Programmering 2
Nhut Pham

Teorioppgaver 1.1

- Class

  En klasse er en del av OOP (Objektorientert Programmering) som er en gitt måte å strukturerekoden vår på. Klasse er noe vi kan reflektere egenskapen til noe i den «virkelige» verden, dette er ikke alltid sant da det finnes eksempler for at dette ikke stemmer, eks når vi jobber med UI.
  Vi bruker klasse når vi ønsker å samle/gruppere like objekter sammen. Klasse er bygd opp av variabler og metoder. Klassenavnet skrives med stor forbokstav, og deretter stor forbokstav på det første bokstaven i ordene.

- Object (konseptet, ikke klassen)

  Objekt konseptet bruker vi når vi ønsker å konstruere et spesifikt objekt eller en instans til en klasse. Eksempel på dette kan være at en person klasse har forskjellige instansvariabel. Disse variablene bruker vi for å lage spesifikke objekter som vi ønsker å ha. Magnus Karlsen er et objekt av klassen person, Emilia Clarke er et annet objekt til person klassen.

- Instansvariabel

  En instansvariabel brukes når vi det er snakk om objektorientert programmering, dette er en variabel som defineres i en klasse, men utenfor metoden. Hvert objekt (konseptet) i en klasse bruker disse instansvariabel for å lage en separat kopi.

- Overloading

  Overloading betyr at vi kan lage flere metoder med samme navn. Det som skiller disse metodene, er antall parameter og rekkefølgen på datatyper. Selv om overloading kan returnere forskjellige retur-typer, så klarer den ikke å se forskjellen på disse. Vi kan bruke overloading for som et alternative for å opprette objekter av en klasse.

  Eks:

  Disse to metodene vil fungere siden det differensiere på antall parameter.

  I de to neste metodene kan vi se at den andre metoden ikke fungerer da de har den samme datatypen i samme rekkefølge, selv om vi har byttet plassering på x og y.

```
public int sum(int x, int y) {
return x + y;
}

public int sum(int x, int y, int z) {
return x + y + z;
}

public double sum(double x, int y) {
return x + y;
}

public double sum(double y, int x) {
return x + y;
}
```

- Overriding

  Ved å bruke overriding så kan vi «overstyre» arvede metoder. Det betyr at vi kan tilpasse metoden som har blitt arvet fra en annen klasse for det gitte

barneklassen. Selv om vi kan tilpasse dette, så må fortsatt returtype og navn være den samme. Det er parameter og kodelogikken som kan defineres om. Overriding er et konsept under Polymorfi.

Får å bruke overriding har vi metoden toString() fra Objekt-klassen, denne metoden brukes til å skrive ut objekt-beskrivelsen.

For å være sikker på at arvet metode blir overskrevet kan vi bruke @Override, hvis det ikke gjøres vil vi få «Error».

- Extends

  Extends brukes når vi har en barneklasse som skal arve metoder som en foreldreklasse har.

  Eks: Vi har en Person-klasse, denne klassen er da en foreldre-klasse eller «Super» class. Under denne klassen kan vi ha flere barne-klasse som kan eks. være «Elektriker». Får å kunne arve metoden fra en foreldre-klasse må vi bruke Extends.

  public class Elektriker extends Person {

  public antallSikringskap()

  public antallProsjekt()

  }

  Det som skjer, er at konstruktøren i barne-klassen bruker det som er fra konstruktøren i foreldreklassen.

- Polymorphism

  Ett av kjerne-prinsippene i OOP. Det finnest to typer polymorfi, «Metode-polymorfi» og «Objekt-polymorfi».

  Metode-polymorfi er det vi tidligere i oppgaven har skrevet om, «Overriding» og «Overloading».

  Objekt-polymorfi vil si at et objekt av en barneklasse er også samtidig et objekt av foreldreklassen den arver fra.

- Private, public, protected (klasse, variabel, metode)

  Er det vi bruker for å definere en klasse, variabel og metoder. Når vi bruker public, så kan det det brukes i andre klasse og metoder enn der det er blitt laget. For å bruke dette i andre klasser så bruker vi eks. metodeNavn.getName, for å hente ut navn som er lagret i variablen for i den klassen som metoden ligger i.

Bruker vi private, så kan den bare brukes i den klassen som vi har opprettet den i. Har vi da et variabel og en parameter med den samme navn, så må vi definere med *.this for variablen som er i klassen. For å bruke disse metodene andre steder så må vi bruke konstruktør, og i tillegg sette «set/get» metoden for konstruktøren.

Protected betyr at vi kan bruke metoden, variablene i klassen som vi oppretter den i, men også for alle barneklasser.

- This og Super

  This brukes for å referere til seg selv, slik som tidligere nevnt i oppgave 1.1 Polymorphism – Private.

  Super kan brukes når vi ønsker å kalle på foreldreklassen variabler og metoder. Super() brukes når vi ønsker å kalle på konstruktøren.