EDAA45 Programmering, grundkurs

Läsvecka 9: Mönster, Undantag

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2016

9 Mönster, Undantag

- Specialundervisning
- Nyhet: Scala 2.12.0
- Veckans lab: `chords-team`
- Matchning
- Mönster
- Option
- Implementera `equals` med `match`
- Undantag
- `scala.util.Try`

Specialundervisning

Specialundervisning

Under vecka w09 och w10 (till att börja med) kommer vi att organisera **specialundervisning** under dessa **resurstider**:

- **Torsdag kl 8-10** i både **Falk** och **Val** (Gustav, Valthor, Emil)
- **Torsdag kl 10-12** i **Falk** (Maj)
- OBS! Dessa rumtider är till för de som hade 0 eller 1 på kontrollskrivningen och som ansökt om specialundervisning via länk i speciell mejlinbjudan.

Undantag: Om du har mer än 1 på kontrollskrivningen men inte alls har möjlighet att gå på någon annan resurstid än ovan är du också välkommen; anmäl då din situation till handledaren på plats och så får du vara med i gruppen och kan få svar på frågor etc. som vanligt.

Nyhet: Scala 2.12.0

Nyhet: Scala 2.12.0 släpptes 3:e Nov 2016

- Nytt i Scala 2.12:
  - **Optimeringar** ”under huven” som **kräver Java 8**
  - **Snabbare, klarar sig med mindre minne, kortare bytekod, ...**
  - Väsentligt förbättrad **Scaladoc**:
    `http://www.scala-lang.org/api`
  - Du hittar gamla Scaladoc för 2.11.8 här:
    `http://www.scala-lang.org/api/2.11.8/`
- I denna kursomgång och på LTH:s datorer kommer vi att stanna kvar vid **2.11.8** (nästa kursomgång kör vi 2.12)
- Observera att 2.12 **inte är bytekodskompatibel** med 2.11 så du måste kompilera om all gammal kod om den ska funka med nykompilerad kod om du installerar 2.12.
- Med `sbt` (se appendix G) är det enkelt att ha många olika versioner av Scala-kompilatorn igång på samma maskin.

För den intresserade, läs mer här: `http://www.scala-lang.org/news/2.12.0`

Veckans lab: chords-team

Veckans lab: `chords-team`

Övergripande syfte:

- Träna på case-klasser, matchning, undantag
- Jobba med ett större program med flera klasser i olika filer
- Jobba flera personer på samma program

Innehåll:

- Skapa och spara ackord på gitarr (6 strängar) och ukulele (4 strängar)
- Spela upp ackord med Javas inbyggda midispelare inkapslad i `SimpleNotePlayer`
- Rita ackord med `SimpleWindow`

Hur mycket ni gör beror på hur många ni är i gruppen och hur stora ambitioner ni har. Diskutera detta med handledare på resurstid.

