Seminar uke 3

Arv (subklasser)

I Java er det mulig å arve attributter og metoder fra en klasse til en annen.

- En klasse beskriver objekter med visse felles egenskaper
- En subklasse beskriver objekter som har de samme egenskapene, men i tillegg har objektene i subklassene flere/spesielle egenskaper
- En subklasse *arver* egenskapene til superklassen

Subklasse (barn) - klassen som arver fra en annen klasse

Superklassse (opphav) - klassen den arver fra

Hvorfor bruke arv?

→ Nyttig når man skal gjenbruke kode, gjenbruke attributter og metoder av en klasse når man skal lage en ny, lignende klasse. Samme felles egenskaper i stedenfor å kode det samme flere ganger.

Generalisering: Samle de felles metodene og attributtene

Spesialisering: Skille ut de metodene og attributtene som kun gjelder for en spesifikk klasse

Begreper

`Private` i en klasse gjør at ingen subklasser kan se denne egenskapen

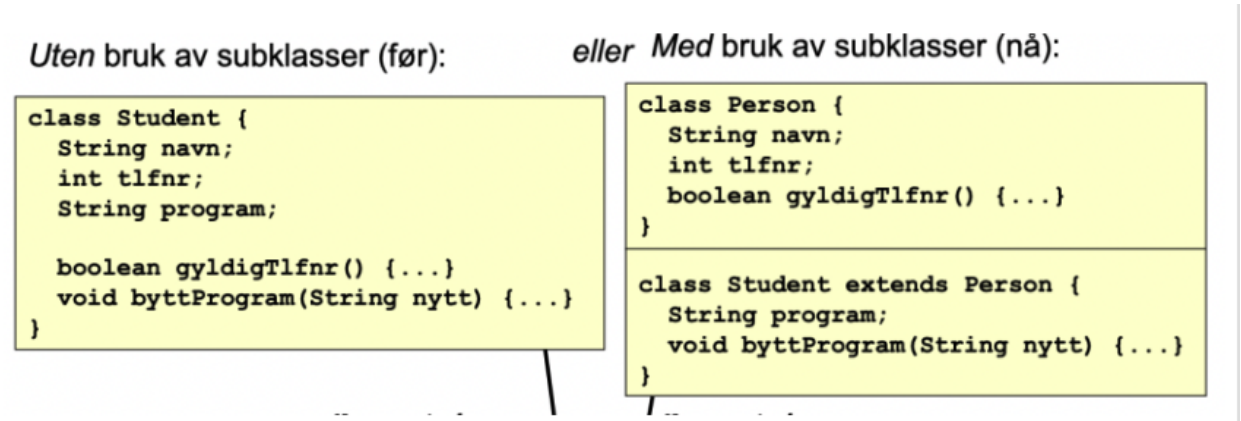
`Protected` i en klasse gjør at alle subklasser kan se denne egenskapen, men ingen utenfor klassen (bortsett fra i samme katalog/pakke)

`Public` er som før

`Final` gjør at klassen ikke kan arves, det er den siste klassen (kommer feilmelding *dersom man bruker **extends** fra en **Final** class)*

```
final class Vehicle {
 ...
}
```

💡 For å arve bruker vi Java nøkkelordet `extends`



Eksempel 1

```
class Vehicle {
  protected String brand = "Ford";        // Vehicle attribute

  public void honk() {                    // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
}
```

```
class Hovedprogram {
  public static void main(String[] args) {
      // Create a myCar object
      Car myCar = new Car();

      // Call the honk() method (from the Vehicle class) on the myCar object
      myCar.honk();

      // Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from the Car class
      System.out.println(myCar.brand + " " + myCar.modelName);
    }
}
```

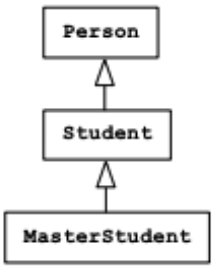
- **Car** er subklassen
- **Vehicle** er superklassen
- Vehicle sin attributt **brand** blir satt til `protected`
  - Hadde den vært satt til private som så ville ikke **Car** hatt tilgang til den
  - Om det hadde stått “*private* String brand”, så ville ingen subklasser til **Vehicle** kunne se denne egenskapen

