Seis

- Baastest:
  - 25 sai läbi
  - 8 ei saanud
  - 6 pole teinud
- Kontrolltöö:
  - 9 sai 20..18 punkti
  - 9 sai 17..10 punkti
  - 5 sai 9..1 punkti
  - 2 said 0 punkti
  - 14 ei teinud kt-d

Kava

1. Näide Scala programmist. (1. loeng)
2. Kiire ülevaade peamsitest omadustest.
3. Vaatame mõnda keele aspekti detailselt.

Scala

```
object MinuEsimeneScalaProgramm {
  def main(args: Array[String]): Unit = {
    println("Tere maailm!")
  }
}
```

Scalas ...

1. meetodid on seotud *objektidega*
2. peameetodi nimi on `main`
3. peameetodi ainsa argumendi tüübiks on `Array[String]`
4. peameetodi tagastustüübiks on `Unit`

Näide: *Evil Hangman*

```
def main(args: Array[String]): Unit = {
  // loeme sõnastiku
  val wordSet: Set[String] =
    io.Source.fromFile("s6nad.txt").getLines.toSet

  // äraarvatava sõna pikkus
  print("Mitu tähte: ")
  val wordLength: Int = StdIn.readInt()

  // jätame alles õige pikkusega sõnad
  val filteredWords = wordSet.filter(_.length == wordLength)

  if (filteredWords.nonEmpty)
    play(new GameState(filteredWords))
  else
    println("Kahjuks sellise pikkusega sõnu pole.")
}
```

Abifunktsioonid

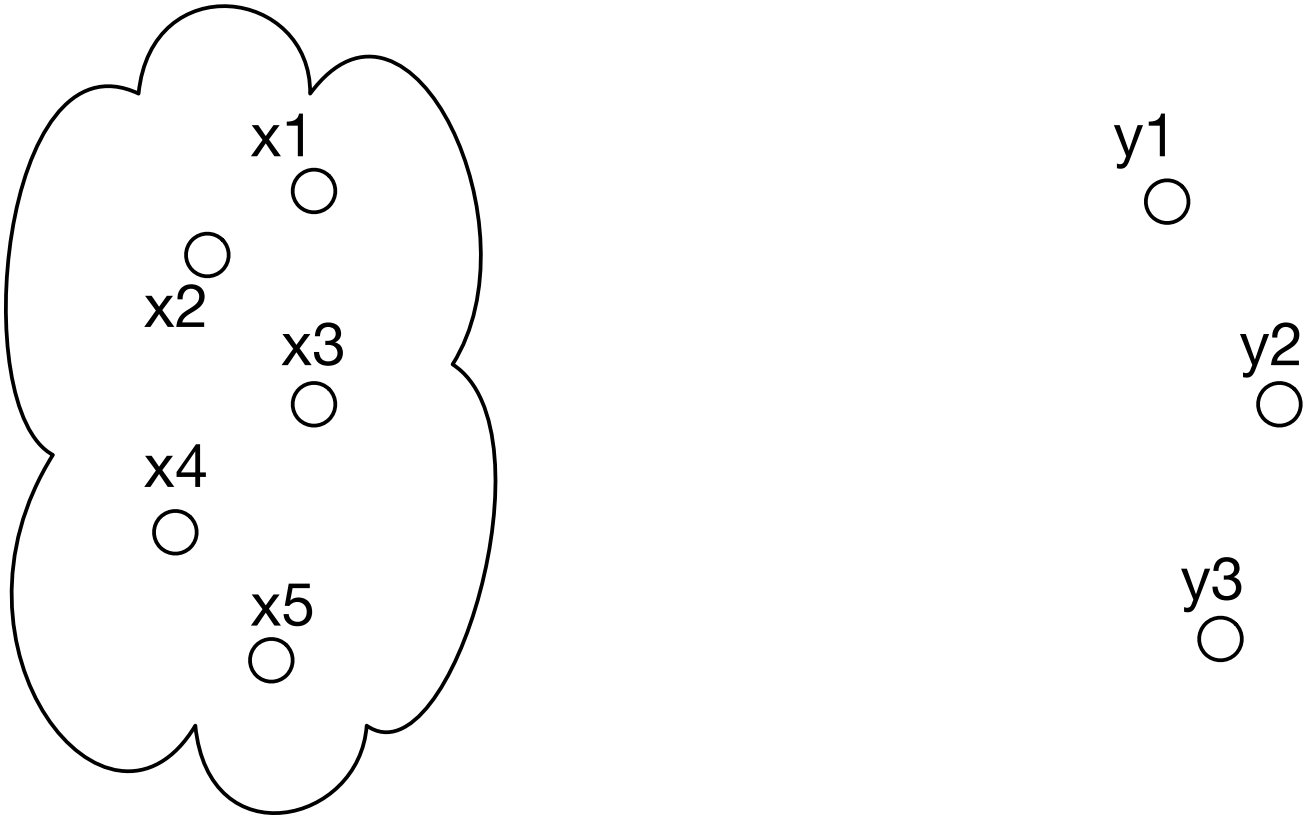
```
def readChar: Char = {
    print("Paku täht: ")
    try {
      StdIn.readChar()
    } catch {
      case _: Throwable => println("Viga!"); readChar
    }
  }

sealed trait Status
case object Correct extends Status
case object Wrong extends Status
```

