EDAA45 Programmering, grundkurs

Läsvecka 7: Arv

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2016

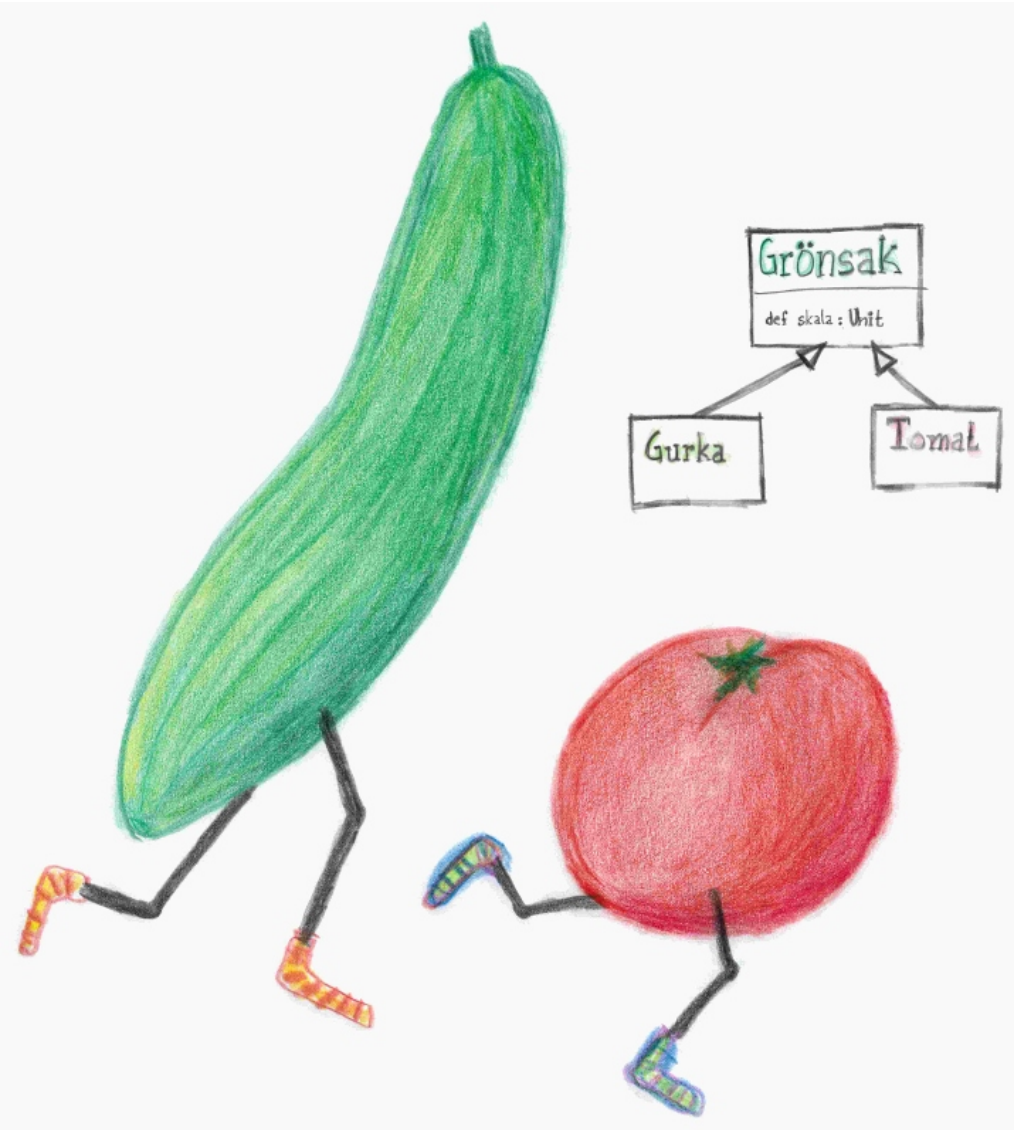
7 Arv

- Arv och nyckelordet `extends`
- Överskugging
- Trait eller abstrakt klass?

Arv och nyckelordet `extends`

Vad är arv?

Med arv kan man beskriva relationen
X **är en** *Y*



Varför behövs arv?

- Man kan använda arv för att dela upp kod i:
  - **generella** (gemensamma) delar och
  - **specifika** (specialanpassade) delar.
- Man kan åstadkomma **kontrollerad flexibilitet**:
  - Klientkod kan **utvidga** (eng. *extend*) ett givet API med egna specifika tillägg.
- Man kan använda arv för att deklarera en gemensam **bastyp** så att generiska samlingar kan ges en mer specifik elementtyp.
  - Det räcker att man vet bastypen för att kunna anropa gemensamma metoder på alla element i samlingen.

