennom dette begynnerkurset. Da jeg kom over denne oppgaven på Internett syntes jeg dette var den muligheten til å få satt lys på en del av studentenes problemer.

Min metode.

Når studentene arbeider har jeg sett på hvordan de arbeider, hva gjør de for å komme seg igjennom oppgavene. Hvordan løser de oppgavene? Hvordan angriper de problemene sine? Hva er det de gjør som forbereder dem på oppgaven? Hvordan fordeler tidsforbruket på disse aktivitetene seg på studentene etter deres nivå som programmerere? Er det mulig å si noe om nivået studentene er på i forhold til hvordan de arbeider.

I denne oppgaven har jeg analysert videoopptak som ble tatt av studenter som løser oppgaver i et begynnerkurs. Jeg har analysert hva studentene har problemer med, analysert hvor mye tid de bruker på forskjellige relasjonene mellom domenene jeg har funnet og hvor mye tid de bruker på forskjellige aktiviteter. Ved analyse av enkelt gruppene og ved å sammenligne gruppene får jeg et inntrykk i hva studentene har problemer med og hva de bruker mesteparten av sin tid på.

2. Teori.

2.1. Java og andre problemer.

Det er mye litteratur som er skrevet om nybegynnere innen programmering og om deres problemer med å lære og forstå programmering. De fleste fokuserer på bakgrunnen til at studentene feiler. Det er ikke mye litteratur jeg har funnet som peker på direkte problemer studentene har med språket de lærer, slikt som for eksempel parametere som jeg peker på senere i oppgaven.

2.1.1 Forståelse av objektorientering.

I (Granerud et al, 2005) er det gjort to forsøk med læring av objekt orientering for 6 og 9 klasse elever. Mens elevene ble undervis ble det tatt opp film av grupper på tre og tre som jobbet sammen, i tillegg ble det tatt opp film av skjermen og samtalene til elevene. Alle elevene i 6 og 9 klasse brukte Lego Mindstorms med Robolab, men 6 klasse ble undervist i mest prosedural programmering mens 9 klasse ble instruert i Java programmering.

Lego Mindstorms med Robolab er basert på LOGO språket, som er designet til å programmere gulv skilpadder, som kan brukes til å lage forskjellige geometriske figurer på et papir. Lego (2005) tok opp denne ideen i deres design i leke konstruksjonen Mindstorms og i Robolab (2005) programmeringsomgivelsen.

Under forsøket ble det registrert at elevene hadde problemer med å forstå forskjellen mellom punktum og parentesene de skal bruke når de kaller en metode med parametere og bruken av parametere, ved at de ikke klarer å navigere seg til rett parameter.

De fant ut at hvis elevene brukte genitivs ”s” når de snakket om kallet på metoder klarte de fleste å få bedre forståelse av hva som skal stå i koden. 9 klasse elevene virket som om de klarte å skille mellom klasser og objekter, men de hadde ikke lært at referanser og objekter er forskjellige enheter, men dette ble heller ikke undervist.

Ut i fra disse forsøkene vil det være feil å si at noen barn forstår objekt orientering, men man kan se at kompetansen i objekt orientert programmering blir konstruert ved et antall av ferdigheter ved å bruke forskjellige objekt orienterte konstruksjoner slik at programmet fungerer slik læreren syntes er akseptabelt. Elevene må også kunne relatere koden til den virkelige verden, samtidig som de må få tak i den sosiale kompetansen de trenger for å programmere.

Problemet med å skille mellom punktum og parentesene i metodekallene har ikke blitt registrert tidligere, men en del forfattere har sett slike problem i nybegynner kurs i programmering. Det er heller ikke dokumentert noe om problematikken rundt navigeringen til parametrene.

I (Pea 1986) sin artikkel belyses studenters problemer med at de bruker programmeringsspråket som om maskinen var en person. Pea har hentet sin erfaring fra flere år med Logo programmering med 8 – 12 og 14 – 17 år gamle barn. Han fant tre forskjellige ”bugs”, som er ”parallelism bug”, ”intentionality bug” og ”egocentrism bug”. ”Parallelism bug” er at man tror at flere linjer kode kan være aktive eller er kjent av maskinen på samme tidspunkt. ”Intentionality bug” er at man tror at maskinen **vil** utføre for eksempel en metode. ”Egocentrism bug” er at man antar at det er mer mening i det man vil oppnå i programmet enn det man faktisk presenterer i koden man har skrevet. Disse tre ”bugs” kan samles i en overordnet bug som Pea kaller for ”super bug”. Dette ”super bug” er at det antas at det er en gjemt hjerne i programmeringsspråket som har intelligente og fortolkende krefter. Denne brukes som en default strategi for å forstå problemer de kommer over, ved at de bruker analogien til naturligspråk samtaler. Dermed kan man si at det første steget i læring av programmering er å gå fra menneskelig tenking til program konsepter.

I (Spohrer et al. 1986) brukte studenter i humanistiske fag som tok introduksjonskurs i programmering. De fant tre problemer, som er ”composition” som er sammenslåingen av beskyttelsen og kalkulasjonen, ”optimization” som er opprinnelig duplisert output og ”previous experience” som er output etter kalkulasjon som har blitt brukt i forrige programmer. De fant tre bug typer, som er ”plan komposisjon”, ”kanskje språk” og ”språk”. Lærere observerte at studentene mister kontrollen på hva de har gjort, kan ikke sette sammen deler av større program, ser ikke konsekvensene når de forandrer et større program og starter uten noen plan. De rapporterte at hovedhindringen for å lære programmering er å putte deler av kode sammen. Dermed virker det som om det er et steg fra forståelse av program konsepter til å håndtere kode ferdighetene.

2.1.2. Problemer med læring av objektorientering.

I (Detienne 2002) nevnes det at nybegynnere innen programmering men også noen eksperter, under visse forhold, må forholde seg til to typer av problemer. Disse problemene er vanskeligheten med å identifisere klasser og vanskeligheten med å koble sammen deklarative og prosedurale aspekter i løsningen. Nybegynnere bruker en del tid på å identifisere og forkaste problemområdeobjekter som potensielle klasser i løsningen sin. Det viser seg at mange av disse klassene de kommer på i den første designfasen vil vise seg som ubrukelige i en senere fase, og klasser de ikke ventet at skulle være med viser seg å være essensielle.

De deklarative og prosedurale aspektene i løsningen kobles ikke sammen før seint i designet. Nybegynnere, oftere enn eksperter, beskriver objektene og handlingene eller prosedyrer separat i deres første versjon av løsningen. Assosiering mellom de deklarative og prosedurale aspektene betyr ofte dekomponering av store prosedyrer, utviklet til å behandle et av problem målene, til mindre funksjonsenheter. Nybegynnere assosierer enkelte ganger hele greia med en enkelt klasse i stedet for å dekomponere prosedyren, selv om dette ikke går etter objektorienteringens ortodoksi.

2.2. Arbeidsmetoder.

Det er skrevet mye litteratur om hvordan studenter jobber og hvilke metoder de bruker når de løser oppgaver. En del av litteraturen baserer seg på hva slags metoder studenter bør løse oppgaver på med å gjøre det best mulig. Jeg vil se på de enkle og observerbare aktivitetene ”lese oppgave”, ”lese kode”, ”taste kode”, ”kompilere” og ”kjøre program”. Kan man, ut i fra disse aktivitetene, si noe om studentene bruker noen form for arbeidsmetode.

2.2.1. Problemløsning i introduksjonskurs i informatikk.

Artikkelen av (Barnes et al. 1997) er basert på ”How To Solve It” av George Polya (1957), som har hatt stor påvirkning i matematisk utdanning. (Barnes et al. 1997) vil hjelpe studenter, som er i første året i sin utdannelse innen informatikk, som har problemer gjennom hele begynner kurset. De vil særlig hjelpe de studentene som sier noe sånt som ”Jeg vet ikke hvordan jeg skal starte med denne oppgaven..” De overfører ideen Polya har til en programmerings kontekst.

Polya laget en tabell som viser en fire stegs prosess (eller mer en syklus) for løsing av problemer. De har spesifisert en liknende sammendrag av metoden som de kaller ”How To Program It”. Denne tabellen er strukturert i fire faser som er Forståelse, Design, Skrive og Refleksjon.

I Forståelse oppfordres studentene til å forstå problembeskrivelsen godt, slik at de forstår fullt ut hva den sier. Studentene skal spørre ”Hva om?” spørsmål for å utfordre områder som er

uklare. Dette gir en samling med input og ventede output som ikke bare hjelper til å avklare tenkingen, men gir begynnelsen til en mengde test data som kan brukes senere i prosessen.

I Design skal den pedagogiske vekten legges på å se etter relaterte problemer som de allerede har løst eller liknende problemer som de er familiære med allerede. Dette passer godt med måten å undervise på hvor man bruker oppgaver til å oppmuntre til å bruke tanker rundt det nåværende kurs materialet, dette for å sikre at oppgavene gir maks nytte til studentene. Her kan de bruke deres input og output for å sjekke komplettheten og konsistensen i designet.

Skrive steget involverer man det å gjøre et design til et program som fungerer. Her er det nødvendig å fokusere på implementasjons språket, de tidligere stegene kan stort sett være språk uavhengige. Hvis forenklede versjoner av problemet har vært fokuset i design steget kan disse bli utført gjennom implementasjonen og fungere som bygge blokker, eller skisse løsninger, for større problemer er verdien av å få selvtillit ved i hvert fall å få noe resultat noe som ikke bør minimaliseres.

Slutt steget Refleksjon som bruker tid på reflektering og tilbakeblikk på produktet en gang etter at leveringsfristen er passert. Målet her er å konsolidere læringsprosessen og det å sette pris på det man har lært. Her burde man bygge opp erfaring man kan bruke i fremtidige design faser.

De gir også studentene spesielle språk spesifikke råd i tabell form. Det gir muligheten til å gi eksempler for å illustrere designet og programmeringsstegene i prosessen og diskutere programmeringsprosessen i en mer spesifikk måte.

Dette ble prøvd ut på et kurs på tre sesjoner i sommer semesteret i 1996 og de så effektiviteten ved at de som hadde problemer klarte å bestå eksamen. Dette førte til at de endret på fire av kjerne kursene i første året i studiet ved å innføre denne modellen, de fire stegene, i kursene.

2.2.2. Det å lære å programmere.

I doktoravhandlingen til (S. Booth 1992) studerte hun introduksjons studenter og hun fant ut at det er tre måter studenter forstår programmering på. Den første er ”Programmering som en maskinorientert aktivitet”, den andre ”Programmering som en problemorientert aktivitet” og den siste ”Programmering som en produktorientert aktivitet”. Den første er at man oppfatter programmering som en aktivitet som fokuserer på maskinen, den andre er at man har hovedfokus på problemet som programmeringsaktiviteten er ment for å løse, i stedet for maskinen selv og til slutt den siste som er at man har hovedfokus på programmet som et produkt, på den måten at programmering er den aktiviteten som produserer et program for potensielle brukere og som også skal kunne bli brukt til vedlikehold av andre programmerere.

Studenter forstår læring av programmering på fire måter. Disse fire måtene er ”Lære et programmerings språk”, ”Lære å skrive et program i et programmeringsspråk”, ”Lære å løse problemer i formen av et program” og ”Bli en del av et programmerings samfunn”. De to første måtene har med kode ferdigheter, den tredje måten må studenten klare bruke ferdighetene på rett måte når de konfronteres med et problem som ikke blir uttrykt i

programmerings termer. Den siste måten er å anse seg som et medlem av et programmerings team som også involverer brukere.

Booth fant ut at studenter løser lærebok oppgaver på fire forskjellige måter. Disse måtene er: ”Middel” hvor fokuset er på å produsere et komplett program fra begynnelsen ved å ta bruk av et eksisterende program eller ved å tilpasse noen kjente program til kravene i problemet, ”Konstruksjon” hvor fokuset er på at man gjenkjenner detaljer i problemet ved kjennetegn i programmeringsspråket som kan brukes til å lage et program, ”Operativ” hvor fokuset er på at man redegjør for operasjonene som programmet skal gjennomføre, deretter skriver de programmet og ”Strukturell” hvor de først redegjør problemet innen dets eget domene, deretter strukturering og til slutt koding.

Studentene beveget seg generelt mot mer avanserte nivåer, og de som ikke gjorde dette stod sjelden i kurset. Denne observasjonen impliserer at undervisningen burde trene studentene i å redegjøre problemet innen dets domene og strukturere programmet. Likevel er det ikke nødvendig at redegjørelse og strukturering hjelper studentene til å overkomme deres start problem. Uten å vite delene et program kan bli konstruert av og de vanlige mønstrene for å sette de sammen har ikke studentene noen mulighet til å strukturere programmet.

2.2.3. Utforsking av didaktiske modeller.

I starten av denne diskusjonen i (Kaasbøll et al. 1998) blir artikkelen over (Barnes et al. 1997) diskutert. Det blir påpekt at siden deres modell legger vekt på design og skriving og i tillegg introduksjons analyse og konkluderende evaluering vil denne modellen være enda bedre tilpasset studenter som ser hvordan læringsmålene relateres til læringen. I problemløsningsmetoden kan man inkludere menneskelige og organisatoriske temaer i problemene, som skal løses, og i metodene, og derfor kan det bli inkludert i undervisningen.

En innvending mot modellen er at problem løsningsmetoden likner vannfallsmodellen til program utvikling og dermed hinter om at programmer er utviklet uten iterasjoner og faser. Barnes åpner for at strategien til programmeringen i deres modell er mer av ”hackete” måte ved kopiering og tilpassing enn en ”strukturert” måte ved stegvis raffinering. De unngår dette ved å si at studenter som ikke vet hva løsningen kan være kan ikke bruke en strukturert metode.

Av dette konkluderer Kaasbøll at prosessen man har for å lære seg et program er forskjellig fra software utviklingsprosessen, dermed trengs det gjennomtenking av software utviklingsmodeller i undervisningen.

Kaasbøll refererer også til (Booth 1992), men han konkluderer med at Booth mener at når det gjelder programmer er kommunikasjon mellom programmerere en mer avansert nivå av kompetanse. Korresponderende er forståelse av programmer som kommunikasjon mellom programmerere og brukere er en annet nivå av forståelse. Det kan sies at læring er hacking, mens programmering i en strukturert måte betyr at du ikke har mer å lære.

2.2.4. Hva skal jeg finne ut?

Jeg vil finne ut hva slags arbeidsmetoder, det vil si om studentene bruker noen form for strategi når de løser oppgaver i objektorientert språk, studentene tar seg bruk av. Hvordan jobber de og hva gjør de? Finnes det noen sammenheng mellom hva slags arbeidsmetoder studentene bruker og nivået de er på som programmerere? Hvordan kan måten disse studentene jobber på passe inn med arbeidsmetodene til (Booth 1992) og (Barnes et al.1997). Kan det være noen spesielle aktiviteter lærer bør oppmuntre til under læringen for at studentene skal komme best ute av det?

2.3. Domene resonnering og relasjoner.

I (Kaasbøll et al. 2004) prøver de å finne ut hvordan studenter får tak i de objektorienterte konseptene og hvordan deres forståelse påvirker deres læring av programmering og hvilke temaer studentene har problemer med. I denne (Kaasbøll et al. 2004) har de laget et bilde av hvilke domener og relasjonene mellom disse domenene studentene fokuserer på.

2.3.1. Læring av objektorientert programmering.

Studentene som er med i undersøkelsen i denne artikkelen er fra et introduksjonskurs med Java på universitetsnivå, de fleste har ikke vært borte i noe særlig programmering før. Sekvensen de lærer på er som følger: variable, kontroll strukturer, metoder, array, fil operasjoner, grafiske interfaces, klasser, objekter og pekere, objekt mengder, arv, sub- og superklasser, abstrakte klasser og unntaks håndtering. Disse studentene må gjennomføre fem obligatoriske oppgaver gjennom semesteret. Studentene ble observert i åtte timer under forelesninger og noen av disse studentene igjen ble observert i ca 20 timer på gruppe timene. Observatørene to notater i løpet av tiden de tilbrakte med studentene, ved noen tilfeller hjalp observatørene studentene med oppgavene dermed fikk de muligheter til å prate med studentene om programmene de lagde og dermed ble det tatt færre kommentarer under denne veiledningen.

Når studenter skal utvikle objekt orienterte applikasjoner må de ta en del domener i betraktning. Disse er ”det virkelige verdensdomenet” som applikasjonene skal tjene, ”objektene” som programmet vil generere i løpet av kjøringen, ”samspillet” med det kjørende programmet når de kjører et syntaktisk riktig program og ”koden” med klasse beskrivelser som må skrives og diskutert. Å holde disse temaene fra hverandre kan være trivielt for en erfaren programmerer, som kan se bort i fra interface’s og objekter i perioder og kan konsentrere beskrivelser av den virkelige verden. Dermed ble det observert blant noen av studentene at de ikke hadde noen klar formening om objekter gjennom programmeringen og dermed lagret de alle forrandringer til programmet på fil.

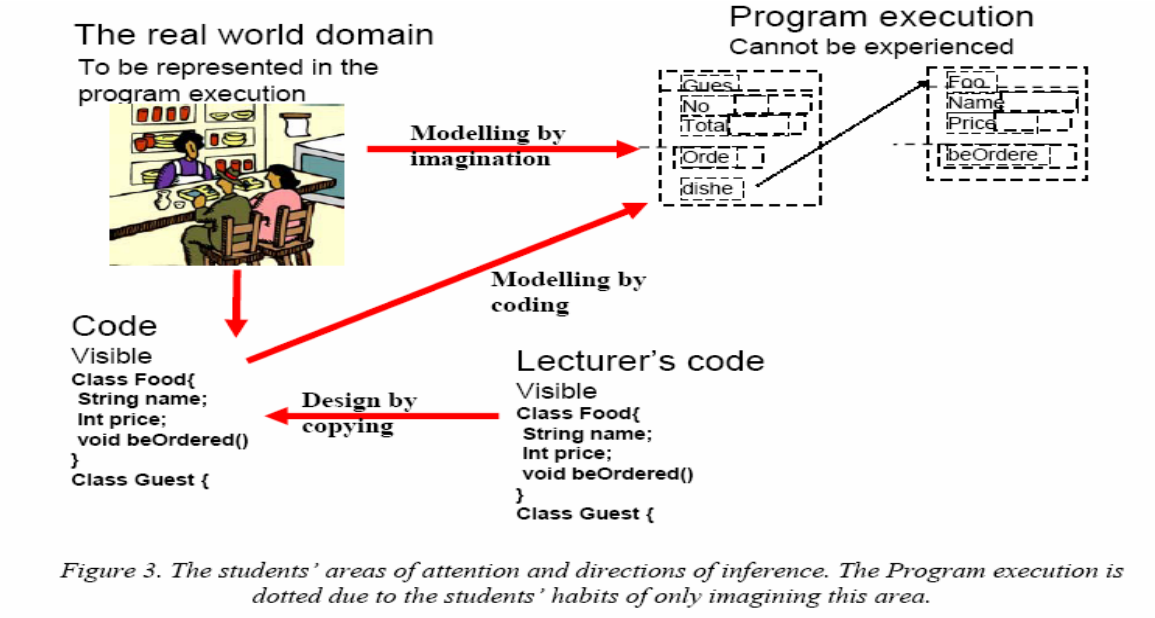


Figure 3. The students' areas of attention and directions of inference. The Program execution is dotted due to the students' habits of only imagining this area.

Domenene disse studentene fokuserte på er "det virkelige verdensdomenet", "program utførelse", "kode" og "lærers kode" som vist i figure 3. Dette er disse områdene innen programmeringsomgivelsene studentene i dette forsøket fokuserer på.

Det ble observert at studentene lagde design modeller i stedet for modeller av den virkelige verden, og det er ikke overraskende da de konseptene de hadde lært var implementasjons orientert. Av og til tegnet studentene modeller men brukte dem ikke eller at de ikke klarte å overføre dem til kode. Samtalene om array'ene og pekerne de lagde hadde ikke grunnlag fra tegningene, men heller fra den synlige koden på maskinen. Denne type modellering kalles "modellering ved fantasi". Når studenter som bruker denne type modellering spurte gruppelæreren om hjelp, prøvde denne læreren og tegne hva den trodde var modellen til studenten og ga svar i forhold til den modellen.

De studentene som ikke klarte å kode i forhold til modeller og har bare mulighet til å arbeide nedenfra og opp fra kode til program, dette kaller vi "modellering ved koding". Dette er omvendt vei fra det de mer erfarne studentene gjør. De studentene som dermed lager kode ved å kopiere programmer fra lærere og modifiserer programmet kalles "design ved kopiering".

Objekt orientering gir to utfordringer når man skal lære program design. Det ene er at studentene må lære seg å kunne lage modeller av virkeligheten, som objektene deres skal representere. Det andre er at studentene må lære hvordan de skal designe de andre komponentene i programmet, som interface og fil behandling og hvordan disse skal forholde seg til modellen av den virkelige verden. Studentene i undersøkelsen demonstrerte at de drev med hasardiøs programmerings utvikling ved å se bort i fra virkelighets sjekker og ved å kode uten visuelle modeller.

Ved å redusere antall områder studentene skal være fokusert på kan man få studentene til å lære seg de grunnleggende kunnskapene i forhold til programmeringen. Ut i fra at studentene i undersøkelsen hadde hatt prosedyremessig programmering fra før ser det ikke ut til å ha påvirkning på deres forståelse av objekt orientering.

2.3.2. Hva skal jeg finne ut?

Jeg vil finne ut hva slags domener de arbeider innenfor og hvordan de tar hensyn til de andre domenene (relasjonene) når de arbeider i et av dem. Finnes det noen sammenheng mellom hva slags domener og relasjoner studentene bruker og nivået de er på som programmerere? Hvordan er sammenhengen mellom domenene og relasjonene i artikkelen til (Kaasbøll et al. 2004) og domenene og relasjonen studentene i dette eksperimentet brukte? Er det noe spesielt lærerne bør oppmuntre til under læringen for at studentene skal komme best ute av det?

2.4. Undervisningsmetoder i objektorientert programmering.

I dette kapitlet vil jeg se på artikler som tar for seg forskjellige måter å undervise objektorientert programmering. Ut i fra disse artiklene og det jeg har sett på i denne oppgaven skal jeg se om jeg kan komme fram til hva lærere bør undervise.

2.4.1. Å starte med objektorientert programmering.

I (Borge et al. 2003) sier de at man må definere objektorienterte konsepter slik at man vet hva som er objektorientert tenkning, for å vite om hva man skal legge vekt på om man skal undervise objektorientert programmering først. Det er nevnt i tidligere litteratur at ”abstraction”, prosessen hvor man drar ut den essensielle informasjonen om virkeligheten, ”encapsulation”, det å gruppere data sammen med operasjonene som skal utføres på det, ”polymorphism”, overbelastning eller tilsidesetting av tidligere definerte metoder og ”inheritance”, det at klasser kan arve informasjon og oppførsel fra foreldre klassene, er de viktigste objektorienterte begrepene. De nevner at de objektorienterte konseptene kan undervises i den rekkefølgen læreren syntes passer best, men det betyr ikke at studentene vet hvordan de skal bruke dem.

De nevner at noen måter å lære på fører til at programmet ikke lager output, og uten output fra programmet vil ikke studentene motta feedback fra maskinen angående koden de skrev og dermed blir læringens hovedstimuli borte. KarelJ er en objektorientert Java omgivelse som fungerer godt for illustrering av ”encapsulation”og ”inheritance”, men også ”polymorphism” og for det tilfellet ”abstraction. I KarelJ blir alle objektene studentenes program generer synlige. Fokuset blir tidlig på noen av de grunnleggende objektorienterte konseptene. Det negative med KarelJ er at man ikke læres hvordan man løser et problem som baserer seg på den virkelige verden.

Robocode er et programmeringsdomene begrenset til et skytespill mellom tankser, hvor små sekvenser av Java kode kan lages slik at det blir et komplekst program. Det baserer seg på aktive objekter, som er en entitet som kan stå alene med en egen oppførsel og som kan påvirke andre objekter på forskjellig måte avhengig av situasjonen. Robocode kan vise ”inheritance”, ”encapsulation” og ”polymorphism”.

For å modellere verden kan man bruke programmeringsomgivelsen BlueJ, her trenger man bare de grunnleggende objektorienterte konseptene. Man kan inspisere objektene ved å dobbeltklikke på dem og hver metode til et objekt er ledig for trigging gjennom en meny. Dette gir den input og/eller output som er nødvendig for å håndtere modellen. Det som er nyttig med å modellere verden er at studentene skjønner at programmet skal være brukbar på en eller annen måte og at programmering bør være basert på kundenes ønsker.

Hvilken metode man skal bruke i undervisningen må velges ut i fra hvilket mål man har med undervisningen og hvilke kriterier man har til å avgjøre studentenes innsats. De mener om målet er å få studentene til å stå på eksamen, hvis den inneholder ”event” håndtering, kan det være lurt å undervise i Robocode eller noe liknende. Men for å vite hva som er best å undervise i må man følge studentene under og etter et introduksjonskurs for å få bedre innsikt i effekten til de forskjellige metodene, som vil gi læreren bedre grunnlag for valget av metode.

2.4.2. Objektorientert programmering som en modellerings metode.

(Groven et al. 2003) syntes at lærebøker og kurs er dårlige på grunn av mangel på et forenende perspektiv og mangelen på en pedagogisk metode som fokuserer på modellerings-(eksperimentell/utforskende) aspekter ved objektorientering. De starter heller med eldre proseduralorienterte perspektiver innen det objektorienterte språket før de introduserer de grunnleggende objektorienterte konseptene. De mener at objektorientert programmering bør sees på som modellering og at de grunnleggende konseptene av objektorientering bør undervises tidlig i studiet.

De nevner de den skandinaviske objektorienterte modelleringstradisjonen starter med ”tilstrekkelig komplekse programmer” og nødvendiggjør introduksjonen av alle de grunnleggende objektorienterte konseptene fra starten av. Modellering handler om å lage en beskrivelse av fenomener og konsepter fra et gitt applikasjonsdomene, for eksempel den delen av verden man er interessert i, men ikke vet nok om ved bruk av objektorienterte perspektiv eller paradigme. Fenomenene og konseptene beskrives som klasser og objekter. Klassene er termene som man bruker for å beskrive modellen, de former språket som modellereren bruker. En modell er en abstraksjon av et eller annet, med den hensikt å forstå det.

Modelleringsprosessen er utforskende og kreativ prosess som bør utføres i grupper. En av fordelene med objektorientering er at den gir et integrert perspektiv på analyse, design, modellering og implementasjon.

3. Kurs, studenter og faginnhold.

Studentene som er med i dette eksperimentet er studenter som går på Universitetet i Oslo og tar begynnerkurset i objektorienterting. Dette kurset heter inf1000 – grunnkurs i objektorientert programmering og er et kurs som kan taes både i vår og høst semesteret.

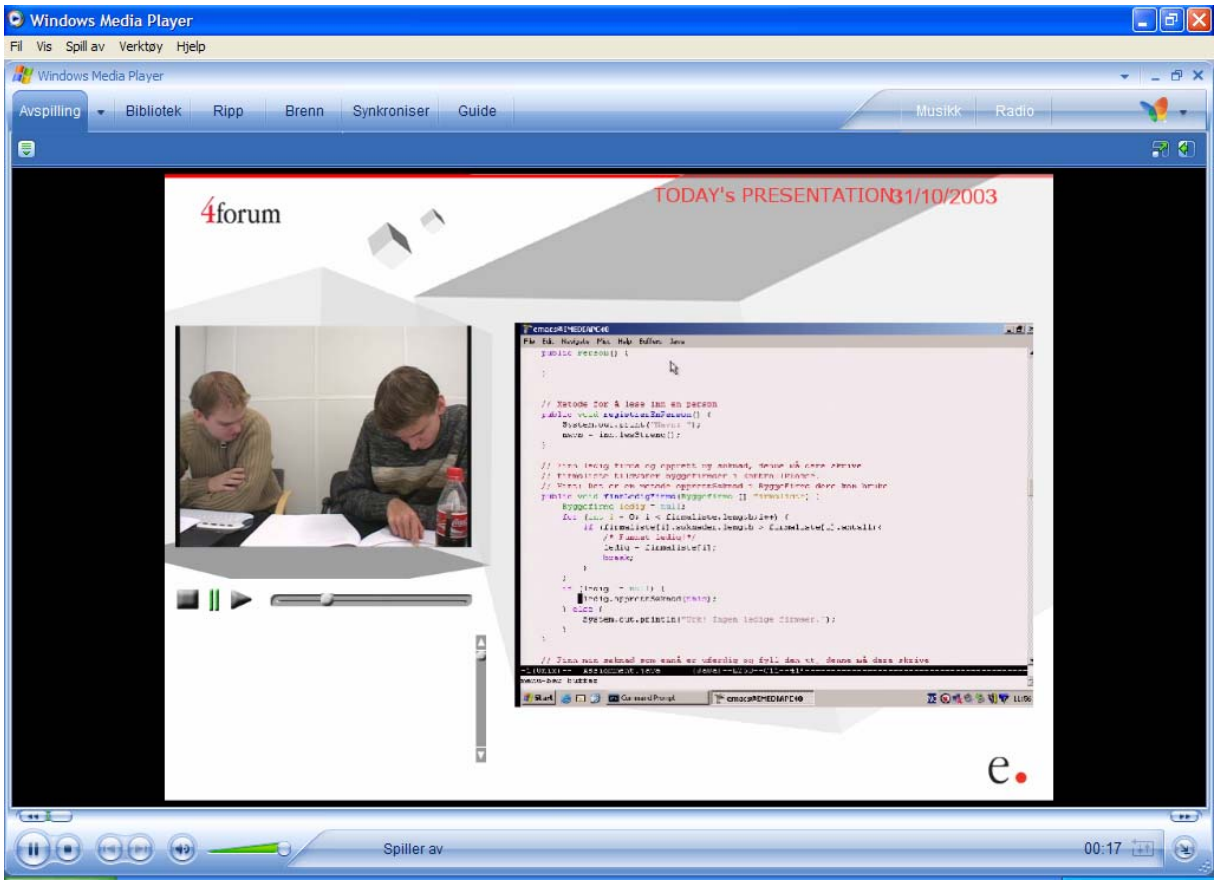
Innholdet i kurset og hva man skal lære forklares i teksten under og er hentet fra hjemmesiden til inf1000 (http://www.uio.no/studier/emner/matnat/ifi/INF1000/): En første innføring i objektorientert programmering i Java og utvikling av algoritmer, inkludert bruk av arrayer og andre typer tabeller, kommunikasjon med bruker og filhåndtering. Flere sentrale mekanismer i objektorientert programmering vil bli gjennomgått, inkludert bruk av klasser, objekter og referanser. Samspill mellom objekter av noen få klasser. Bruk av Javas klassebibliotek. Modellering i UML av små systemer (klassediagrammer). Konsekvenser av IKT. Personvern. Studentene skal etter endt kurs være i stand til å utvikle egne programmer i Java ut fra en enkel problembeskrivelse. De skal ha tilstrekkelig oversikt og forståelse over klassebiblioteket i Java til at de selv kan finne nødvendig informasjon. Studentene forventes å nå et nivå hvor de vil kunne sette seg inn i andre, tilsvarende programmeringsspråk.

Undervisningen i inf1000 består av to timer forelesning og fire timer gruppe undervisning, to av gruppe timene er i klasserom og de to andre er på terminalstuer. Forelesningene holdes i store forelesningssaler og det er ikke pliktig oppmøte på forelesning, så studentene må selv passe på å få med seg forelesning eller det som ble forelest på en forelesning de ikke møtte opp på. Alle forelesningsfoiler som gåes igjennom på forelesning legges ut på internett, slik at man kan lese over materialet i ettertid. Alle gruppetimene veiledes av en gruppelærer som selv er student på informatikk studiet på Universitetet i Oslo. I klasserommet får studentene mulighet til å jobbe sammen og gå igjennom oppgaver på tavlen, det er mulighet for å stille spørsmål til og diskutere oppgavene de løser. På terminalstuene sitter studentene på maskiner og jobber med oppgaver, de har her mulighet til å få hjelp av gruppelæreren til å løse problemer de har med oppgavene på datamaskinen.

