# ex4types

Joachim von Hacht

# Subtyper för Generiska Typer

```
Box<Integer> m1 = new Box<>();
Box<Double> m2 = new Box<>();

Box<Object> m3 = m1;                 // No
Box<Object> m4 = (Box<Object>) m1;   // No
Box<Object> m5 = m2;                 // No
```

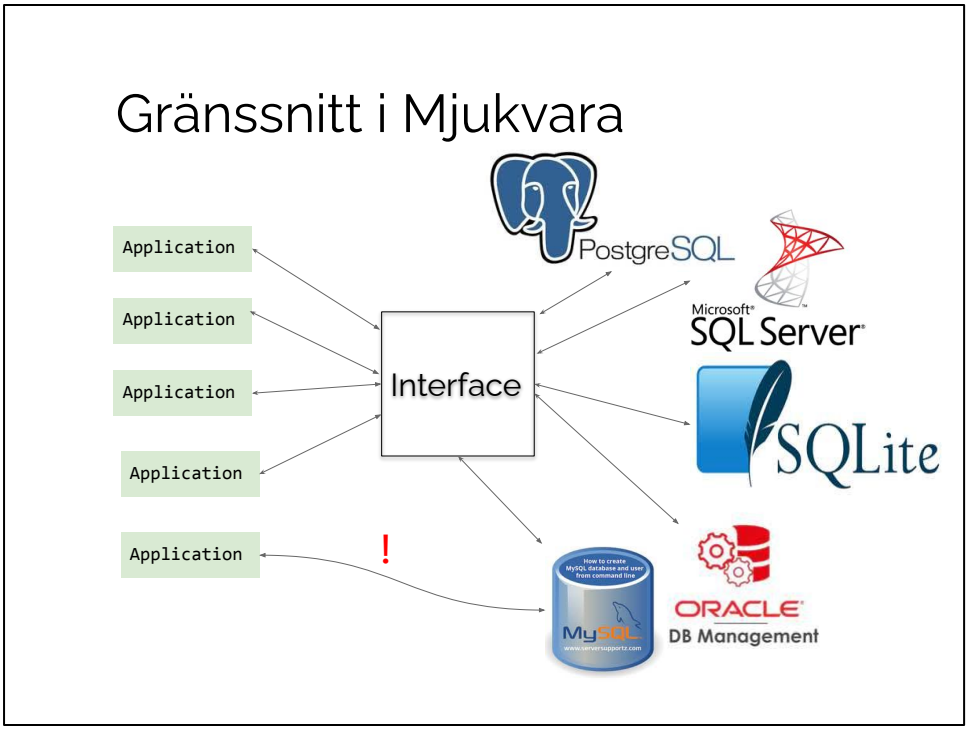
Generiska typer har inget super/subtype förhållande.

- Även om typargumentet har super/sub
  - Integer <: Object men:  Box<Integer> är inte subtyp till Box<Object>
- Går att åstadkomma super/subtyp relation, avancerat kommer i följande kurser.

# Gränssnitt i Hårdvara

Idéen med gränssnitt...

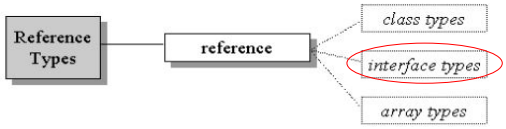
- ...handlar om ett möte mellan två olika intressenter (levande eller döda)
- ...är att dölja hur saker sker internt.
    - Uttaget vet inget om vilken apparat kontakten sitter på. Vet bara om gränssnittet (två pinnar + jord + formen). Om gränssnittet stämmer levereras ström.
    - Kontakten vet inget om hur tillverkning/leverans av ström fungerar. Spelar ingen roll, det intressanta är om det går att få ström.
    - Vi kan alltså använda vilket uttag som helst för vilken apparat som helst bara de uppfyller gränssnittets krav.
- Om kontakten inte uppfyller gränssnittet fungerar det inte.