Mängu loogika

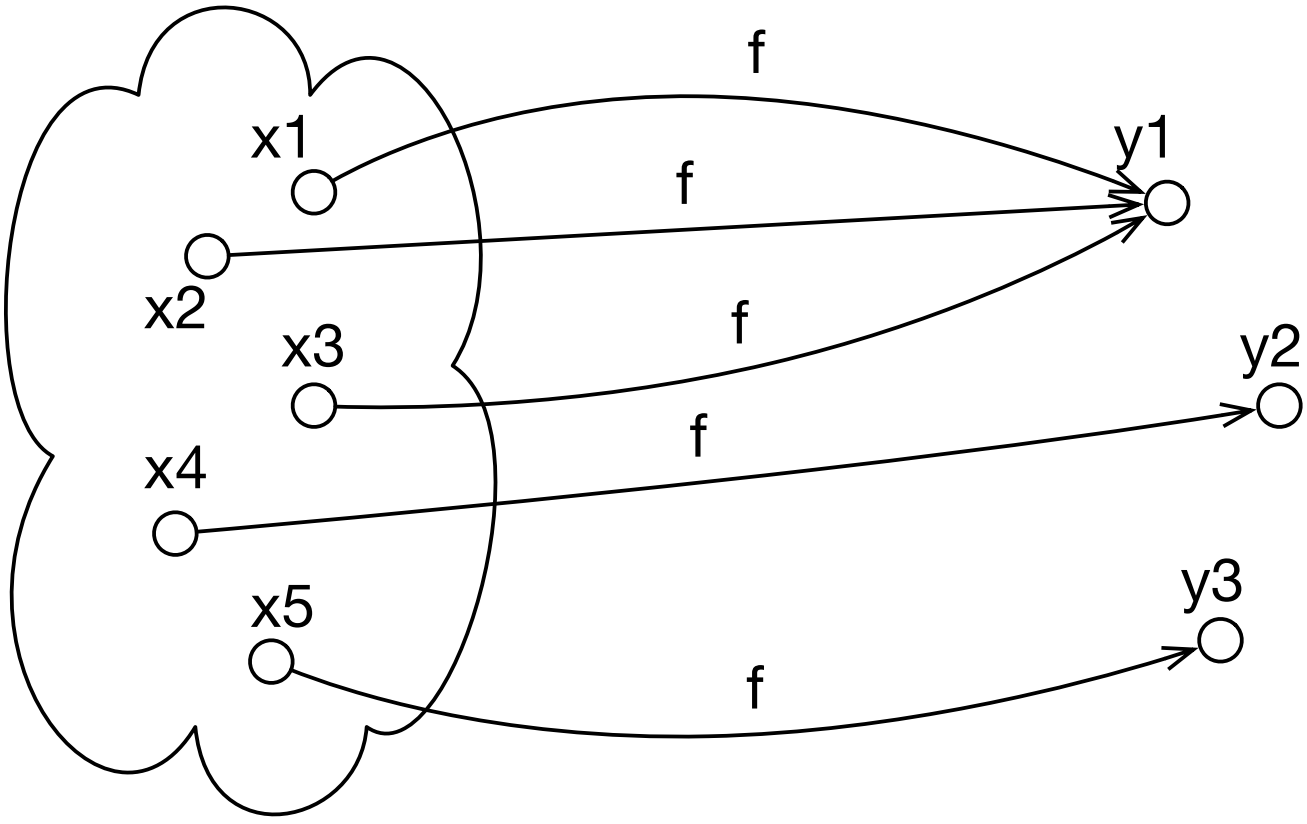
```
class GameState(
    var candidateWords: Set[String],
    var movesLeft: Int   = 21,
    var guessed: Set[Char] = SortedSet.empty)
{

  def blankOtherChars(word: String, charSet: Set[Char]): String =
    word.map(c => if (charSet(c.toLower)) c else '_')

  // Nüüd asendame selles sõnes kõik mitte-pakutud tähed alakriipsuga.
  def blankedWord: String =
    blankOtherChars(candidateWords.head, guessed)

  // Kui meie sõnes pole enam ühtegi alakriipsuga peidetud täht jäänud,
  // siis on sõne ära arvatud ja mäng on läbi.
  def allGuessed: Boolean = !blankedWord.contains('_')
```

```
// Mängu juhtimise meetod, mis teeb ühe käigu ära.
def move(guess: Char): Status = {

  // Nüüd siis peamine töö. Tahame pakkumise järgi liigitada sõnad.
  val groups: Map[String, Set[String]] =
    candidateWords.groupBy(blankOtherChars(_, Set(guess)))

  // Kui on liigitatud, siis tuleks lihtsalt valida need sõnad,
  // kus on kõige rohkem kandidaatsõnu.
  val (pattern, newWords): (String, Set[String]) =
    groups.maxBy{ case (pat, set) => set.size }

  candidateWords = newWords
  movesLeft     -= 1
  guessed       += guess

  if (pattern.contains(guess))
    Correct
  else
    Wrong
}
```