Gjennom semesteret studentene tar inf1000 må de levere inn 4 obligatoriske oppgaver og få disse godkjent av gruppelærer for å kunne gå opp til eksamen. De to første obligatoriske oppgavene har lett eller middels vanskelighetsgrad og er ikke alt for store i omfang, men de to siste er det større vanskelighetsgrad på og disse er to større oppgaver. Studentene i forsøket har levert inn de tre første obligatoriske oppgavene og det er rett før de skal begynne på den siste obligatoriske oppgaven. Studentene er dermed ferdig med ca 70 % av kurset på det tidspunktet de er med i forsøket.

4. Metode.

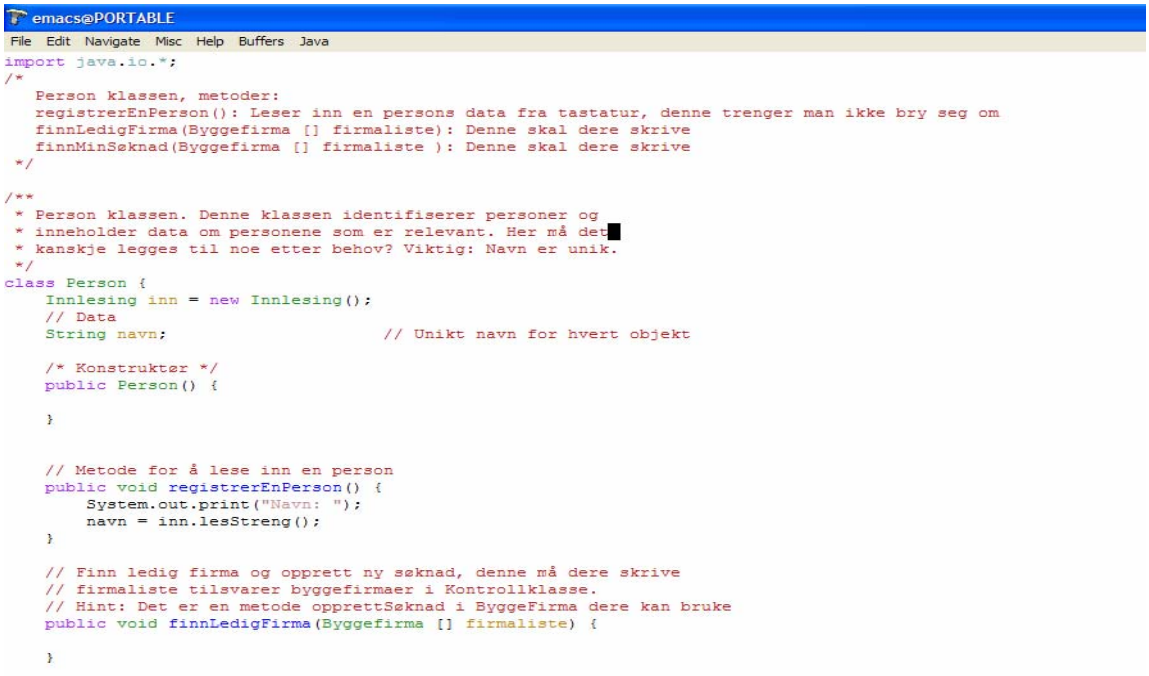
Høsten 2003 ble det gjort et eksperiment med åtte studenter, de ble puttet sammen to og to hvor de fikk noen oppgaver de skulle løse. Det ble tatt opp film og lyd av studentene mens de jobbet, og i tillegg ble det tatt opp film av det studentene gjorde på dataskjermen. Det er laget en total ”film” av dette hvor lyd, bilde og skjermbildet er synkronisert sammen i ett bilde, som vises i bilde 1 under. Jeg kan hele tiden velge å trykke på bildet av studentene eller koden for å få dette over hele skjermen slik at bildet blir større og slik at man ser bedre hva som skjer.



Bilde 1: Et illustrerende bilde av det jeg ser på skjermen min når jeg ser filmene.

De fire forskjellige gruppene var innkalt til eksperimentet på hvert sitt tidspunkt. Det ble satt av ca 4 timer til hvert forsøk med hver gruppe. Filmene er derfor ca 4 timer lange, men studentene tar noen pauser og noen av gruppene har noen tekniske problemer som gjør at den totale tiden med film som jeg har analysert er mindre enn fire timer per film.

Kort fortalt handler oppgaven om å lage et program, som brukes av mennesker som vil bygge hus, og som finner firmaer til hvert oppdrag som kan bygge husene. Programmet skal også ordne med søknader for bygging av hus til firmaene og som skal ordne godkjenningene og kostnadene til disse søknadene. Det meste av koden er ferdig kodet, mens de metodene som har noe å si for det oppgaven skal gå ut på står tomme og det er studentenes oppgave å fylle disse ut med en kode som fungerer, se bilde 2 under. Selve oppgave teksten og resten av koden kan man lese i appendiks kapittelet bakerst.



Bilde 2: utdrag av koden til studentene, med en ferdig og en tom metode.

Jeg startet med å se gjennom alle filmene og ta notater på hva som skjer og når det skjer. Etter dette kategoriserte jeg disse notatene på tre måter. De tre forskjellige måtene er basert på ”Java og andre problemer”, ”arbeidsmetoder” og ”domene resonnering og relasjoner”. Deretter så jeg igjennom filmene en gang til mens jeg tok flere notater. Dette gjorde jeg for å finne ting studentene hadde problemer med og for å kunne ta å utforske visse temaer nærmere. Jeg fant ut hvilke tidspunkter og temaer i de forskjellige filmene som var verdt å utforske mer. Deretter tok jeg å så disse sekvensene på nytt hvor jeg skrev ned hva disse studentene sa og gjorde, dette for å få fram studentenes perspektiv i oppgaveløsningen. Dette er en meget krevende jobb da studentene til tider snakker i munnen på hverandre og de mumler. Jeg har dermed ikke fått skrevet av alt det de sier helt ordrett men jeg har prøvd i det lengste å få det så riktig som mulig.

Jeg har også studert filmene og tatt ut noen eksempler som viser hvordan studentene sliter. Disse eksemplene gir oss en pekepinn på hvilke områder studentene sliter med. Jeg beskriver hva de gjør og hva de sier, dette for å få fram alt som kan beskrive hvordan studentene sliter. Når jeg vurderte problemene studentene hadde baserte jeg ikke det på hver enkelt gruppe i dette forsøket, men generelt for alle studentene totalt sett i forsøket. Der valgte jeg ut problemer som viste seg flere ganger hos flere av gruppene. Dermed fikk jeg plukket ut representative problemer som studentene hadde.

Deretter samlet jeg dette i et dokument hvor jeg skriver alt som skjer utfyllende slik at man får med hva som skjer og hvorfor de sier det de sier. Den første gruppen som ble filmet ble et slags forsøk for utførelsen av eksperimentet, hvor det var en del problemer underveis og dermed har jeg valgt å se bort i fra denne gruppen da det ikke gir noe godt inntrykk av hva disse studentene har problemer med. Dette på grunn av at mange av studentenes problemer er frembrakt av lærerne og deres kommunikasjons problemer. Det viste seg at læreren som hadde kodet programmet trodde studentene skulle gjøre noe annet enn læreren som hadde skrevet oppgave teksten, og dermed ble studentene avbrutt i tankegangen og de måtte begynne på nytt med nye oppgaver.

Etter at jeg hadde fått dokumentert det viktigste av det studentene gjør og sier, gikk jeg gjennom én av oppgavene studentene skulle løse, som vises i filmene, for alle de resterende gruppene. Dermed gikk jeg gjennom de sekvensene i de forskjellige filmene og noterte ned hva slags aktiviteter de holder på med og hvor lenge de holder på med hver aktivitet. Dette for å se forskjellen mellom de forskjellige studentene og se hva de som lykkes gjør i forhold til de som har mer problemer. Deretter laget jeg noen tabeller som viser hvor mye tid de bruker på disse aktivitetene og hvilke aktiviteter de er innom og hvilken rekkefølge de kommer i.

Jeg prøvde å analysere filmene og finne ut hvilke domener i artikkelen (Kaasbøll et al.2004) som passet inn for studentene i dette eksperimentet. Ved å ha domenene fra artikkelen i tankene og hvis jeg så det hensiktsmessig å lage andre/flere domener gjorde jeg det ut i fra hva studentene holder på med.

Relasjonene mellom disse forskjellige domenene jeg fant, forklares her: ”modellering ved koding”: studentene koder koden basert på hvordan programmet er sammensatt og hvordan det skal funger. ”Oppgavetekst sjekking”: studentene prøver ut koden de har laget for å se om den gjør det oppgaven har skrevet at de skal gjøre. ”Design motivert av oppgavetekst”: studentene lar oppgaveteksten veilede dem til hva de skal skrive i koden og ikke den koden som allerede er skrevet. ”Fil sjekking”: studentene leser filene for å forstå koden bedre, slik at de vet hva de forskjellige ord og tall i filene betyr og kan dra nytte av dette. ”Design ved kopiering” studentene lar andres kode påvirke og være et utgangspunkt i sin egen kode. ”Virkelighets sjekking”: studentene har behov for å forstå modellen i den virkelige verden og sjekker ut det i modellen de trenger å forstå med den virkelige verden. ”Justering av modellen”: studentene har behov for å endre på hvordan modellen ser ut for å få en kode som fungerer til det den skal. De forskjellige bildene av hvordan disse domenene og relasjonene henger sammen finner man i kapitlene 5, 6 og 7.

Overgangene mellom disse relasjonene er ikke bråe, men de er gradvise. Det er enkelt å se når studentene tar seg bruk av kopiering for da ser man at de peker på arkene med kode og sier at man kan ta seg bruk av det og begynner å skrive det inn. Etter at de har skrevet inn det de vil kopiere fokuserer de tilbake på ”modellering ved koding” for å se hvordan det de kopierte passer inn i modellen. ”Oppgavetekst sjekkingen” er lett å merke overgangen til, det er den delen der de sjekker, ved å kjøre programmet, om programmet gjør det oppgaven sier at det skal gjøre. Overgangene mellom ”modellering ved koding” og ”design motivert av oppgavetekst” er vanskeligere å se, det jeg har lagt vekt på er at når de koder modellen har de fokus på hvordan alt skal henge sammen i en helhet, mens når de designer ved hjelp av oppgavetekst så har de fokus på det oppgaveteksten sier og bare det og prøver å løse oppgaven med det som grunnlag. Overgangen til ”fil sjekking” er lett å se da de begynner å prate om hvordan filene ser ut og leter dem fram og snakker om hvordan den ser ut. Deretter diskuterer de hvordan filen er satt sammen, og finner ut hva alt betyr. Når de har diskutert det de skal i filen fortsetter de med det de gjorde før de sjekket filen.

Etter at jeg hadde funnet ut hvilke domener studentene fokuserer på og hvilke relasjoner mellom dem de fokuserte på noterte jeg ned hvor mye tid de brukte på hver av disse relasjonene. Dette gjorde jeg for alle de tre gruppene slik at man kan sammenlikne disse gruppene. For å kunne sammenlikne dem enda bedre laget jeg kakediagrammer for å se totalt forbruk av tid på de forskjellige relasjonene. Av dette kan man se hvor mye tid i forhold til hverandre de bruker på de forskjellige relasjonene. Som et avsluttende diagram laget jeg et

graf diagram med brukt tid for de forskjellige gruppene slik at man ser de i samme diagram og ser forskjellen i brukt tid.

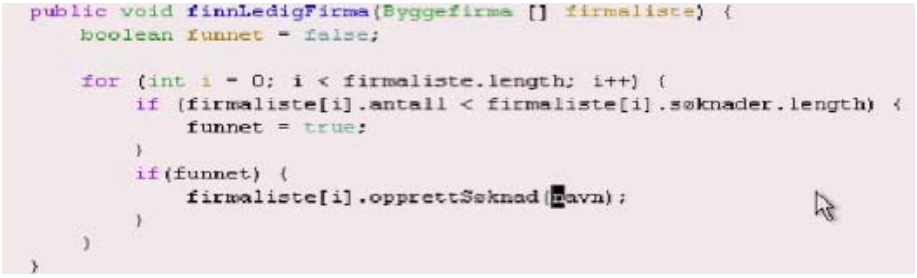
Deretter analyserte jeg disse eksemplene og diagrammene og sammenlignet gruppenes resultater og diagrammer. Dermed kunne jeg basere diskusjonen og konklusjonen på dette.

5. Java og andre problemer.

5.1. Person objekt og String navn.

Her vises et eksempel fra en av gruppene som var med i forsøket hvordan de ikke helt klarer å se forskjellen mellom et objekt av Person og String navn som er et attributt i klassen Person.

Det er et metodekall (firmaliste[i].opprettSøknad(navn)) i koden til studentene, som studentene har skrevet selv, som vist under i figur 4:



figur 4: utklipp fra video av studentenes kode i klassen Person.

Metoden opprettSøknad tar et personobjekt og den ser slik ut:

```
public void opprettSøknad(Person pers) {
    søknader[antall] = new ByggeSoknad(pers, this);
    søknader[antall].statusSøknad = "uferdig";
    antall++;
}
```

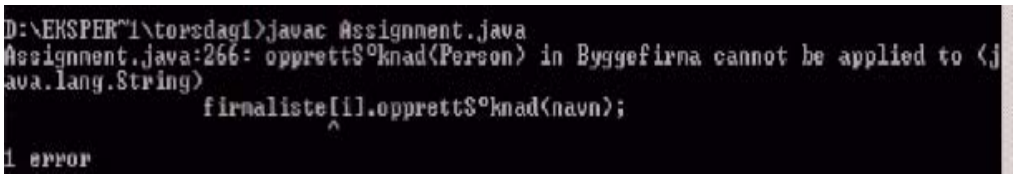
figur 5: utklipp av metoden opprettSøknad i klassen Byggefirma fra koden studentene fikk utdelt.

Under kommer samtalen mellom de to studentene når de skal løse problemet med hva parameteren i kallet til opprettSøknad skal være. Det illustrerer litt vanskeligheten studentene har med forståelsen av forskjellen mellom for eksempel et objekt av klassen Person og String navn som er et attributt i et objekt av klassen Person.

Her ser vi hvordan studentene prøver å finne ut hva parameteren skal være.

Student1: Jeg er usikker på den parameteren vi har puttet inn i...
Student2: Navn...

I figur 6 ser man feilmeldingen de får opp på skjermen.



figur 6: utklipp fra studentenes skjerm hvor de har fått opp en feilmelding etter kompileringen.

De lener seg veldig mot skjermen og stirrer intenst på denne feilmeldingen.

Student1: Han viser her men mener der, opprettSøknad(Person) in Byggefirma cannot be applied to (java.lang.String). Ehh… såå.. vi må vel ehh..
Student2: Skal vi bare skrive person?
Student1: Hva er pekeren i person objekt?

De stirrer på arket med koden de har fått utlevert og det blir stille en stund.

Student2: Den ligger.., assa her er jo den..
Student1: Nå skal vi vise til den personen som allerede har registrert seg, den som har skrevet navnet sitt og sånt.
Student2: mm…
Student1: Hvordan, hva kaller vi den for i parentesen her? Assa vi skal legge inn pers fra klassen Person.
Student2: Hva er dette for noe?
Student1: Ehh.. hvor lese.. registrerNyPerson() eller hva det heter, leslnnPerson() hvor har vi den metoden?
I kor: registrerEnPerson().

De peker på arket med koden mens de prater og diskuterer for å "bekrefte" det de sier.

Student1: Navn er inn.lesStreng(); Det er den vi skal ha så vidt jeg vet. Putt inn her.
Student2: Men hvor er innLes…
Student1: Vi er i, men vi er jo inne i, denne metoden er inne i person klassen, så..

Student1 blar i arket med koden for å lete etter det student1 lurer på.

Student2: Men hvor ligger den innlesningsgreia? Var det ikke en klasse som het innlesning?
Student1: Jo.
Student2: Public person leslnnPerson… person… pers…

De finner stedet på arket og det blir stille en stund. Så peker de på arket og skjermen mens de diskuterer videre.

Student1: Det er antall personer som ligger i arrayen. Hvilken metode vi kaller, ja det er denne metoden, navn lik inn.lesStreng, men vi henter denne, metoden vi lager i klassen..

En lærer kommer inn i rommet for å gi råd.

Lærer: Dere har kommet fram til at det er en parameter i opprettSøknad?
Student1: Ja.
Lærer: Hvilken type er den parameteren?
Student1: Det er jo en string.
Student2: Ja.

De ser litt forvirret ut og at det lærer sier er overveldende. Blikket veksler mellom lærer, arket med kode og skjermen.

Lærer: Har dere sett på definisjonen av metoden opprettSøknad?
Student1: Den er jo en Person pers, den er av klassen Person.
Lærer: Ja, og denne metoden dere lager er lokal i en klasse Person.
Student1: mm… riktig.
Lærer: Da kan du legge med deg selv, så du har en klasse som har en metode lokalt i seg og det er den dere lager nå.
Begge: mm…
Lærer: Og det den metoden opprettSøknad ber om er å få et Person objekt som parameter, da kan dere legge med seg selv.

Student1 ser på student2 med litt forvirring i blikket.

Student1: Er du med?
Student2: Nei, ikke på like, hva mener du med legge med seg selv?
Lærer: Legge med seg selv, det er noe som heter this.
Student1: this.navn.
Lærer: Har dere hørt om det?
Student1: Ja, har så vidt vært borte i det, har ikke brukt det.
Student2: Nei.

Blir litt stille, lærer snakker med noen utenfor, studentene tenker og de stirrer på arket med kode og skjermen.

Student1: Skal vi prøve det enkleste bare?
Lærer: Prøv this

Under ser man hva studenten skriver i koden etter denne hjelpen fra læreren:

```
if(funnet) {
    firmaliste[i].opprettSøknad(this.navn);
}
```

figur 7: utklipp fra koden til studentene i klassen Person i metoden finnLedigFirma.

En lærer kommer inn i rommet igjen når de ser hva studentene skriver.

Lærer: Ikke, ikke noe this.navn bare this.
Student1: This ja, kun this okay.

Deretter kompilerte programmet feilfritt.

Her ser vi et eksempel på at studentene har feil inntrykk av objekter, som her at de tror String navn er det samme som selve person objektet. De har ikke forstått at String navn bare er en tekst streng men objektet inneholder for eksempel denne tekst strengen men eventuelle flere attributter og variable.

5.2. Parameter og peker.

Her vises et eksempel hvor skal studentene skrive metoden finnLedigFirma(Byggefirma [] firmaliste) hvor de får arrayen firmaliste inn som en parameter. Firmaliste er den samme arrayen som byggefirmaer vist i figur 9. I figur 8 ser vi klassen Byggefirma og de attributter

og pekere som finnes i denne klassen. Dette eksemplet viser hvilke problemer studentene har med pekere og parametere.

```
 */
class Byggefirma {
    Innlesing inn = new Innlesing();
    // Data
    ByggeSoknad[] søknader;          // Array med søknader firmaet har inne
    String navn;                     // Navn på firma
    int antall;

    /* Konstruktør */
    public Byggefirma(String navn, int kapasitet) {
        this.navn = navn;
        søknader = new ByggeSoknad[kapasitet];
        antall = 0;
    }

    /* Oppretter en ny søknad, denne kan benyttes i oppgave 2
       pers er en parameter som peker på personen som kalte metoden*/
    public void opprettSøknad(Person pers) {
        søknader[antall] = new ByggeSoknad(pers, this);
        søknader[antall].statusSøknad = "uferdig";
        antall++;
    }
```

figur 8: utklipp fra koden studentene fikk utlevert, viser klassen Byggefirma med dets attributter og metoden opprettSøknad() som studentene skal ta seg bruk av i oppgaven de skal løse.

```
class Kontrollklasse {
   private Innlesing inn;          // Klasse som inneholder innlesningsverktøy
   Person [] husbyggere;           // Array med personer
   Person person;                  // Midlertidig peker til person
   Byggefirma byggefirma;          // Midlertidig peker til byggefirma
   Byggefirma [] byggefirmaer;     // Array med byggefirmaer
   Saksbehandler saksbehandler;    // Peker til en saksbehandler
   int antallPersoner;
```

figur 9: utklipp fra koden studentene fikk utlevert, viser klassen Kontrollklasse med dets atributter og pekere. Det er fra denne klassen studentene bruker pekeren byggefirmaer og byggefirma.

I figur 10 ser vi det første forslaget studentene har til hvordan metoden finnLedigFirma(Byggefirma [] firma) som ligger i klassen Person skal se ut, de kompilerer koden og får feilmeldinger de prøver å rette på.

```
public void finnLedigFirma(Byggefirma [] firmaliste) {
    boolean funnet = false;
    for (int i = 0; i < byggefirmaer.length; i++) {
        if (antall < kapasitet) {
            funnet = true;
        }
        if(funnet) {
            opprettSøknad(navn);
        }
    )
}
```

figur 10: utklipp fra studentenes kode, dette er metoden finnLedigFirma() som ligger i klassen Person.

De stirrer på feilmeldingene, som vist i figur 11, og leser dem høyt. Deretter prøver de å finne løsninger til disse feilmeldingene ved å foreslå ting de kan skrive i koden som kan rette feilene.

```
Command Prompt

D:\EKSPER~1\torsdag1>javac Assignment.java
Assignment.java:260: cannot resolve symbol
symbol  : variable byggefirmaer
location: class Person
        for (int i = 0; i < byggefirmaer.length; i++) {
                          ^
Assignment.java:261: cannot resolve symbol
symbol  : variable antall
location: class Person
            if (antall < kapasitet) {
                ^
Assignment.java:261: cannot resolve symbol
symbol  : variable kapasitet
location: class Person
            if (antall < kapasitet) {
                         ^
Assignment.java:265: cannot resolve symbol
symbol  : method opprettS°knad (java.lang.String)
location: class Person
                opprettS°knad(navn);
                ^
4 errors
```

figur 11: utklipp fra command promt vinduet til studentene.

Student1: Byggefirmaer finnes i klassen?

Student2: Der er byggefirma, stor B? Eller? Nei byggefirmaer.
Student1: Kanskje vi må rett og slett det ja.
Student2: Det var jo liten B på den andre, der vi fant den.
Student1: Jo men vi må ha en peker, konstruktør peker eller noe sånt. Jeg lurer på om det må se sånn ut faktisk.
Student2: Det gjorde det jo ikke på den..
Student1: Vi prøver og feiler.
Student2: Skal se der jeg fant, der er den, det er jo akkurat den der vi gjorde.
Student1: mm… Men det er kanskje at den metoden er inne, at vi nå er, skal vi se, hvilken klasse er vi inne i når vi lager denne metoden? Ehh..
Student2: Byggefirma er vi i.

En lærer kommer inn for å veilede studentene litt, han lurer på om de kan ha missforstått noen kommentarer som er skrevet i koden.

Lærer2: Det kan være en kommentar hos oss som kan være litt forvirrende.
Student2: Ja ha..?
Lærer2: Hvis dere ser her.

Læreren peker på skjermen, stedet han peker på er vist i figur 12, til studentene mens han forklarer hvordan de kan ha missforstått.

```
// firmaliste tilsvarer byggefirmaer i Kontrollklasse.
// Hint: Det er en metode opprettSoknad i ByggeFirma dere kan bruke
public void finnLedigFirma(Byggefirma [] firmaliste) {
```

figur 12: utklipp fra studentenes kode.

Lærer2: I koden står det at firmaliste tilsvarer byggefirmaer.
Student1: mm…
Lærer2: Det vil jo ikke være helt riktig å si for du ser her at grunnen til at dere får den første feilen er at den ikke kjenner igjen. Dere var på rett vei når dere skrev bare byggefirmaer men det var firmaliste det skulle stå for det er den som kommer inn.
Student1: Okay.
Lærer2: Det var, men jeg tror dere kan ha blitt forvirret av at det står tilsvarer byggefirmaer.
Student1: Ja, ja..
Student2: mm…
Student1: Firmaliste.
Lærer2: Så ja, det er grunnen til den feilen. Det kan hende at kommentaren var litt misvisende, derfor sier jeg i fra.
Student1: Eller at vi lette et annet sted eller..
Lærer2: Ja. Ehh..
Student1: mm.. flott, yes, takk da har vi den. Skal vi se.

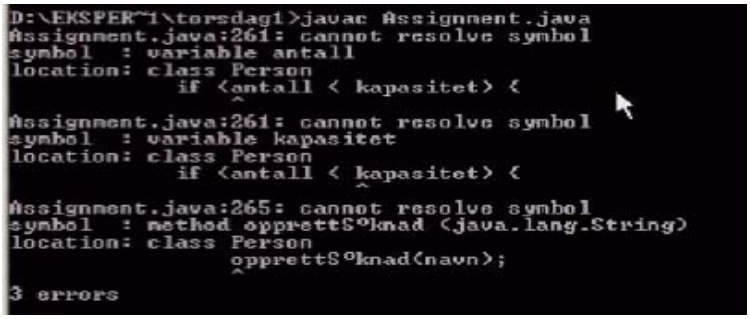
Det virker ikke ut som de forstår helt hva lærer sier, men de later som eller så tror de at de forstår det med å bruke firmaliste og ikke byggefirmaer. Skriver inn firmaliste.length, som vist i figur 13, i for- løkken i stedet for byggefirmaer.length.

```
for (int i = 0; i < firmaliste.length; i++) {
    if (antall < kapasitet) {
        funnet = true;
```

figur 13: utklipp fra studentenes kode i metoden finnLedigFirma().

Student2: Du må lagre, prøve den og se hva den sier nå.
Student1: At han kanskje nå godtar de andre nå.. ehe.. som var feil, det kan jo være.

Studentene lagrer og kompilerer koden, så ser de på feilmeldingene. Dermed er den første feilmeldingen borte. De diskuterer hva som kan være årsaken til resten av feilmeldingene. De har noen peker feil og noen variabel feil, som vist i figur 14.



figur 14: utklipp fra command promt vinduet til studentene, med feilmeldingene etter kompileringen.

Student1: Må vi her bruke pekere? Eller? Antall og.. de er jo også i Byggefirma.

Deretter diskuterer de mer om hvordan de skal få brukt disse variablene og hvilke variable de skal bruke i if- testen. De klarer ikke overbevise dem selv om hva som er den rette løsningen.

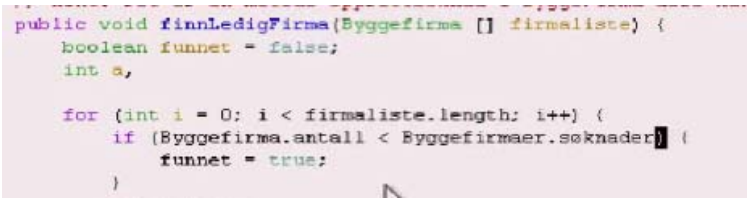
Student2: Men firmaliste da? Jeg skal se om jeg finner den firmaliste tingen.

Studentene har ikke skjønt eller fått med seg at firmaliste kommer inn som en parameter i metoden finnLedigFirma() og dermed vil de lete i arkene med koden etter det stedet de fikk firmaliste fra. De finner ikke stedet der de tror de fikk firmaliste fra. De prøver å skjønne hvor firmalisten kom fra.

Student1: Den stod som en parameter i..
Student2: listoppFirmaer(firmaliste)…

De blar videre i de samme arkene og sitter stille og tenker på hva de skal gjøre. De sier lite når de er usikre på hva de skal gjøre. De prøver å finne eksempler fra arkene med koden som de skriver ned på ark og som kan løse feilmeldingene de har fått opp.

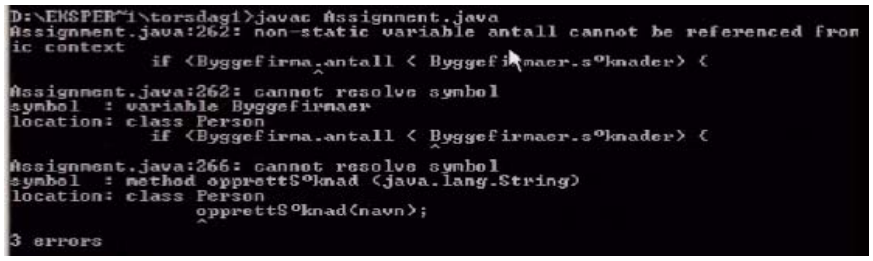
Student1: Jeg lurer på om jeg skal prøve med en sånn konstruktør, la meg bare prøve å se om de godtar det.
Student2: Ja.
Student1: Antall, dott søknader.
Student2: Lagre det. Se om de godtar det.



figur 15: utklipp fra studentenes kode.

I figur 15 ser man det neste forslaget studentene har til å lage pekere for å få tak i de variablene de vil teste på. De lagrer og kompilerer koden, som vi ser i figur 15 har de gjort en slurve feil med å deklarere int a og ikke avslutte med ”;”, og dermed fikk de noen feilmeldinger de ikke helt klarer å forstå. Læreren kommer inn og hjelper dem med å få bort

disse feilene, da ikke det er viktig å bruke tid på sånne feil. Deretter lagrer de på nytt og kompilerer igjen. De ser på feilmeldingene de får opp, som er vist i figur 16.



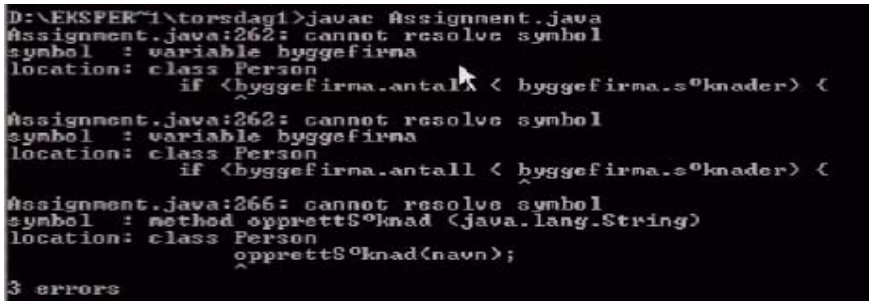
figur 16: utklipp fra command promt vinduet til studentene, med feilmeldingene etter kompileringen.

Student2: Nei.
Student1: Nei, han liker ikke de. Kanskje det skal være liten bokstav på byggefirmaer?

```
for (int i = 0; i < firmaliste.length; i++) {
    if (byggefirma.antall < byggefirma.søknader) {
        funnet = true;
    }
```

figur 17: utklipp fra studentenes kode.

De skriver inn forslaget med små bokstaver, som er vist i figur 17, lagrer løsningen og kompilerer den. Feilmeldingene de får opp er vist i figur 18, men de får ikke noe mer rett etter denne kompileringen. De tenker på hva de kan gjøre og kommer med et forslag om å lage en teller variabel som kan løse feilene de får opp. Så kommer læreren inn for å hjelpe dem inn på rett spor, da læreren har skjønt at de ikke forstår bruken av parameteren de får inn i den metoden de skal lage.



figur 18: utklipp fra command promt vinduet til studentene, med feilmeldingene etter kompileringen.

Lærer1: Skal vi ta en briefing?
Student1: Ja.
Lærer1: Hva er en parameter?
Student2: mm..

Man ser på studentene at de er usikre på hva en parameter er, eller at de ikke vet hva de skal svare på et slikt spørsmål.

Lærer1: Det er et eller annet navn som man kan dytte noe inn i en metode.
Begge: mm..
Lærer1: Da står det helt opp til deg selv å velge navn lokalt, hva du vil kalle den. Dere er veldig nærme når dere snakker om byggefirmaer, ikke sant? Men hva kalles parameteren i denne metoden?
Student1: Firmaliste. Ehh… ja.
Lærer1: Da er vi er lite stykke videre, å når vi snakker om array av objekter er det snakk om arrayer av adresser til objekter. Det er ikke objektene som blir dyttet inn..
Student1: Nei.

