Oblig2 OOP Teori

Oppgave 1

* Class

Alt i java skjer i en eller flere klasser. En klasse er en type oppskrift/mal for et element/objekt. Klassen spesifiserer hvordan et objekt lages og hvordan dataen skal håndteres. Objektet er da en instans av klassen. Når vi skal registrere et element angir vi et klassenavn, og i denne klassen angir vi hva som skal registreres for elementet. For eksempel om vi vil registrere studenter og kurs får vi disse to klassene hvor vi i student blant annet vil ha fornavn, etternavn, alder, studentID og kurs som instansvariabler, mens i klassen med kurs vil vi ha kursnavn, kurskode og studiepoeng som instansvariabler. Med kun disse instansvariablene har vi ikke noe objekt enda, for at det skal bli et objekt må vi oppgi det med «Student student1 = new Student ();» i en annen klasse. Da opprettes en student kalt student1 ved at vi henviser til klassen Student først. Deretter kan vi oppgi verdier som navn og alder ved å skrive «student1.fornavn = ‘fornavn’; student1.alder = 19;».

* Object (konseptet, ikke klassen)

Object er en realisering av en klasse. Objekter er et element som opprettes basert på klassen, og i objektet setter vi verdier for dette nye elementet. Når vi setter verdier for objektet henviser vi til klassen, for eksempel student.fornavn for å sette verdi for fornavnet til studenten.

For å opprette et objekt benytter vi oss av klassen, for eksempel «Student student1 = new Student ();». Her navngir vi en variabel, det er derimot ikke et objekt enda da det ikke har noen verdi men kun er en variabel som kan referere til et objekt. For at det skal bli et objekt må vi angi noen verdier «student1.fornavn =’Tora’; student1.alder=19;».

* Instansvariabel

Instansvariabler er objektets egenskaper, for eksempel når det kommer til en student har han/hun blant annet egenskapene fornavn, etternavn og studentID som vi da kan ha som instansvariabler.

* private,public,(protected) (klasse,variabel,metode)

Når vi angir at variabler i en klasse er private vil det si at verdien av variablene kun kan endres av klassen de ligger i. Dersom vi vil kunne endre verdien til en variabel utenfor denne klassen blir vi nødt til å lage metoder som er public som henter ut og kan sette verdien til variabelen vi vil bruke.

Om du setter en variabel som public istedenfor private, tillater du alle å endre på eller hente ut verdien til variabelen avhengig av om du bruker set-er eller get-er. Da kan verdien endres eller hentes ut i andre klasser, pakker og subklasser.

For å kunne tillate andre klasser å hente ut verdien til en variabel i en annen klasse blir vi nødt til å angi en public get-metode i klassen til variabelen. I denne get-metoden returnerer vi variabelen. Da kan man hente ut variabelen i andre klasser ved å bruke getVariabelNavn();.

Samme gjelder dersom vi vil kunne endre verdien av variabelen i en annen klasse. Da må vi lage en public set-metode i klassen hvor variabelen ligger. I denne set-metoden skriver vi «this.variabelNavn = variabelNavn;» og denne endres ved at vi kan skrive «student1.fornavn = ‘Kari’;» hvis vi fortsetter med samme eksempel på klasse som over i forklaringen på objekt og klasse.

Altså ved å ha variabler som private kan vi beskytte de fra å bli endret eller hentet ut i andre klasser. Dette gjør det også mulig å kontrollere og sette visse krav for verdien til variabelen. For eksempel hvis vi vil kreve at fornavnet skal inneholde minst tre tegn kan vi angi dette i en if-setning; «if (fornavn.length == 3)» i set-metoden som ligger i klassen hvor variabelen er.

* Overloading

Vi kan både overloade metoder og overloade konstruktører, og overloadingen skjer alltid i en og samme klasse. Når det kommer til overloading i forhold til metoder er det slik at overloading gir en klasse muligheten til å ha mer enn en metode med samme navn så lenge parameterne er forskjellige. Når dette er tilfellet fortelles metodene at de er overloadet. Dette er en måte å implementere polymorfisme. For å overloade en metode trenger du kun å lage to forskjellige versjoner av samme metode også gjør kompilatoren resten av jobben. Derimot er det viktig at parameterne er forskjellig i de to metodene, men det som returneres kan også være forskjellig og de kan også ha i ulikt antall parametere. Når koden kjøres vil den metoden som matcher med parameteret/ene til spørringen/ene i main-filen være de som utføres. Dersom du har flere metoder som heter «overload» hvor en av dem har ett int-parameter mens den andre har to, og det står «Overload overload1 = new Overload(); overload1.overload(1,2); System.out.println();» vil metoden med to parameter være den som kjøres og printes ut.