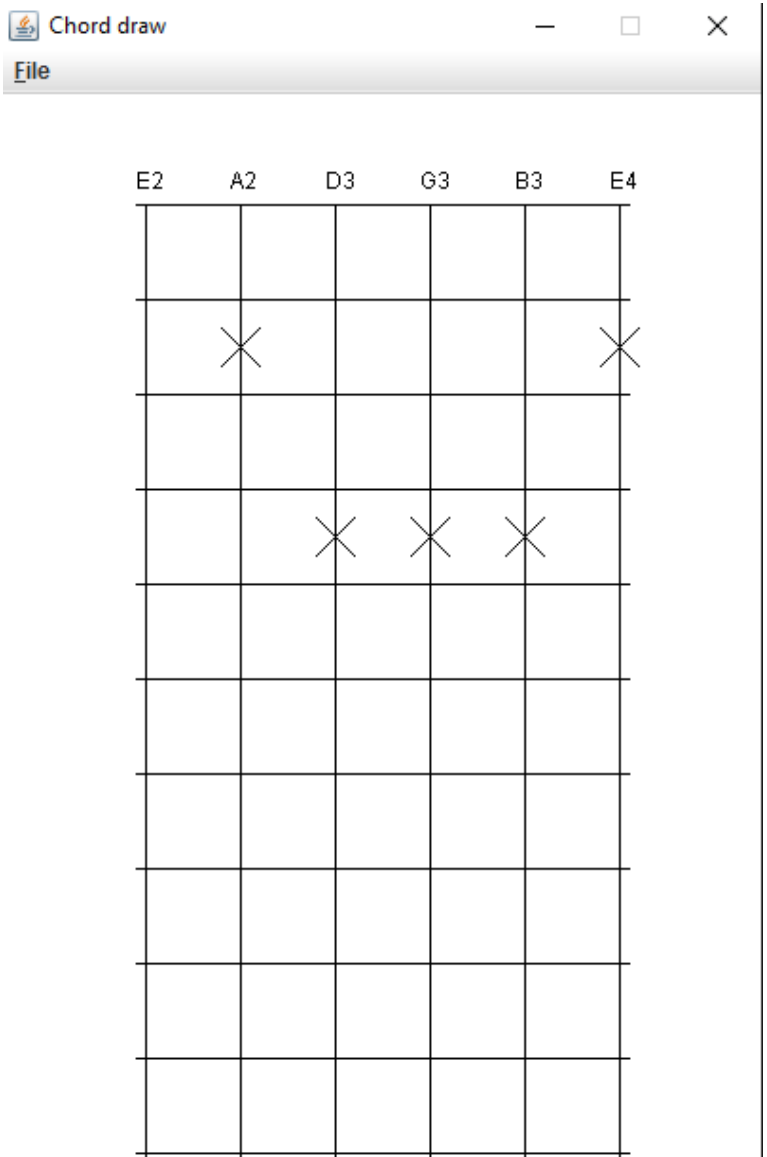
Toner, oktaver och ackord

- Det finns 12 toner som har speciella namn:
  C, C#, D, D#, etc. (uttalas: c, ciss, d, diss, etc.)
- Jämför vita och svarta tangenter på ett piano:
  avståndet mellan varje tangent är ett s.k. *halvt tonsteg*.
- Toner återkommer i oktaver, modulo 12.
- Tonen som representeras av strängen `"D2"` är tonen D i andra oktaven.
- Tonen `"D2"` motsvarar heltalet 26 på labben.
- Ett ackord består av flera toner.

```
1  scala> val notes = Vector("C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B")
2
3  scala> notes.size
4  res0: Int = 12
5
6  scala> notes(26 % 12)
7  res1: String = D
```

Toner på ett stränginstrument

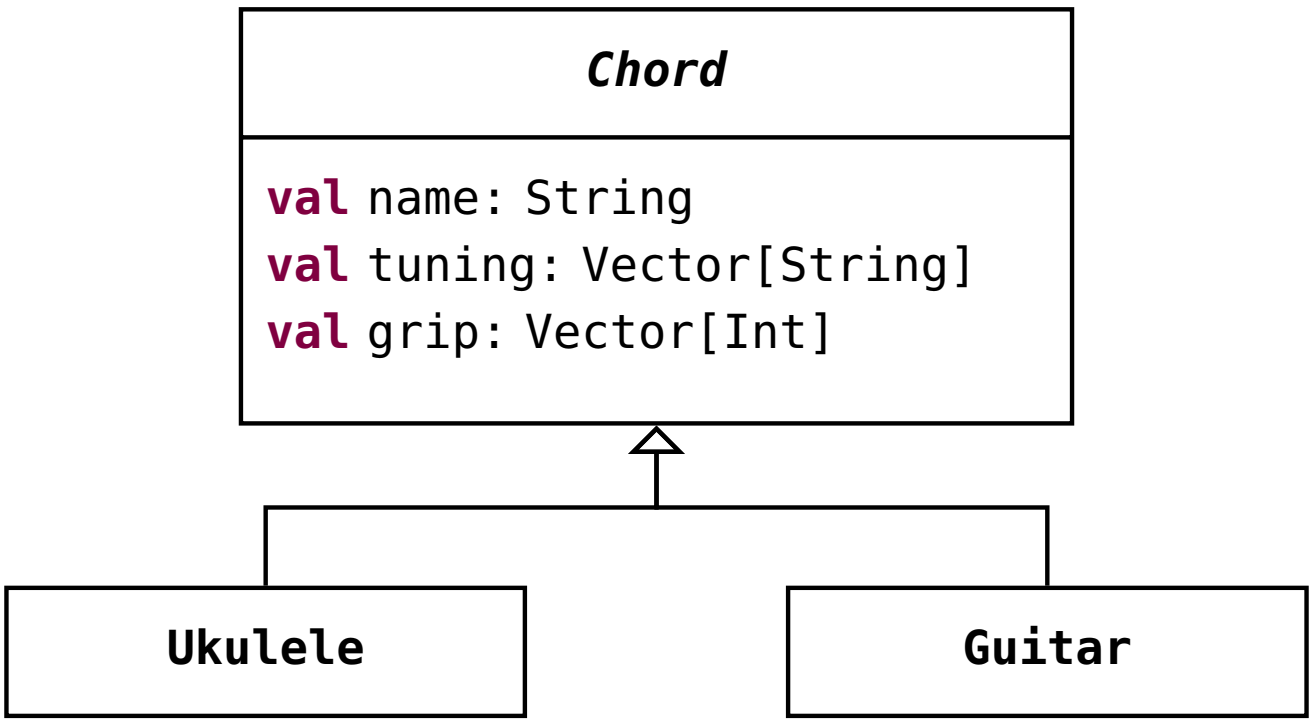
- Gitarr och ukulele har 6 resp. 4 strängar och en greppbräda med s.k. band.
- Om man trycker ned ett finger på första bandet höjs tonen ett halvt tonsteg.
- Exempel: om en sträng är stämd i D3 blir tonen om man trycker ned fingret på fjärde bandet F#3.
- www.gitarr.org
- www.stefansukulele.com