Lærer1: I arrayen, men referansene, som det så fint heter, som ligger i arrayen søknader for eksempel. Nå vil jeg ikke hjelpe mer.

```
for (int i = 0; i < firmaliste.length; i++) {
    if (firmaliste.antall < firmaliste.kapasitet) {
        funnet = true;
    }
```

figur 19: utklipp fra studentenes kode. Her skriver studentene firmaliste. og ikke firmaliste[i]. dermed får de feil i kompileringen.

```
D:\EKSPER~1\torsdag1>javac Assignment.java
Assignment.java:262: cannot resolve symbol
symbol  : variable antall
location: class Byggefirma[]
            if (firmaliste.antall < firmaliste.kapasitet) {
                           ^
Assignment.java:262: cannot resolve symbol
symbol  : variable kapasitet
location: class Byggefirma[]
            if (firmaliste.antall < firmaliste.kapasitet) {
                                           ^
```

figur20: utklipp fra command promt vinduet til studentene, med feilmeldingene etter kompileringen.

Læreren sier mye av det samme på nytt men på en annen måte og ber dem skifte navn for å komme videre, deretter går han ut av rommet igjen. Studentene skifter navnet på pekeren men glemmer [i] i array pekeren, som vist i figur 19, de får dermed feil etter de har lagret og kompilert koden på nytt, det vises i figur 20. De diskuterer litt om hva firmaliste er og det virker ikke som de forstår at det er pekere til Byggefirma objekter i denne arrayen. De har ikke forstått sammenhengen mellom firmaliste og byggefirmaer som ”identiske”, men den ene er en lokalt navngitt array i den metoden de er i og den andre en global navngitt array som ligger i en annen klasse enn det de er i. På nytt kommer læreren inn igjen for å hjelpe studentene på rett spor.

Lærer1: Dere er nesten der men dere føler ikke at dere er der. Ehh.. ehh.. I den metoden dere har laget har dere laget en teller som dere kaller i.
Begge: mm..
Lærer1: Og den telleren går på..
Student1: Byggefirmaer.
Lærer1: Antall elementer her.
Begge: mm..
Lærer1: Og dere er nesten der, som jeg sier, og det har med litt at en array har mange elementer, og dere kan ikke se på alle elementene samtidig, men den telleren som der har i løkka deres går jo for hvert element.
Student1: Ja, ja.
Lærer1: Og et byggefirma er jo et element.
Student1: mm..
Lærer1: Som har en array inni seg igjen som er søknader. Er dere litt på..?
Student1: Jeg vet hva neste ting jeg vil prøve.
Lærer1: Ja prøv det, så vil ikke jeg si mer, så kan dere diskutere hva dere prøver.

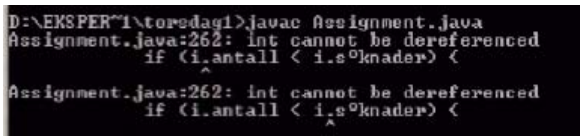
Lærer forlater studentene. Studentene skriver det de tror er løsningen, som vises i figur 21.

```
for (int i = 0; i < firmaliste.length; i++) {
    if (i.antall < i.søknader) {
        funnet = true;
    }
```

figur 21: utklipp fra studentenes kode, de har problemer med å vite hvilke pekere de skal bruke eller hva en peker er.

Student1: Så tror jeg ikke det er kapasitet, men søknader. i.søknader. Prøver det.
Student2: mm..
Student1: Så kan vi heller sjekke, lage noe med den metoden etterpå.

Så lagrer og kompilerer studentene dette forslaget, men fremdeles får de feilmeldinger, som vist i figur 22, men nå noen andre en det de har fått før.



figur 22: utklipp fra command promt vinduet til studentene, med feilmeldingene etter kompileringen.

Student1: Ja! Eller nei.
Student2: Nei.

Læreren ser at de ikke har forstått helt det han mente så han kommer inn i rommet igjen. Læreren tar tegningen som studentene har fått utdelt og viser på arket mens han forklarer.

Lærer1: Her refererer vi til firmaliste og dette er firmaliste. Kan dere gi en presis beskrivelse av hva den cellen der er? Hvis vi bruker i da. Det elementet der i arrayen.
Student1: Det er et Byggefirma objekt.
Lærer1: Ja.
Student1: Som inneholder..
Lærer1: Ja det vi kaller firma, byggefirma, men den plassen i arrayen, men arrayen heter, heter ikke lokalt her byggefirmaer men firmaliste for det er en parameter.
Student1: mm..
Lærer1: Firmaliste refererer til hele..
Student1: mm..
Lærer1: Det elementet der hva er det? Da vi har disse klammeparentesene.
Student1: Ja, ja firmaliste[0].
Lærer1: 0 ja, er du med på den og?
Student2: Ja.
Lærer1: Og den her heter firmaliste…?
I kor: 1.
Lærer1: 1 riktig ehh.. firmaliste[0] er den som inneholder en adresse til et Byggefirma objekt.
Student2: mm..
Lærer1: Firmaliste[1] inneholder peker til et annet.
Student1: Men vi er interessert i, i vel?
Lærer1: I ja.
Student1: mm.. i.. og i er jo 0 når vi går gjennom for- løkken.
Lærer1: Hvis ikke kapasiteten er full på første firma, prøv det.
Student1: Ja.
Lærer1: Men skjønte dere det? Det jeg sa?
Student2: mm..
Student1: Jeg tror det.
Lærer1: Dere kan jo diskutere litt.
Student1: Var du med?
Lærer1: En ting er å få et riktig svar og en feilfri kompilering, men dere kan jo diskutere det jeg sa.
Begge: mm..

Læreren går ut av rommet igjen.

Student1: Jeg har gjort mange tilsvarende feil når jeg har holdt på med obliger og oppgaver i det hele tatt, så..
Student2: Det er alltid så vanskelig, det vanskeligste, jeg forstår på en måte okay hvordan ting skal gjøres, men alltid det vanskeligste er å finne det som skal stå inne i løkkene og...
Student1: Ja, ikke sant.

De skriver inn firmaliste[i] i koden sin, som vist i figur 23, og dermed har de fått til pekerne i if- testen inne i for- løkken.

```
for (int i = 0; i < firmaliste.length; i++) {
    if (firmaliste[i].antall < firmaliste[i].kapasitet) {
        funnet = true;
    }
```

figur 23: utklipp fra studentenes kode, her har de fått til pekerne som de skal se ut.

Student1: Men og..
Student2: Men jeg skjønner hva man skal gjøre..
Student1: Akkurat det å holde rede på, jeg syntes det er vanskelig å holde rede på det jeg selv kaller de forskjellige variablene og array pekerne og her har noen allerede laget de for oss så det gjør det..
Student2: Skal du ha med .antall også?
Student1: Ja det er jo antall, og der skal det stå søknader eller kapasitet.

De har en del problemer da de har lyst å bruke kapasiteten til et Byggefirma i if- testen, men kapasiteten er ikke deklarert globalt, se figur 8, og dermed kan de ikke bruke denne. Men kapasiteten kan sees på som lengden av en array, som kalles søknader, som ligger i samme klasse, da lengden til arrayen er basert på kapasiteten. Studentene kommer også på å bruke arrayen søknader men de glemmer at de må skrive .length også. Lærerne kommer inn igjen for å gi studentene hjelp til å finne ut hva det skal stå der de har skrevet kapasitet eller søknader. Det studentene skriver i koden etter dette ser man i figur 24. De får også hint av lærerne før de går ut at de trenger en peker til det siste metodekallet de har skrevet, dette metodekallet ligger innenfor for- løkken, så dermed kan de bruke noe av det de har gjort tidligere i samme metode.

```
for (int i = 0; i < firmaliste.length; i++) {
    if (firmaliste[i].antall < firmaliste[i].søknader.length) {
        funnet = true;
    }
```

figur 24: utklipp fra studentenes kode, hvor de har fått til å sjekke på riktige variable i if-testen.

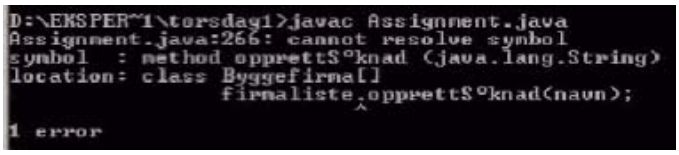
Lærer1: Det er litt av clouet med objektorientering, hvis du vil ha tak i det objektet nyter det ikke bare å be om det, du må gå hele veien..
Student2: be om det som peker på det.
Lærer 1: Ellers mister du det.
Student1: Hvilken klasse ligger opprettSøknad().
Student2: I Byggefirma.
Lærer2: Det har dere jo allerede begynt å peke på ting, for antall og søknader ligger i samme klassen.
Student1: Firmaliste.

Lærerne går ut etter litt mer diskusjon om variabel sjekkingen de har holdt på med, slik at studentene skal skjønne hva de har gjort. Deretter skriver studentene firmaliste.opprettSøknad(), som vist i figur 25, etter hintet de fikk og kompilerer denne koden.

```
for (int i = 0; i < firmaliste.length; i++) {
    if (firmaliste[i].antall < firmaliste[i].søknader.length) {
        funnet = true;
    }
    if(funnet) {
        firmaliste.opprettSøknad(navn);
    }
}
```

figur 25: utklipp fra studentenes kode, her prøver studentene å ta bruk av de hintene de fikk av lærerne i kallet på opprettSøknad().

Student2: Byggefirma, ikke firmaliste? Eller?
Student1: Ja.



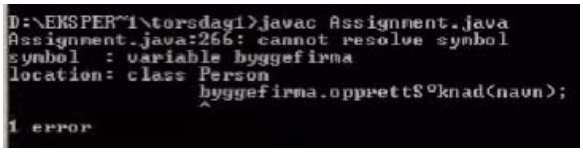
figur 26: utklipp fra command promt vinduet til studentene, med feilmeldingene etter kompileringen.

De ser på skjermen og leser feilmeldingen. Siden de får en feilmelding nå også tror de det er feil peker, og dermed tror de at de må finne på noe annet at de ikke kan bruke firmaliste lengre.

Student1: Skal vi prøve å bruke.. ehh..
Student2: Byggefirma.
Student1: Ja, at vi nå ikke er inne i firmaliste lengre, men vi er inne i Byggefirma med liten, og så er jeg ikke sikker på den parameteren.
Student2: Nei ikke jeg heller.
Student1: Det får vi se.
Student2: Vi prøver det ellers får vi prøve noe annet.
Student1: Det skal i hvert fall puttes inn en parameter.

De skriver det de vil prøve ut i koden, som vist i figur 27, deretter lagrer de og kompilerer den. De ser på feilmeldingene som kommer opp, se figur 28, og innser at ikke dette var riktig måte heller.

figur 27: utklipp fra studentenes kode, hvor de prøve å finne riktig peker for å kalle på opprettSøknad().



figur 28: utklipp fra command promt vinduet til studentene, med feilmeldingene etter kompileringen.

Begge: Nei.
Student1: Liker ikke den heller.

De blar i arket med koden de har fått utdelt og leter etter ideer for å finne riktig peker.

Student1: Det kan være byggefirmaer. Nei.
Student2: Nei for pekeren til Byggefirma var byggefirma.
Student1: Du må ha..
Student2: opprettSøknad() i Byggefirma, gå inn på der..
Student1: Byggesøknad.
Student2: opprettSøknad() ligger den i klassen Byggesøknad, er det den gjør?

De leter gjennom arkene for å finne stedet der den ligger, de finner den i Byggefirma. Deretter kommer læreren inn igjen for de ser at de er på feil spor, og ikke har brukt hintet og det de har sagt til dem om array pekere.

Lærer1: På nytt, parameteren heter ikke byggefirma…
Student1: mm.. vi prøvde først med firmaliste, men det var sannsynlig (”mumler”), det antagelig feil parameter i parentesen.
Lærer1: Vi var ikke på jakt etter hele..
Student2: i-en.
Student1: Å ja..

Lærer1: Ikke hele arrayen, men ett element i arrayen.
Begge: mm..
Student1: Og fremdeles holder vi på i i-en.
Lærer1: Dere skjønner logikken bak arrayer? Jeg bare spør.
Student2: Ja.

De diskuterer mer om arrayer og pekere, slik at læreren er mer sikker på at de forstår det de gjør og ikke bare skriver ned en riktig løsning. Deretter skriver de inn riktig peker, se figur 29, lagrer, kompilerer koden og ser at den feilen er borte.

```
if(funnet) {
    firmaliste[i].opprettSøknad(navn);
}
```

figur 29: utklipp fra studentenes kode, her har studentene skrevet inn rett peker til metodekallet.

Dermed ser koden til studentene ut slik det er vist i figur 30.

```
public void finnLedigFirma(Byggefirma [] firmaliste) {
    boolean funnet = false;

    for (int i = 0; i < firmaliste.length; i++) {
        if (firmaliste[i].antall < firmaliste[i].søknader.length) {
            funnet = true;
        }
        if(funnet) {
            firmaliste[i].opprettSøknad(navn);
        }
    }
}
```

figur 30: utklipp fra studentenes kode, her har de fått til alle pekerne og variabel sjekkinger.

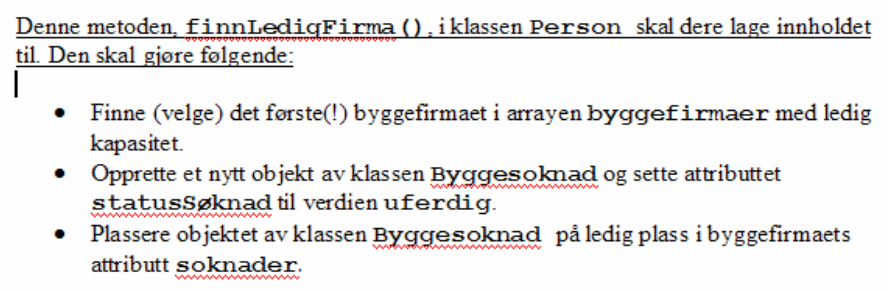
I dette eksemplet viser jeg at studentene har problemer med bruk av parametere, både det og motta en parameter og bruke den, men også det å sende med en parameter av riktig type. De har også problemer med pekere da de ikke har forstått hva en peker er. Det viser seg også at studentene har dårlig forståelse av programmeringsspråket Java og dermed begrenser det studentenes mulighet til å gjøre det bra på egenhånd.

6. Gruppe 1.

6.1. Arbeidsmetoder.

6.1.1. Hva gjør de?

Studentene begynner med å prøve ut programmet, deretter går de gjennom koden på ark. De velger å tegne farger på klasser, metoder og andre viktige ting i programmet slik at de lettere skal finne igjen ting. De går igjennom programmet ark for ark, de leser koden linje for linje. De bruker tid på å finne ut hvor metoden de skal lage er, så de leter i koden etter det stedet der det ligger. Deretter leser de oppgaven en gang til (se figur 31 under), så kommer de med noen forslag til hvordan oppgaven skal løses.


Denne metoden, `finnLedigFirma()`, i klassen `Person` skal dere lage innholdet til. Den skal gjøre følgende:

- Finne (velge) det første(!) byggefirmaet i arrayen `byggefirmaer` med ledig kapasitet.
- Opprette et nytt objekt av klassen `Byggesoknad` og sette attributtet `statusSøknad` til verdien `uferdig`.
- Plassere objektet av klassen `Byggesoknad` på ledig plass i byggefirmaets attributt `soknader`.

figur 31: utdrag fra oppgaveteksten studentene fikk utlevert.

Student1: Okay, finne det første Byggefirmaet med ledig kapasitet.
Student2: Vi må ha en eller annet løkke.
Student1: Det har vi vel.. først.. vi skal vel gå.. Det er en metode opprettSøknad i Byggefirma. FinnLedigFirma, her skal vi i hvert fall..

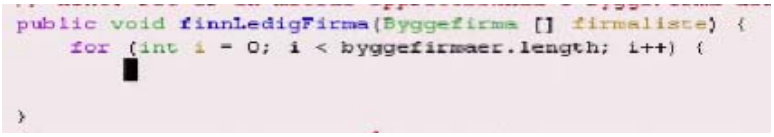
De ser forvirret ut og de blir stille.

Student1: Hva er det står på menyen igjen, eller den er en del av menyen er den ikke?
Student2: mm..? Her er den.
Student1: Her er det bare vis liste over firmaer..
Student2: Å ja, ja. Det er registrer ny søknad, skal vi se hvor er den, ser vi her. For i registrer ny søknad så kaller vi på, på finnLedigFirma(). For den skal brukes i det du registrerer for da at i det du registrerer deg skal den finne et ledig firma.
Student1: Yess, å.. ehh.. variabelen som puttes inn i finnLedigFirma() er byggefirmaer så..
Student2: Den er deklarert et sted…

De diskuterer denne parameteren videre, og konkluderer med at de kan bruke arrayen med navn byggefirmaer og ikke navnet firmaliste som den blir gitt i metoden finnLedigFirma(). Så blar de i arkene de har fått utdelt og blir stille mens de leser og tenker. Etter det blar de i arkene for å finne hvor ting er eller for å finne ideer til oppgaven de skal løse. Studentene diskuterer om hva kapasiteten er, hva den gjør og hvilke funksjon den har i programmet. De leser litt mer i koden og de blir stille mens de tenker. Deretter tegner og skriver de på papir mens de diskuterer hvordan de skal løse oppgaven. De diskuterer mer om kapasiteten og de antar at den er gitt ett sted i koden uten å ha funnet det noe sted. Deretter kommer de med flere forslag til hvordan de kan løse oppgaven.

Student1: Hvilke variable trenger vi? Det er vel det første vi må tenke på tror jeg.
Student2: Kapasiteten, mm.. kommer vi jo til å trenge for jeg tenkte at hvis vi lager en if løkke.
Student1: mm..
Student2: Høres ikke det bruk..
Student1: Ja.
Student2: brukbart ut?
Student1: En teller variabel rett og slett.
Student2: mm.. så hvis.. ehh..
Student1: Skal vi kalle den teller?
Student2: Jeg tenkte noe sånt som at hvis det.. at vi må sette opp et eller annet som sier at hvis.. ehh.. kapasitet byggefirma ehh.. er full så går den opp en gang til og fortsetter på neste Byggefirma.
Student1: mm..
Student2: Da vil den jo stoppe når den har oppfylt løkken sin holdt jeg på å si, Byggefirma, Byggefirma med kapasitet.
Student1: Trenger vi både en teller for firma og en teller for kap.. nei eller en teller for kapasitet, trenger vi to?
Student2: Ehh.. Skal vi bare prøve å se for meg..
Student1: Vi gjør vel kanskje det? Vi skal telle gjennom firma.. Først går vi vel i en for- løkke for å gå igjennom alle, gjør vi ikke det og så..
Student2: Ja, mm..
Student1: En if test inne i for- løkken.

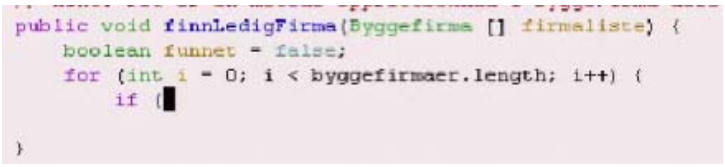
Student2: mm..



figur 32: utdrag fra studentenes kode.

Basert på dette begynner de å skrive ned litt kode (se figur 32 over). Så noterer de ned noe på ark hvordan de kan gjøre ting. Dette gjør de ved å bruke eksempler fra koden ellers. Når de begynner å kode tar de ikke bruk av parameteren som de får inn i metoden. Nå kommer de med enda flere forslag til hvordan de kan løse ting. De blar mer i arkene med koden, noe som tyder på at de er usikre på hvordan de skal løse oppgaven. Så kommer de med flere ideer til det de skal løse, etter det leser de mer i arkene med koden og de bruker tid på å tenke. De vet ikke hva de skal gjøre, og de kommer med flere forslag til hvordan de skal løse oppgaven.

Student2: Skal vi ha en funnet variabel, if funnet er lik..
Student1: Ja en sånn boolean funnet.
Student2: Ja. Hvis du da har funnet at den er.. har ehh.. kapasitet.. Ja..
Student1: Må vi kjøre en for løkke til med en gang kanskje, må vi ikke det?
Student2: For hva da tenker du på?
Student1: For å finne ut kapasiteten.
Student2: Men holder det ikke med en if, med en if løkke hvis den har ledig kapasitet..
Student1: Ja, mm..
Student2: Så bruk den..
Student1: Ja, jeg er enig.
Student2: Og hvis ikke ledig, så skal den da ta neste..
Student1: Ja, mm..
Student2: gå videre i løkka, må det jo bli på en eller annen måte, må det ikke det?

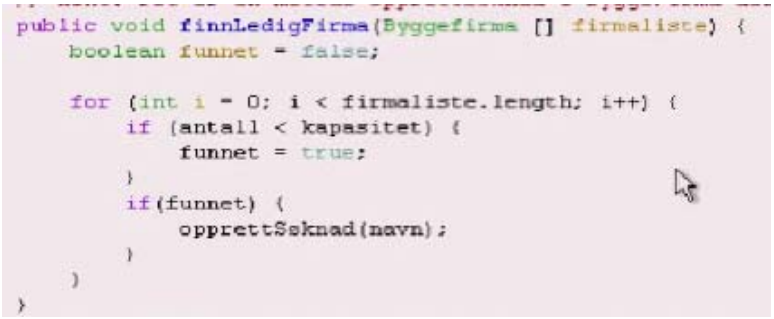


figur 33: utdrag fra studentenes kode.

Så kommer de med et forslag som er et skritt i riktig retning (se figur 33 over og figur 34 under), og de diskuterer hvordan de kan bruke det. De noterer noe ned på ark for deretter å diskutere dette forslaget nærmere.

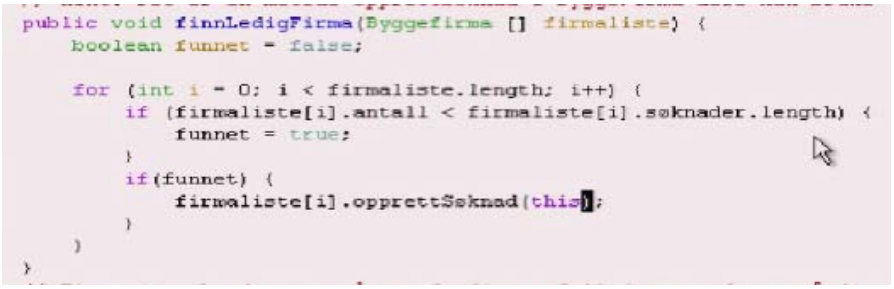
Student2: Altså søknader antall hvis det er lik da, hvis den er lik.. Hvis antall søknader er lik byggesøknader kapasitet, så må den jo være full.
Student1: mm..
Student2: Hvis den der er lik den der, hvis det er fem, hvis det er fem søknader og de har oppgitt kapasiteten fem da er den jo full og vi skal ikke ha den. Er du?
Student1: Ja.
Student2: Er du med på hva jeg tenker? Holdt jeg på å si.. For her, søknader det er lik da byggesøknad kapasitet, de oppgir en kapasitet som vi puttet inn der.. Så du får byggesøknad, det er jo et array da som da kanskje har fem plasser, ikke sant?
Student1: mm..
Student2: Til det ene firmaet, firma en har fem plasser, firma to har tre plasser ett eller annet så er det søknad, de sender inn søknad som blir da telt hvor du da får søknader av til.. selvfølgelig velge første firma, hvor de da har søknader, den er da kanskje et array som er satt til, hvor det da blir puttet inn, hvor det da kanskje har kommet inn to søknader.
Student1: Ja, mm..
Student2: Og hvis den der ikke er lik den, så skal man velge den. At hvis den kapasiteten der ikke er lik, hvis antall og kapasitet er like..

Student1: mm..
Student2: Så må vi videre til neste firma, hvis de er ulike, hvis den er mindre enn den så skal den sette statussøknad uferdig.
Student1: mm.. Ja vi skal gjøre hele punkt to, opprett et nytt objekt av Byggesøknad..



figur 34: utdrag fra studentenes kode.

Deretter skriver de inn i koden det de tror kan være rett. De leser og tenker litt mer før de fullfører forslaget (se figur 34 over). De har en del feil i koden som tyder på at de ikke har forstått det å ta bruken av en parameter som kommer inn i en metode, det å bruke pekere til objekter med deres metoder og variable, hva kapasiteten er og hvordan man kan få tak i denne verdien og det å sende med parametere av riktig type. Med en del veiledning fra lærer kom de fram til ett rett resultat (se figur 35 under).

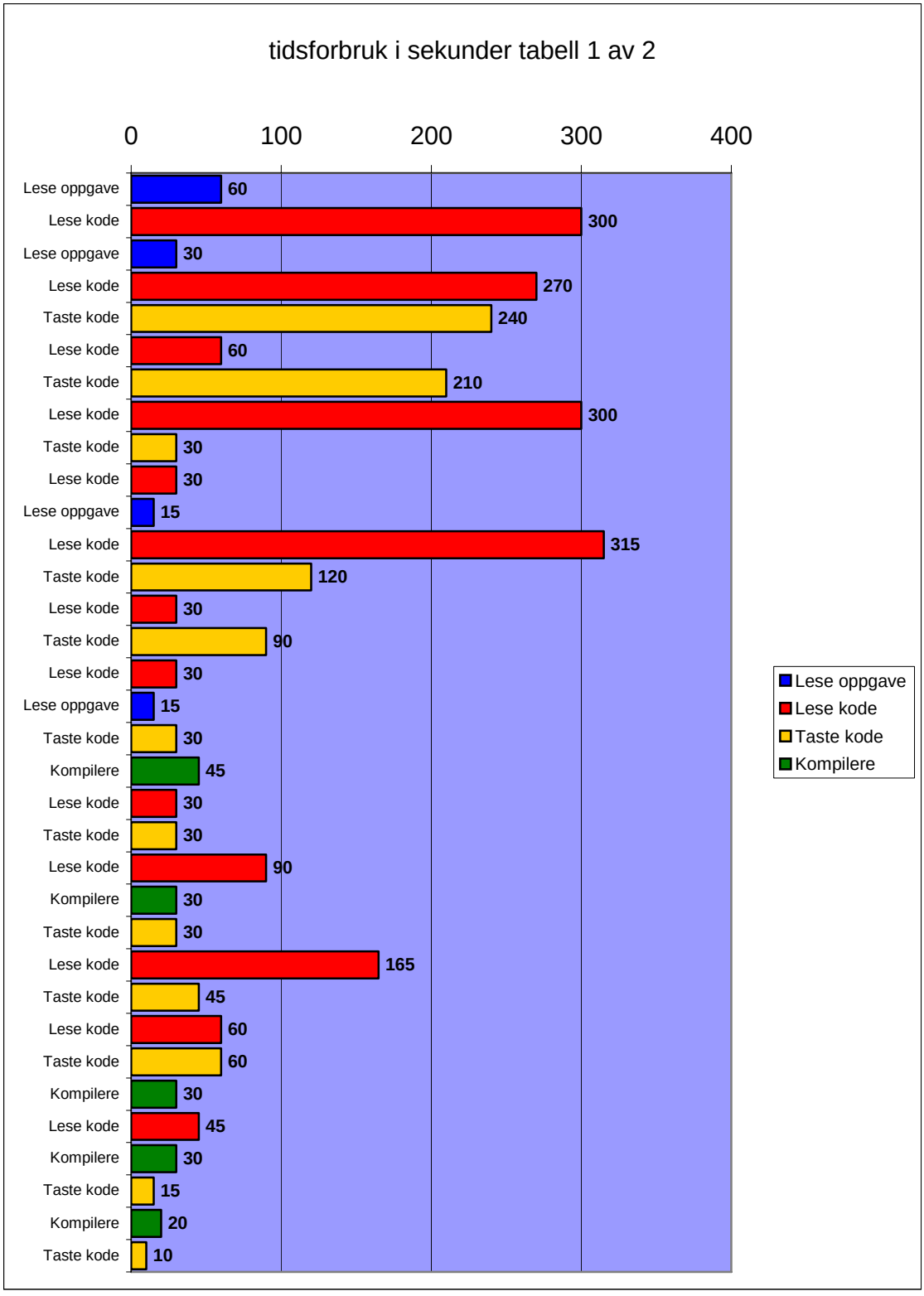


figur 35: utdrag fra studentenes kode.

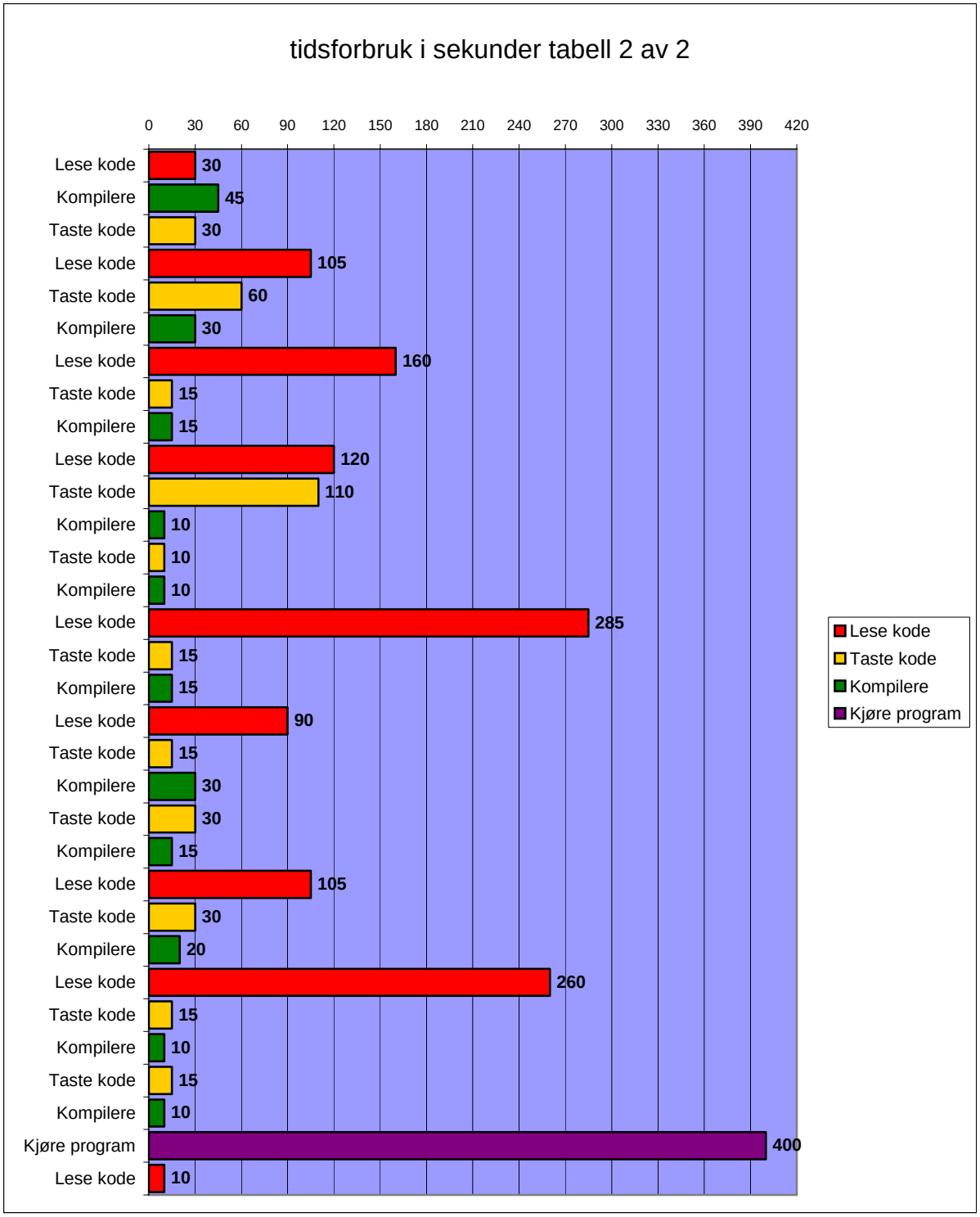
I dette eksempelet ser vi hvordan studentene har problemer med hva de skal gjøre og hvordan de skal løse problemer. De viser hvordan de har problemer med å forstå programmet og dermed sliter de med oppgaveløsningen.