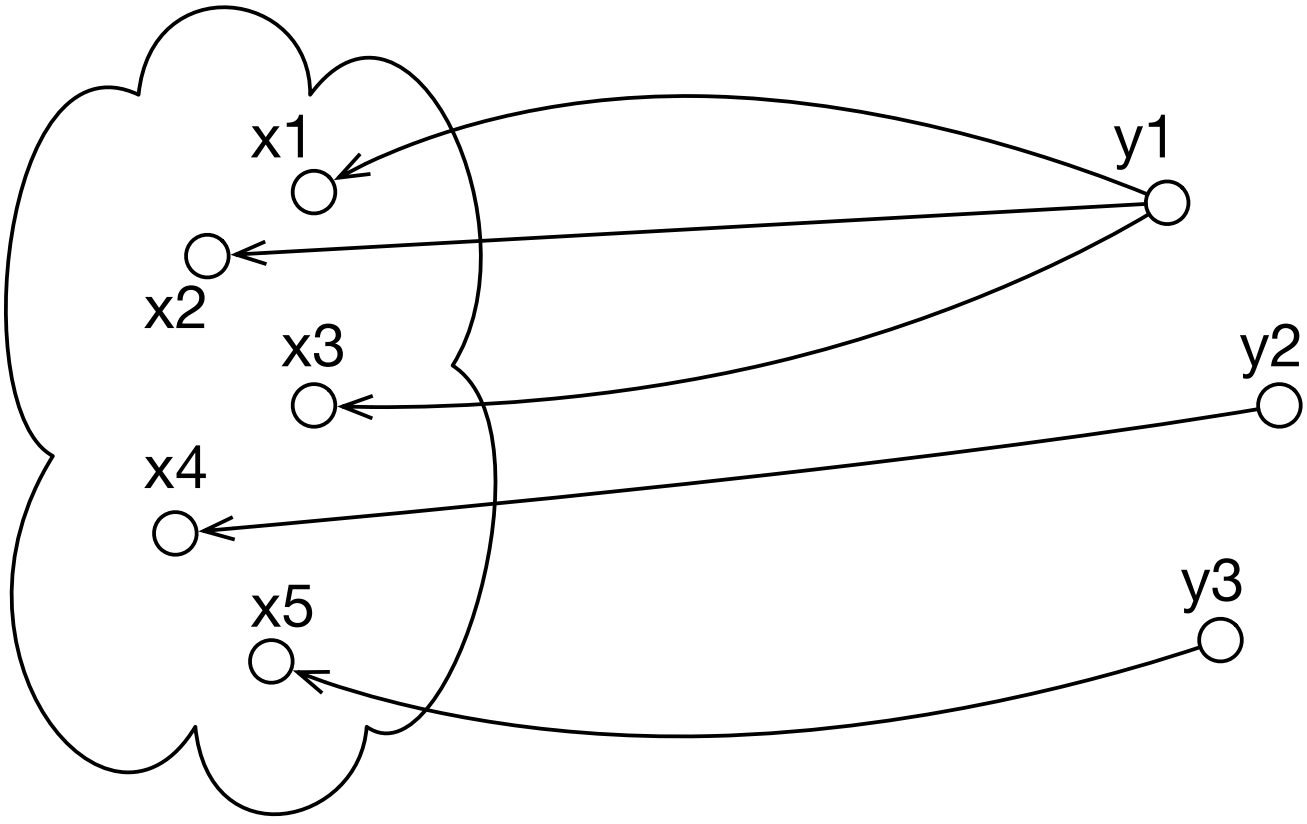
groupBy (1)



