IN1010 Gruppetime uke 2

Gruppe 2 - Design

Subklasser

- Klasser som arver egenskaper fra en annen klasse
  - Slike klasser kalles *subklasser*
  - Vi kaller klassen som arves fra for en *superklasse*
- Bruker nøkkelordet **extends** for å si at en klasse er en subklasse av en annen klasse
- Subklasser arver instansvariabler og metoder fra superklassen (og evt superklassens superklasser)

```
class Klasse { /* ... */ }

class Subklasse extends Klasse { /* ... */ }
```

Hvorfor arv?

- Noen klasser vi ønsker å skrive kan være veldig like hverandre og ha delte egenskaper, men også noen tilleggsegenskaper
  - For eksempel, te og kaffe er begge koffeinholdige drikker, men de er forskjellige type drikker
- Vi kan samle objekter som arver fra samme klasse
- Slipper å gjenskrive kode som er felles for flere klasser

Implisitt og eksplisitt casting

- Man kan bruke en referanse til en klasse til å lagre et objekt av en subklasse til denne klassen
  - Referanser har også en type, ikke bare objekte
- Dersom man lagrer et objekt i en referanse med typen til en superklasse, så vil objektet konverteres automatisk
  - Implisitt casting
- Hvis man vil konvertere dette objektet tilbake til dens opprinnelige subklasse (eller en annen superklasse), må man *eksplisitt* konvertere objektet
  - Eksplisitt casting

```java
public class Sirkusdyr { }

public class Elefant extends Sirkusdyr { }

public class Hvalross extends Sirkusdyr { }
```

```java
public static void main(String[] args) {
    Sirkusdyr sirkusdyr = new Elefant(); // Implisitt casting
    Sirkusdyr sirkusdyr2 = new Hvalross();

    if (sirkusdyr instanceof Elefant) { // Før eksplisitt konvertering
        Elefant elefant = (Elefant) sirkusdyr;
    }
}
```

Abstrakte klasser

- Abstrakte klasser deklareres med nøkkelordet *abstract*
- Det er ikke lov å lage objekter av abstrakte klasser
  - Egner seg derfor godt for arv, siden vi kan ha subklasser som er ikke-abstrakte

Object

- Klassen object er en abstrakt klasse som *alle* objekter i Java arver fra
- Til og med klasser vi definerer selv ender opp med å være en subklasse av Object selv om vi ikke definerer det
- Derfor har alle objekter noen felles metoder