6.1.2. Arbeidsaktiviteter.

Under ser vi to tabeller som hører sammen, gruppe 1 var innom så mange aktiviteter at det ble for vanskelig å bare lage en tabellsom får plass på en side, derfor ble det laget to stykker i stedet. Tidene ble mål under en oppgave studentene skulle løse, dvs. fra de får oppgaven utlevert til den er ferdig laget. Disse tabellene viser hvilke aktiviteter de er innom, hvor mye tid de bruker på hver aktivitet og i hvilken rekkefølge de er innom aktivitetene.

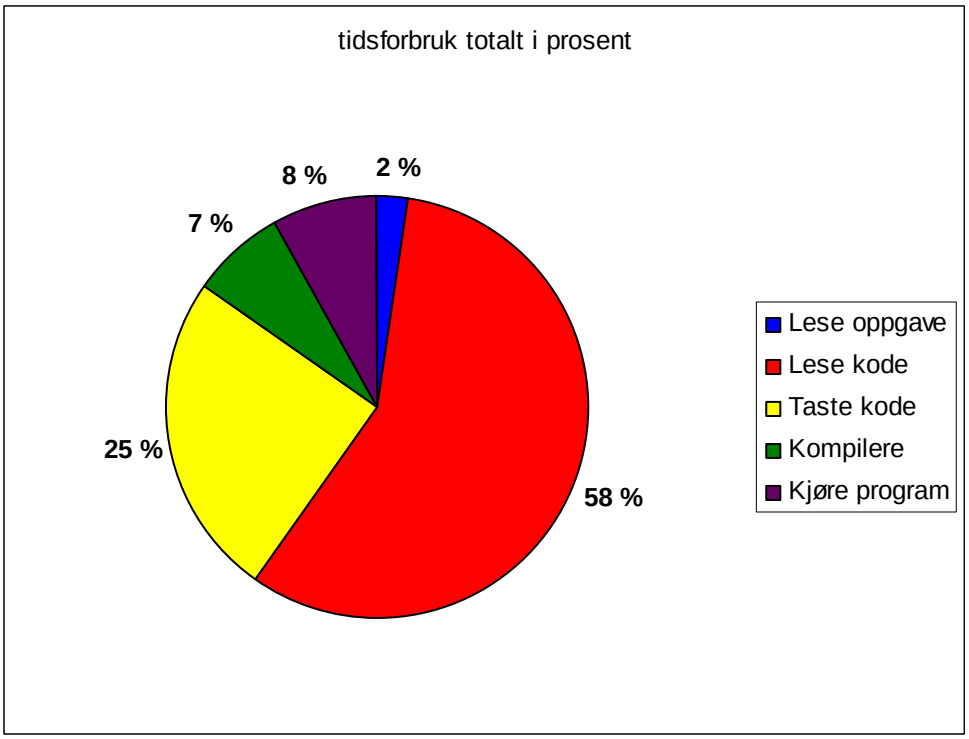


tabell 1: tabell en av to for arbeidsaktivitetene til gruppe 1.



tabell 2: tabell to av to for arbeidsaktivitetene til gruppe 1.

Disse tabellene viser hvilke aktiviteter studentene er innom og i hvilken rekkefølge de er innom disse. Disse studentene er innom mange aktiviteter ofte på grunn av at de vakler mye fram og tilbake, dette kan skyldes mye deres problemer med forståelsen av Java. Meste parten av tiden bruker disse studentene på å lese koden, deretter er det å taste kode.



Tabell 3: kakediagram av totalt brukt tid på de forskjellige aktivitetene.

Dette kakediagrammet viser det totale tidsforbruket på de forskjellige aktivitetene, som vises i tabellene over.

6.2. Domene resonnering og relasjoner.

I denne seksjonen viser jeg ved utdrag av samtaler mellom studentene hvilke relasjoner, mellom domenene fra artikkelen til (Kaasbøll et. al 2004), de er innom. Jeg bruker disse utdragene av samtalene mellom studentene for å vise at de faktisk forholder seg til disse domenene. Deretter laget jeg mitt eget bilde av hvilke domener og relasjoner disse studentene er innom, slik at man får et overblikk på hvordan deres fokus er. Jeg har også laget en tabell som viser tidsforbruket de bruker på hver av disse relasjonene fra bildet, der ser man også hvordan de veksler mellom de forskjellige relasjonene. Deretter laget jeg et kakediagram som viser totalt brukt tid på de forskjellige relasjonene.

Relasjonen ”design motivert av oppgavetekst”:

De leser i oppgaveteksten og leter i koden etter stedet der de skal kode.

Student1: FinnLedigFirma skal vi lage. Denne er det vi skal begynne med å lage, public void finnLedigFirma.
Student2: Velge det første byggefirmaet i arrayen byggefirmaer med ledig kapasitet, opprette et nytt objekt av klassen byggesøknad og sette attributtet statussøknad til verdien uferdig, plassere objektet av klassen..
Student1: Følgende.., finne det første byggefirma i arrayen byggefirmaer med ledig kapasitet.
Student2: Da må vi ha en eller annen løkke.

Slik forsetter de, og diskutere litt mer hva de kan skrive.

Dette eksemplet viser hvordan studentene tar utgangspunkt i oppgaveteksten.

Relasjonen ”design ved kopiering”

De leter i koden etter hjelp som kan løse det problemet de har.

Student2: Der har du listoppFirmaer, byggefirmaer.length..
Student1: Ja riktig.. vi kan bruke samme løkken som er der..

Så kopierer de det som står i den koden inn i din egen kode.

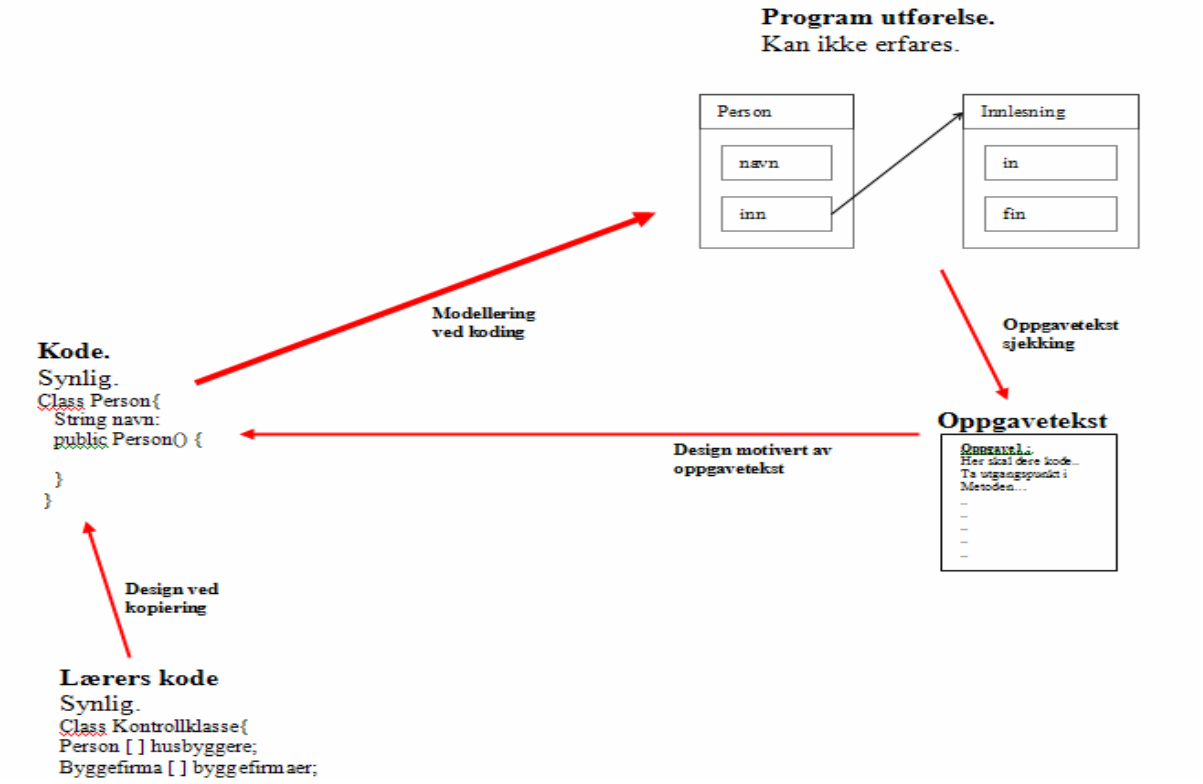
Dette eksemplet viser at studentene kopierer ting de finner inn i sin egen kode.

Relasjonen ”modellering ved koding”:

De ser på koden de har skrevet ned og diskuterer hva som skjer.

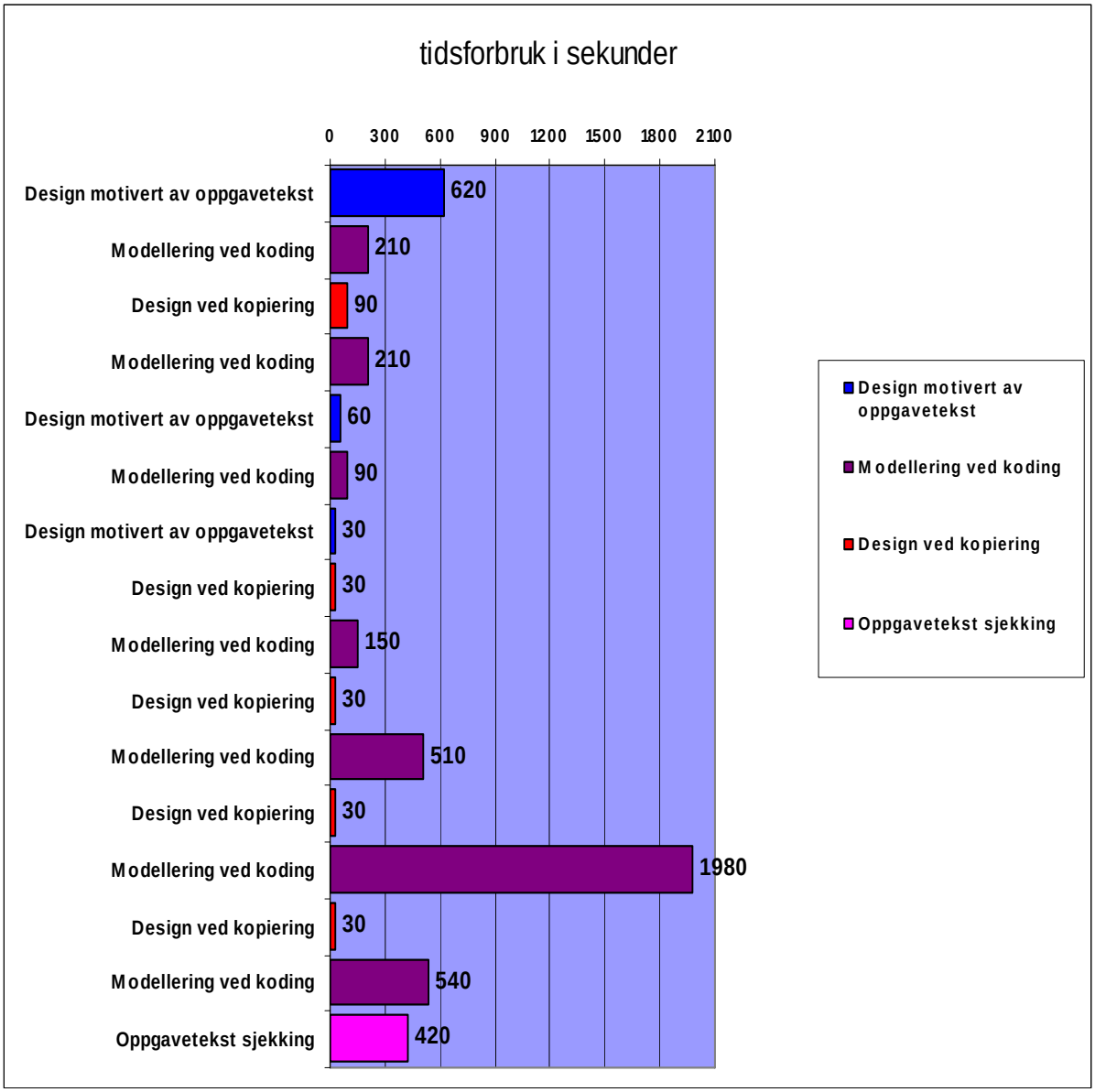
Student1: Den skal bare opprette et søknadsskjema, så får han søknadsskjema siden..
Student2: For at den der, den metoden der, den blir jo kalt på i det..
Student1: Jeg tenkte for langt.
Student2: Ja, nei, i det du skriver søknaden så skriver du inn navn, adresse og what ever, alt det der. Og så sjekker den, så går den hit og sjekker er det noen ledig, ja det er ledig der søknaden din blir puttet inn der, ja.
Student1: Helt med.

Deretter har de sin klare formening om hvordan modellen ser ut. Dette eksemplet viser hvordan studentene koder modellen, det vil si hvordan de får koden til å gjøre det program utførelsen skal.



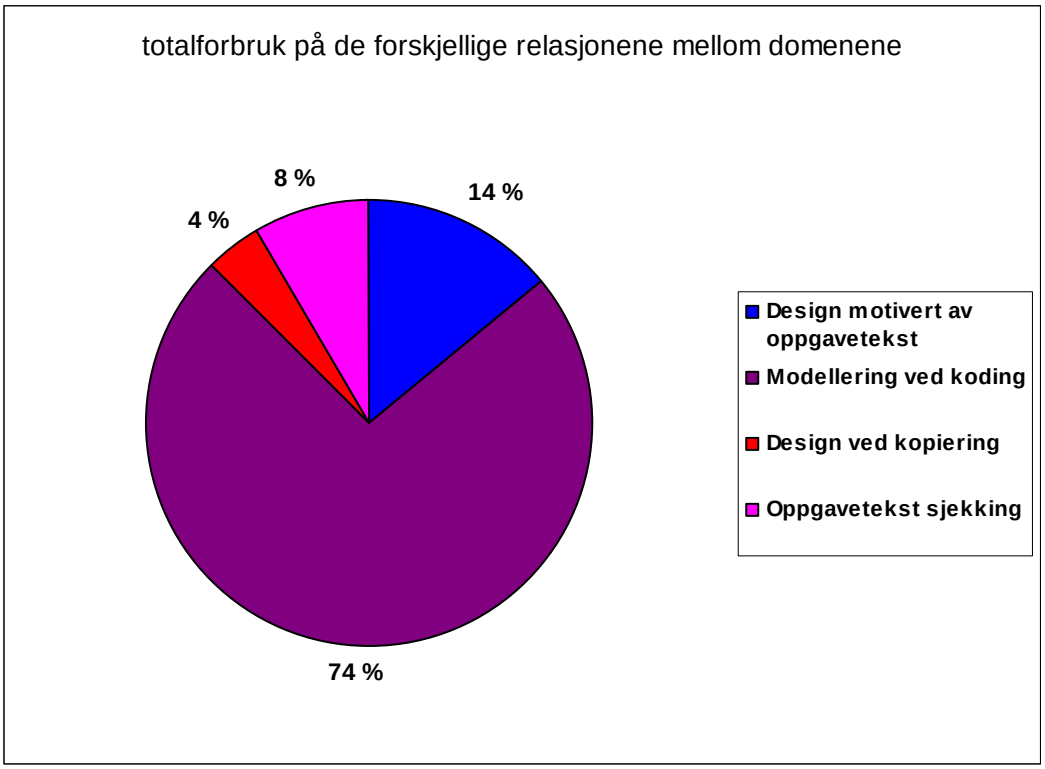
Figur 45: Oversikt over hvilke domener disse studentene fokuserer på.

Figur 45 viser hvilke områder studentene fokuserer på mens de programmerer og løser programmeringsoppgaver. Disse studentene fokuserer mest på relasjonen (pilen) ”modellering ved koding”. De bruker mindre tid på de relasjonene som er med tynnest pil.



Tabell 4: Oversikt over hvilke domener disse studentene fokuserer på.

Tabell 4 viser hvor mye tid i sekunder studentene bruker på de forskjellige relasjonene (pilene) i figur 45 over.



tabell 5: Gruppe 1 og deres fordeling av tidsforbruk på de forskjellige relasjonene mellom domenene.

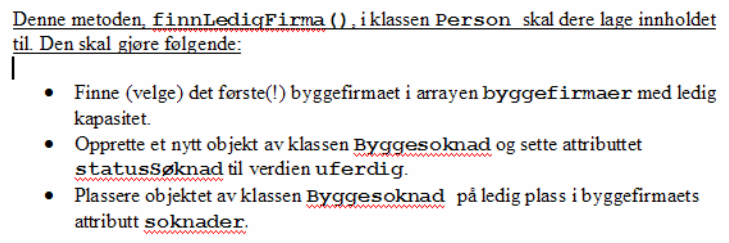
Denne tabellen viser totalforbruket av tid de bruker på de forskjellige relasjonen. Her angir jeg tidsforbruket i prosent for at man lettere kan sammenlikne de tre gruppene. For denne gruppen ser vi at modellering ved koding er den relasjonen som opptar studentene mest.

7. Gruppe 2.

7.1. Arbeidsmetoder.

7.1.1. Hva gjør de?

I dette eksemplet viser jeg en gruppe til hvordan de løser oppgaver de får utdelt. De bruker ca 5 minutter på å bli kjent med programmet, det virker ikke som om de prioriterer å lese oppgaven nøye og forstå koden. Deretter får de oppgaven utlevert og de begynner å tenke hvordan de skal løse denne oppgaven.



Denne metoden, `finnLedigFirma()`, i klassen `Person` skal dere lage innholdet til. Den skal gjøre følgende:

- Finne (velge) det første(!) byggefirmaet i arrayen `byggefirmaer` med ledig kapasitet.
- Opprette et nytt objekt av klassen `Byggesoknad` og sette attributtet `statusSøknad` til verdien `uferdig`.
- Plassere objektet av klassen `Byggesoknad` på ledig plass i byggefirmaets attributt `soknader`.

figur 36: Utdrag fra oppgaven studentene fikk utlevert

De begynner med å lese oppgaveteksten, se figur 36. De har ikke gjort seg nok kjent med koden så det kan se ut som de vet ikke hvor de har de forskjellige klassene og hvilke metoder som ligger i de forskjellige klassene og hva disse gjør. De leter etter den metoden de skal skrive innholdet i, men finner ikke stedet den ligger. Derfor leser de oppgaveteksten en gang til. Deretter finner de metoden de skal skrive og ser hvilken klasse den er i. De finner ut at de skal kalle en metode inne i denne metoden. De har ikke enda noen forslag til hvordan de skal gjennomføre oppgaven. De blar i arkene med koden de har fått utlevert. Deretter kommer de med ideer på hvordan de kan løse denne oppgaven og begynner å skrive deler av forslaget inn basert på det de skriver under.

Student2: Men de firmaene er jo der fra før av, som vi skal bruke ikke sant?
Student1: Ja.
Student2: Når vi kaller det, skal vi få String navn og det blir navnet på firma, ikke sant?
Student1: mm..
Student2: Også int kapasitet, kapasitet er kapasiteten til firmaet. Men ehh..
Student1: Det vi må gjøre er at vi må ha en løkke som går igjennom firmaliste.
Student2: Ja, det må vi jo.
Student1: Eller det trenger vi ikke, vi trenger bare en while- løkke vet du.. For det han skal gjøre er at den skal finne..
Student2: Men se her, vi kan bruke den løkka her, if antall er større enn null så skal den kjøre den for-løkka der. Så forandrer vi det inni ikke sant? Ellers kan, else kan den lage den første.

Studentene leser mer i oppgaveteksten.

Student1: mm.. Finne og velge det første Byggefirma objektet med ledig kapasitet.
Student2: Da må vi sjekke den kapasitet saken og.
Student1: mm..
Student2: Vi sjekker kapasitet og så..
Student1: Vi må ha da en variabel funnet i hvert fall, så kan vi ha en while- løkke som sjekker på.

De peker på arket de har fått utlevert hvor det er noe de kan bruke.

Student2: Hvis vi har den..
Student1: Sjekker vi på den der Byggefirma, på firmalisten.
Student2: Hvis vi har den her på, vi kan bare sette en boolean på det.
Student1: mm..
Student2: Vi må ha en som begrenser at det, at den, at hvis den ikke har kapasitet skal den ikke legge inn flere. Ikke sant?
Student1: mm.. int antall.. se her, de har en variabel..
Student2: Ja.
Student1: Byggesøknad det er en array med Byggesøknader det har inne, skal vi se.. Byggesøknad, int antall, int kapasitet.
Student2: Kapasitet ja, den får den jo inn da fra før av. Den setter du jo inn når du skriver inn firma.
Student1: mm.. antall det er antall søknader som er inne, det den er kanskje..
Student2: Ja, for da kan vi ha en if på, if hvis antall er mindre enn eller lik kapasitet ikke sant? Den har registrert hvor mange..
Student1: Nei antall sier hvor mange søknader du har inne.

Student2: Ja som den har inne, ikke sant? Da er det den, er jo en, jo, ja, for hvis antallet søknader er mindre enn kapasitet så kan du legge inn en ny en ellers kan du registrere på den.
Student1: Skal vi se..

Deretter skriver inn litt av det de har tenkt på, og underveis diskuterer de hva de skal skrive videre og hvordan de skal gjøre dette. De har en tendens til å løse oppgaven på en litt vanskelig måte og henger seg opp i uviktige detaljer som man gjerne kan se bort i fra.

Student1: Sånn.
Student2: If antall, men vi må jo kalle den riktige antall, det må bli punktum antall.
Student1: mm..
Student2: Det må bli en løkke på det da, for å gå igjennom byggefirmaene.
Student1: mm.. Skal vi da bruke en while- løkke?
Student2: mm.. Ja.
Student1: Int i er lik null.
Student2: Vi kan jo bare bruke en for og da egentlig, eller ehh.. Vi kan jo bare kjøre en while på boolean'en vår. While !false.
Student1: Ja while ikke funnet mener du?
Student2: Ja while !funnet.
Student1: mm..
Student2: Da må vi kanskje..
Student1: Problemet er da å, problemet er da alle sammen har ett eller annet oppdrag.
Student2: Men da har vi den på en else, ikke sant?
Student1: Vi må ha to sjekker faktisk.
Student2: Ja først skal vi sjekke..
Student1: Først sjekke om hvem som har ledig kapasitet og så sjekke hvem som har minst å gjøre.
Student2: Ja, hvem som har minst å gjøre må vi sjekke på ja. Vi må gå igjennom alle og finne hvem som har minst, for vi skal ha fortest ferdig og.
Student1: Skal vi se..
Student2: Da må vi jo se størrelsen på prosjektet og for to små kan gå like fort som en.
Student1: Ja, hvordan gjør vi det da?

```
public void finnLedigFirma(Byggefirma [] firmaliste) {
    boolean funnet = false;
    int i = 0;

    while(i < firmaliste.length && !funnet) {

}
```

figur 37: Utdrag fra studentenes kode.

På slutten av denne diskusjonen over smiler studentene litt ekstra da de begynner å skjønne at de løser oppgaven på en litt for vanskelig måte. De begynner å skrive inn innholdet i while-løkken basert på det de har sagt over, se figur 37. Disse studentene får med seg at de får inn en parameter og tar delvis bruk av den.

Student2: Er den faktisk der den der?
Student1: Ja det er den, for når du kaller på den finnLedigFirma, så får du inn arrayen med byggefirma.
Student2: Okay.
Student1: Som er tilgjengelig, som kan sjekkes på. Ehh.. mm..

De tenker og blir stille en liten stund.

Student2: Så må vi ha en ny løkke.
Student1: Eller trenger vi to løkker? Vi kan ha to if tester.
Student2: mm.. Ja.
Student1: If, så else if og så else.
Student2: mm..
Student1: Kan det gjøre jobben kanskje?
Student2: Jo, det gjør jo det.

```
while(i < firmaliste.length && !funnet) {
    if (firmaliste[i].antall == 0) {
        fyllUtSøknad();
        funnet = true;
    }
```

figur 38: utdrag fra studentenes kode.

Deretter skriver de inn innholdet i if testen, se figur 38. Siden de har løst oppgaven litt vanskelig ender de bare med å ha en if test og de lar resten av if testene være. Dette fører til at de ikke gjør hele oppgaven fullstendig. Disse studentene løser oppgaven underveis mens de leser, skriver ned og tester kode. De tar ikke med pekere til metoder i første omgang som skal kalles i andre klasser. Siden de ikke har lest oppgaveteksten og koden godt nok kaller de på feil metode, etter en stund ser de at det er feil metode de kaller. De har heller ikke helt orden på objekter og det virker som om at de tror String navn i klassen person er det samme som person objektet. Det blir stille og de tenker og de har problemer med å hva de skal gjøre. De trenger hjelp av lærer for å lese et hint som står i koden, se figur 39, slik at de ser hvilken metode de skal kalle på.

```
// firmaliste tilsvarer byggefirmaer i Kontrollklasse.
// Hint: Det er en metode opprettSøknad i ByggeFirma dere kan bruke
public void finnLedigFirma(Byggefirma [] firmaliste) {
```

figur 39: utdrag fra studentenes kode.

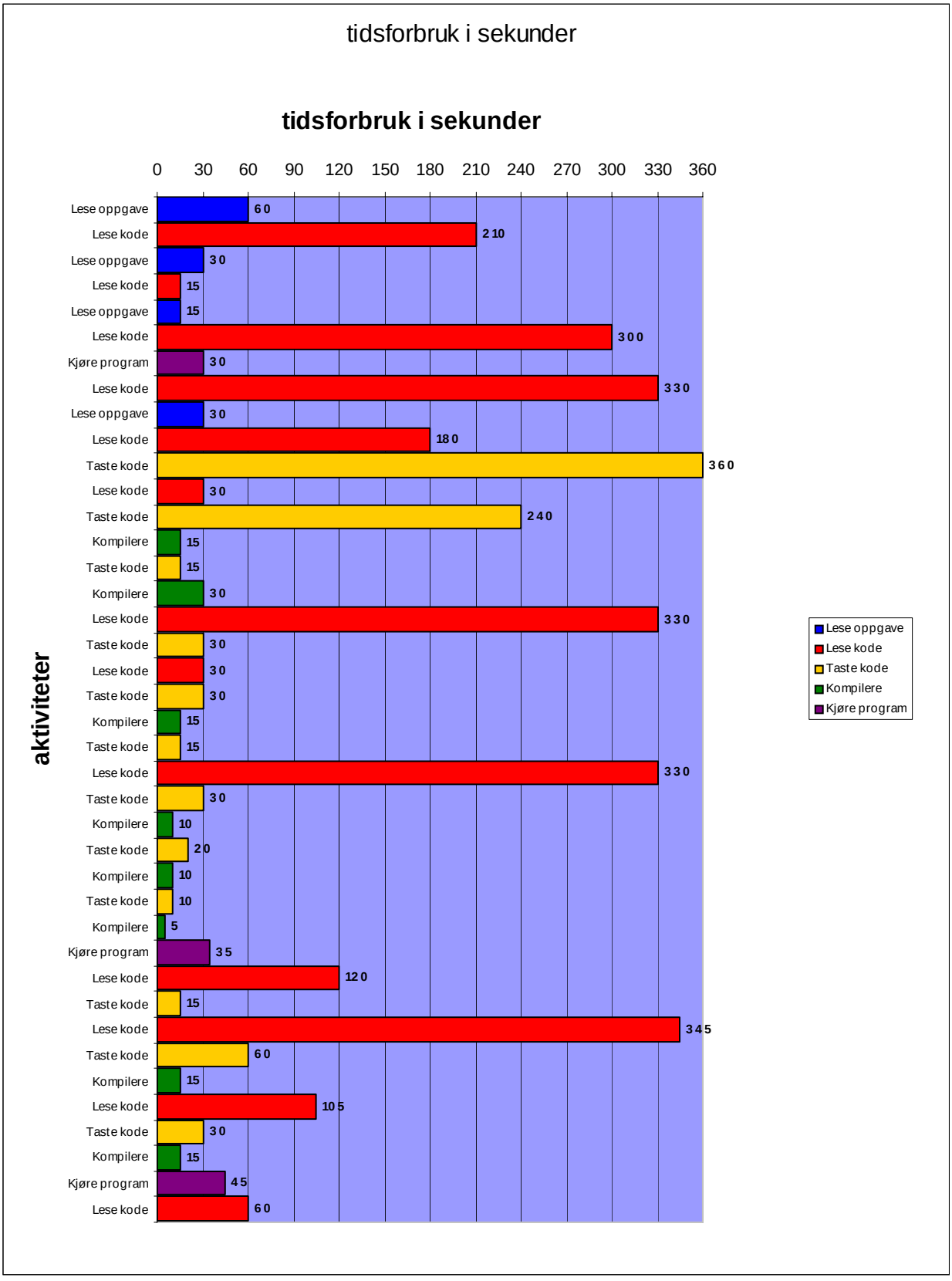
Når de får rett metode de kaller på utglemmer de pekeren til riktig objekt hvor denne metoden skal utføres i. Det kan derfor virke som om de ikke har fått godt nok overblikk på hvordan programmet fungerer og dermed har de problemer med å vite hvordan de skal gjøre ting. De kommer ikke forbi kallet på opprettSøknad() da de ikke får til pekeren eller parameteren uten hjelp. Læreren kommer derfor inn og gjør dem oppmerksom på at metoden ligger i klassen Byggesøknad. Ved å bare fokusere på pekeren først klarer de å bruke riktig peker. Deretter må lærer hjelpe studentene til å sende med riktig type parameter. Deres ferdige metode er vist i figur 40.

```
public void finnLedigFirma(Byggefirma [] firmaliste) {
    boolean funnet = false;
    int i = 0;

    while(i < firmaliste.length && !funnet) {
        if (firmaliste[i].antall == 0) {
            firmaliste[i].opprettSøknad(this);
            funnet = true;
        }
        i++;
    }

}
```

figur 40: utdrag fra studentenes kode.

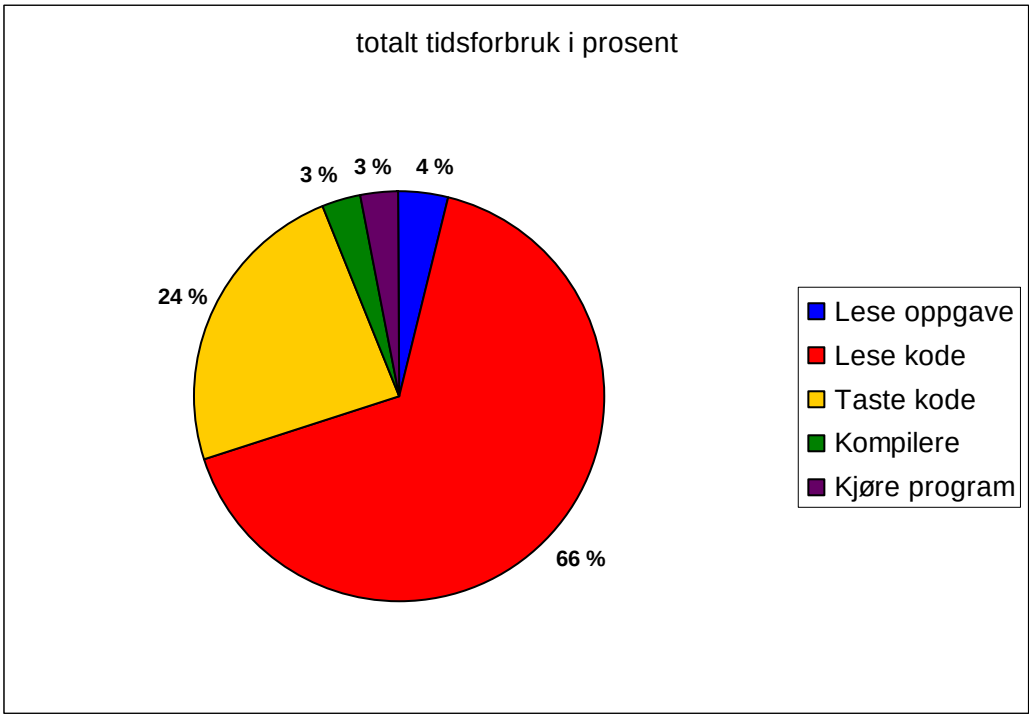
Dette eksemplet viser hvordan disse studentene jobber. De jobber ganske ustrukturert og mangler forståelse av hvordan programmet fungerer. Dermed får de noen problemer under veis ved at de ikke har kontroll på hva som skjer. Dessuten har de en tendens til å gjøre ting vanskeligere enn det er, og dermed får de problemer med å finne en god og enkel løsning. Disse studentene kunne Java koding ganske godt, men det kunne virke som om forståelsen av pekere og parametere var litt mangel full og dermed fikk de litt problemer med dette.