* Overriding

Hodet er likt men vi endrer innholdet i kroppen. Altså metodenavnet, returtypen og parameteret er det samme men verdien/innholdet på det vi returnerer er annerledes. Grunnen til at vi bruker override er at koden blir mer stabil og det gjør det mulig å få hjelp av IntelliJ til å si i fra hvis vi har gjort noe feil, for eksempel små skrivefeil som å stave parameteret feil.

Override blir benyttet når vi vil benytte oss av samme metode som i en annen klasse men vi ønsker å gjøre noe annet i metoden. Altså lar vi metoden hete det samme, vi bruker samme returtype og parametere men vi endrer logikken i metoden, altså vi returnerer noe annet. Vi kan kun benytte oss av metoder og variabler som er public, altså om vi ønsker å bruke en variabel fra en annen klasse i logikken må vi bruke get-metoden. En klasse kan kun arve en metode som er public.

Ved å bruke override overskriver vi metoden vi arver fra til å gjøre noe annet. Altså vi endrer på innholdet/logikken i den første metoden men ved å gjøre det i vår egen klassen med override.

* Extends

Extends benytter vi for å lage en subklasse. Det blir brukt i for at en klasse skal arve egenskaper fra en annen. Da kan man i subklassen referere til variabler i den første klassen som om de lå i subklassen. Når vi da vil lage ny objekter til subklassen kan vi angi verdier for variabelene både fra subklassen og fra klassen som subklassen arver av.

* this og super

Når en metode kjøres blir den automatisk henvendt til et argument som referer til det påkalte objektet og denne henvendelsen kalles for this. Altså med «this» refererer man til det aktuelle/nåværende objektet som metoden eller konstruktøren kaller på/henviser til. «This» er altså for å fortelle hvilken variabel du henviser til eller ikke. For eksempel når variabelen i en konstruktør heter det samme som instansvariabelen du skal bruke. Da kan du benytte «this.variabelNavn = variabelNavn;», for å angi at det er den første variabelen som skal være lik den i konstruktøren.

Når du lager en subklasse som arver av en annen klasse kaller vi den andre klassen for superklasse. Hvis du har en metode som overrider en av superklassen sin sine metoder kan du påkalle metoden som blir overridet ved å bruke «super». Altså for å kunne referere til metoden i superklassen i subklassen må det i overriden kalles på metoden i superklassen med super først med bruk av super. For eksempel «super.printUtInfo();» dersom metoden heter printUtInfo();.

* Polymorphism

Polymorfi er at alle objekter kan utgi seg for å være et objekt av alle klassene objektet arver fra. Polymorfi gjør det mulig å behandle en gruppe med forskjellige figurer likt, for eksempel rektangel, sirkel og kvadrat. De behandles da likt som samme klasse.

Polymorfisme betyr flerformhet. I java innebærer det at et objekt eller en instanse av en klasse kan ha flere former/typer. Altså kan blant annet en student også være en person som igjen er et objekt. Da er studenten både en student, person og et objekt. Samme blir med ansatt som både kan være ansatt, person og objekt. Derimot er ikke en ansatt også en student.

Alt er et objekt og alt arver fra objekt. Altså er en person et objekt og en student eller ansatt er også objekter.

En student og en ansatt arver begge fra person. Dermed kan vi lage arraylister hvor vi definerer type objekt som student eller ansatt, og henter ut informasjonen som ligger i variabelen til det valgte objektet. Dersom vi ønsker å hente ut informasjonen om alle personene uavhengig av om de er studenter eller ansatte, kan vi lage en arrayliste med person som objekt. Da kan vi legge til alle personene i lista og skrive ut info om alle i lista ved hjelp av en for-løkke.

Oppgave 2

Når vi kompilerer et java-program gjøres teksten og koden om til bytekode som JVM’en

forstår. Derfor må vi kompilere koden slik at JVM’en skal kunne kjøre programmet og for at det skal fungere i terminalen. Så fort koden er kompilert kan man kjøre koden i terminalen. Denne bytekoden som koden vår kompileres til lagres på maskinen i noen nye filer med samme filnavn men med endelse «.class», derfor må alle klasser som er public lagres i forskjellige java-filer hvor filnavnet er likt som klassenavnet.