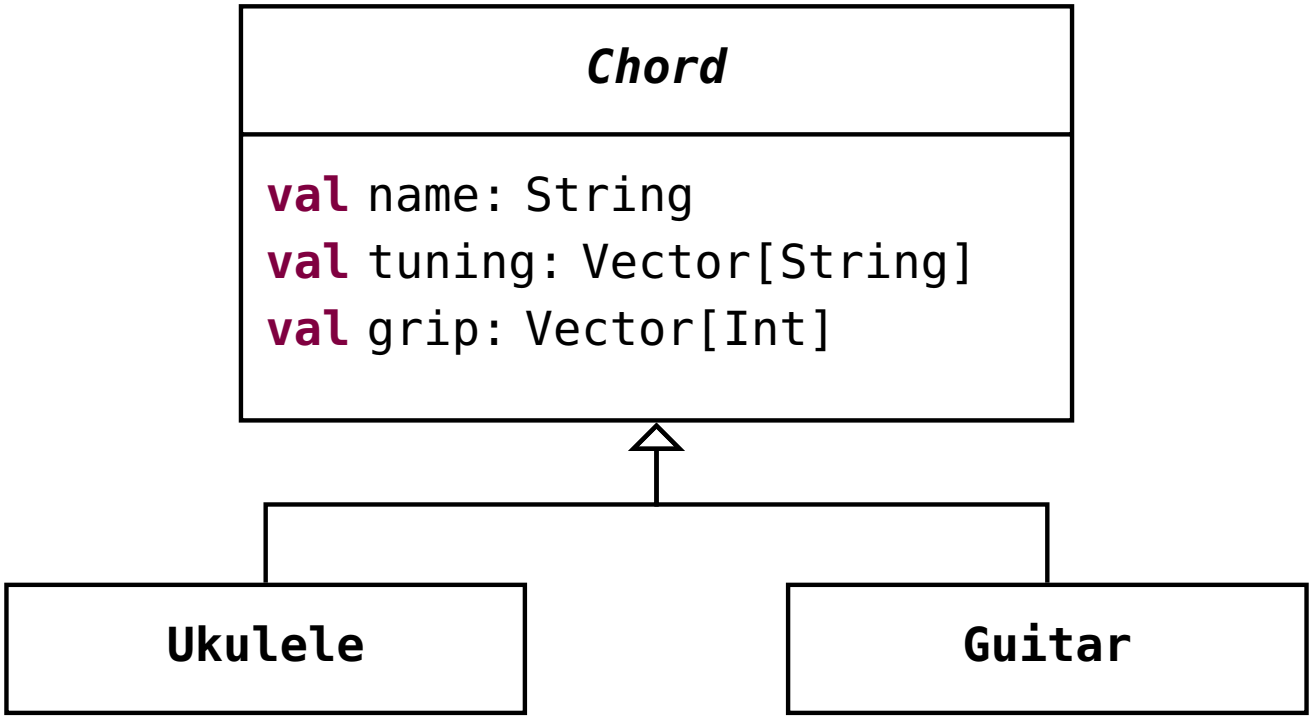
Modell av gitarr och ukulele

```
object model {
  type Tuning = Vector[String]
  type Grip = Vector[Int]

  trait Chord {
    def name: String
    def tuning: Tuning
    def grip: Grip
  }

  case class Guitar(name: String, grip: Grip) extends Chord {
    val tuning = Vector("E2", "A2", "D3", "G3", "B3", "E4")
  }

  case class Ukulele(name: String, grip: Grip) extends Chord {
    val tuning = Vector("G4", "C4", "E4", "A4")
  }

}
```

