Abstract Classes and Interfaces
kapitel 12

Joakim Iversen

January 13, 2025

Disposition:

1.

Noter:

Main concepts

- abstract Classes
- Interfaces
- Multiple Inheritance

KODE EKSEMPEL: FOXES-AND-RABBITS SIMULATION

Simulation

Kode eksemplet tager udgangspunkt i at simulere populationen af ræve og kaniner i et indelukket reservat og er bare et enkelt eksempel på *predator-prey simulations*

Klasserne Simulation, Fox og Rabbit er de spændende klasser og dem vi vil have fokus på. Fox og Rabbit giver os nogle basale handlinger for rovdyr og bytte. Det er dog ikke en biologisk præcis genskabelse af kaniner og ræve.

Simulation klassen står for at lave start stadiet af simulationen og kontrollere og kører det. Det indeholder collections af ræve og kanniner og giver hver af disse lov til at agere i et "step" af deres livscyklus. på den måde repræsentere "steps" en mængde tid.

Abstract Classes

Vi ser at klasserne, som de er givet i starten, for kanniner og ræve vil have mulighed for at blive samlet under en superklasse kaldet *Animal*, i det der er meget kode duplikation på tværs af disse to klasser - og det vil gøre noget af vores kode i simulation klassen nemmere.

Vi starter med at samle fælles elementer fra **Fox** og **Rabbit** klassen til en superklasse. Dette er, felt-variabler og metoder.

Efter at have gjort dette kan vi nu pynte vores simulation klasse yderliggere. I stedet for at ittere over to forskellige lister kan vi samle det hele til en fælles collections af animals, og så itterere over den

```
for(Iterator<Animal> it = animal.iterator(); it.hasNext(); ) {
    Animal animal = it.next();
    if(animal instanceof Rabbit) {
        Rabbit rabit = (Rabbit) animal;
        rabbit.run(newAnimals);
    }
    else if(animal instanceof Fox) {
        Fox fox = (Fox) animal;
        fox.hunt(newAnimals);
    }
    else {
        System.out.println("found unkown animal");
    }

    // Remove dead animals from the simulation.
    if(! animal.isAlive()) {
        it.remove();
    }
}
```

vi kan se vi stadig skal sortere efter hvlket dyr vi har fat i. Men vi har samlet det hele, gennemløbning af ræve og kanniner, til en fælles gennemløbning. En måde at undgå dette er at lave en metode i superklassen og lade subklasserne overskive denne metode. På den måde behøver vi ikke at skulle tage forbehold for de to forskellige dyr.

```
// Let all animals act.
for(Iterator<Animal> it = animals.iterator(); it.hasNext(); ) {
    Animal animal = it.next();
    animal.act(newAnimals);
    // Remove dead animals from the simulation.
    if(! animal.isAlive()) {
        it.remove();
    }
}
```

Vi kan lave denne *act* metode abstract hvilket er kendetegnet ved to detaljer:

- Prefixed med keyword **abstract**
- Det har ikke en metode krop, men dens *header* er afsluttet med et semikolon.

```
abstract public void act(List<Animal> newAnimals);
```

En klasse vil også kunne være abstrakt. En klasse der ikke er abstrakt - Alle klasser der ikke er definiere som abstrakt vil automatisk ikke være det - er det vi kalder *Concrete classes*.

Der er tre fordele ved at lave en klasse abstrakt:

- Man kan ikke lave et objekt af en abstrakt klasse
  - At bruge det reserverede ord **new** til at skabe en abstrakt klasse vil give en fejl
  - Dette sikre man aldrig vil kunne skabe et objekt af en abstrakt klasse, men det kune fungere som en superklasse til dets subklasser.
- Kun abstrakte klasser kan have abstrakte metoder.
  - Dette sikre at en konkret klasse ikke vil have metoder, som ikke kan udføres.
- En abstrakt klasse med abstrakte medtoder tvinger subklassen til at overskrive disse metoder.
  - Så for at en sub klasse skal blive en konkret klasse, skal den overskrive alle metoderne af den abstrakte superklasse. Da den ellers selv ville være en abstrakt klasse, og man vil ikke kunne lave et objekt af den type.

Interfaces

Der er nogle ting man skal huske når man bruger et interface:

- **Interface** keyword istedet for klasse i headeren
- Interface har ikke nogle konstruktør
- Interface har ikke nogle instans variabler
- Kun static og final class fields med public synlighed må være inde i et interface.

Hoved forskellen på et interface og en abstrakt klasse, er at et interface ikke har defineret nogle af sine metoder, derudover ikke have feltvariabler.

Når vi vil lave et interface bruger vi følgende header:

```
public interface Actor
{
    body
}
```

Her kan vi se at vi har byttet *class* ud med *interface*. Derudover har det et andet reserveret ord til når en klasse skal implementere et interface

```
public class Fox extends Animal implements Actor
{
    Body
}
```

Her er **Fox** en subklasse af Animal og implementere Actor interfacet.