Scala 2.7 (ja varem)

- Soov on operatsioonid tõsta klassihierarhias kõrgemale:

```
trait scala.Iterable[A] {
  // üks abstraktne meetod
  def elements: Iterator[A]

  // palju konkreetseid meetode
  def isEmpty: Boolean = ...
  def map[B](f: A => B): Iterable[B] = ...
  def dropWhile(p: A => Boolean): Iterable[A] = ...
}
```

- Täpsed tüübid lähevad kaduma:
  - `List(...).map(...): Iterable`
  - `HashSet(...).dropWhile(...): Iterable`
  - `TreeSet(...).map(...): Iterable`

Scala 2.8 kuni 2.12

- Kasutame implitsiitseid argumente:

```
trait MyCanBuildFrom[-From, -Elem, +To] {
  def apply(): MyBuilder[Elem, To]
}
trait MyBuilder[-Elem, +To] {
  def +=(elem: Elem): MyBuilder.this.type
  def clear(): Unit
  def result(): To
}


trait MyIterable[+A] {
  def map[B, That](f: A => B)
          (implicit bf: MyCanBuildFrom[MyIterable[A], B, That]): That
    val b = bf()
    for (x <- this)
      b += f(x)
    b.result
  }
  ...
}
```

Implitsiitsed parameetrid

Kuidas saab kirjutada:

```
val x1 = List(1,2,3).max
val x2 = List("aabits", "zorro").max
val x3 = List(false, true).max
```

Kui listidel (`List[+A]`) on meetod:

```
def max[B >: A](implicit cmp: Ordering[B]): A = ...
```

Järjestus on enam-vähem selline:

```
trait Ordering[A] { def compare(x: T, y: T): Int }
```

Kuskil on defineeritud:

```
implicit object A extends Ordering[Int] { ... }
implicit object B extends Ordering[Boolean] { ... }
implicit object C extends Ordering[String] { ... } //leks. järjestus
```

Scala 2.8 kuni 2.12

- Kasutame implitsiitseid argumente:

```
// Kuskil defineeritud:
implicit def cbfm[C,A,B]: MyCanBuildFrom[C, (A,B), MyMap[A,B]] = ...
implicit def cbfl[U]: MyCanBuildFrom[MyList[_], U, MyList[U]] = ...
implicit def cbfs: MyCanBuildFrom[MyList[_], Char, String] = ...

// Meie koodis:
val x : MyList[Int] = MyList(1,2,3,4)
val z : MyList[Char] = x.map(_.toChar)
val z : String = x.map(x => (x + 'a'.toInt).toChar)
val q : MyMap[Int, Char] = x.map(x => x -> (x+'a'.toInt).toChar)
```

- Ülipaindlik
- Veateated kohutavad, `map` tüüp kohutav
- Väga keeruline ja suur süsteem!

Kollektsioonid Scalas oli suur süsteem!

```
trait GenTraversableOnce[+A] extends Any

trait TraversableOnce[+A] extends Any with GenTraversableOnce[A]

trait GenIterable[+A] extends GenIterableLike[A, GenIterable[A]]
  with GenTraversable[A] with GenericTraversableTemplate[A, GenIterable]

trait GenericTraversableTemplate[+A, +CC[X] <: GenTraversable[X]]
  extends HasNewBuilder[A, CC[A] @uncheckedVariance]

trait GenIterableLike[+A, +Repr] extends Any
  with GenTraversableLike[A, Repr]

trait GenTraversableLike[+A, +Repr] extends Any
  with GenTraversableOnce[A] with Parallelizable[A, ParIterable[A]]

trait TraversableOnce[+A] extends Any with GenTraversableOnce[A]
```

Väga keeruline ja suur süsteem!

```
trait TraversableLike[+A, +Repr] extends Any with HasNewBuilder[A, Repr]
  with FilterMonadic[A, Repr] with TraversableOnce[A]
  with GenTraversableLike[A, Repr] with Parallelizable[A, ParIterable[A]]

trait Traversable[+A] extends TraversableLike[A, Traversable[A]]
  with GenTraversable[A] with TraversableOnce[A]
  with GenericTraversableTemplate[A, Traversable]

trait IterableLike[+A, +Repr] extends Any with Equals
  with TraversableLike[A, Repr] with GenIterableLike[A, Repr]

trait Iterable[+A] extends Traversable[A] with GenIterable[A]
  with GenericTraversableTemplate[A, Iterable]
  with IterableLike[A, Iterable[A]]
```

Uued Scala kollektsioonid (2.13)