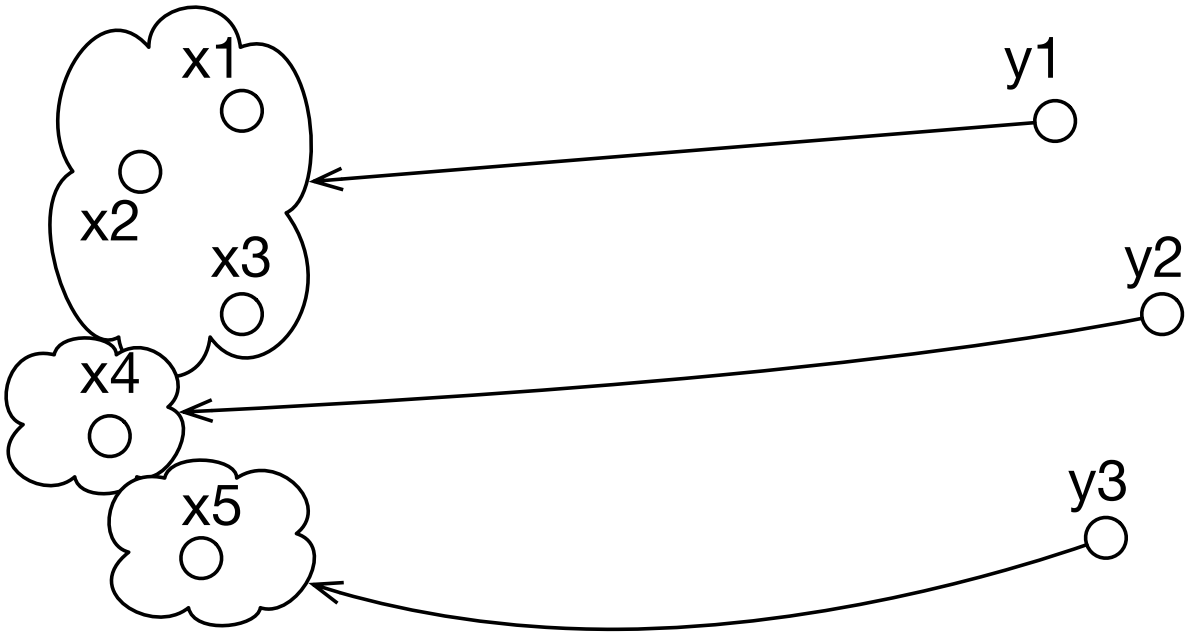
groupBy (2)



groupBy (3)



groupBy (4)



Suhtlus kasutajaga

```
def play(gs: GameState): Unit = {
  if (gs.movesLeft == 0) {
    println("Kaotasid! 6ige vastus oli: " + gs.candidateWords.head)
    return
  }

  printf("Pakutud tähed: %s\nPakkumisi jäänud: %d\n\n%s\n",
    gs.guessed.mkString(", "), gs.movesLeft, gs.blankedWord.mkString(" "))

  if (gs.allGuessed) {
    println("Palju õnne!")
    return
  }

  gs.move(readChar) match {
    case Correct => println("V2ga tubli!")
    case Wrong => println("Vale t2ht!")
  }

  play(gs)
}
```

Scala ülevaade

1. Meetodid, muutujad ja väärtused. Süntaks
2. Lihtsad tüübid ja väärtused.
3. OOP, `case`-klassid ja mustrisobitus.
4. Puhta Scala väärtustamine.
5. Keerulisemad tüübid.
6. Nähtavus, implitsiitsus.