En gemensam bastyp för olika ackord



En gemensam bastyp för olika ackord




```
scala> import model._

scala> val uc = Ukulele("C", Vector(0, 0, 0, 3))
uc: model.Ukulele = Ukulele(C,Vector(0, 0, 0, 3))

scala> val ge = Guitar("E", Vector(0, 2, 2, 1, 0, 0))
ge: model.Guitar = Guitar(E,Vector(0, 2, 2, 1, 0, 0))
```

Grupparbete

- Förslag på arbetssätt:
  - Träffas nu på rasten och boka nästa gruppmöte
  - Förberedelser inför första gruppmötet: individuella studier av labbinstruktioner och koden som är given i workspace
  - Träffas gärna i ett studierum med whiteboard
  - På mötet: gå igenom uppgift och given kod så att alla fattar vad det går ut på; bestäm omfattning och ansvarsuppdelning
  - När ni träffas, skissa upp din kod på whiteboard och få feedback
  - På varje gruppmöte, bestäm tid för nästa möte och vad var och en ska försöka hinna tills dess
- Ni får **lov att ändra på omfattningen** efter antalet gruppmedlemmar, ambition och förmåga: diskutera detta med handledare på resurstid
- **Minimikrav**: att med textkommando kunna skapa/spara/ladda gitarr- och ukulele-ackord och att ni tränar på matchning

Matchning

Vad är matchning?

Matchning gör man då man vill jämföra ett värde mot andra värden och hitta överensstämmelse (eng. *match*).

Vad är matchning?

Matchning gör man då man vill jämföra ett värde mot andra värden och hitta överensstämmelse (eng. *match*).

Detta kan man t.ex. göra med nästlade if-else-satser:

```
val g = scala.io.StdIn.readLine("grönsak:")

if (g == "gurka") println("gott!")
else if (g == "tomat") println("gott!")
else if (g == "broccoli") println("ganska gott...")
else println("inte gott :(")
```

Javas switch-sats

De flesta C-liknande språk (men inte Scala) har en **`switch`**-sats som man kan använda istället för (vissa) nästlade if-else-satser:

```
import java.util.Scanner;

public class SwitchNoBreak {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
        case "tomat":
            System.out.println("gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println(" mindre gott...");
            break;
        }
    }
}
```

Funkar bara för primitiva typer och några till (t.ex. String).

Javas switch-sats utan break

Saknad **break**-sats ”faller igenom” till efterföljande gren:

```
import java.util.Scanner;

public class SwitchNoBreak {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
        case "tomat":
            System.out.println("gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println(" mindre gott...");
            break;
        }
    }
}
```

En glömd **break** kan ge svårhittad bugg...

Javas switch-sats med glömd break

```
import java.util.Scanner;

public class SwitchForgotBreak {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
            System.out.println("gott!");
        case "tomat":
            System.out.println("gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println("mindre gott...");
            break;
        }
    }
}
```

Javas switch-sats med glömd break

```java
import java.util.Scanner;

public class SwitchForgotBreak {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Skriv grönsak:");
        String g = scan.next();
        switch (g) {
        case "gurka":
            System.out.println("gott!");
        case "tomat":
            System.out.println("gott!");
            break;
        case "broccoli":
            System.out.println("ganska gott...");
            break;
        default:
            System.out.println("mindre gott...");
            break;
        }
    }
}
```

```
$ java SwitchForgotBreak
Skriv grönsak:
gurka
gott!
gott!
```

Scalas `match`-uttryck

Scala har ingen `switch`-sats men erbjuder i stället ett **`match`-uttryck** som är mångsidig och kraftfull och ger ett värde.

```
val g = scala.io.StdIn.readLine("grönsak:")
val msg = g match {
  case "gurka" => "gott!"
  case "tomat" => "jättegott!"
  case "broccoli" => "ganska gott..."
  case _ => "mindre gott..."
}
```

Och den ”faller inte igenom” som Javas `switch`-sats!
Default-grenen skrivs så här: **`case _ =>`**