7.1.2. Arbeidsaktiviteter.



tabell 6: tabellen for arbeidsaktivitetene til gruppe 2.

Denne tabellen viser gruppe 2 sine aktiviteter, tiden og rekkefølgen de bruker på hver aktivitet, på samme oppgave som gruppe 1. Disse studentene bruker mest tid på å lese kode, deretter å taste kode. Disse studentene vakler også mellom de forskjellige aktivitetene, men mindre enn gruppe 1. Disse studentene hadde bedre forståelse av Java programmering, dette fordi de løste oppgavene lettere enn forrige gruppe og at de lettere finner løsninger. De kuttet ned på tidsbruk på de forskjellige aktivitetene og totalforbruket av tid på hele oppgaven.



Tabell 7: kakediagram over totalt brukt tid på aktivitetene.

Dette kakediagrammet viser totalt brukt tid på de forskjellige aktivitetene, som vist i tabellen over.

7.2. Domene resonnering og relasjoner.

I denne seksjonen viser jeg ved utdrag av samtaler mellom studentene hvilke relasjoner, mellom domenene fra artikkelen til (Kaasbøll et. al 2004), de er innom. Jeg bruker disse utdragene av samtalene mellom studentene for å vise at de faktisk forholder seg til disse domenene. Deretter laget jeg mitt eget bilde av hvilke domener og relasjoner disse studentene er innom, slik at man får et overblikk på hvordan deres fokus er. Jeg har også laget en tabell som viser tidsforbruket de bruker på hver av disse relasjonene fra bildet, der ser man også hvordan de veksler mellom de forskjellige relasjonene. Deretter laget jeg et kakediagram som viser totalt brukt tid på de forskjellige relasjonene.

Relasjonen ”design motivert av oppgavetekst”:

Studentene har lest oppgaveteksten og ser i koden og de prøver å finne ut hva de skal gjøre, de finner metoden i koden.

Student2: finnLedigFirma den skal vi lage.
Student1: Finne og velge det første byggefirma i arrayen med byggefirmaer.

De leser enda litt mer.

Studnet1: Finne et firma med ledig kapasitet.

Dette eksemplet viser hvordan oppgaveteksten gir studentene motivasjon til hva de skal kode.

Relasjonen ”modellering ved koding”:

De ser i koden og ser hvordan programmet fungerer mens de diskuterer.

Student1: Her er regNySøknad, kalt opp når vi taster..
Student2: Se på koden og menyen.
Student1: Registrer en ny søknad.
Student2: Ja kjør, gå inn i menyen så ser vi hva den gjør om den går innom noe annet først, den må jo skrive..
Student1: Her kommer den inn.
Student2: Navn ja, hva heter du?
Student1: Så sjekker den om husbyggeren personen ligger inne.
Student2: mm.. da blir det, da blir den..
Student1: og så den punktum finnLedigFirma, byggefirmaer det er da..

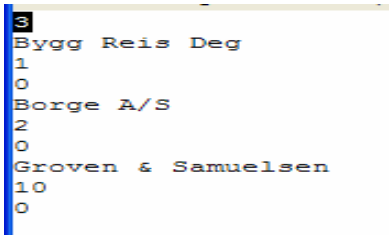
De ser i koden og prøver å finne stedet der arrayen med byggefirmaer er deklarert.

Student1: Byggefirmaer det er jo da arrayen med alle de ledige byggefirmaene, ikke sant. Og den sender han ned hit..

Dette eksemplet viser hvordan studentene ser at koden de har skrevet fungerer når de kjører koden. Dermed viser de at forstår sammenhengen mellom koden og program utførelsen.

Relasjonen ”fil sjekking”:

De titter i filen med byggefirmaene i og diskuterer hva alt betyr.



figur 46: Utdrag av filen med firmaer studentene har fått utlevert.

Student2: Firmaer.txt, det der er jo antagelig kapasitet, nei 1,2,0, hva er det 3 tallet øverst der? Hva er det 3 tallet for noe? Det er jo 3 firmaer inne.
Student1: Sikkert.

Student2: Det er jo..
Student1: Navn, nummer, antall jobber.
Student2: Det er jo kapasiteten, det er antall jobber antakeligvis.

Deretter leser de videre og diskuterer litt mer rundt innholdet i filen.

Dette eksemplet viser at studentene skjønner at filene har innvirkning på koden og at de bør forstå innholdet for å få bedre forståelse av koden.

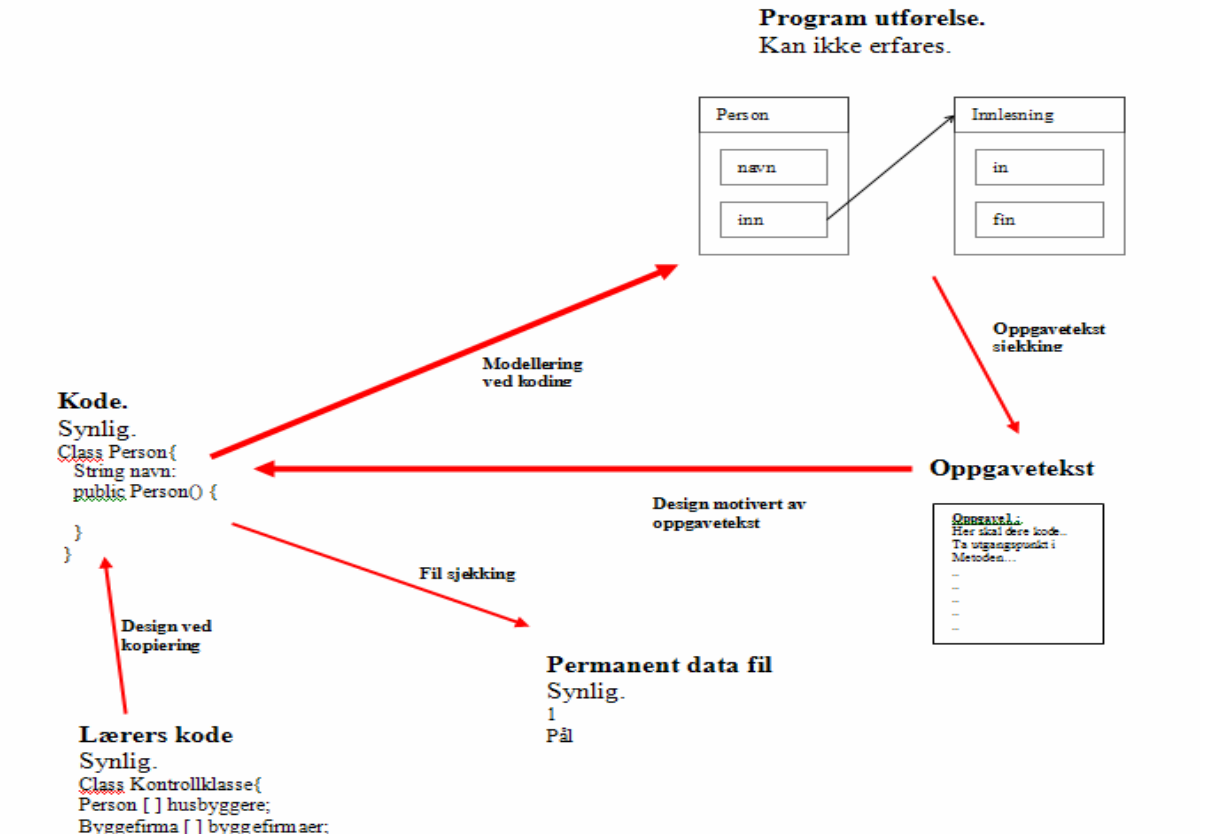
Relasjonen ”design ved kopiering”:

Studentene prøver å finne løsninger på oppgaven de skal løse og den ene studenten finner noe de kan bruke i lærers kode.

Student2: Vi kan bruke den her.
Student1: Vi kan bruke en while løkke.
Student2: Se her, se her..
Student1: Ja.
Student2: Vi bruker, vi kan bruke den løkka der helt fint, if antall er større enn null, så skal den kjøre den for løkka der og så forandrer vi det inni ikke sant. Ellers så, else så kan den lage den første.
Student1: mm..

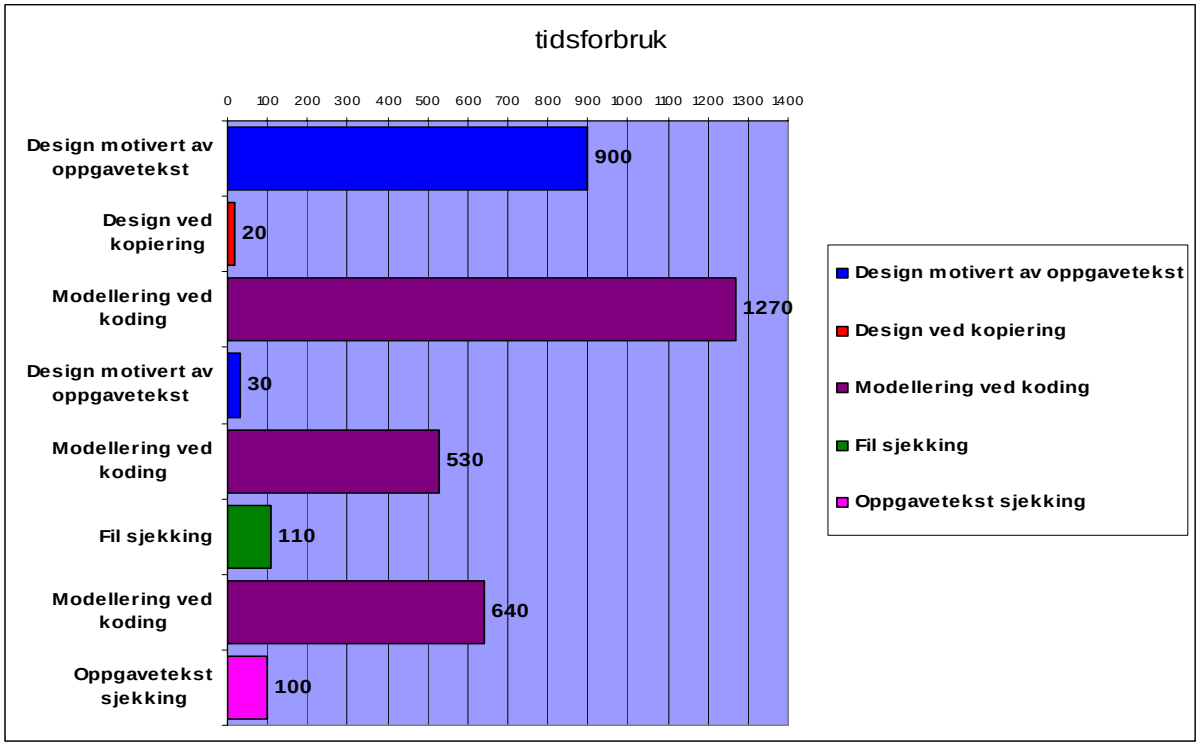
De leser videre og bruker noe av det de finner i lærers kode når de begynner å skrive inn kode.

Dette eksemplet viser at studentene finner motivasjon og eksempler i andres kode til å skrive sin egen kode. De viser at de prøver å gjøre andres kode til sin egen.



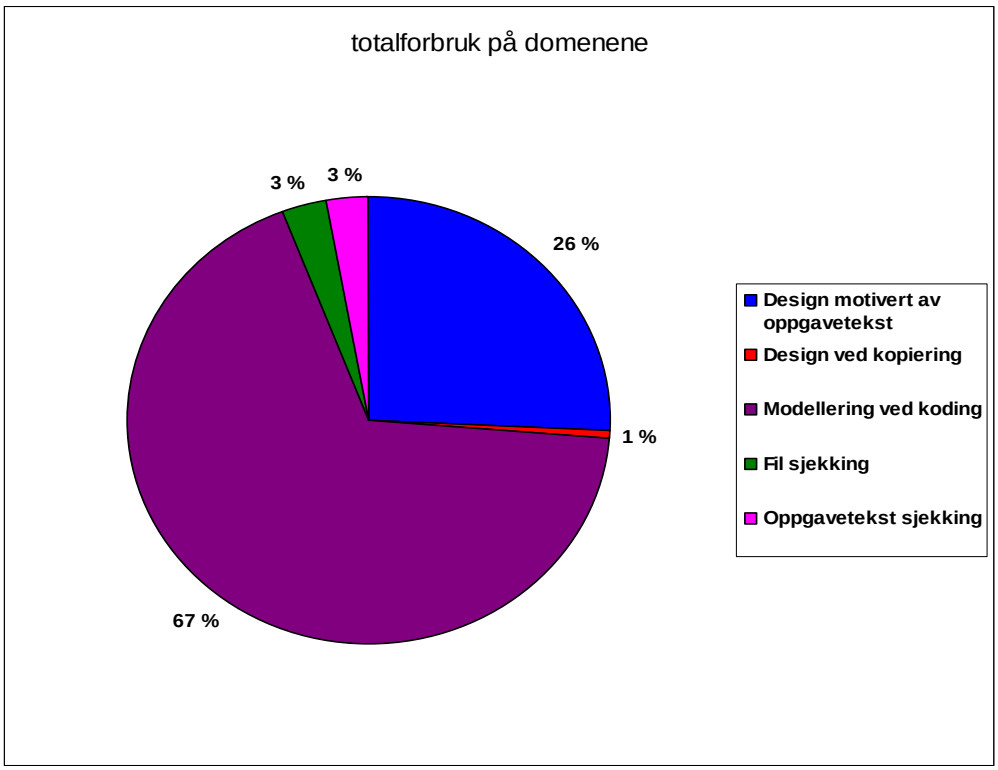
Figur 47: Oversikt over hvilke domener disse studentene fokuserer på.

Figur 47 viser hvilke områder studentene fokuserer på mens de programmerer og løser programmeringsoppgaver. Disse studentene fokuserer mest på relasjonene ”modellering ved koding” og ”design motivert av oppgavetekst”. De bruker mindre tid på de relasjonene som er med tynnest pil.



Tabell 8: Oversikt over hvilke domener disse studentene fokuserer på.

Tabell 8 viser hvor mye tid i sekunder studentene bruker på de forskjellige relasjonene i figur 47 over.



tabell 9: Gruppe 2 og deres fordeling av tidsforbruk på de forskjellige relasjonene mellom domenene.

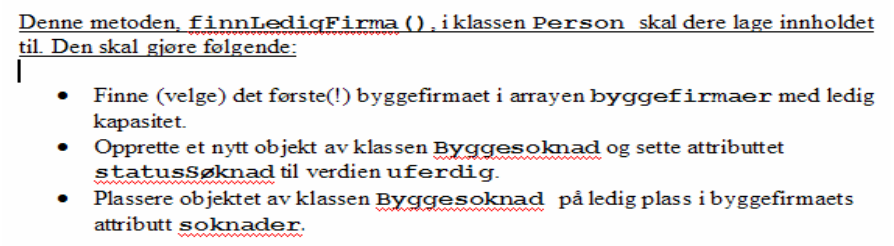
Denne tabellen viser totalforbruket av tid de bruker på de forskjellige relasjonen. Her angir jeg tidsforbruket i prosent for at man lettere kan sammenlikne de tre gruppene. For denne gruppen ser vi at ”modellering ved koding” og ”design motivert av oppgavetekst” er de relasjonene som opptar studentene mest, men de bruker klart mer tid på ”modellering ved koding”.

8. Gruppe 3.

8.1. Arbeidsmetoder.

8.1.1. Hva gjør de?

Disse studentene bruker ca 30 minutter på å bli kjent med programmet, som er del av aller første oppgave. Først leser de oppgaveteksten nøye før de begynner på koden. Her prøver de ut koden og diskuterer hva som skjer i programmet når noe skjer. De forklarer for hverandre hvordan noe fungerer og det kan se ut som de får en god forståelse av programmet. De bruker mest tid på koden på arket og går også gjennom koden hvor de jobber seg gjennom programmet steg for steg. Ved å stille spørsmål om hva som skjer eller hva ting betyr får de gått gjennom hele programmet.



Denne metoden, `finnLedigFirma()`, i klassen `Person` skal dere lage innholdet til. Den skal gjøre følgende:

- Finne (velge) det første(!) byggefirmaet i arrayen `byggefirmaer` med ledig kapasitet.
- Opprette et nytt objekt av klassen `Byggesoknad` og sette attributtet `statusSøknad` til verdien `uferdig`.
- Plassere objektet av klassen `Byggesoknad` på ledig plass i byggefirmaets attributt `soknader`.

figur 41: utdrag fra oppgaveteksten studentene fikk utlevert.

Deretter får de utlevert første oppgave hvor de skal skrive kode (som vist i figur 41 over). Før de begynner å løse oppgaven diskuterer de kort om hvem som skal bruke dette programmet da det ikke er skrevet ned noe sted, men de kommer fram til at det er husbyggerne som skal ta seg bruk av programmet. Da det er disse som skal engasjere Byggefirmaer til en sak. De bruker noen minutter på å lese oppgaven og forstå det som står, dermed vet de hva de skal gjøre. De diskuterer hva som kan gjøres for å gjennomføre denne oppgaven. Dette sier de om oppgaven før de begynner å skrive:

Student2: Så det eneste den skal gjøre er å finne det firmaet som det skal kalles opprettSøknad på.

De diskuterer litt rundt kapasiteten i arrayen søknader og om antall, hva de sier i forhold til hverandre. Som de kommer fram til som vist under i samtalen at lengden på arrayen vil tilsvare kapasiteten til et Byggefirma, og antallet er hvor mange søknader som Byggefirmaet har inne.

Student1: Hvis ikke antall er lik sizeOf eller lengthOf array saken, ikke sant?
Student2: mm.. så skal det være ledig kapasitet der.
Student1: Den er jo grei nok.
Student2: Og igjen så har vi ikke noe beskyttelse av data objektet så da er det rett og slett bare å kalle på, hente ut eller rett og slett finne lengden og antall. Bare gå igjennom hele, hver eneste.. ja. Det er ganske enkelt det der. Bare hoppe inn i det kanskje?
Student1: Ja.

De finner stedet i koden der de skal endre koden, de har et forslag basert på det de har sagt som de skriver rett inn. Siden de dobbelt sjekker at de har rett variabel navn før de fortsetter med oppgaven, typer på at de har kontroll på hvilke variable, metoder og klasser de skal ta hensyn til. De får med seg riktige pekere og eventuelle parametere på første forsøk. De sjekker metodekall de skal kalle om disse skal ha parametere og sender med en parameter som er av samme type. Som man kan se i eksemplet under.

Student 2: Så er det vel søknader den heter. Er det ikke det den heter den arrayen i Byggefirma?

De blar i arkene med koden og letter etter klassen Byggefirma.

Student2: Der byggefirma, ehh.., den heter søknader ja.

De prøver å finne ut videre hva de skal gjøre og hvordan de skal gjøre det.

Student2: Hva var det den metoden het? OpprettSøknad eller noe sånt?

De leser i koden for å finne metoden opprettSøknad. De finner den til slutt.

Student2: OpprettSøknad ja, nei hva var det den het, jo opprettSøknad. Kall opprettSøknad, stor S. Så skal den ha er person objekt så vidt jeg så her.

De ser i koden og peker på metoden opprettSøknad.

```
public void finnLedigFirma(Byggefirma [] firmaliste) {
    Byggefirma ledig = null;
    for (int i = 0; i < firmaliste.length;i++) {
        if (firmaliste[i].søknader.length > firmaliste[i].antall){
            /* Funnet ledig!*/
            ledig = firmaliste[i];
            break;
        }
    }
    if (ledig != null){
        ledig.opprettSøknad(this);
    } else {
        System.out.println("Urk! Ingen ledige firmaer.");
    }
}
```

figur 42: utklipp fra studentenes kode.

Deretter skriver de inn resten av metoden (vist i figur 42 over) og tester ut det de har gjort og det fungerer bra. Det virker som om de vet hele tiden hva som skjer hvor i programmet og dette kan være en av grunnene til at de kommer fram til gode løsninger fort. Denne oppgaven brukte de ca 24 minutter på å løse.

Deretter får de den neste oppgaven utlevert og de leser oppgaveteksten (vist i figur 43 under). Det ser ut til at de skjønner at de må lage relasjoner mellom objektene slik at oppgaven de skal løse er mulig å gjennomføre.

Oppgave 3

Denne oppgaven går også ut på å forandre programkoden:

Ta utgangspunkt i metoden `fyllUtSøknad()` i klassen `Kontroll`. Inni denne metoden kalles først metoden `finnHusbygger()`, som ber om et personnavn og finner riktig objekt i arrayen `husbyggere` (brukt i forrige oppgave også). Dersom personobjektet finnes så kalles dette objektets metode `finnMinSøknad()`.

Denne metoden, `finnMinSøknad()`, i klassen `Person` skal dere lage innholdet til. Den skal gjøre følgende:

- Som navnet tilsier, finn MIN søknad, skal søknadsobjektet som allerede er opprettet for personobjektet finnes. Dette er lagret i attributtet `søknader` hos det valgte byggefirmaet (jfr. forrige oppgave).
- Når riktig søknadsobjekt er funnet kalles (den ferdiglagde) metoden `fyllUtSøknad()` for å lese inn alle søknadsdataene fra tastaturet. Denne metoden er lokal i søknadsobjektet.
- Helt til slutt skal vi finne byggefirma(objektet) som er ansvarlig for søknaden. og kalle dennes metode `sendSøknadTilSaksbehandler()`. Denne metoden er allerede ferdiglaget. (Men det kan være en god ide å vente med å gjøre dette kallet til punktene over er løst.)

figur 43: Oppgave teksten til samtalen under.

Student2: Her kommer det problemet jeg tenkte på. Hvordan har de, hvordan er mapping mellom en person og en søknad på en måte?

De diskuterer mer om hva de gjorde og at disse objektene ikke er relatert til hverandre. De ser at når de oppretter en søknad så sender man med et person objekt men man gjør ikke noe med det selv om man oppretter ett nytt Byggesøknad objekt i den metoden. De finner det vanskelig å gjennomføre oppgaven uten å ha relatert Byggesøknad og Person objektene sammen og deretter bruker de tid på å linke disse sammen i konstruktøren til Byggesøknad. De endrer litt på konstruktøren slik at de får satt pekeren til Person i Byggesøknad.

Student1: Må vi da gå igjennom ale søknader og sjekke om, for det er..
Student2: Det er jo.

Student1: Byggesøknad som peker på en person, for en person peker ikke på en søknad. Da er det bare å løpe igjennom alle byggesøknadene å sjekke om en av de peker på personen.
Student2: mm.. Stemmer det, ehh så må vi kanskje se sjekke om..
Student1: Kan vi ikke bruke hvis de er like skal vi bare sjekke om de peker på samme person så slipper vi å bruke en masse annet tull.
Student2: mm.. jeg mener det kan funke, ikke sikkert i hodet, men vi kan jo teste bare. Vi kan alltids teste, korrigere og teste så det er greit. Nå skal vi i hvert fall ha relatert..
Student1: personen inni der i hvert fall.
Student2: Ja.

Deretter kompilerer de koden for å sjekke om koden de skrev når de relaterte objektene sammen er rett og de får en feilfri kompilering.

Student2: Den eneste metoden vi skal skrive er tydeligvis den å finne søknad.
Student1: Det er finnMinSøknad(), er det ikke?
Student2: Så da er det bare å gå opp i personene.

De leter etter stedet i koden der denne metoden ligger, både på arkene og på skjermen.

Student2: Der, finnMinSøknad(). Der har de startet Byggesøknad søknad = null; Det samme som vi gjorde her oppe. Det samme prinsippet. Vi skal lage en for løkke, for vi får jo firmaene, vi må først gå igjennom firmaene, så gå igjennom søknadene pr. firma.
Student1: Nei, ikke søknader.
Student2: Søknadene ligger i arrayen til firmaet. Vi vet jo ikke hvilket firma personen har registrert på så vi må jo faktisk bla igjennom alle firmaene.

De diskuterer igjen hvordan de skal løse oppgaven og de har et forslag til hvordan de skal gjøre oppgaven (som vist over). Igjen sjekker de variabel navn slik at de skriver ned rett navn med en gang og om metoder har parametere. Det kan se ut som om de kan veldig mye og kommer lett fram til løsninger. De tester ut det de har gjort og det fremkommer en liten ”bug” når de kjører programmet. Etter litt diskusjon ser det ut som om de skjønner at det er en feil hvor de sammenligner to objekter og ikke attributtet i objektet og ved å endre på koden blir feilen rettet opp. Sluttresultatet av koden de skrev er vist i figur 4 under. Denne oppgaven bruker de ca 42 minutter på.

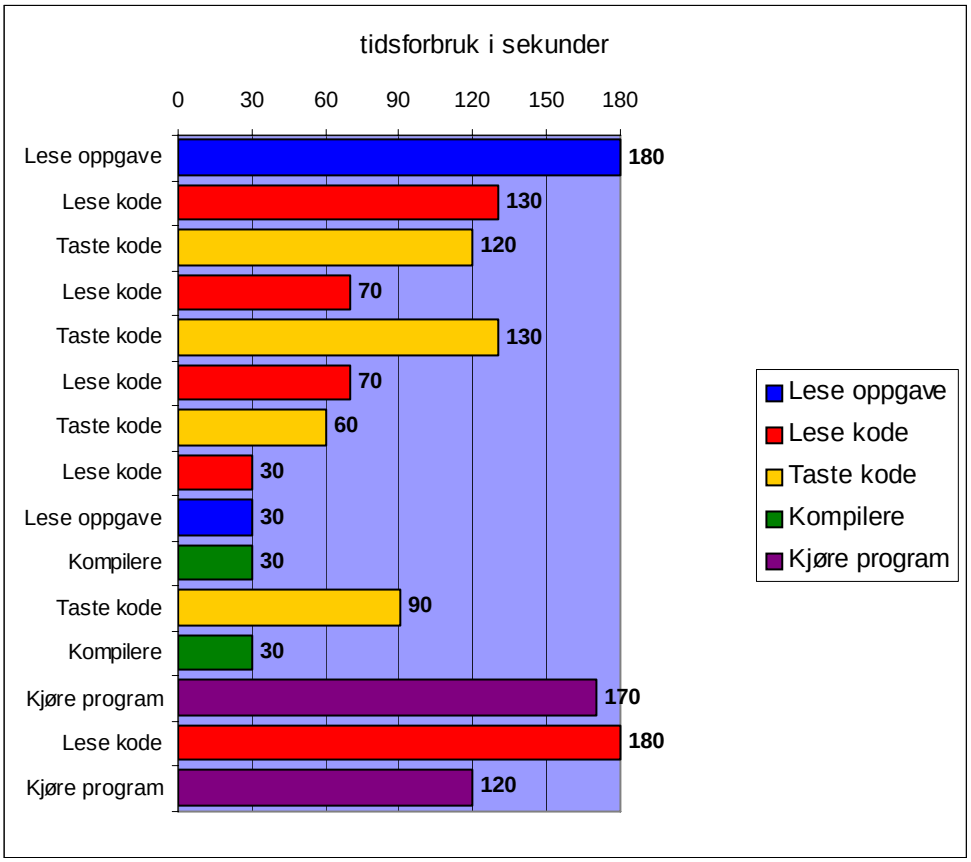
```
public void finnMinSøknad(Byggefirma [] firmaliste ) {
    ByggeSøknad søknad = null;
    for (int firma = 0; firma < firmaliste.length; firma++) {
        for (int s = 0; s < firmaliste[firma].søknader.length; s++) {
            if (firmaliste[firma].søknader[s] != null &&
                firmaliste[firma].søknader[s].husbygger.navn == this.navn) {
                søknad = firmaliste[firma].søknader[s];
                break;
            }
        }
        if (søknad != null) {
            break;
        }
    }
    if (søknad != null) {
        søknad.fyllUtSøknad();
        søknad.firma.sendSøknadTilSaksbehandler(søknad);
    } else {
        System.out.println("Dust! DU har ingen søknader inne!");
    }
}
```

figur 44: Utdrag fra studentenes kode.

Disse studentene kunne Java koding godt, de jobber strukturert og leser det meste grundig. Det kan se ut som om de får et overblikk på hvordan programmet fungerer og at finner de løsninger fort og enkelt. I dette eksemplet viser jeg hvordan denne gruppen jobbet og kommer fram til løsninger på oppgavene. Denne gruppen var de som kunne mest og fikk til mest, de har en god del kunnskaper i Java språket og i programmering og jeg så at de lett kom med løsninger på hvordan de skal gjøre oppgavene de fikk. De jobbet strukturert og det var lett å

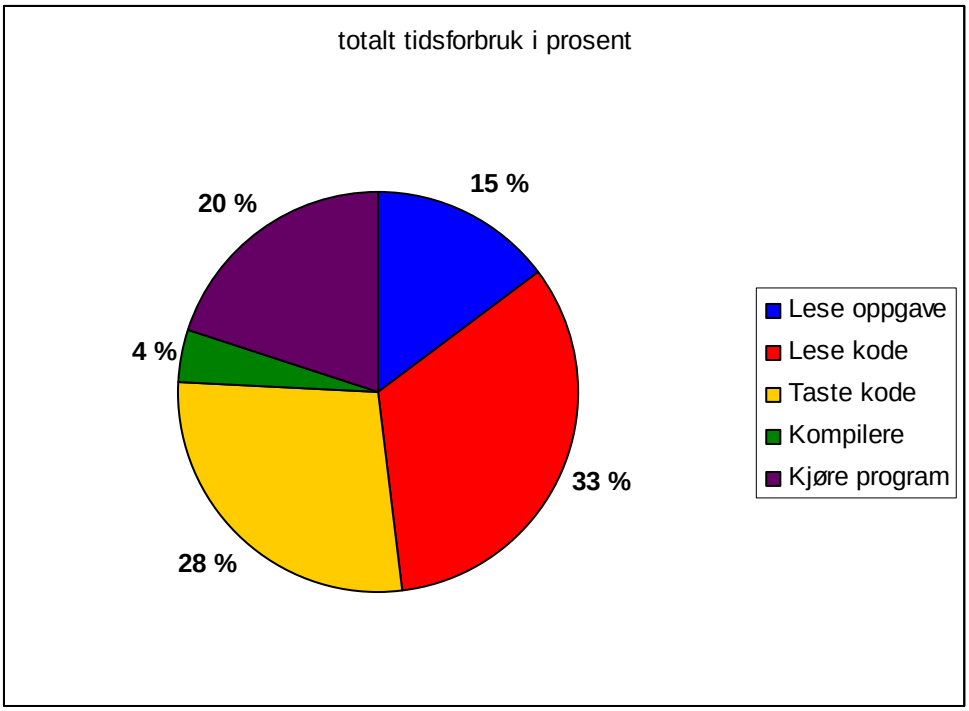
se hva de gjorde til en hver tid. Jeg observerte at de tok bruk av tidligere erfaringer innen programmering når de løse oppgavene.

8.1.2. Arbeidsaktiviteter.



tabell 10: tabellen for aktivitetene til gruppe 3.

Denne tabellen viser gruppe 3 sine aktiviteter, tiden og rekkefølgen de bruker på hver aktivitet på samme oppgave som gruppene over. Disse studentene brukte også mest tid på lesing av koden, men de har mer jevnt fordelt tidsforbruk på de forskjellige aktivitetene. Disse studentene hadde god Java kunnskap og dermed ble forbruket av tid kuttet drastisk ned i forhold til de to andre gruppene.



Tabell 11: kakediagram for totalt brukt tid på aktivitetene.

Dette kakediagrammet viser totalt brukt tid på de forskjellige aktivitetene, som vist i tabellen over.