Eksempel 2

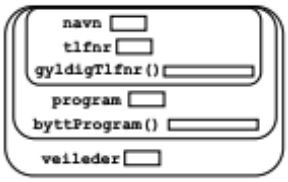
```java
class Person {
    String navn;
    int tlfnr;
    boolean gyldigTlfnr() { . . . }
}
```

```java
class Student extends Person {
    String program;
    void byttProgram(String nytt) { . . .  }
}
```

```java
class MasterStudent extends Student {
    String veileder;
}
```



MasterStudent-objekt




```java
class Person {
  protected String navn;
  protected int tlfnr;

  public boolean gyldigTlfNr(){
    return tlfnr >= 10000000 && tlfnr <= 99999999;
  }
}

class Student extends Person {
  protected String program;

  public Student(String p){
    program = p;
  }

  public void byttProgram(String nyttProgram){
    program = nyttProgram;
  }
}

class MasterStudent extends Student {
  protected String veileder;

  public void giVeileder(String navn){
    veileder = navn;
  }
}
```

Abstract

→ Skjule visse detaljer og bare vise viktige detaljer av et objekt

Abstract class: kan ikke lage objekter (har tilgang til klassen, men må arves fra en annen klasse). Forskjell fra ***final,*** kan arve videre men ikke opprette objekter av klassen.

Abstract metode: kan kun brukes i en abstract class. Kan arves fra subklassen. Metode uten innhold, men som "fullføres" en en subklasse som ikke er abstract.

En abstract klasse kan både ha abstract og vanlige metoder

Eksempel 3

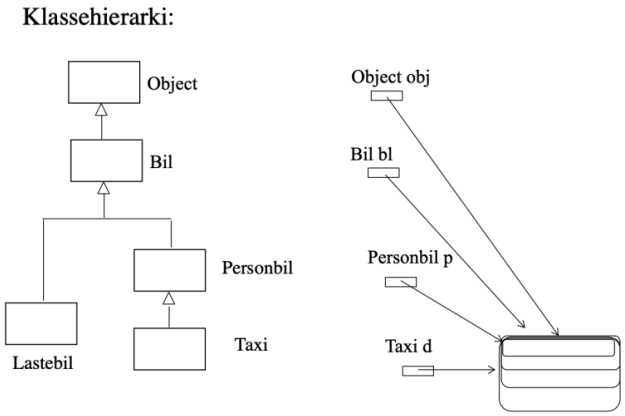
```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}
```

```
class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

- Ikke lov med `new() Animal`
- Ikke lov med `animal.animalSound()`, men f.eks. `myPig.animalSound()` er lov

Class Object

- Alle klassers mor
- Alle klassen i Java er subklaser av kalssen *Object*
  - `class Person {...}` tolker Java som `class Person extends Object {...}`
- Alle klasser har automatisk mange metoder (ikke ordentlig gjennomgått enda)
  - toString(), equals() etc.



Casting

```
class Student extends Person {...}

Student stud = new Student();
// ---
Person pers;
pers = stud;
```

Foretatt en implisitt konvertering fra Student- til Person-referanse (”bytter” klasse referanse)

Konverteringen kan ikke gå “oppover” - kun “nedover” / ”bortover”

- Person pers kan referere til et Student objekt
- Student stud kan **ikke** referere til et Person objekt (class Person “høyere opp”)

```
Student stud2 = (Student) pers;
```

Eksplisitt konvertering tilbake til *Student* for å få tak i de spesielle Student-egenskapene

→ Dette kalles **“casting”** (class-cast / typekonvertering)

Instanceof

→ Sjekker hva slags objekt det er (nyttig for å unngå feil)

```
class A {}
class B extends A {}
class C extends A {}
class D extends C {}

class TrueOrFalse {
  public static void main(String[] args) {
    A a = new A();
    B b = new B();
    C c = new C();
    D d = new D();
    A e = new D();
    A f = new B();

    System.out.println("a) " + (b instanceof A)); // true
```

```
    System.out.println("b) " + (c instanceof D)); // false
    System.out.println("c) " + (d instanceof A)); // true
    System.out.println("d) " + (e instanceof C)); // true
    System.out.println("e) " + (f instanceof C)); // false
  }
}
```