Matchning med gard

Matchning efter typ

Stora/små begynnelsebokstäver vid matchning

Mönster

Mönstermatcning ”plockar isär” en datastruktur

Mönstermatchning och case-klasser

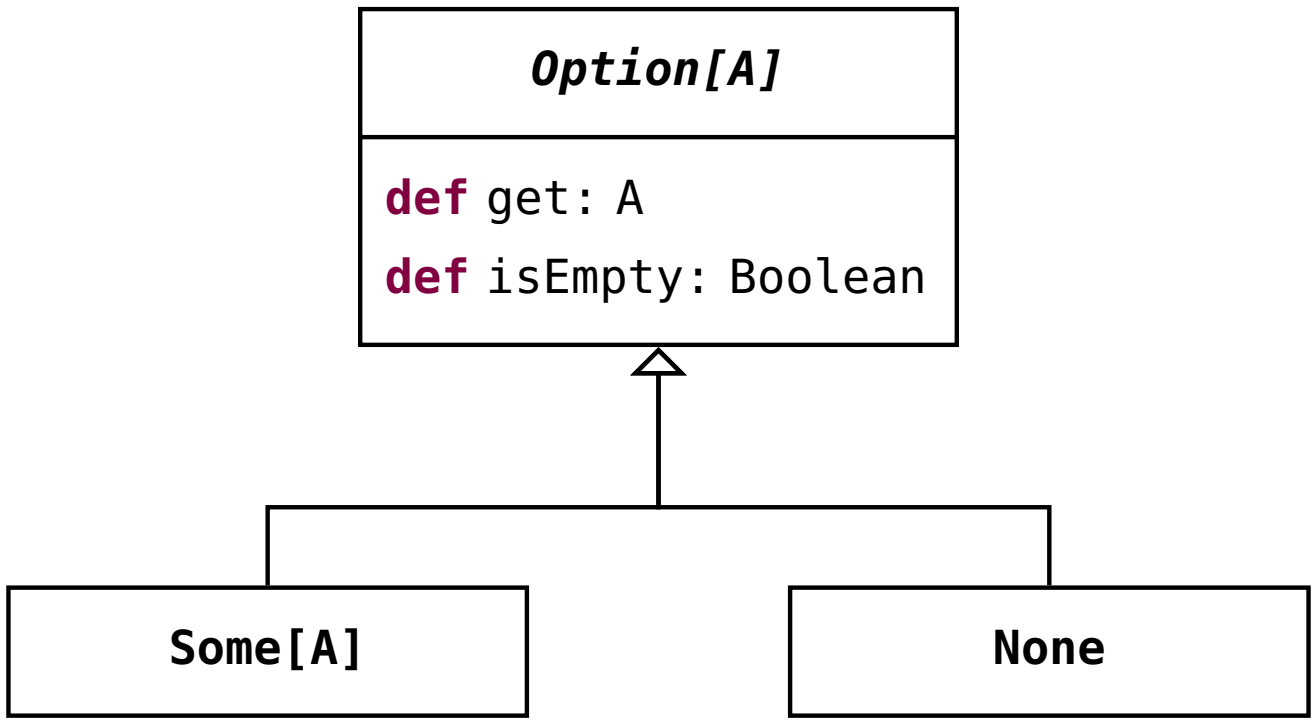
Mönstermatchning och metoden `unapply`

Mönstermatchning och förseglade typer

Option

Option för hantering av ev. saknade värden

En gemensam bastyp för ett värde som kanske saknas



Implementera `equals` med `match`

Undantag

Vad är ett undantag (eng. *exception*)?

Undantag representerar ett fel eller ett onormalt tillstånd som upptäcks under exekvering och som behöver hanteras på särskilt sätt vid sidan av det normala exekveringsflödet.

sv.wikipedia.org/wiki/Undantagshantering

Exempel på undantag:

Vad är ett undantag (eng. *exception*)?

Undantag representerar ett fel eller ett onormalt tillstånd som upptäcks under exekvering och som behöver hanteras på särskilt sätt vid sidan av det normala exekveringsflödet.

sv.wikipedia.org/wiki/Undantagshantering

Exempel på undantag:

- Indexering utanför vektorns indexgränser.
- Läsning bortom filens slut.
- Försök att öppna en fil som inte finns.
- Minnet är slut.
- Division med noll.
- `"hej".toInt` resulterar i `java.lang.NumberFormatException`

Fånga undantag med `try-catch`-uttryck

scala.util.Try

En gemensam bastyp för något som kan misslyckas