8.2. Domene resonnering og relasjoner.

I denne seksjonen viser jeg ved utdrag av samtaler mellom studentene hvilke relasjoner, mellom domenene fra artikkelen til (Kaasbøll et. al 2004), de er innom. Jeg bruker disse utdragene av samtalene mellom studentene for å vise at de faktisk forholder seg til disse domenene. Deretter laget jeg mitt eget bilde av hvilke domener og relasjoner disse studentene er innom, slik at man får et overblikk på hvordan deres fokus er. Jeg har også laget en tabell som viser tidsforbruket de bruker på hver av disse relasjonene fra bildet, der ser man også hvordan de veksler mellom de forskjellige relasjonene. Deretter laget jeg et kakediagram som viser totalt brukt tid på de forskjellige relasjonene.

<u>Relasjonen ”design ved hjelp av oppgavetekst”:</u>

De leser oppgaveteksten og de begynner å se på koden.

Student2: Vi skal lage innholdet i finnLedigFirma, der er den vi ser her framme er det ikke, Person, finnLedigFirma den er tom. Så det eneste den skal gjøre er rett og slett finne det firmaet som man skal kalle opprette søknad metoden på.

Dette eksemplet viser at studenten bruker oppgaveteksten som motivasjon for å løse oppgaven og den brukes som veiledning for å skrive kode.

Relasjonen ”virkelighets sjekking”:

De har akkurat fått utdelt oppgaven og de har lest gjennom oppgaveteksten og begynt å se på koden.

Student1: Dette programmet, er det husbyggerne som skal bruke?
Student2: Ehh.. Ja. Det ville vel være logisk kanskje.

Deretter fortsetter de videre med kode lesing.

Dette eksemplet viser at de underveis sjekker programutførelsen passer inn med virkeligheten. De ser for seg hvem som skal bruke programmet og de sjekker om koden gjør stemmer med det virkeligheten sier at de skal gjøre.

Relasjonen ”modellering ved koding”:

De holder på å kjøre programmet og det kan se ut som om de vil forstå det som skjer når de kaller meny valget registrer en ny søknad.

Student1: Tror den er litt uferdig her jeg..
Student2: Nei, vi får ikke noen melding tilbake bare, men nå skal den ha registrert en ny søknad. For den har nå bare funnet personen og bare kalt finnLedigFirma og så har finnLedigFirma bare kalt opprettSøknad. Da har du opprettet en søknad for Pål, sånn..!

Den ene studentene knipser for å vise hvor fort det går å registrere en søknad.

Dette eksemplet viser at studentene har behov for å forstå forholdet mellom ”koden” og ”program utførelse”. De sjekker forholdet mellom koden og programutførelsen, slik at de skjønner hva som skjer.

Relasjonen ”fil sjekking”:

De ser etter filen hvor firmaene lagres og diskuterer innholdet når de finner den.

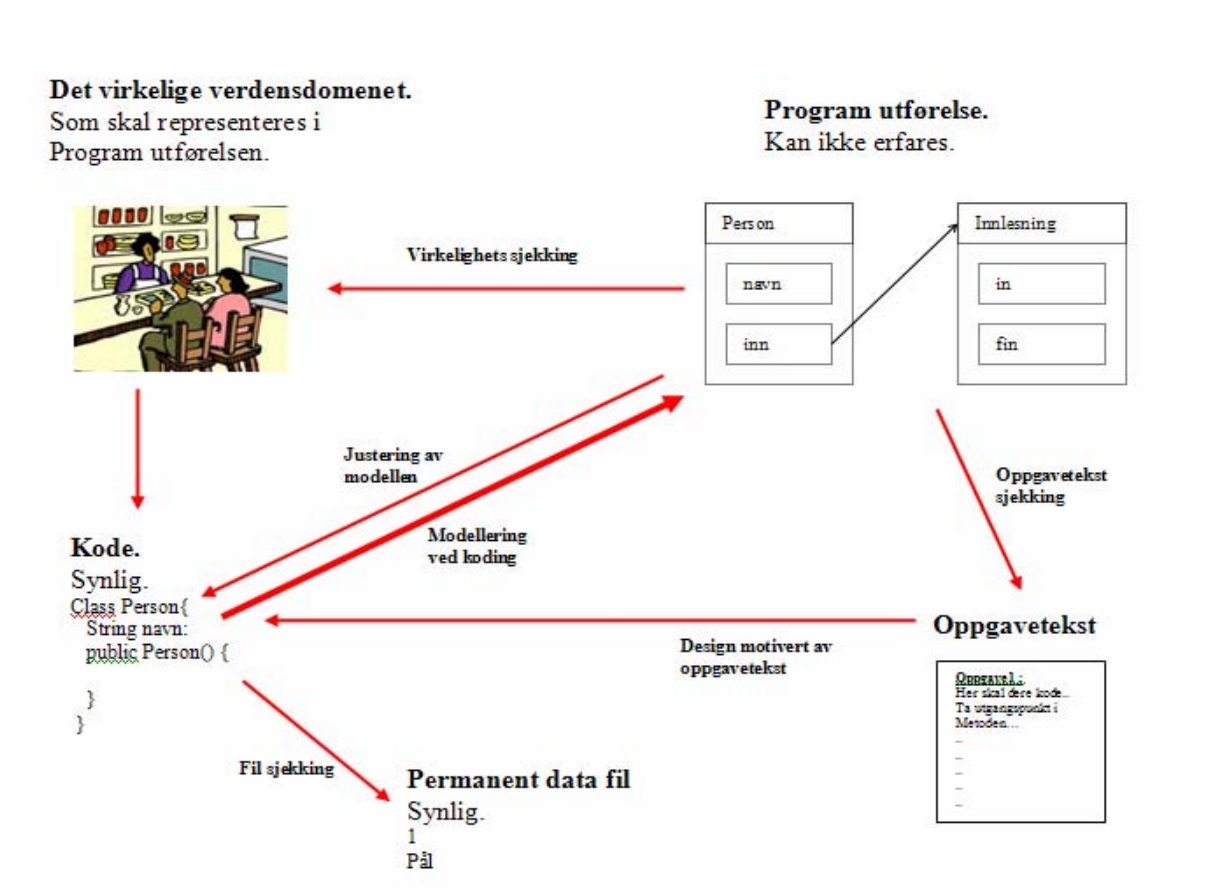
Student1: Her har du, vet du, her har du saken her, filen..
Student2: Ja, ikke sant, det sier vel, skal vi se hvordan de leser inn.., skal vi se.. Den første linja sier hvor mange byggefirmaer man har.

De ser i filen, mens en av studentene leser i koden for innlesning av fil.

Student2: For da opprettes det en array som er like lang som antallet som leses inn, så leses inn..
Student1: Navnet..
Student2: Så leses inn navnet ja..

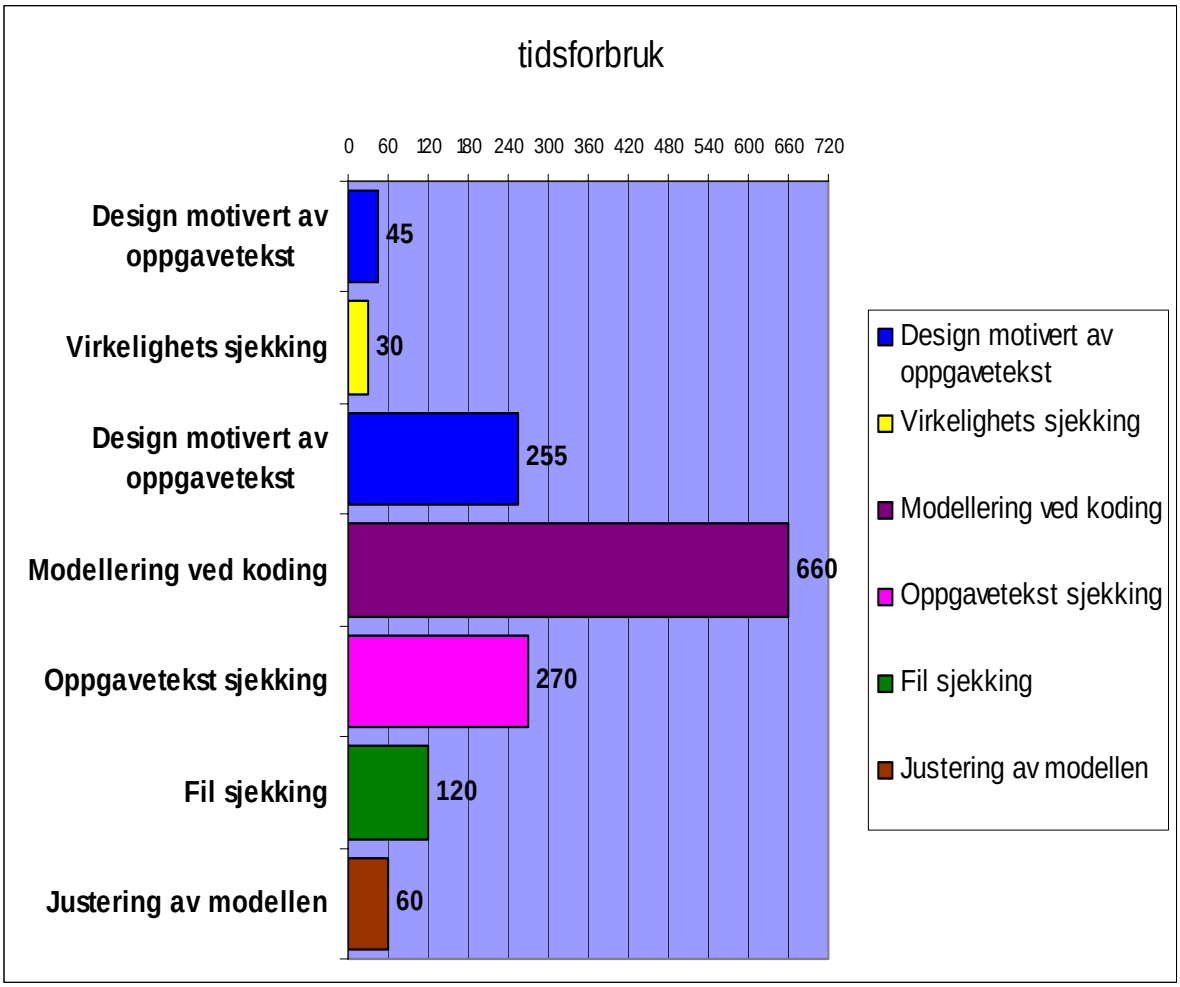
De går igjennom koden hvor filen leses inn og ser hva som gjøres samtidig som de ser i filen det leses inn fra for å se hvordan det vil gjøres i denne filen. Slik fortsetter de gjennom filen til de diskutert alt som står i filen og eventuelt ting som kan stå i filen, men som ikke står der nå.

Dette eksemplet viser at studentene tar hensyn til filer som har innvirkning på programmet. Ved å ta hensyn og forstå innholdet i filene vil de forstå mer av programmet, og klare oppgavene de skal løse lettere.



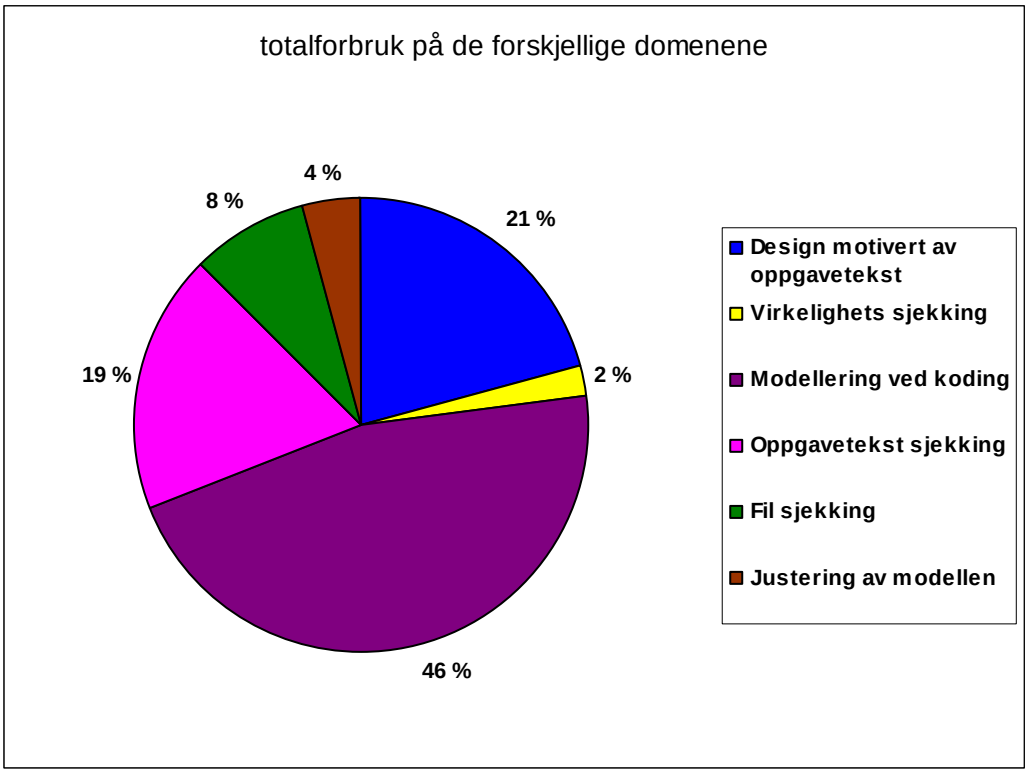
Figur 48: Oversikt over hvilke domener disse studentene fokuserer på.

Figur 48 over viser områdene disse studentene fokuserer på. De pilene som er tjukkere enn de andre påpeker at studentene vier mer tid og krefter på disse områdene, det vil si pilen som representerer ”modellering ved koding”. Deretter vier de mindre tid på pilene som er tynnest. Disse studentene bruker mest tid på å kode slik at program utførelsen oppfører seg slik oppgaveteksten sier og de justerer modellen ettersom de ser at programutførelsen ikke gjør det den skal. I dette programmeringsforsøket er det ikke lagt opp til at studentene skal modellere virkeligheten, det jeg mener er at virkeligheten er allerede modellert for studentene i koden og oppgaveteksten. Det eneste studentene trenger å gjøre er å forstå koden og dermed vet de hvordan programmet skal fungere og av den grunn vet de hvordan virkeligheten ser ut. Det studentene dermed kan gjøre er å ta en virkelighets sjekk og se at programmet er slik som man vil, slik at programmet viser virkeligheten slik den er. Det vil være lite behov for å endre på relasjonen ”justere modellen” da modellen er lagt opp fra før av. Men i løpet av oppgavene studentene skal løse har de behov for å gjøre et par mindre justeringer av modellen.



Tabell 12: tidsforbruk på de forskjellige relasjonene mellom domenene studentene kan fokusere på.

Tabell 12 viser hvor mye tid studentene bruker på de forskjellige relasjonene og tiden de bruker på dem, som man ser i figur 48.



tabell 13: Gruppe 3 og deres fordeling av tidsforbruk på de forskjellige relasjonene mellom domenene.

Denne tabellen viser totalforbruket av tid de bruker på de forskjellige relasjonen. Her angir jeg tidsforbruket i prosent for at man lettere kan sammenlikne de tre gruppene. For denne gruppen ser vi at modellering ved koding er den relasjonen som opptar studentene mest, men de bruker klart tiden mer jevnt fordelt utover de forskjellige domenene.

9. Diskusjon og Konklusjon.

Det jeg har observert i dette eksperimentet er at gruppe 3 er de som er flinkest og kan mest, deretter kommer gruppe 2, som har litt mindre kunnskap, og til slutt gruppe 1, som kunne minst. Gruppe 3 jobbet strukturert og kom fort med løsninger som kunne brukes. Gruppe 2 kunne mer enn gruppe 1, men de fikk ikke løst alle oppgavene helt ferdig på grunn av at de gjør ting litt vanskelig. Likevel kunne de ganske mye mer enn gruppe 1 slik som jeg observerte det. De kunne lettere komme med egne løsninger hvis læreren ga dem veiledning som gjorde at de kom i rett retning, mens gruppe 1 måtte ha svarene forklart ned i minste detalj. Dermed kan jeg si at gruppe 2 er flinkere enn gruppe1. Det utmerker seg at de studentene som kan mye bruker mindre tid på oppgaven enn de studentene som kan mindre.

9.1. Java og andre problemer.

Det viste seg at de største problemene studentene fra dette eksperimentet hadde var med pekere, parametere og forståelsen av objekter. Disse områdene skapte problemer for de fleste gruppene, det kom frem av videoopptakene.

Gruppe 1, den gruppen med størst problemer, registrerte at de fikk inn en parameter men de viste ikke hvordan de skulle bruke den. De trodde at parameteren ga dem en mulighet til å bruke det opprinnelige navnet på variabelen som de fikk inn som parameter, og at de ikke måtte bruke navnet på parameteren i stedet. Dette førte til mange problemer som studentene ikke klarte å løse på egenhånd, men de måtte ha veileding av læreren for å skjønne hva de skulle gjøre.

Disse studentene hadde også problemer med pekere, først og fremst viste de ikke hvordan de skulle bruke pekere eller at de måtte bruke pekere der det var nødvendig. De hadde en oppfatning om at så lenge de hadde laget en relasjon fra et objekt til et annet i en setning vil denne relasjonen gjelde for hele metoden under denne setning. Det jeg prøver å få frem er at de var inne i et personobjekt og gjennom en for løkke gikk de gjennom en array med byggefirmaer. Det de trodde var at for hvert steg i denne for løkken var de inne i et byggefirma objekt og kunne bruke de verdiene som var i dette byggefirmaet. De skjønte ikke at de måtte bruke den pekeren til det ene byggefirmaet de får fram i for løkken og bruke denne pekeren for hver verdi de ville ha tak i. Dette trengte de også hjelp fra læreren til å forstå.

Denne gruppen hadde også problemer med å skille mellom objekter og attributtene og variablene som ligger inne i et objekt. Dette viste seg når de skulle sende med et objekt til en annen metode og i stedet for å sende med et objekt sendte de med attributtet navn til objektet. De klarte ikke å se hvorfor dette ble feil. Her trengte også studentene veiledning fra læreren for å skjønne at det må være samsvar mellom disse to verdiene, og at navnet på et objekt er ikke det samme som selve objektet.

Gruppe 2 hadde også noen problemer med de samme områdene som gruppe 1, unntatt parameter problematikken. Disse hadde også mindre problemer på hvert område enn gruppe 1. De tok seg bruk av parameteren der de trodde de trengte den, men grunnet problemer med å forstå pekere brukte de den ikke på alle stedene der det var nødvendig. Disse studentene hadde også problemer med å forstå at de er nødt til å bruke pekere til de objektene de vil ha verdier fra for å kunne få tak i dem. De hadde litt av samme forståelse som gruppen over at når de var inne i en for løkke var inne i objektet som lå i arrayen de gikk gjennom. Men de var på et stadium der de var på grensen til å forstå hvordan det skal gjøres på riktig måte. De trengte bare litt hjelp av lærer for å tenke i riktig retning for å skjønne hva de skulle gjøre.

Denne gruppen hadde også litt problemer med å skille mellom objekter og navnet på objektet. De trodde det samme som gruppe en, at attributtet navn var det samme som objektet med dette navnet. Det som skiller disse fra gruppe 1 er at de med litt veiledning finner ut selv hva de skal gjøre. De måtte bli veiledet inn i tanken på at parameteren de sender må være den samme som parameteren de skal ta i mot.

Den siste gruppen, gruppe 3, hadde ikke noen problemer med disse problemområdene. Deres Java kunnskaper var gode og de kunne bruke kreftene sine til selve løsningen. Dermed kunne

de fokusere på det at de skriver en kode som virkelig gjør det oppgaveteksten vil at de skal gjøre.

Er man en helt fersk student som ikke har vært borte i Java før, vil forståelsen av parametere, pekere og objekter være vanskelig. Undervisningen av Java bør kanskje fokusere mer på disse temaene, slik at studentene kan forstå hvordan de skal bruke dem. Alle studentene i dette forsøket virker som studenter som er interessert i å lære og har lyst til å lykkes. Jeg syntes det virker som om disse temaene får for lite oppmerksomhet i dagens undervisning, da studentene har tydelige problemer med disse områdene.

Hvis jeg ser på artikkelen av (Granerud et al. 2005) så ser jeg sammenhengen mellom at studentene i eksperimentet og elevene i artikkelen sliter med parametere og pekere. Problemene er på forskjellige nivåer, det er ikke alt som læres til elevene i artikkelen og dermed kan det føre til at de får en del problemer. Men jeg ser sammenhengen i at studentene har problemer med disse områdene, elevene i artikkelen lærer ikke alt som har med pekere og parametere og de har problemer med å forstå disse uten å ha lært det. Studentene i eksperimentet jeg har holdt på med sliter også med å skjønne hvordan parametere og pekere skal brukes selv om det har blitt undervist og de er voksne. Dette kan implisere at pekere og parametere ikke er intuitive å forstå og at det sier ikke seg selv for nye studenter hvordan disse skal brukes. Dette kan fortelle oss at det bør legges vekt på forståelse av disse to temaene.

I (Pea 1986) sin artikkel om ”superbugs” hvor man kan si at det første steget i læring av programmering er å gå fra menneskelig tenking til program konsepter, dette er noe jeg delvis har observert. Studentene sliter med å vite hva de skal skrive for at programmet skal forstå hva de koder, de skjønner hva som skal gjøres men sliter med kodingen. (Spohrer et al. 1986) rapporterte at hovedhindringen for å lære programmering er å putte deler av kode sammen. Det er noe noen av studentene hadde problemer med. Det at noe kode allerede var ferdig og at de skulle lage koden som manglet førte til problemer med å vite hva de skulle gjøre da de syntes det var vanskelig å skrive den koden som passet inn med den eksisterende koden.

Problemene, som er vanskeligheten med å identifisere klasser og vanskeligheten med å koble sammen deklarative og prosedurale aspekter i løsningen, som nevnes i (Detienne 2002) kommer ikke fram hos studentene i denne oppgaven, på grunn av at klasser, prosedyrer med mer er ferdiglaget for studentene. Dermed blir de ikke satt ovenfor de problemområdene hun beskriver, det er meget mulig at studentene ville hatt sånne problemer om de hadde laget programmet fra begynnelsen av.

9.2. Arbeidsmetoder.

I begynnelsen vet ikke gruppe 1 hvordan de skal begynne å løse oppgaven. De sliter med å komme i gang og de sliter med å vite hvilke verdier de skal bruke og hvordan de skal sjekke på de verdiene. Grunnet dårlige Java kunnskaper forstår de ikke alt de leser i koden, men de prøver å akseptere eller anta hvordan noen av tingene er. Dette fører til at de ikke får med seg hva programmet gjør hele tiden. De glemmer også å følge metode kall mellom metoder slik at de vet hvordan delene av programmet hører sammen, så de får et dårlig inntrykk av hvordan

programmet fungerer. Disse studentene lar være å lese kommentarer der det står hint om hvordan de kan gjøre noen ting, som gjør oppgaven de skal løse enda vanskeligere.

Disse studentene skjønner overordnet hva de skal gjøre, men de sliter med å forstå hvordan de skal løse oppgaven i Java. De sliter med å vite hvordan de skal skrive det de tenker i kode. Dermed blir det mye prøving og feiling fra første kode biten de skriver ned til den ferdige koden. De kommer med mange forslag til hva løsningen kan være, men virker aldri overbevist selv om forslaget kan være riktig. De har ikke har forstått programmet helt riktig slik at de bruker variable de ikke kan få tak i og de klarer å bruke pekere og andre Java biter.

De klarer ikke komme videre uten hjelp fra læreren som nesten må fortelle studentene eksakt hva de gjør feil, slik at de endrer koden sin til det riktige. Likevel virker det ikke som de forstår det de gjør. De gjør en del feil om og om igjen selv om lærer kommer inn til dem og forteller dem hva de gjør galt. Mye av disse studentenes problemer skyldes at de har dårlige Java kunnskaper og ikke har programmert mye tidligere. De mangler erfaring på hvordan de gjør visse ting, og de kan ikke relatere det de gjør nå til andre ting de har fått til før.

Hvis vi ser på tabellen for de arbeidsaktivitetene denne gruppen er innom, ser vi at de bruker mesteparten av tiden de bruker på oppgaven til å lese i koden. Dette skyldes at de ikke har forstått koden godt nok før de startet på oppgaven. Dermed må de bruke mye tid på å lete i koden etter variable, metoder og andre ting de trenger å forstå. De sliter også med å vite hva de skal gjøre dermed bruker de også tid på å lese i koden for å finne eksempler de kan bruke i oppgaven sin.

Det som utmerker seg hos gruppe 1 er at de hopper mye fram og tilbake mellom de forskjellige arbeidsaktivitetene. Dette kan skyldes som jeg har beskrevet over at de er usikre på hva de skal gjøre og dermed er det mye prøving og feiling av forskjellige forslag de kommer med. De hadde ikke kommet så nærme løsningen de gjorde uten at de hadde jobbet slik, fordi de er så usikre på hvordan ting gjøres og dermed må de prøve ut det de tenker for å sjekke om det kan være riktig.

Gruppe 2 har mindre problemer med å komme i gangen gruppe 1, og de har bedre Java kunnskaper enn gruppe 1. Disse studentene kommer ganske raskt med forslag til måter de kan løse oppgaven på, men det som gjør at disse studentene får en del problemer med å løse oppgaven er at de tenker litt for vanskelig. De lar uviktige faktorer, som vanskeliggjør koden deres, påvirke hvordan de skriver koden dette fører til at de får noen problemer med å fullføre oppgaven 100 %. Men når de kommer med forslag klarer de stort sett å skrive det ned riktig. Disse sliter mest med pekere og dette er den delen av Java kodingen de strever mest med. Men med lett veiledning fra læreren kommer de på hvordan pekerne skal se ut.

Disse studentene får noen problemer underveis på grunn av at de ikke bruker nok tid før de starter på oppgaven til å forstå programmet. De bruker bare kort tid på dette og dermed får de ikke et godt nok inntrykk i hva programmet gjør og hva alt betyr. Dette gjør at de får problemer med hva de skal skrive i koden siden de ikke vet hva som betyr hva og hvor ting ligger. Som gruppe 1 over glemmer disse studentene også å lese kommentarene med hint som kan hjelpe dem.

Disse studentene jobber ganske ustrukturert og hopper fra ide til ide og de konsentrerer seg ikke om en ting av gangen. Dette fører til som sagt over at de begynner å tenke vanskelig siden de ikke forholder seg til en ting av gangen. Disse studentene er på nippet til å forstå Java

godt, men de ligger akkurat under det punktet der det går opp for studentene hvordan ting henger sammen, de mangler litt kunnskap før dette skjer. De løser mye av oppgaven på den erfaringen de har fra tidligere og når de sliter og får hjelp av lærer trenger han bare å gi et lite puff i riktig retning så kommer studentene selv på hvordan de kan løse problemet.

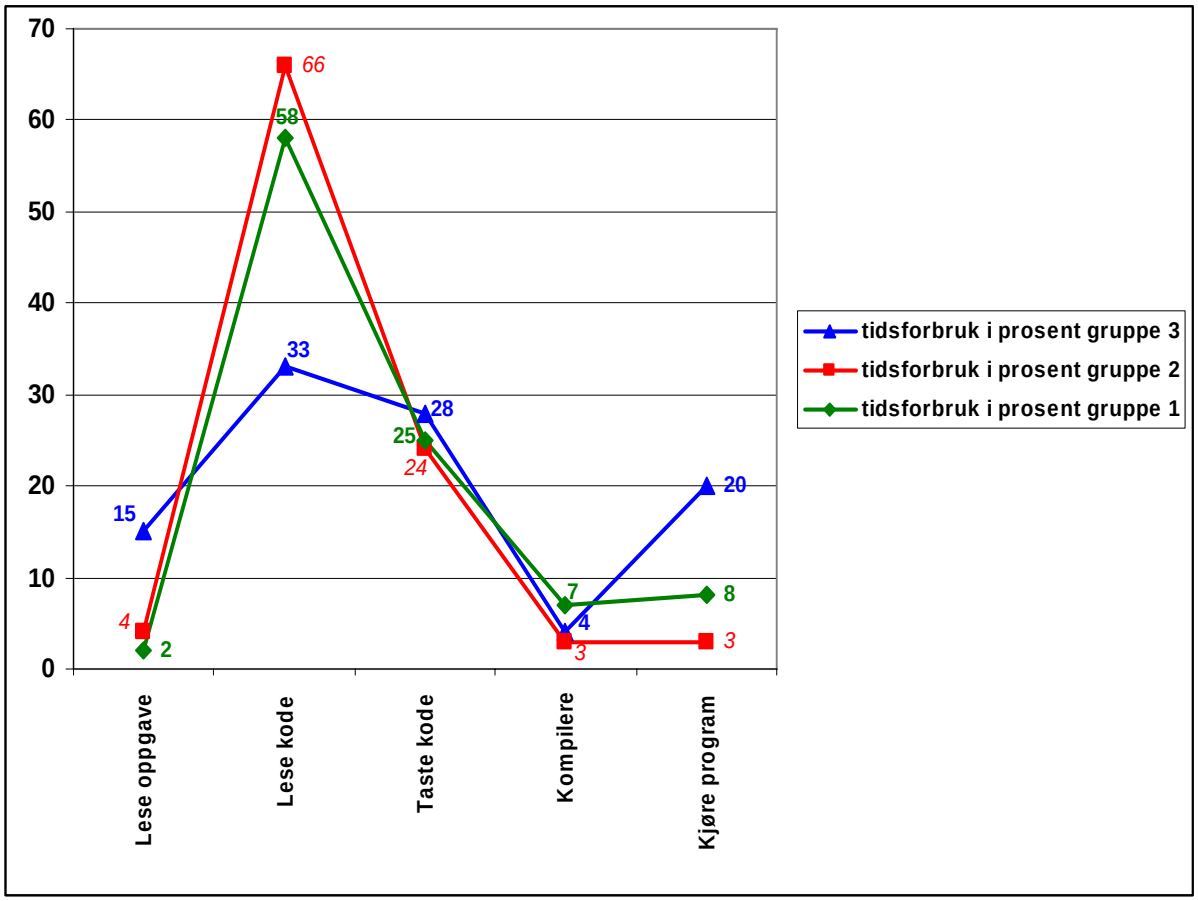
Hvis vi ser på gruppe 2 sin tabell over arbeidsaktiviteter bruker disse også mest tid på å lese kode. Dette skyldes også her at de ikke har brukt nok tid på å forstå programmet før de starter på oppgaven. Disse studentene hopper også mye fram og tilbake mellom aktivitetene, men likevel mindre enn gruppe 1. Det at gruppe 2 hopper mindre mellom aktivitetene enn gruppe 1 kan skyldes at gruppe 2 er mye mer stødig på Java, dette grunnet at jeg observerte at denne gruppen hadde bedre Java kunnskaper og at de ikke trengte å lete etter svar så mange forskjellige steder og så ofte som gruppe 1. De hadde bedre forutsetning for å få til ting enn de andre. Deres hopping mellom aktiviteter skyldes for det meste at de ikke er helt stødige på pekere og på forståelse av programmet.

Det som kjennetegner gruppe 2 er det at de hopper for fort forbi forståelsen av programmet og begynner rett på oppgaven og at de har en tendens til å jobbe for ustrukturert som fører til at det de tenker blir for vanskelig.

Gruppe 3 bruker nok tid, før de starter på oppgaven, på å sette seg inn i programmet og hva alt betyr og gjør. De husker på å følge metodekall og får med seg hva en hver del gjør i programmet, etter dette bruker de tid på å forstå oppgaven. De får dermed et godt grunnlag for å løse oppgaven. De kommer med forslag som de vet hvordan de skal skrive i Java. De har et godt grunnlag i forståelse av Java og dermed lager ikke dette noe problem for dem under oppgave løsningen. Gruppe 3 sin forståelse av Java var langt bedre enn noen av de andre gruppenes forståelse av Java. Grunnet at de har forstått programmet godt fører dette til at de med en gang kommer med gode forslag til hvordan de skal løse oppgaven og dermed ikke bruker tid på å lure på hva de skal gjøre.