Behovet av gemensam bastyp

```
1  scala> class Gurka(val vikt: Int)
2
3  scala> class Tomat(val vikt: Int)
4
5  scala> val gurkor = Vector(new Gurka(200), new Gurka(300))
6  gurkor: scala.collection.immutable.Vector[Gurka] =
7    Vector(Gurka@60856961, Gurka@2fd953a6)
8
9  scala> gurkor.map(_.vikt)
10 res0: scala.collection.immutable.Vector[Int] = Vector(200, 300)
11
12 scala> val grönsaker = Vector(new Gurka(200), new Tomat(42))
13 grönsaker: scala.collection.immutable.Vector[Object] =
14   Vector(Gurka@669253b7, Tomat@5305c37d)
15
16 scala> grönsaker.map(_.vikt)
17 <console>:15: error: value vikt is not a member of Object
18        grönsaker.map(_.vikt)
```

Kan vi inte ordna en mer specifik typ än `Object`?

Scalas typhierarki

Any, AnyVal, AnyRef, Object

Skapa en gemensam bastyp

Behovet av gemensamma delar

Överskugging

Medlemmar, arv och överskuggning

Olika sorters överskuggningsbara medlemmar i klasser och traits i **Scala**:

- `def`
- `val`
- `lazy val`
- `var`

Medlemmar, arv och överskuggning

Olika sorters överskuggningsbara medlemmar i klasser och traits i **Scala**:

- `def`
- `val`
- `lazy val`
- `var`

Olika sorters överskuggningsbara instansmedlemmar i **Java**:

- variabel
- metod

Medlemmar som är `static` kan ej överskuggas (men döljas) vid arv.

Medlemmar, arv och överskuggning

Olika sorters överskuggningsbara medlemmar i klasser och traits i **Scala**:

- `def`
- `val`
- `lazy val`
- `var`

Olika sorters överskuggningsbara instansmedlemmar i **Java**:

- variabel
- metod

Medlemmar som är `static` kan ej överskuggas (men döljas) vid arv.

- När man överskuggar (eng. *override*) en medlemmen med en annan medlem med samma namn i en subtyp, får denna medlem en (ny) implementation.
- När man konstruerar ett objektorienterat språk gäller det att man definierar sunda överskuggningsregler vid arv. Detta är förvånansvärt knepigt.
- Singelobjekt kan ej ärvas. Medlemmar i singelobjekt kan ej överskuggas.

Fördjupning: Regler för överskuggning i Scala

En medlem M1 i en supertyp får överskuggas av en medlem M2 i en subtyp, enligt dessa regler:

1. M1 och M2 ska ha samma namn och typerna ska matcha.
2. `def` får bytas ut mot: `def`, `val`, `var`, `lazy val`
3. `val` får bytas ut mot: `val`, och om M1 är abstrakt mot en `lazy val`.
4. `var` får bara bytas ut mot en `var`.
5. `lazy val` får bara bytas ut mot en `lazy val`.
6. Om en medlem i en supertyp är abstrakt *behöver* man inte använda nyckelordet `override` i subtypen. (Men det är bra att göra det ändå så att kompilatorn hjälper dig att kolla att du verkligen överskuggar något.)
7. Om en medlem i en supertyp är konkret *måste* man använda nyckelordet `override` i subtypen, annars ges kompileringsfel.
8. M1 får inte vara `final`.
9. M1 får inte vara `private` eller `private[this]`, men kan vara `private[X]` om M2 också är `private[X]`, eller `private[Y]` om X innehåller Y.
10. Om M1 är `protected` måste även M2 vara det.

Fördjupning: Regler för överskuggning i Java

`http://docs.oracle.com/javase/tutorial/java/IandI/override.html`

Trait eller abstrakt klass?

Trait eller abstrakt klass?

Använd en **trait** som supertyp om...

- ...du är osäker på vilket som är bäst. (Du kan alltid ändra till en abstrakt klass senare.)
- ...du vill kunna mixa in din trait tillsammans med andra traits.
- ...du bara har abstrakta medlemmar.

Använd en **abstract class** som supertyp om...

- ...du vill ge supertypen en parameter vid konstruktion.
- ...du vill ärva supertypen från klasser skrivna i Java.
- ...du vill minimera vad som behöver omkompileras vid ändringar.