Oblig 2 – Programmering 2 - Teori

Skriv en oversikt over hva følgende ord/begreper betyr:

- Class
- Object (konseptet, ikke klassen)
- Instansvariabel
- Overloading
- Overriding
- Extends
- Polymorphism
- private,public,(protected) (klasse,variabel,metode)
- this og super

Class:

En klasse kan forklares ved å tenke på klasse som en blueprint (plantegning) over noe du skal lage (et objekt). Klassen inneholder alt du trenger å ha for å kunne lage noe av disse plantegningene. Ved hjelp av klasser kan vi skape noe (data) som speiler det som finnes rundt oss i verden. En bil kan være et eksempel – du har en blueprint som spesifiserer hvordan lage et objekt (bilen, med chassis, vinduer, dører, osv).

Object:

Objekter er det vi lager ved hjelp av klassene (blueprint). Objekter er instanser av klassen. Hvis vi fortsetter med bilen som eksempel så kan vi lage forskjellige biler som har ulike egenskaper. En bil kan være rød med 4 dører, en annen er en blå cabriolet med 2 dører.

Instansvariabel:

Instansvariabler er variabler som defineres i klassen. Med bilen som eksempel kan vi f.eks sette instansvariabler som farge, motortype, hestekrefter, etc. Instansvariabler kan ha hvilken som helst datatype definert til seg. Instansvariablene kan få en verdi satt når de defineres, eller senere i en konstruktør-metode. Det anses som best-practice å sette access-modifier til «private» for instansvariablene da dette innebærer å innkapsle variablene slik at det ikke direkte adgang til klassens «indre tilstand».

Overloading:

Overloading er det at en klasse kan ha mer enn en metode (funksjon) med samme navn, så lenge parametere som metoden tar er forskjellige. Overloading kan være gunstig i at det kan gjøre koden enklere å lese og forstå da forskjellige metoder altså har samme navn, men forskjellige input-parametere. Overloading kan også gjøre det enklere å bruke klassen/metoden da det er mye lettere å huske ett metode-navn for en metode som utfører lignende operasjoner.

Overriding:

Når en metode i en subklasse har samme navn, return type og parametere som en metode i superklassen (foreldreklassen) så innebærer dette at metoden i subklassen «overrider» metoden i foreldreklassen. For eksempel toString() metoden er en metode som er definert i superklassen «Object», så kan vi override denne ved å spesifisere hvordan vi ønsker at en utskrift av f.eks et Planet-Object skal se ut (da metoden uten overriding printer ut en ikke så lesevennlig utskrift av objektet).

Extends:

Extends brukes når vi skal spesifisere at en klasse arver fra en annen klasse. Det som arves er attributtene og metodene. Konstruktørene arves ikke. Arv er en av nøkkel-prinsippene til OOP. Selv om konstruktører ikke arves kan en klass som har arvet kalle på konstruktøren til foreldreklassen ved å bruke super(). Extend (det å arve) er begrenset til en klasse. Altså, for eksempel kan klassen «bil» bruke extend for å arve fra klassen «kjøretøy». Bil kan ikke nå bruke enda en «extends» til å arve fra en annen klasse.

Polymorphism:

Det finnes to typer polymorfisme: compile-time og run-time polymorfisme. Polymorfisme betyr noe som «mange former» - speiler at en metode kan fungere på litt forskjellige måter basert på hvilket objekt metoden brukes på.

Compile-time: denne typen polymorfisme skjer via metoder og overloading av disse, altså hvor vi har metoder med samme navn men med forskjellige inputparametere. Metodene har da forskjellig method signature, og compilern bruker denne signaturen til å skille mellom metoder som er overloaded. Signaturen består av metode-navnet og parameterne som tas.

Run-time: denne typen polymorfisme skjer via metode-overriding og bestemmes i run-time, derav navnet run-time polymorphism. Dette skjer altså når en metode i en sub-klasse har en metode med samme navn, return type og parametere som sin superklasse: Igjen kan eksemplet med toString() brukes. toString() arves hos alle objekter fra den øverste superklassen Object i Java – derfor kunne jeg override denne til å skrive ut planetene på en måte som bedre passer det som oppgaven(e) krever, og for leselighetens skyld.

Private, public, protected – klasser, variabler og metoder:

«private»: Variabler og metoder som settes som private kan kun nås fra klassen hvor de er definert. Disse kan ikke nås utenfor klassen, ikke heller fra klasser som arver fra klassen hvor de er definert. Dette gjelder selvfølgelig så lenge det ikke er satt public setters /getters for disse private variablene.

I java kan vi ikke ha det som kalles top-level, altså øverste nivå klasser som er private.

Oppsummert så er «private» en restriktiv access-modifier, den begrenser tilgang utenfor selve klassen hvor variabler og metoder er definert.

«public»: Class – som nevnt tillater ikke java top-level klasser å være «private». Public klasser kan nås fra alle andre klasser. Variabler og metoder som defineres som public kan også nås fra alle andre klasser.

Public er en tillatende access-modifier – den tillater adgang til klasser, variabler og metoder uten restriksjoner.

«**protected**»: variabler og metoder som er satt til protected kan nås innen samme «package» men også fra sub-klasser selv om disse er i forskjellige «packages». Protected begrenser noe adgang, men ikke like mye som private gjør.

Klasser (top level) tillates ikke å være satt til protected, likt som for private.

«this» og «super»:

This: brukes for å referere til f.eks instansvariablene i en klasse når disse skal brukes i en metode eller en konstruktør. This.radius = radius, hvor this vil referere til instansvariabelen radius. Brukes for å differensiere mellom metode/konstruktørvariabler som har samme navn. This refererer altså tilbake til klassen hvor vi har våre instansvariabler.

Super: brukes ved arv for å referere til foreldreklassen. Hvis klassen «Car» har arvet fra klassen «Vehicle» via extends, kan vi bruke super for å f.eks kalle på en metode i Car-klassen (som er definert i forledreklassen Vehicle) eksempelvis: super.drive();