Det at de har såpass gode Java kunnskaper fører til at denne gruppen får tid til å løse flere oppgaver enn de andre gruppene, og i tilegg får de muligheten til å drive med finpussing. Det viste seg at studentene i gruppe 3 har holdt på med programmering i en del år før de har begynt med programmering på universitetet. Det viser seg tydelig på filmene, da det er veldig lett å følge med på hva de gjør siden de jobber strukturert og jeg forstår hva som skjer.

Når vi ser på tabellen over arbeidsaktiviteter for gruppe 3 ser vi at det er mye mer jevnt fordelt tidsforbruk på de forskjellige aktivitetene. Disse studentene hopper mye mindre mellom de forskjellige aktivitetene enn de andre gruppene. Jeg observerte at de brukte mer tid på forståelse av programmet og at de har bedre Java kunnskaper enn de andre studentene og jeg følte dette var en av grunnene til at disse hoppet mindre mellom aktivitetene men også mindre tid på hver aktivitet. Man kan se at gruppe 3 kan fokusere på selve oppgave løsningen enn å fokusere på programmet og Java kodingen.



Tabell 14: graf for totalt brukt tid på de forskjellige aktivitetene.

I tabell 14 kan man se forskjellen på tidsforbruket på de forskjellige aktivitetene i prosent mellom gruppene. Gruppe 1 vises i den grønne grafen, gruppe 2 vises i den røde grafen og gruppe 1 vises i den blåe grafen. Den beste gruppen som var gruppe 3 vises i den blå grafen og man ser at denne gruppen har en mer jevn graf. Kompileringen vil ofte være mindre i tid enn de andre aktivitetene så denne vil gjøre grafen buete. Siden de to gruppene med mest problemer har en graf som svinger mye og den gruppen som gjør det bedre har en graf som er mer jevn, kan det se ut som om at jo jevnere kurve det er på grafen jo bedre gjør studentene det i oppgaveløsningen. Vi ser også at de gruppene som gjorde det dårligst brukte veldig liten tid på å ”lese oppgave” og at de bruke ganske mye mer tid på å ”lese kode” i forhold til gruppen som gjorde det best, som brukte mer tid på å ”lese oppgave” og som ikke hadde et så veldig stort hopp i tidsforbruk opp til å ”lese kode”. Dette kan implisere at man bør passe på å vie oppgaveteksten tid nok til å forstå hva den sier og ikke lese kode for å forstå oppgaveteksten.

Det kan merkes stor forskjell på de forskjellige gruppene. Gruppe 1 har store problemer med å løse oppgavene, mens gruppe 2 har en del problemer med å løse oppgavene og gruppe 3 har svært få problemer med å løse oppgavene. Største forskjellen på gruppene ligger på Java kunnskapene, det er også forskjeller på hvordan de jobber med å forstå programmet. De som bruker nok tid slik at de forstår programmet godt gjør det bedre under oppgaveløsningen. Så lenge studentene tar seg tid til å forstå programmet og får litt mer undervisning om pekere, parametere og objekter så vil disse studentene klare seg godt gjennom et nybegynnerkurs i Java på universitetet.

Det kan være nyttig for lærerne å informere studentene om at man bør bruke litt tid på å forstå kode, hvis det er noe ferdiglaget kode, og oppgavetekst, dette for at de virkelig skal skjønne hva de skal gjøre. Det kan være nyttig for studentene at lærerne sier fra at det er viktig å sette seg ordentlig inn i kode og oppgave for å kunne løse oppgaven slik de skal.

Hvis vi ser på gruppe 3 og tabell 10 så kan det se ut som om at den ”metoden” for å gjøre det best mulig i oppgaveløsningen består av tre steg, nemlig forståelse, skrive og teste. Forståelsessteget består av å lese oppgave og kode, skrivesteget kan sees på som tasting av koden og testesteget kan sees på som kompilering og kjøre program. Disse stegene er ikke strengt atskilte men de har en fin overgang dem i mellom. Jo bedre man fokuserer på hvert av stegene og gjør disse nøye nok før man tar fatt på det neste jo bedre vil studenten kunne klare oppgaven. Ved å utvide disse stegene kan man se likheten til artikkelen som diskuteres under.

I artikkelen av (Barnes et al. 1997) er det lagt opp til at studentene bør følge stegene ”Forståelse”, ”Design”, ”Skrive” og ”Refleksjon” for å løse problemer. Der de oppfordres til å stille spørsmål for å forstå program beskrivelsen, deretter finne liknende problem, så skrive det oppgaven sier og til slutt lese over det man har gjort en gang til. Eksperimentet jeg analyserte var lagt opp slik at studentene kunne ha klart å følge slike steg. Blant de aktivitetene studentene gikk gjennom kan man se ”Forståelses” steget som aktivitetene ”lese oppgave” og litt av ”lese kode” og at ”Design” steget også går inn i aktiviteten ”lese kode”, men det bør gjøres mer enn bare å ”lese kode”. ”Skrive” steget kan sees på som aktiviteten ”taste kode” og ”Refleksjons” steget inneholder aktivitetene ”kompilere” og ”kjøre program”, men dette steget krever mer evaluering over det arbeidet man har gjort enn de to aktivitetene som studentene er innom.

Alle gruppene gikk gjennom program beskrivelsen og prøvde å forstå alt i detalj, men ikke alle gjorde dette like grundig. Det at de ikke gjør det grundig nok kan føre til mer vanskeligheter senere i programmeringen. De leser også koden for å forstå oppgaven, vi ser her at studentene er i ”forståelses” steget. ”Design” steget er en fase studentene ikke er mye innom. Det med å finne eksempler på liknende problem kommer ikke så godt fram i dette eksperimentet jeg har sett på, men de studentene som fikk til mer hadde sannsynlig flere eksempler å ta fram i minnet enn de som sliter mer. Dette kan tyde på at dette steget er viktig for å løse problemer. Det at studentene bruker koden lærere har skrevet til å løse deler av oppgaven kan type på at de er innom ”Design” steget. ”Skrive” steget gjorde alle gruppene, det innebærer at de skriver ned den faktiske koden de kommer fram til at er løsningen. Tilslutt i ”Refleksjons” steget skal man også se tilbake på det man har gjort, det kan man se på som kompileringen og kjøringen av programmet, men det innebærer også at de evaluerer det de har gjort ved å reflektere over koden men dette gjorde ingen av gruppene. Det er noe som ikke faller seg naturlig for studenter uten at de blir bedt om det spesielt.

S. Booth (1992) at studenter løser oppgaver på 4 måter. Jeg kan gjenkjenne at studentene i dett forsøket løser oppgaver på den måten hun kaller ”operativ”, hvor de først redegjør operasjonene som programmet skal gjennomføre og deretter skriver de programmet. Det studentene gjør når de får en oppgave er å lese oppgaveteksten godt og deretter lese nødvendig kode i det eksisterende programmet for å finne rett sted å skrive koden. I denne fasen redegjør de for hva programmet skal gjøre og dermed hva operasjonene som programmet skal gjennomføre. Etter det begynner de å skrive ned koden som fullfører programmet.

9.3. Domene resonnering og relasjoner.

Jeg har funnet ut at studentene i eksperimentet bruker en del av domenene og relasjonene fra artikkelen (Kaasbøll et al. 2004). Det nye domene jeg har funnet er oppgavetekst, ellers bruker de domenene som nevnes i denne artikkelen. Jeg måtte dermed måtte jeg få med dette i bildet jeg laget av disse domenene og relasjonene. Relasjonene jeg har funnet ekstra er de relasjonene mellom domenet oppgavetekst og de andre domenene. Slik som studentene i artikkelen (Kaasbøll et al. 2004) klarte ikke studentene i dette eksperimentet heller å kode i forhold til modeller og de hadde bare mulighet til å arbeide fra kode til program, som vi kalte ”modellering ved koding”.

Som det sies i (Kaasbøll et al. 2004) at objekt orientering gir to utfordringer når man skal lære program design. Det ene man må lære seg er å kunne lage modeller av virkeligheten og det andre er at man må lære hvordan de skal designe de andre komponentene i programmet, hvordan disse skal forholde seg til modellen av den virkelige verden. Studentene i dette eksperimentet viste at disse utfordringene ba på vanskeligheter. De fleste studentene så bort i fra virkeligheten og de viste ikke tegn på at de trengte å vite noe om hvordan virkeligheten faktisk er. Studentene i de to første gruppene i dette eksperimentet demonstrerte at de drev med hasardiøs programmerings utvikling ved å se bort i fra virkelighets sjekker og ved å kode uten visuelle modeller slik som studentene i artikkelen (Kaasbøll et al. 2004).

Domenene jeg kom fram til at studentene i dette eksperimentet er innom er ”Program utførelse”, ”Det virkelige verdensdomenet”, ”Kode”, ”Oppgavetekst”, ”Permanent data fil” og ”Lærers kode”. Relasjonen mellom ”Program utførelse” og ”Det virkelige verdensdomenet” er ”Virkelighets sjekking”, relasjonene mellom ”Program utførelse” og ”Kode” er ”Modellering ved koding” og ”Justering av modellen”, relasjonen mellom ”Program utførelse” og ”Oppgavetekst” er ”Oppgavetekst sjekking”, relasjonen mellom ”Oppgavetekst” og ”Kode” er ”Design motivert av oppgavetekst”, relasjonen mellom ”Kode” og ”Permanent data fil” er ”Fil sjekking” og relasjonen mellom ”Kode” og ”Lærers kode” er ”Design ved kopiering”.

De domenene som gruppe 1 forholdt seg til var ”program utførelse”, ”kode”, ”lærers kode” og ”oppgaveteksten”. Det stedet studentene tilbringer tiden sin på er relasjonene mellom disse domenene. For gruppe 1 forholdt de seg på relasjonene ”design ved kopiering”, ”modellering ved koding”, ”oppgavetekst sjekking” og ”design motivert av oppgavetekst”. Denne gruppen bruker klart mest tid på ”modellering ved koding”. I starten forholder de seg til ”modellering ved koding” og ”design motivert av oppgavetekst”, men etter hvert går de over til ”modellering ved koding” og ”design ved kopiering”. Dette på grunn av at i starten har de behov for å vite hva oppgaveteksten sier at de skal gjøre og dermed lar dette påvirke det de gjør og når de begynner å gå vekk fra oppgaveteksten tar de bruk av kopiering for å få skrevet ned det de tenker, for de vet ikke ellers hvordan de skal skrive ting.

Det at denne gruppen virker usikker og at vi ser at de veksler en del mellom relasjonene kan ha en mulig forklaring i at de ikke vet hvordan oppgaven skal løses. Deres dårlige Java kunnskaper vises ved at de er innom relasjonen ”design ved kopiering”. Denne gruppen var den som gjorde det dårligst av alle gruppene og de var den gruppen som forholdt seg til færrest domener. Det at de forholdt seg til få domener kan skyldes at de ikke har så gode Java kunnskaper og må konsentrere seg så få domener som mulig.

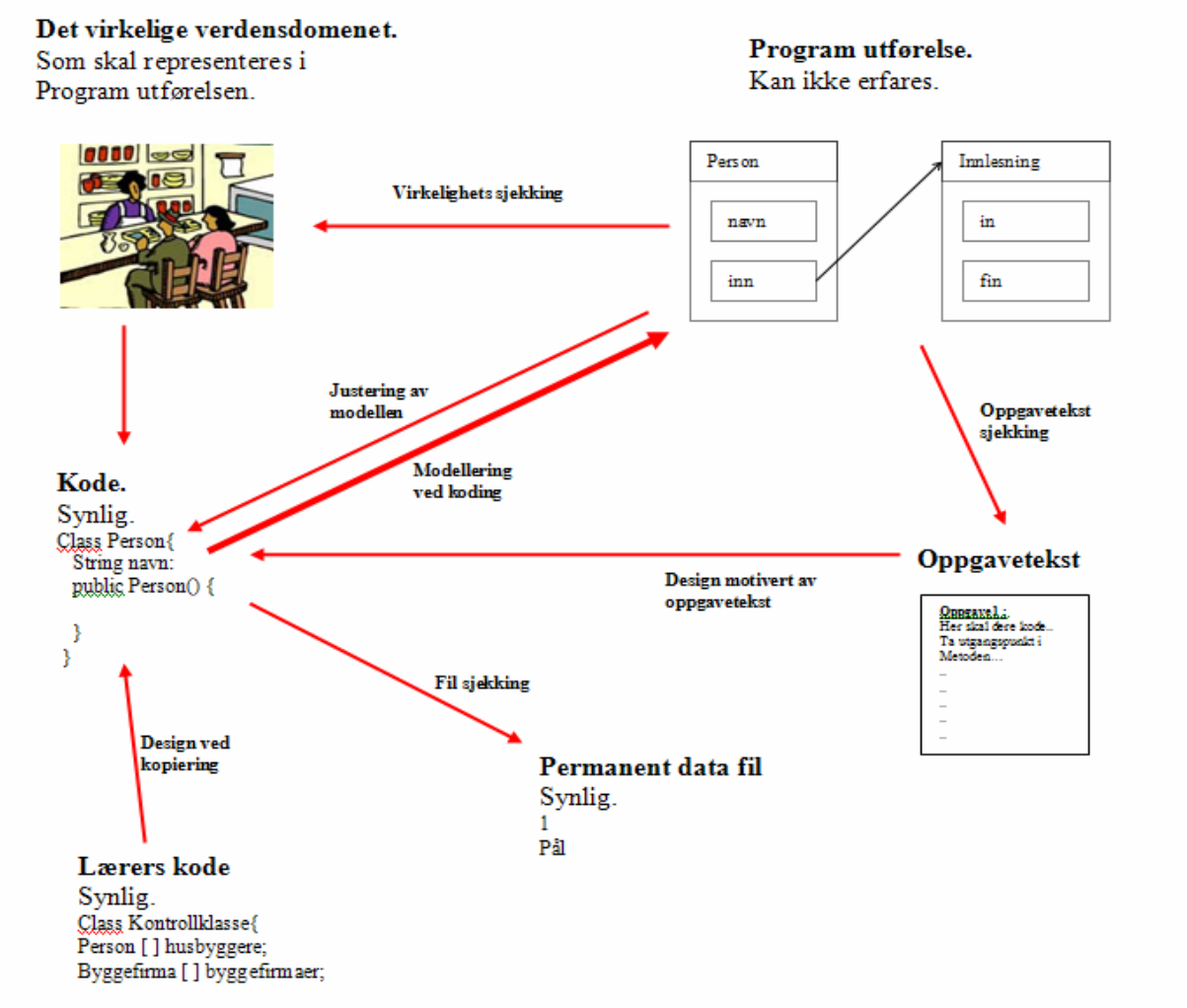
Gruppe 2 forholdt seg til domenene ”lærers kode”, ”kode”, ”programutførelse”, ”oppgavetekst” og ”permanent data fil”. Relasjonene de for holdt seg til mellom disse domenene var alle de som er beskrevet for gruppe 1, men i tillegg relasjonen ”fil sjekking”. Denne gruppen hadde behov for å forstå filene for å se sammenhengen mellom koden og filen, de brukte lite tid på denne relasjonen men for å forstå måtte de være innom denne relasjonen.

De bruker klart mest tid på relasjonen ”modellering ved koding” og deretter ”design motivert av oppgavetekst. Det at de har fokus mest på disse to relasjonene skyldes at de har dårlig oppfatning av programmet og er nødt til å bruke tiden på disse to relasjonene for å løse oppgaven. Disse studentene veksler også en del mellom de forskjellige domenene, men likevel mindre enn det gruppe 1 gjorde. I forhold til gruppe 1 bruker disse mindre tid på relasjonen ”design ved kopiering”, dette grunnet at de kan mer Java og har ikke så stort behov for dette. De fleste gangene klarer de å komme på løsninger uten at de trenger å se eksempler andre steder i koden.

Gruppe 3 er innom domenene ”kode”, ”program utførelse”, ”det virkelige verdensdomenet”, ”oppgavetekst” og ”permanent data fil”. De bruker de samme relasjonene som gruppe 2 unntatt relasjonen ”design ved kopiering”, denne relasjonen bruker de ikke i det hele tatt. Denne gruppen bruker i tilegg relasjonene ”justering av modellen” og ”virkelighets sjekking”.

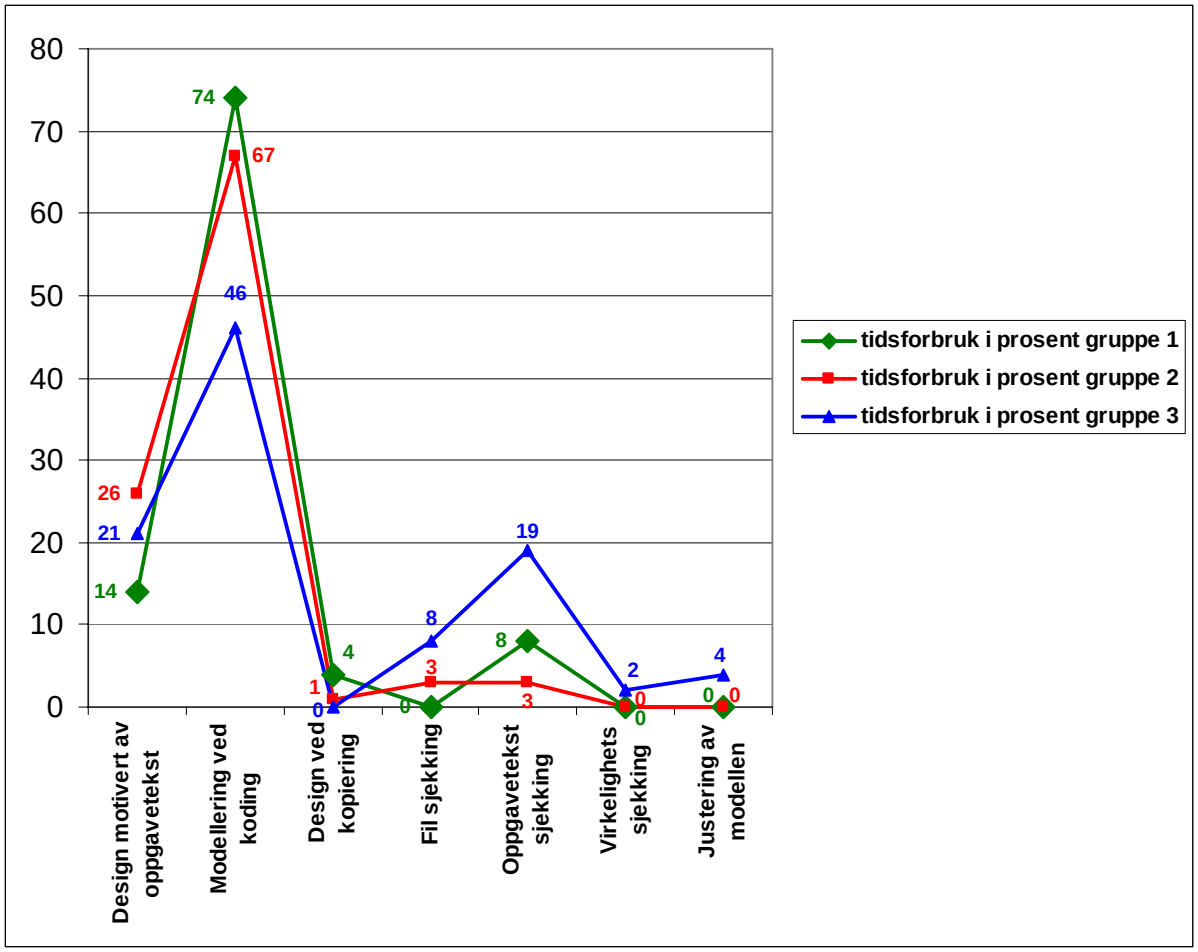
Denne gruppen bruker også mest tid på relasjonen ”modellering ved koding”, men mindre enn de andre gruppene. Disse bruker tid mer jevnt fordelt på de forskjellige relasjonene enn de andre gruppene. Dette kan skyldes at de har mer Java kunnskaper enn de andre gruppene og at de er flinkere og kan ta hensyn til flere faktorer på en gang, uten å la det bli forvirrende. Disse studentene har såpass god erfaring med Java at de har nok eksempler i hukommelsen til å løse oppgaven slik at de ikke har behov for å kopiere fra lærerens kode.

Denne gruppen har behov for å sjekke hvordan virkeligheten blir vist i koden for å få en bedre forståelse av programmet. De skjønner at de må forstå omgivelsene rundt programmet og hva det viser for å kunne skrive et program som kan fungere. Dette viser at disse studentene er ”modne” programmerere. Dette på grunn av at de skjønner at et program skal fungere i en helhet med omgivelsene. Gruppe 3 bruker også relasjonen ”justering av modellen”, dette fordi de merker at de bør forandre på modellen og koden for å få en kode som fungerer godt. I denne oppgaven var det lagt opp til at de måtte forandre på modellen litt, men dette ble beskrevet litt senere i oppgaven. Det som utmerket seg for denne gruppen var at de skjønte at de måtte forandre på modellen før dette ble beskrevet, dette viser at de ser helheten i programmet og skjønner ting må henge sammen.



Figur 49: Oversikt over hvilke domener alle studentene fokuserer på.

I figur 49 er det avbildet hvilke domener studentene fokuserer på. Denne figuren viser hvilke domener og relasjoner studenter kan ta hensyn til, de flinkeste tar hensyn til de fleste, mens de som kan mindre tar bar hensyn til noen få av de. De som kan minst vil nok utover i studiet sitt utvide sitt fokusområde og dermed ta hensyn til flere og flere av disse domenene og relasjonene etter hvert. I det eksperimentet jeg har sett på kan det virke som om studentene har de relasjonene og domenene som vist i figur 49 som utgangspunkt for sin forståelse av oppgave og program, i tillegg til å bruke dem som grunnlag for å løse oppgavene.



tabell 15: graf for tidsforbruket i prosent på relasjonene mellom domenene.

Tabell 15 er en graf som viser forskjellene mellom gruppene. Tiden de bruker på hver relasjon er angitt i prosent av den totale tiden de bruker på de forskjellige relasjonene. I dette forsøket kan det se ut som om at jo mer jevn kurve, det vil si ikke mye svingninger på kurven", jo bedre er studentene til å løse oppgaver i Java. Dette kan jeg si på grunn av at jo bedre gruppen ble observert til å løse oppgaver i Java jo jevnere kurve i grafen hadde de. Det kan være slik at studentene minsker bruken av relasjonen "design ved kopiering" ettersom de blir mer erfarne programmerere. Dermed kan man kanskje "se bort i fra" denne relasjonen når man ser etter om studentene er flinke til å løse objektorienterte oppgaver. "Design ved kopiering" brukes av studenter som er usikre og som enda ikke har fått nok programmeringserfaring, dette kan implisere at "design ved kopiering" er noe studenter som har kontroll ikke tar seg bruk av. Den fungerer som et hjelpemiddel til å komme med forslag til hvordan oppgaver skal løses. Studentene som har kontroll har ideer i minnet hvordan de kan løse oppgaver og trenger ikke lete etter eksempler andre steder for å finne løsninger.

Hvis vi ser på tabell 15 og ser på de forskjellige grafene, hva de står for og hva de sier oss, ser vi at det er forskjeller mellom gruppene. Blå kurve beskriver den gruppen som gjorde det best, deretter kommer gruppen med rød strek og til slutt gruppen med grønn strek. Dette stemmer godt overens med min observasjon av disse tre gruppene, blå kurve var den som gjorde det best som er gruppe 3. Etter det jeg observerte var gruppe 3 den gruppen som hadde mest kunnskap og fikk til mest på oppgavene. Deretter kom gruppe 2 som gjorde det nest best og

som vises i den røde kurven, deretter gruppe 1, som gjorde det ”dårligst” som vises i den grønne kurven.

Relasjon	Gruppe 1	Gruppe 2	Gruppe 3
Design motivert av oppgavetekst	710	930	300
Modellering ved koding	3690	2440	660
Design ved kopiering	210	20	0
Fil sjekking	0	110	120
Oppgavetekst sjekking	420	100	270
Virkelighets sjekking	0	0	30
Justering av modellen	0	0	60
Totalt	**5030**	**3600**	**1440**

Tabell 16: tabell for totalt brukt tid på relasjonene mellom domenene i sekunder.

Denne tabellen viser verdiene fra grafen over i en tabell men her viser tabellen verdiene i antall sekunder, slik at man lett kan sammenligne gruppenes tidsforbruk på de forskjellige relasjonene. Hvis vi ser på tabell 16 ser vi at det totale forbruket av tid ser man at gruppe 1 bruker mye mer tid en gruppe 2 og gruppe 2 bruker mer tid enn gruppe 3. Jeg gjorde om tiden til timer og minutter for å lettere se forskjellen i tid og dermed ser jeg at gruppe 3 bruker ca 24 minutter på en oppgave, mens gruppe 2 bruker ca 1 time på samme oppgave, mens gruppe 1 bruker ca 1 time og 24 minutter på den samme oppgaven. Man kan se her at det er stor forskjell på tidsforbruket på oppgaven de skal løse, og fra dette kan det sees ut som om gruppe 3 er den beste gruppen og gruppe 1 den gruppen som hadde mest problemer, dette passer godt med det jeg har observert.

Alle studentene bruker mest tid på ”modellering ved koding”, så ”design motivert av oppgavetekst” og så ”oppgavetekst sjekking”. Ut i fra min observasjon på hvor godt gruppene gjorde det kan det se ut som jo bedre gruppene gjorde det jo flere av relasjonene i tabell 12 var de innom.

9.4. Råd til undervisningen.

Jeg vil anbefale at pekere, parametere og forståelsen av hva et objekt er vil bli gjennomgått mer i undervisningen. Dette er de delene av Java programmeringsspråket studentene sliter mest med. Jeg vil tro at studentene lykkes bedre hvis disse temaene blir bedre gjennomgått slik at studentene forstår hva og hvordan de skal brukes. Studenter kan ofte få problemer med ting de ikke skjønner og syntes er abstrakt, det er derfor viktig at man forklarer alt slik at studentene får et forhold til det de har problemer med.

Det å ta hensyn til flere relasjoner gjør at sjansen for å lykkes er bedre enn hvis man bare tar hensyn til noen få relasjoner. Likevel vil jeg ikke anbefale at lærere skal få studentene til å ta hensyn til alle disse relasjonene, dette på grunn av at mange studenter vil få problemer om de skal komplisere sitt arbeid. Studenter bør sannsynligvis i starten bare ta hensyn til så få relasjoner som mulig slik at de får orden på de områdene før de utvider sitt fokusområde. Det

jeg vil oppfordre til er at lærere informerer studentene om at det er en del domener og relasjoner som man bør prøve å ta hensyn til for at man skal få en kode som er bra, slik at studentene har dette i bakhodet og at de kanskje drar nytte av det.

Det kan være lurt å lære studentene måter å løse oppgaver, slik at de vet hva de bør tenke på når de har fått en oppgave. Det å informere studentene om hva som kan være lurt å gjøre når man har fått utlevert en oppgave, kan gjøre arbeidet til studentene lettere. Mange av studentene i nybegynnerkurs vet ikke hvordan man løser oppgaver i objektorientert programmering og dermed trenger de hjelp til å komme seg over dette hinderet. Mange av disse studentene sliter med å vite hva de skal starte med, dermed kan lærere gjøre den delen av oppgaven lettere ved å gi studentene en metode, det vil si et eksempel, for hvordan de kan gå løs på oppgaver.

Som det nevnes i (Borge et al. 2003) må man ta en vurdering av alle de forskjellige metodene man kan undervise objektorientert programmering i mens de er i bruk. Dette for å se hvordan studentenes prestasjoner er i forhold til metoden det undervises i og man må også ta hensyn til hvilke mål man har med undervisningen når man skal velge undervisningsmetode. Verken (Borge et al. 2003) eller (Groven et al. 2003) gir noen råd om hva som kan være riktigst metode å bruke, men de nevner hvordan man kan undervise og hva dette innebærer.

Slik det er i dag er det mange studenter som ikke fullfører informatikkstudiet, jeg vil dermed anta at det er viktig for lærerne å bruke en metode som gjør at studentene gjennomfører et introduksjonskurs og dermed har et godt grunnlag for videre utdanning innen informatikk. Det kan virke som om dagens metode for undervisning av informatikk ikke lærer studentene nok.

Undersøkelsen viste at studentene ikke er så flinke nok til å tenke på at programmet skal være en avbilding av virkeligheten og at de glemmer at det de lager er ment for bruk av en eventuell kunde. Jeg ville dermed satset på å sette fokus mot disse temaene. Hvis jeg skulle anbefalt en metode til undervisningen ville jeg satset på bruk av verktøyet BlueJ. Det første prinsippet som læres i BlueJ er klasser og objektmodell aspekter av verden, som blir fokusområdet i starten i læringen. Programutførelsen skal representere aspekter av den virkelige verden, og dermed fungerer den slik som en dynamisk modell av den virkelige verden hvor forandringer skjer. Studentene kan følge objektene når de lages og hva som skjer med objektene under kjøringen av programmet.

Jeg mener at studentene vil komme lettere forbi de Java problemene, pekere, parametere og forståelsen av objekter, som er beskrevet i denne oppgaven og at studentene lærer seg måter å angripe problemet de settes foran i de oppgavene de får hvis de bruker BlueJ. For eksempel hvis man skal kalle en metode og den tar i mot en parameter ser man et vindu hvor man ser metodekallet med en åpen plass til parameteren som man skal fylle inn selv og metoden som kalles med parameteren metoden har. Men for å være sikker på at studentene lærer parametere og pekere bør lærerne passe på at studentene leser igjennom og forstår den faktiske koden BlueJ lager slik at de ser hvordan ting henger sammen.

BlueJ skal være enkelt å bruke for studentene og er dermed egnet for nybegynnere. BlueJ er en integrert Java-omgivelse som er spesielt designet for læring i introduksjonskurs, den ble utviklet på et Universitet og med det spesielle målet om å lære bort objektorientert programmering i Java språket. Dessuten er BlueJ gratis og er dermed vil ikke Universitet eller høyskoler tape på å legge det inn på maskinene sine.

Jeg ville anbefalt bruk av BlueJ grunnet at den tar hensyn til den faktiske virkeligheten, men det er også viktig at studentene også lærer å programmere på måten vi gjør i dag, med å kode rett inn i Emacs, slik at de ser en sammenheng mellom det de gjør i BlueJ og det de skriver fra bunnen av. Jeg tror BlueJ kan være nyttig å bruke i starten, slik at studentene lærer seg de viktigste begrepene, som klasser, objekter, pekere, parametere m.m., i objektorientering og at de har fått noen eksempler de kan bygge videre på. Disse begrepene kan studentene lære seg med interaksjon med BlueJ, slik vil de få erfaring og forståelse over disse begrepene. På slutten av studiet kan det være lurt å la de skrive kode fra bunnen av i Emacs slik som de gjør i dag for å se om de klarer å overføre det de har lært i BlueJ til ren kode uten å ha noe verktøy som lager koden. BlueJ kan også hjelpe studentene med å lære dem hvordan man kan starte med å løse oppgaver. Det er lagt opp til en måte man skal løse oppgavene, man må først lage klasser og deretter lage metoder og attributter som klassen skal inneholde og bygge ut programmet steg for steg, og dermed vil studentene lære seg en måte å begynne å løse oppgaver på. Det kan likevel være at studentene ikke lærer seg hvordan man jobber uten et verktøy som BlueJ, dette fordi studentene bare lærer hvordan man skal bruke BlueJ og ikke ser sammenhengen mellom å jobbe med BlueJ og det å løse oppgaver fra bunnen av. Dermed bør denne sammenhengen påpekes av lærerne, slik at studentene er klar over sammenhenger og eventuelle forskjelligheter.

Slik som det er i dag vil ikke studentene erfare noe av domenet program utførelse, som nevnt tidligere i oppgaven, det domenet er bare noe som skjer uten at studentene ser hva som skjer. Men i BlueJ vil studentene kunne erfare det som skjer med objektene grafisk. Når man lager et nytt objekt vil man erfare hva som skjer når man lager et objekt. Dermed blir ikke programutførelsen så abstrakt og uvirkelig for studentene. De vil kunne få mer mening med hva som skjer i programmet de skriver. Når man kjører metoder i koden vil man kunne se hva det fører til ved at man ser hvordan objektene forandrer seg. Studentene vil også kunne se den faktiske koden som programmet er skrevet i om det er ønskelig. De vil kunne skrive direkte i koden om de ville det, for å endre noe av koden. I BlueJ vil sammenhengen mellom den virkelige verden, koden og programutførelsen være lett og se. De vil se hvordan ting i koden, ting i verden og i program utførelsen henger sammen. Hvis det blir noen forandringer på en av domenene vil man se hvordan dette forandrer de to andre domenene. De vil kunne se at disse tre domenene henger sammen og at dette er noe man må ta hensyn til.