Meetodid

deklaratsioon, näiteks: `def +(x:Int): Unit = { println(x); x+1 }`

meetodi kutse:

$\texttt{nimi}_{\texttt{objekt}}\texttt{.nimi}_{\texttt{meetod}}\texttt{(avaldis}_1\texttt{, ..., avaldis}_n\texttt{)}$

näiteks: `o.+(5)`

infixne kutse (kui on üks argument):

$\texttt{nimi}_{\texttt{objekt}}\ \texttt{nimi}_{\texttt{meetod}}\ \texttt{avaldis}$

näiteks: `o + 5`

Väärtused ja muutujad

Objektide (ja meetodide) sisse saab defineerida väärtuseid, muutujaid ja meetode.

```
object ScalaProgramm {
  var muutuja: Int = 10
  val v22rtus: Int = 100
  def main(args: Array[String]): Unit = {
    muutuja = 20
    val summa = muutuja + v22rtus
    println(summa) // prindib 120
  }
}
```

Sulud ja meetodite kutsed

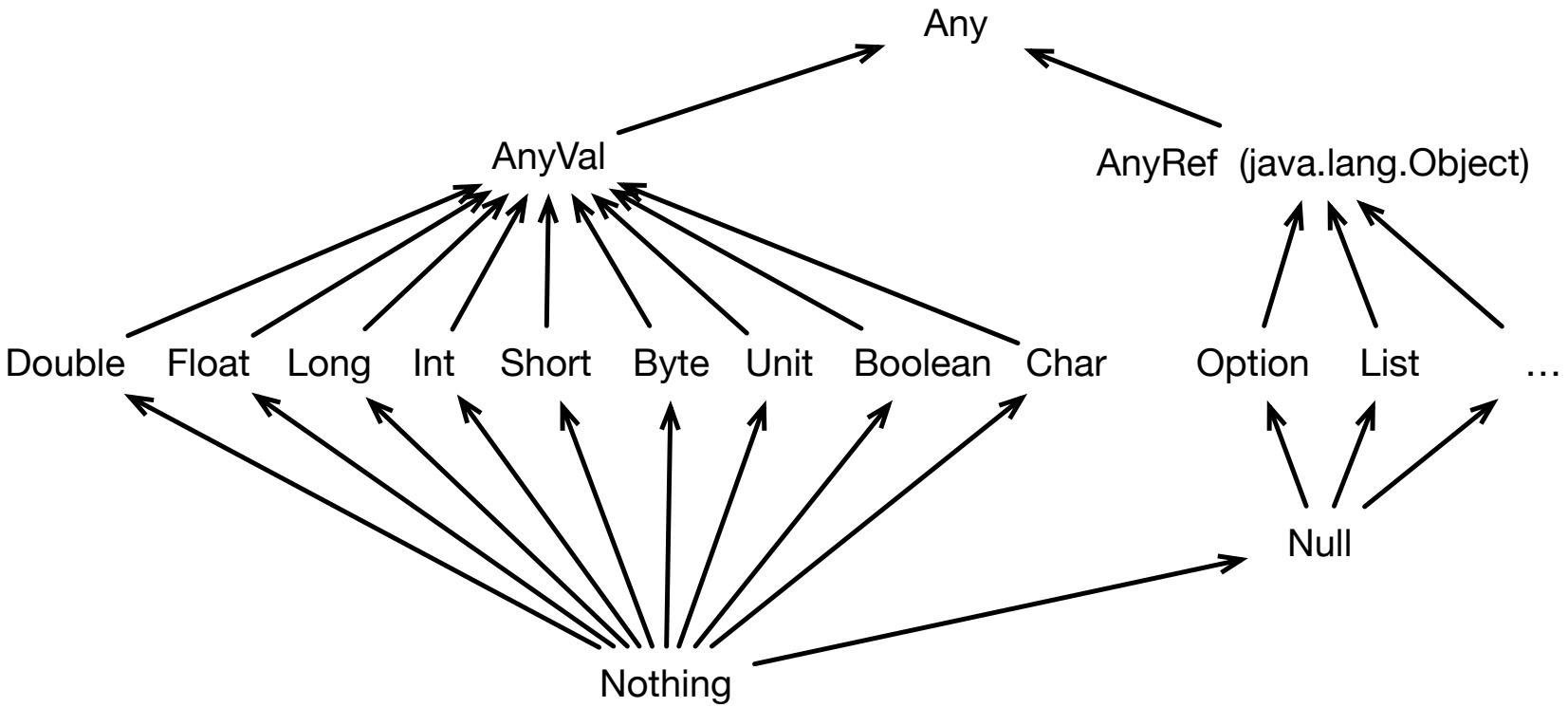
- Kui meetodil puuduvad argumendid, jätame enamasti sulud ära.
- Sulge on soovitatav kasutada, kui meetod muudab objekti.

```
object Sulud {
  def x(): Int = 5
  def y  : Int = 5
  val z  : Int = 5

  def main(args: Array[String]): Unit = {
    val summa = x + x() + y + z
//    summa += y() + z()  <- ei tööta
  }
}
```