Genom att applikationerna använder ett gränssnitt kan vilken databas som helst (som uppfyller gränssnittet) användas.

- Man kan dessutom byta ut databasen utan att ändra i koden i applikationen.
- Applikationen som inte går via gränssnittet kan bara använda MySQL (använder att databasspecifikt protokoll)

# Gränssnitt




```
// No implementation only declaration
// of operations (methods)
public interface MyList ... {
    boolean isEmpty();
    boolean add(int i);
    int get(int index);
}

MyList myList = new . // No can't instantiate
```

En typ definierar vilka operationer som finns men inte hur de är implementerade (kan finnas flera tänkbara implementationer)!

- En klass definierar operationer och implementering (på samma gång).
    - Inte säkert vi vet hur implementering skall gå till
    - Kanske vill kunna byta implementering (även under körning)
- Dvs. i vissa fall onödigt att binda sig till en fix implementering (i klassen)

Ett **gränssnitt** (interface) introducerar en typ och därmed ett antal operationer, .. men

- Implementeringen av metoderna finns inte med!
    - D.v.s. metoderna saknar kroppar. De går inte att exekvera (de är abstrakta).
    - Implementeringen av metoderna får någon annan sköta (senare)
    - Vi kan använda typen, deklarera variabler samt utföra operationer på dessa utan att vi vet exakt vad som kommer att exekveras (senare)

Man kan inte instansiera ett gränssnitt med new-operatorn.

- Vi kan inte skapa ett objekt som saknar körbara metoder.

# Implementera ett Gränssnitt

```
public class MyListImpl implements MyList {
    @Override
    boolean isEmpty(){ … }     // Method bodies here
    @Override
    boolean add(int e) { … }
    @Override
    int get(int index){ … }
    …
}
```

För att få körbara kod för metoderna deklarerade i gränssnittet låter man någon klass implementera (implements) gränssnittet.

- Innebär att alla metoder från gränssnittet måste implementeras i klassen.
    - Kontrolleras av kompilatorn.
    - Vi anger @Override så att kompilatorn kontrollerar att vi verkligen använder samma metodhuvud som i gränssnittet.
- Vi har på detta sätt fått ett objekt som kan utföra operationerna deklarerade i gränssnittet.

OBS! Att vi på detta sätt kan få flera olika objekt som kan utföra operationerna givna i gränssnittet.

- Vi kan alltså välja mellan olika implementationer!

# Subtyper för Gränssnitt

```
interface Sayable {
   String say();
}

class Cat implements Sayable {
   @Override
   public String say() {
       return "Mjau";
   }
}


Sayable sayable = new Cat();  // Super = sub
sayable.say();                // Will run ok
```

Olika gränssnitt har inget super/sub förhållande

- Däremot kommer implementerande klass att bli subtyp till gränssnittstypen.
- Naturligt: Kompilatorn kontrollerar att alla metoder som deklareras i gränssnittet finns implementerat i klassen. Kommer alltid att fungera.

Typomvandlingar till/från gränssnitt komplicerat, återkommer.

- Vi konstaterar att man kan typomvandla gränssnittstyper till/från vilken annan typ som helst.

# Gränssnitt och Abstraktion

```java
// This works for Cat, Dog, ..any
// that implements Sayable
// Abstraction: Don't care what it
// is, just that it can say()
void doSay(Sayable s) {
   out.println(s.say());
}
```

```java
interface Sayable {
   String say();
}

class Cat implements Sayable {
   @Override
   public String say() {
       return "Mjau";
   }
}

class Dog implements Sayable {
   @Override
   public String say() {
       return "Voff";
   }
}
```

Gränssnitt ger oss möjlighet att hantera olika typer genom att bara beakta en viss aspekt av objekten, nämligen vilket gränssnitt de implementerar.

- Genom att använda gränssnitt som parametrar och/eller returtyper kan metoder hantera olika objekt på ett uniformt sätt.
- Vi får en generellare kod.
- Man kan se det som att objekt kan ha flera olika typer, de är mångformiga (**polymorfa**)