Domenet permanent data fil vil ikke få stor vektlegging. Det avhenger av om studenten selv ønsker å skjønne sammenhengen mellom fil, kode og program. Hvis man vil at studentene skal få med seg denne sammenhengen må man undervise slik at studentene føler for å forstå det. Domenene oppgavetekst og lærers kode vil fremdeles være som før, da oppgavene fremdeles vil motivere studentene i oppgaveløsningen og studentene vil bruke tidligere eksempler for å finne løsninger.

Det å bruke BlueJ er basert på antagelser om hva som kan være lurt. Det er ikke sikkert at dette er den lureste løsningen, jeg har ikke lest noen resultater om BlueJ vil føre til at flere studenter kommer seg gjennom studiet. Men jeg tror kanskje at et slikt verktøy gir studentene bedre mulighet for suksess.

10. Referanser.

1. ”Learning Object-Oriented Programming” (2004) av Jens J. Kaasbøll, Ola Berge, Richard Edvin Borge, Annita Fjuk, Christian Holmboe og Terje Samuelsen. *Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group*, Institute of Technology Carlow, Ireland, 5 – 7th April 2004.

2. ”Chapter 4 – Children’s Understanding of Object-Orientation” (2005) av Roar Granerud, Jens Kaasbøll, Richard Borge og Christian Holmboe. (*Ikke publisert enda våren 2005*)

3. ”Introductory Problem Solving in Computer Science” (1997) av David J. Barnes, Sally Fincher og Simon Thompson. *5th Annual Conference on the Teaching of Computing*, Dublin City University, pp. 36 – 39.

4. ”Exploring didactic models for programming” (1998) av Jens J. Kaasbøll, *Norsk Informatikkonferanse NIK’98*, Høgskolen I Agder 23 -25 november 1998.

5. ”Language-independent conceptual ”bugs” in novice programming” (1986) av Roy Pea. *Journal of Educational Computing Resarch*, 2, 1, 1986.

6. “Novice mistakes: are the folk wisdoms correct?” (1986) av James C. Spohrer and Elliot Soloway. *Communications of the ACM*, 29, 7, 1986, 624-632

7. “Learning to program: a phenomenographic perspective” (1992) av Shirley Booth. *ACTA Universitatis Gothoburgensis*, Göteborg studies in educational sciences 89.

8. “What is “OO first”?” (2003) av Richard Edvin Borge og Jens Kaasbøll. *Seventh Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts, ECOOP 2003*. http://prog.vub.ac.be/~imichiel/ecoop2003/workshop/#Documents

9. ”OO learning, a modelling approach” (2003) av Arne-Kristian Groven, Håvard Hegna og Ole Smørdal. *Seventh Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts, ECOOP 2003*. http://prog.vub.ac.be/~imichiel/ecoop2003/workshop/#Documents

10. ”Software Design- Cognitive Aspects” (2002) av Françoise Detienne. *Practitioner series*, Springer. Translator and Editor: Frank Bott.

11. Appendiks.

11.1. Oppgavetekst.

Kort beskrivelse av oppgavens tema

Denne oppgaven handler om personer som skal sende *byggesøknad*, som må godkjennes før de får lov til å bygge hus.

Det første personene må gjøre er å finne et *byggefirma* som kan være ansvarlig for byggesøknaden. Byggefirmaene har begrenset kapasitet med hensyn på hvor mange byggesøknader de til enhver tid kan være ansvarlige for.

Flyten av handlinger knyttet til godkjenning av en byggesøknad er som følger:
Først finner personen et byggefirma med ledig kapasitet.
På et senere tidspunkt, når et byggefirma med ledig kapasitet er funnet, vil personen som ønsker å søke om byggetillatelse fylle ut data i søknaden. Deretter sender byggefirmaet(!) søknaden til saksbehandler (i kommunen).
Saksbehandler behandler søknaden og godkjenner (eventuelt avslår) den. Beskjed om resultat sendes til byggefirma.
Dersom søknad er godkjent sendes regning fra byggefirma til person som søkte om byggetillatelse.

Oppgave 1

Vi har laget en første versjon av et program som skal løse oppgavene nevnt over. Dere skal i første omgang bli kjent med dette programmet, deretter skal dere modifisere programkoden.

Start først opp programmet . Det første dere blir spurt om er "**Har du en personfil klar?**". Svar **N**, nei, på dette spørsmålet.

Opprett isteden fire forskjellige objekter av klassen **Person** ved å gjøre fire kall på metoden **lesInnPerson()**som er tilgjengelig som et av flere menyvalg i klassen **Kontroll**. Metoden **lesInnPerson()**ber deg oppgi navn på personobjektet som deretter opprettes og legges inn i arrayen kalt **husbyggere** i klassen **Kontroll**. Arrayen **husbyggere** er en array av referanser til personobjekter. Hva betyr det? (Diskuter kort.)

Skriv deretter ut en testutskrift til skjerm som lister opp alle registrerte personer i arrayen **husbyggere**. Benytt dere av en passende ferdiglaget metode som finnes som menyvalg i klassen **Kontroll**. Hva sier testutskriften om arrayens struktur/organisering? (Diskuter kort.)

Skriv ut innholdet i arrayen **husbyggere** til fil ved hjelp av et kall på metoden **skrivHusbyggereTilFil()** før dere avslutter programmet. Denne metoden finnes også som et menyvalg.

Neste gang dere starter opp programmet kan dere svare **J**, ja, på spørsmålet "**Har du en personfil klar?**". Da leses samtlige personnavn fra fil og personobjekter opprettes og lagres i arrayen **husbyggere**.

Oppgave 2

Enhver person, representert som et personobjekt i arrayen **husbyggere**, skal kunne lete etter et byggefirma, i arrayen **byggefirmaer**, for å finne et firma med ledig kapasitet til å være ansvarlig for byggesøknaden til personen.

Arrayen **byggefirmaer** inneholder pekere til objekter av klassen **Byggefirma**. Alle data om tilgjengelige byggefirmaer blir lest inn fra fil når programmet vårt starter. Ta gjerne en testutskrift av innholdet i arrayen **byggefirmaer** ved å kalle metoden **listOppFirmaer()** fra menyen.

Det første byggefirmaet med ledig kapasitet skal velges. Hvordan finner vi ut om et byggefirmaet har ledig kapasitet? Attributtet **søknader** i objekter av klassen **Byggefirma** er en array av referanser til objekter av klassen **Byggesoknad**. Dersom et gitt byggefirma har ledig kapasitet vil det være ledig plass i arrayen **søknader**. Diskuter hva det vil si at det er ledig plass i en array av referanser til objekter!

Denne oppgaven går ut på å forandre programkoden:

Ta utgangspunkt i metoden regNySøknad()**i klassen** Kontroll. **I denne metoden kalles først en metode kalt** finnHusbygger() **som ber om et personnavn og finner riktig objekt i arrayen** husbyggere**. Dersom personobjektet finnes så kalles dette objektets metode** finnLedigFirma()**.**

Denne metoden, **finnLedigFirma()**, i klassen **Person** skal dere lage innholdet til. Den skal gjøre følgende:

Finne (velge) det første(!) byggefirmaet i arrayen **byggefirmaer** med ledig kapasitet.
Opprette et nytt objekt av klassen **Byggesoknad** og sette attributtet **statusSøknad** til verdien **uferdig**.
Plassere objektet av klassen **Byggesoknad** på ledig plass i byggefirmaets attributt **soknader**.

Kompiler deretter koden, rett feil, kompiler på nytt osv.

Når den er fri for syntaksfeil kan dere prøve følgende:
Skriv en testutskrift av personene som finnes i arrayen **husbyggere**
Kall metoden **regNySøknad()**fra menyen i **Kontroll** klassen tre ganger, dvs. for tre forskjellige personer.
regNySøknad()kaller som sagt metoden **finnLedigFirma()**som dere nettopp har skrevet.
Skriv til slutt ut en testutskrift av innholdet i arrayen **byggerfirmaer**.

Diskuter om resultatet er som forventet.

Oppgave 3

Denne oppgaven går også ut på å forandre programkoden:

Ta utgangspunkt i metoden fyllUtSøknad()**i klassen** Kontroll. **Inni denne metoden kalles først metoden** finnHusbygger(), **som ber om et personnavn og finner riktig objekt i arrayen** husbyggere **(brukt i forrige oppgave også). Dersom personobjektet finnes så kalles dette objektets metode** finnMinSøknad()**.**

Denne metoden, **finnMinSøknad()**, i klassen **Person** skal dere lage innholdet til. Den skal gjøre følgende:

- Som navnet tilsier, finn MIN søknad, skal søknadsobjektet som allerede er opprettet for personobjektet finnes. Dette er lagret i attributtet **søknader** hos det valgte byggefirmaet (jfr. forrige oppgave).
- Når riktig søknadsobjekt er funnet kalles (den ferdiglagde) metoden **fyllUtSøknad()** for å lese inn alle søknadsdataene fra tastaturet. Denne metoden er lokal i søknadsobjektet.
- Helt til slutt skal vi finne byggefirma(objektet) som er ansvarlig for søknaden. og kalle dennes metode **sendSøknadTilSaksbehandler()**. Denne metoden er allerede ferdiglaget. (Men det kan være en god ide å vente med å gjøre dette kallet til punktene over er løst.)

NB: Her vil dere garantert støte på problemer fordi dere/vi i oppgave 2 ikke lagde tilstrekkelige relasjoner mellom objektene når ny byggesøknad for en person ble opprettet og lagret hos byggefirma.

Hint: Gå først tilbake til oppgave 2 og utvid metoden(e) dere der laget. Ser dere nøye på definisjonen av **class Person** og **class Byggesoknad** så har vi foreslått noen ekstra attributter i disse klassene som dere muligens kan bruke. Dere trenger imidlertid ikke bruke alle disse attributtene, det avhenger av hvordan dere løser problemet. (Alternativt kan dere la være å bruke noen av disse og heller velge andre løsninger, for eksempel ved å definere andre (typer) attributter. Dette anbefaler vi imidlertid ikke.)

Diskuter, tegn på ark, kompiler, test ut og kompiler igjen osv. inntil dere har løst problemet. Dere har også her muligheter for å teste ut en del ved å legge inn data og å kjøre testutskrifter via menyene i programmet.

Oppgave 4

Saksbehandleren har vi representert som et såkalt aktivt objekt med en egen tråd i programmet. Det betyr at den kjører parallelt med resten av programmet og sjekker periodisk dersom den har fått en ny søknad til behandling. I så fall behandles saken. Å behandle en sak betyr i vårt tilfelle at verdien av attributtet **status_søknad** i objekter av klassen **Byggsoknad** endres fra **sendt**, en verdi den fikk ved kall på metoden **sendSøknadTilSaksbehandler()**, til enten verdien **godkjent** eller verdien **avslag**. Dette gjøres av saksbehandler og nøyaktig hvordan behøver vi ikke å fokusere på her.

Imidlertid vet saksbehandleren nøyaktig hvilket byggefirma den skal forholde seg til, gjennom en referanse til dette byggefirmaet. Når saken er ferdigbehandlet kalles derfor metoden **resultatAvBehandling()** i det aktuelle objektet av klassen **Byggefirma**.

Dere skal lage innholdet i metoden, **resultatAvBehandling()**, i klassen **Byggefirma**. Den skal gjøre følgende:

Fjern aktuell byggesøknad fra byggefirmaobjektets attributt **søknader** dersom byggesøknad er godkjent. Send deretter regning til riktig husbygger ved å kalle husbyggers metode **mottaRegning()** som bare skriver ut beløpet husbygger skylder byggefirma på skjermen.
Dersom byggesøknad ikke er godkjent kalles husbyggers metode **finnMinSøknad()** på nytt. Effekten er at nye data legges inn i søknaden før den sendes på nytt til saksbehandler.

11.2. Kode.

11.2.1. Assignmentklassen.

```
/**
 * Dette er bare en startfil som starter programmet deres.
 * Her er det ikke nødvendig å gjøre noe.
 */
class Assi
gnment {
    static Kontrollklasse oppg;
    public static void main(String [] args) {
            oppg = new Kontrollklasse();
            oppg.startSaksbehandler();  // Starter opp en saksbehandler
            oppg.lesInnData();          // Leser inn data fra fil
    }
}
```

11.2.2. Kontrollklassen.

```
import java.io.*;
/*
  Kontrollklasse, metoder:
  lesInnData(): Leser inn data fra fil, man trenger ikke bry seg om denne
  startSaksBehandler(): Setter igang en saksbehandler, man trenger ikke bry seg om denne
  meny(): Dette er hovedmenyen deres.
  lesInnPerson(): Leser inn en ny person (valg nr 6 i menyen)
  regNySøknad(): Starter registreringen av en ny søknad
  listOppFirmaer(): Lister opp firmaer tilgjengelige i byggefirmaer arrayen
  fyllUtSøknad(): Fyller ut en søknad som har status "uferdig" for husbygger
  listOppPersoner(): Lister opp personer i arrayen deres
  sjekkOmPersonFinnes(): Sjekker om en person ligger i arrayen
  finnHusbygger(String navn): Ser om en husbygger ligger i arrayen, og returnerer en peker til denne husbyggeren, man
trenger ikke bry seg om denne
  utvidArrayMedEn(): Utvider hysbyggere arrayen med en plass, man trenger ikke bry seg om denne
 */

/**
 * Dette er "Kontrollklassen" deres. Her er menyen og hvor dere skal
 * hente inn input.
 */

class Kontrollklasse {
   private Innlesing inn;          // Klasse som inneholder innlesningsverktøy
   Person [] husbyggere;           // Array med personer
   Person person;                  // Midlertidig peker til person
   Byggefirma byggefirma;          // Midlertidig peker til byggefirma
   Byggefirma [] byggefirmaer;     // Array med byggefirmaer
   Saksbehandler saksbehandler;    // Peker til en saksbehandler
   int antallPersoner;

    /* Konstruktør */
   public Kontrollklasse() {
      inn = new Innlesing();
      antallPersoner = 0;
```

```
    }


/* Metode som leser inn all data som ligger i filer, skal ikke modifiseres */

  public void lesInnData() {

     try {
              // Leser inn data fra en fil og lager en array med byggefirmaer
                    byggefirmaer = inn.lesByggefirmaer();
            }

            catch(IOException ie) {
               System.out.println("Innlesing feilet: " + ie);
               System.exit(1);
            }

            System.out.print("Har du en personfil klar (J/N)? ");
            String svar = inn.lesStreng();
            if(svar.equalsIgnoreCase("J")) {
               // Leser inn fra fil og legger i husbygger
               try {
                        husbyggere = inn.lesInnPersoner();
                        antallPersoner = husbyggere.length;
                        utvidArrayMedEn();
               }
               catch(IOException e) {
                        husbyggere = new Person[1];
                        System.out.println("Uff" + e);
               }
               meny();
            }
            else {
               System.out.println("Det finnes ikke noen egen fil med personer, du må skrive inn noen: ");
               husbyggere = new Person[1];
               meny();
            }
    }



    /* Setter igang et saksbehandler objekt */
    public void startSaksbehandler() {
            saksbehandler = new Saksbehandler();
            saksbehandler.start();
    }



    /* Her er interaksjonsmenyen til programmet, skal ikke modifiseres */
    public void meny() {
            System.out.println("*** Velkommen til testen dette husbyggerprogrammet. ***");
            System.out.println("Dette er hovedmenyen: ");
            System.out.println("1: Vis liste over firmaer.");
            System.out.println("2: Vis liste over personer.");
            System.out.println("3: Registrere en ny søknad.");
            System.out.println("4: Fyll ut søknad.");
            System.out.println("5: Sjekk om en person finnes.");
            System.out.println("6: Registrer en ny person.");
            System.out.println("7: Avslutt.");
            System.out.println();
            System.out.print("Hva vil du gjøre? ");
            int valg = inn.lesInt();
            if(valg == 1) {
               listOppFirmaer();
               meny();
            }
```

```
        else if(valg == 2) {
            listOppPersoner();
            meny();
        }
        else if(valg == 3) {
            regNySøknad();
            meny();
        }
        else if(valg == 4) {
            fyllUtSøknad();
            meny();
        }
        else if(valg == 5) {
            sjekkOmPersonFinnes();
            meny();
        }
        else if(valg == 6) {
            lesInnPerson();
            meny();
        }
        else if(valg == 7) {
            inn.skrivUtHusbyggereTilFil(husbyggere);
            System.exit(0);
        }
        else {
            meny();
        }
}

/* Registrere en person, skal ikke modifiseres */
private void lesInnPerson() {
        husbyggere[antallPersoner] = new Person();
        husbyggere[antallPersoner].registrerEnPerson();
        antallPersoner++;
        utvidArrayMedEn();
        System.out.println();
}


/* List opp firmaer, skal ikke modifiseres */
private void listOppFirmaer() {
        System.out.println("List opp firmaer: ");
        for(int i = 0; i < byggefirmaer.length; i++) {
            byggefirmaer[i].skrivUtFirma();
        }
        System.out.println();
}

/* Registrer en ny søknad, skal ikke modifiseres */
private void regNySøknad() {
        System.out.print("Hva heter du? ");
        String husbygger = inn.lesStreng();
        person = finnHusbygger(husbygger);
        if(person == null) {
            System.out.println("Personen finnes ikke!");
        }
        else {
            // Finner ledig firma og oppretter ny søknad
            person.finnLedigFirma(byggefirmaer);
        }
}

/* Fyll ut den uferdige søknaden, skal ikke modifiseres */
private void fyllUtSøknad() {
        System.out.print("Hva heter du? ");
        String husbygger = inn.lesStreng();
        person = finnHusbygger(husbygger);
```

```
            if(person == null) {
               System.out.println("Personen finnes ikke!");
            }
            else {
               // Finn min søknad som er uferdig og fyll ut
               person.finnMinSøknad(byggefirmaer);
            }

   }


   /* List opp personer, skal ikke modifiseres */
   private void listOppPersoner() {
            System.out.println("List opp personer: ");
            for(int i = 0; i < husbyggere.length-1; i++) {
               System.out.println(husbyggere[i].navn);
            }
            System.out.println();
   }

   /* Sjekker om en person finnes, skal ikke modifiseres */
   private void sjekkOmPersonFinnes() {
            System.out.print("Hva heter du? ");
            String husbygger = inn.lesStreng();
            person = finnHusbygger(husbygger);
            // Finn personen
            if(person == null) {
               System.out.println("Personen finnes ikke!");
               System.out.println();
            }
            else {
               System.out.println("Personen finnes.");
               System.out.println();
            }
   }


   /* Returnerer en person dersom han finnes i arrayen, skal ikke modifiseres */
   private Person finnHusbygger(String navn) {
            Person returnerPerson = null;
            for(int i = 0; i < husbyggere.length; i++)
               if(husbyggere[i] != null) {
                        if(navn.equalsIgnoreCase(husbyggere[i].navn))
                           returnerPerson = husbyggere[i];
               }

            return returnerPerson;
   }

   /* Utvider array med en plass, slik at den blir dynamisk, skal ikke modifiseres */
   private void utvidArrayMedEn() {
            Person []temp = new Person[husbyggere.length + 1];
            for(int i = 0; i <husbyggere.length; i++) {
               temp[i] = husbyggere[i];
            }
            husbyggere = temp;
   }

}
```

11.2.3. Personklassen.

```
import java.io.*;
/*
  Person klassen, metoder:
```

```
  registrerEnPerson(): Leser inn en persons data fra tastatur, denne trenger man ikke bry seg om
  finnLedigFirma(Byggefirma [] firmaliste): Denne skal dere skrive
  finnMinSøknad(Byggefirma [] firmaliste ): Denne skal dere skrive
 */

/**
 * Person klassen. Denne klassen identifiserer personer og
 * inneholder data om personene som er relevant. Her må det
 * kanskje legges til noe etter behov? Viktig: Navn er unik.
 */
class Person {
   Innlesing inn = new Innlesing();
   // Data
   String navn;                    // Unikt navn for hvert objekt

   /* Konstruktør */
   public Person() {

   }


   // Metode for å lese inn en person
   public void registrerEnPerson() {
            System.out.print("Navn: ");
            navn = inn.lesStreng();
   }

   // Finn ledig firma og opprett ny søknad, denne må dere skrive
   // firmaliste tilsvarer byggefirmaer i Kontrollklasse.
   // Hint: Det er en metode opprettSøknad i ByggeFirma dere kan bruke

   public void finnLedigFirma(Byggefirma [] firmaliste) {

   }


   // Finn min søknad som ennå er uferdig og fyll den ut, denne må dere skrive
   // firmaliste tilsvarer byggefirmaer i Kontrollklasse.
   // Hint: I ByggeSøknad klassen ligger det en metode fyllUtSøknad()

   public void finnMinSøknad(Byggefirma [] firmaliste) {
            ByggeSoknad søknad = null;

   }


}
```

11.2.4. Byggefirmaklassen.

```
/*
 Byggefirma klassen, metoder:
 opprettSøknad(Person pers): Oppretter en ny søknad for firmaet, her må det kanskje legges til noe?
 skrivUtFirma(): Denne skriver ut info om et firma, denne trenger dere ikke redigere
 sendSøknadTilSaksbehandler(ByggeSoknad søknad): Denne metoden sender en søknad til saksbehandler
 resultatAvSøknad(): Denne metoden skriver ut resultat av søknad
 */

/**
 * Denne klassen inneholder data for byggefirmaene som eksisterer
 * i dette programmet.
 */
class Byggefirma {
   Innlesing inn = new Innlesing();
```

```
    // Data
    ByggeSoknad[] søknader;          // Array med søknader firmaet har inne
    String navn;                     // Navn på firma
    int antall;

    /* Konstruktør */
    public Byggefirma(String navn, int kapasitet) {
            this.navn = navn;
            søknader = new ByggeSoknad[kapasitet];
            antall = 0;
    }

    /* Oppretter en ny søknad, denne kan benyttes i oppgave 2
       pers er en parameter som peker på personen som kalte metoden*/
    public void opprettSøknad(Person pers) {
            søknader[antall] = new ByggeSoknad(pers, this);
            søknader[antall].statusSøknad = "uferdig";
            antall++;
    }

    public void skrivUtFirma() {
            System.out.println();
            System.out.println("Navn på firma: " + navn);
            if(antall > 0) {
               System.out.println("Søknader som er inne: ");
               for(int i = 0; i < antall; i++) {
                        System.out.println("Søknad nummer: " + (i+1));
                        søknader[i].visSøknad();
               }
            }
            else {
               System.out.println("Ingen søknader inne.");
            }
    }

    /* Metode som sender en søknad til saksbehandleren */
    public void sendSøknadTilSaksbehandler(ByggeSoknad søknad) {
            Assignment.oppg.saksbehandler.mottaSøknad(søknad);
    }

    /* Denne metoden kalles når en søknad er ferdigbehandlet hos saksbehandler */
    public void resultatAvSøknad() {
            System.out.println(”Resultat: ”);
            antall--;
    }

}
```

11.2.5. Byggesøknadklassen.

```
/*
  ByggeSoknad klasse, metoder:
  visSøknad(): Viser info om en søknad, her må dere kanskje rette noe?
  fyllUtSøknad(): Fyller ut info om en søknad, her må det kanskje legges til noe?
 */


/**
 * Dette er søknadsklassen. Her ligger det data om alle søknadene
 * som finnes for alle firmaene.
 */
class ByggeSoknad {
    Innlesing inn = new Innlesing();
    String byggetsAdresse;      // Adressen der det skal bygges
```

```
    String byggetsAreal;
    String byggetsEtasjer;
    String statusSøknad; // Status på søknaden
    Byggefirma firma;
    Person husbygger;

    /* Konstruktør */
    public ByggeSoknad( ) {

    }

    public void visSøknad() {
            System.out.println("  Navn på søker: ");
            System.out.println("  Navn på firma: ");
            System.out.println("  Adresse: " + byggetsAdresse);
            System.out.println("  Areal: " + byggetsAreal);
            System.out.println("  Byggets etasjer. " + byggetsEtasjer);
            System.out.println();
    }

    public void fyllUtSøknad(){
            System.out.print("Adresse for byggesak: ");
            byggetsAdresse = inn.lesStreng();
            System.out.print("Areal på bygget: ");
            byggetsAreal = inn.lesStreng();
            System.out.print("Etasjer på bygget: ");
            byggetsEtasjer = inn.lesStreng();
            System.out.println();
            System.out.println();
            visSøknad();
            System.out.print("Trykk enter for å fortsette.");
            inn.lesStreng();
    }


}
```

11.2.6. Innlesningsklassen.

```
import java.io.*;
/*
  Denne klassen skal det ikke skrives noe i, her skal det bare brukes metoder fra
  De dere kanskje får bruk for (men ikke nødvendig vis!) er:
  lesInt(): Leser en int fra tastatur (det kan kun stå en int på linja!)
  lesStreng(): Leser inn HELE strengen fra tastatur
 */

/**
 * Innlesningsklassen. Denne klassen skal ikke redigeres, den inneholder
 * de verktøyene dere trenger. Dere må kanskje kikke på den for å se hva
 * slags metoder som er tilgjengelige da.
 */
class Innlesing {
    BufferedReader in;
    BufferedReader fin;

    public Innlesing() {
            in = new BufferedReader(new InputStreamReader(System.in));
    }

    public Person[] lesInnPersoner() throws IOException {
            Person[] returArray;
            int antPers;
            try {
```

```
            // Leser fra fil
            fin = new BufferedReader(new FileReader("personer.txt"));
        }
        catch(Exception e) {
            System.out.println("Fil ikke funnet!");
            System.exit(1);
        }
        antPers = Integer.parseInt(fin.readLine());
        returArray = new Person[antPers];
        for(int i = 0; i < antPers; i++) {
            returArray[i] = new Person();
            returArray[i].navn = fin.readLine();
        }
        return returArray;
}

public void skrivUtHusbyggereTilFil(Person[]personer) {
        try {
            PrintWriter out
                    = new PrintWriter(new BufferedWriter(new FileWriter("personer.txt")));

            out.println(personer.length-1);
            for(int i = 0; i < personer.length-1; i++) {
                    out.println(personer[i].navn);
            }
            out.close();
        } catch(Exception e) {

        }

}


/*
  Leser inn alle byggefirmaer fra fil og returnerer en array med
  alle byggefirmaene. Denne metoden er litt komplisert, så ikke
  kast bort tid med å skjønne denne nå. Det kan dere gjøre senere
  hvis dere er interessert.
 */
public Byggefirma[] lesByggefirmaer() throws IOException {
        // Lokale variable
        Byggefirma [] arraySomReturneres;
        int antallFirmaer, antallSoknader, søknadskapasitet;
        String tempnavn;

        try {
            // Leser fra fil
            fin = new BufferedReader(new FileReader("firmaer.txt"));
        }
        catch(Exception e) {
            System.out.println("Fil ikke funnet!");
            System.exit(1);
        }
        antallFirmaer = Integer.parseInt(fin.readLine());
        arraySomReturneres = new Byggefirma[antallFirmaer];

        for(int i = 0; i < antallFirmaer; i++) {
            // Leser inn navn på firma:
            tempnavn = fin.readLine();
            // Leser inn kapasiteten til firmaet (stort eller lite firma?)
            søknadskapasitet = Integer.parseInt(fin.readLine());
            // Oppretter et nytt Byggefirma objekt
            arraySomReturneres[i] = new Byggefirma(tempnavn, søknadskapasitet);
            // Finner ut hvor mange søknader firmaet har inne allerede
            antallSoknader = Integer.parseInt(fin.readLine());
            if(antallSoknader > søknadskapasitet)
                    antallSoknader = søknadskapasitet; // Ikke overstige kapasiteten!
```

```
                for(int j = 0; j < antallSoknader; j++) {
                        // Lage et nytt Soknad objekt
                        /*arraySomReturneres[i].søknader[j] = new ByggeSoknad();
                          arraySomReturneres[i].søknader[j].adresse = fin.readLine();
                          arraySomReturneres[i].søknader[j].beskrivelse = fin.readLine();
                          arraySomReturneres[i].søknader[j].status = fin.readLine();
                        arraySomReturneres[i].antall++;*/
                }
            }

            return arraySomReturneres;
    }


    // Denne metoden kan kun lese inn ett tall fra linjen
    // Dersom dere skriver mer enn ett tall, får dere feil
    public int lesInt() {
            int ret = -1;
            try {
                ret = Integer.parseInt(in.readLine());
            }
            catch(Exception e) {
                System.out.println("Feil i innlesing... Returnerer -1");
            }
            return ret;
    }

    // Denne metoden returnerer alt dere har skrevet på en
    // linje...
    public String lesStreng() {
            String ret = null;
            try {
                ret = in.readLine();
            }
            catch(Exception e) {
                System.out.println("Feil i innlesing. Null returnert.");
            }
            return ret;
    }
}
```

11.2.7. Saksbehandlerklassen.

```
/*
  Denne klassen skal det være unødvendig å gjøre noe med, så se bort ifra den
 */

/**
 * Dette er klassen for saksbehandler. Man ser at denne klassen
 * har noe ekstra: extends Thread. Dette betyr at et objekt av
 * denne klassen opererer på egenhånd og blir ikke oppholdt av
 * resten av programmet.
 */
class Saksbehandler extends Thread {

    private ByggeSoknad [] søknader;
    private int antall;
    Kontrollklasse klasse;

    public Saksbehandler() {
            søknader = new ByggeSoknad[100];
            antall = 0;
    }
```

```
   /* Denne metoden kjører den nye tråden (den uavhengige prosessen for saksbehandler) */
   public void run() {
            while(true) {
               try {
                        // Sov i 60 sekunder (60000 millisekunder)
                        sleep(60000);
                        if(antall == 0) {}
                        else {
                           System.out.println("\nSaksbehandleren griper etter en søknad!");
                           sleep(1000);
                           Byggefirma behandles = behandleSøknad();
                           behandles.resultatAvSøknad();
                        }
               }
               catch(InterruptedException ie) {
                        System.out.println("Saksbehandleren døde!");
                        System.exit(1);
               }
            }
   }


   /* Behandle en søknad (en random funksjon) */
   private Byggefirma behandleSøknad() {
            ByggeSoknad tilBehandling = hentEldste();
            // Henter inn byggefirma
            Byggefirma ret = tilBehandling.firma;
            if(((int)(Math.random() * 100)) < 30) {
               // Avslag
               tilBehandling.statusSøknad = "Avslått";
            }
            else {
               // Godkjent
               tilBehandling.statusSøknad = "Godkjent";
            }
            return ret;
   }

   /* Hent ut den eldste søknaden,det  denne som skal behandles. */
   private ByggeSoknad hentEldste() {
            ByggeSoknad returneres = søknader[0];
            for(int i = 0; i < antall-1; i++) {
               søknader[i] = søknader[i + 1];
            }

               søknader[antall-1] = søknader[antall];
            antall--;
            return returneres;
   }

   /* Legge inn en ny søknad i søknads registeret */
   public void mottaSøknad(ByggeSoknad søknad) {
            søknad.statusSøknad = "sendt";
            søknader[antall] = søknad;
            antall++;
   }
}
```

11.3. Filer.

11.3.1. personer.txt

(dette er en fil som lages og lagres etter at studentene har kjørt programmet en gang, det kan være flere eller færre personer i denne filen. Jeg har valgt her å vise det med de personene som vises i oversiktsbildet, som studentene har fått utlevert.)

```
4
Per
Pål
Eva
Ida
```

11.3.2. firmaer.txt

```
3
Bygg Reis Deg
1
0
Borge A/S
2
0
Groven & Samuelsen
10
0
```

11.4. Oversiktsbilde.

Dette er et bilde av deler av datastrukturen som er i programmet.

Oversikt over datastrukturer

(Dette må utvides for å kunne løse hele oppgaven)