Süntaks

- Tingimusavaldised:
  ```
  if(e) s1 else s2
  ```
- tsükklid (lihtsustatult):
  ```
  for (muutuja <- algus to l6pp) { ... }
  for (muutuja <- algus until l6pp) { ... }
  for (muutuja <- kollektsioon) { ... }
  ```
- funktsiooni tüüp:
  ```
  tyyp =>tyyp2
  ```
- lambdad:
  ```
  (muutuja: tyyp, muutja2: tyyp2) => keha
  ```
- blokid (eraldi ridadel võib semikoolonid ära jätta):
  $\{e_1; e_2; \ldots e_n;\}$

Alamtüüpimine



Avaldised

Väärtused (`AnyVal`):

- `() : Unit`
- **`true`** `: Boolean`, **`false`** `: Boolean`
- `'a' : Char`, `'b' : Char`, `...`
- `1 : Byte`, `2 : Short`, `3 : Int`, `3 : Long`
- `1f : Float`, `2 : Double`

- Väärtustel saab samuti meetodeid välja kutsuda. N: `1.+(2)`
- Aritmeetikafunktsioonid. N: `math.max(math.Pi, x*4)`

Klassid ja Objektid

```
class MinuKlass {
  val viis: Int = 5
  def lisaviis(x: Int): Int = x+viis
}
```

- Klassid on objektide tüübid.

```
var o : MinuKlass = null
```

- Objekte saab luua võtmesõnaga **new**.

```
o = new MinuKlass
```

Konstruktorid

- Peamine konstruktor — argumendid klassi nime järel ja keha meetodite ja väljadega segamini.

```
class K(nimi: String, synniaasta: Int) {
  println("Loodi klass K("+ nimi +"," + synniaasta + ")")
  def umbkaudneVanus(): Int = LocalDate.now.getYear - synniaasta
}
```

- Abikonstruktor.

```
class K(nimi: String, synniaasta: Int) {
  def this() = this("nipitiri",LocalDate.now.getYear)
  println("Loodi klass K("+ nimi +"," + synniaasta + ")")
  def umbkaudneVanus(): Int = LocalDate.now.getYear - synniaasta
}
```

- Konstruktori väljakutse:

```
  val x = new K("Donald", 1934)
  val y = new K // = new K("nipitiri", 2018)
```

Case klassid ja objektid

```
trait List[+A] // Scala listide lihtsustus
case object Nil                          extends List[Nothing]
case class Cons[A](x:A, xs: List[A]) extends List[A]

val list = Cons(1, Cons(2, Cons(3, Nil)))
```

- Nagu algebralised andmestruktuurid Haskellis.
- Ei kasutata võtmesõna **new**.
- Defineerib võrduse, mis sõltub konstruktorite argumentidest:

```
Cons(3, Nil) == Cons(3, Nil)
Cons(3, Nil) != Nil
```

- Defineerib `toString` meetodi:

```
list.toString = "Cons(1,␣Cons(2,␣Cons(3,␣Nil)))"
```

Traitid ja pärimine

- Liideste asemel on Scalas **`trait`**-id:

```
trait T {
  var q: Char
  def f(x: Int): Double
  def g(x: Int): Double = f(10) / 2
}
```

- Abstraktsed klassid:

```
abstract class Q {
  def h(x: Int): Boolean
}
```

- Klass laiendab kuni ühte klassi ja suvaline arv *trait*-e.

```
class W extends Q with T {
  override var q: Char = 'a'
  override def f(x: Int): Double = x.toDouble % 10
  override def g(x: Int): Double = f(10) / 3
  override def h(x: Int): Boolean = f(x) > 100
}
```