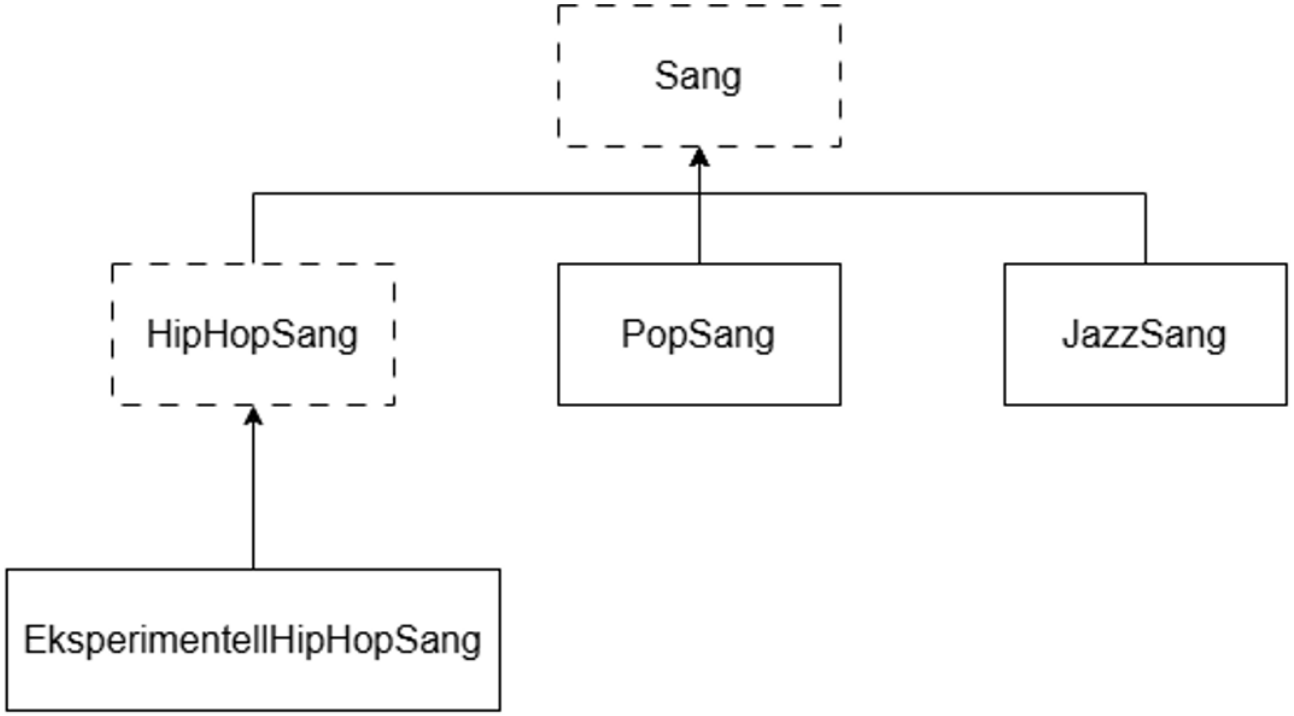
Super

- Dersom man har en superklasse med en konstruktør må subklassene arve konstruktøren
- I subklassens konstruktør, kall på *super*(<argumenter>) øverst i konstruktøren
- Viktig selv om superklassen ikke tar inn argumenter i konstruktøren

Private, public og protected

- Private
  - Kun tilgjengelig i klassens skop
- Public
  - Tilgjengelig for alle
- Protected
  - Kun tilgjengelig for alt i samme 'pakke' og alle subklasser
  - Dersom vi ønsker at en subklasse skal aksessere subklassen sine instansvariabler, må disse deklareres med protected

Klassehierarki



Oppgave 1

Lag et program som administrerer en samling (HashMap) av kjøretøy. Programmet skal ha følgende klasser:

1. Kjøretøy-klasse:
    - Egenskaper: id (String), merke (String), modell (String), år (int)
    - Konstruktør: for å initialisere egenskapene
    - Hent-metoder og sett-metoder for egenskapene (kun hent-metode for id)
2. Bil-klasse (arver fra Kjøretøy):
    - Ekstra egenskap: seter (int)
    - Konstruktør: for å initialisere egenskapene (inkludert de arvede egenskapene fra Kjøretøy)
    - Hent-metode og sett-metode for egenskapen seter
3. Motorsykkel-klasse (arver fra Kjøretøy):
    - Ekstra egenskap: motortype (String)
    - Konstruktør: for å initialisere egenskapene (inkludert de arvede egenskapene fra Kjøretøy)
    - Hent-metoder og sett-metoder for egenskapene
4. Program-klasse:
    - Main-metode: Opprett en samling (HashMap) for å lagre kjøretøy. Legg inn to instanser (objekter) av Bil og to instanser (objekter) av Motorsykkel. Skriv ut detaljene for kjøretøyene i samlingen.

Oppgave 2

Du lever i en verden der kunstig intelligens florerer. Enkelte roboter er mer høyteknologiske enn andre. Her skal vi ta for oss GPT, men merk at det kan eksistere roboter med navn som starter med noe annet enn GPT. Ved opprettelse av objekter, tar konstruktøren til hver robot inn et navn. Samtidig skal det ikke være mulig å opprette instanser/objekter av Robot. Det skal også finnes en metode som returnerer navnet til roboten.

GPT 3, GPT 3.5 og GPT 4 har alle noen delte egenskaper. For eksempel kommuniserer de med mennesker. Altså har de et navn og en samling *delte* egenskaper. I tillegg har GPT 3, GPT 3.5 og GPT 4 en samling *unike* egenskaper de ikke deler med de andre. Når vi snakker om samling, mener vi her enten array eller liste.

a. Tegn et klassehierarki over robotene i robotfamilien beskrevet ovenfor.
b. Implementér klassene med utgangspunkt i klassehierarkiet og informasjonen du har
c. Opprett metoder for å legge til og fjerne egenskaper fra de ulike robotene. Du kan anta at egenskaper som legges til ikke finnes fra før, og du kan anta at egenskaper som skal fjernes eksisterer.
d. Skriv og utfør enhetstester, som tester klassene og metodene du har implementert. Hvorfor er enhetstesting nyttig? Diskutér gjerne med en medstudent.