- Aluseks võetud iteraatorid:

```
trait IterableOnce[+A] extends Any {
  def iterator(): Iterator[A]
}

trait Iterator[+A] extends IterableOnce[A] {
  // abstraktsed meetodid
  def hasNext: Boolean
  def next(): A

  def iterator() = this

  // konkreetsed meetodid
  def dropWhile(p: A => Boolean): Iterator[A] = ...
  ...
}
```

Uued Scala kollektsioonid (2.13)

- Kasutab kõrgemat järku tüübimuutujaid:

```
trait IterableOps[+A, +CC[_], +C] extends Any {
  protected[this] def coll: Iterable[A]

  def iterableFactory: IterableFactory[CC]
  protected[this] def fromSpecificIterable(coll: Iterable[A]): C
  protected[this] def newSpecificBuilder(): Builder[A, C]

  // konkreetsed meetodid
  def size: Int =
    if (knownSize >= 0) knownSize else coll.iterator().length

  def dropWhile(p: A => Boolean): C =
    fromSpecificIterable(View.DropWhile(coll, p))

  def map[B](f: A => B): CC[B] =
    iterableFactory.fromIterable(View.Map(coll, f))
}
```

Uued Scala kollektsioonid (2.13)

- Lihtsustatud näide:

```
trait MyIterableOps[+A, +CC[_], +C] extends MyIterable[A] {
  def ++[B >: A](suffix: MyIterable[B]): CC[B]
  def map[B](f: A => B): CC[B]
  def filter(p: A => Boolean): C
}
sealed abstract class MyList[+T]
  extends MyIterableOps[T, MyList, MyList[T]]
{
  // def ++[B >: T](suffix: MyIterable[B]): MyList[B]
  // def map[B](f: T => B): MyList[B]
  // def filter(p: T => Boolean): MyList[T]
  ...
}
```

Uued Scala kollektsioonid (2.13)

- Kasutab ära ülelaadimist

```
trait SortedSet[A] extends Set[A]
  with SortedSetOps[A, SortedSet, SortedSet[A]]

trait SortedSetOps[A, +CC[X] <: SortedSet[X], +C <: SortedSet[A]]
  extends SetOps[A, Set, C] with SortedOps[A, C]
{
  ...
}
```

- `SortedSet[A]`-l on kaks `map`-i

```
def map[B           ](f: A => B): Set[B]
def map[B : Ordering](f: A => B): SortedSet[B]
```

Uued Scala kollektsioonid (2.13)

- `BitSet`-id:

```
trait BitSet extends SortedSet[Int] with BitSetOps[BitSet]

trait BitSetOps[+C <: BitSet with BitSetOps[C]]
  extends SortedSetOps[Int, SortedSet, C]
{
  def map(f: Int => Int): C = ...
  ...
}
```

- `BitSet`-il on need `map`-id

```
  def map[B](f: Int => B): Set[B]
  def map[B : Ordering](f: Int => B): SortedSet[B]
  def map(f: Int => Int): BitSet
```

Eelnev definitsioon pole Scala 2.11-s mugavalt kasutatav:

```
scala> BitSet(1,3,5).map(_ + 1)
<console>:13: error: missing parameter type for
      expanded function ((x) => x.plus(1))
```

Tüübituletus versioonis 2.11:

1. Püüame leida meetodi tüübi kuju järgi: jääb mitu alternatiivi
2. Üritame leida argumendi tüüpi:
   - Lambda _ + 1 on ilma oodatava tüübita.
   - Ei oska tüüpi leida, kuna liitmine võib olla defineeritud mitmel tüübil.

Keeleuuendused >=2.12

1. Püüame leida meetodi tüübi kuju järgi: jäävad kaks alternatiivi

```
(Function1[Int, B]): SortedSet[B]
(Function1[Int, Int]): BitSet
```

2. Unifitseerime alternatiivide tüübid:

```
(Function1[Int, ?]): ?
```

3. Leiame argumendi tüübi, kasutades meetodite unifitseeritud tüüpi:
   - `_ + 1` tüübitakse oodatava tüübiga `Function1[Int, ?]`
   - See õnnestub tüübiga `Function1[Int, Int]`

4. Tüübitakse alternatiivid, kasutades argumendi tüüpi. Sobib `(Function1[Int, Int]): BitSet`

Mitme operatsiooni tegemine

- Tehes

```
List(1,2,3,4).filter(f).map(g)
```

tekitatakse iga operatsiooni järel uus list.

- Vahetulemust ei genereeri:

```
for (x <- List(1,2,3,4) if f(x)) yield g(x)
```

ehk

```
List(1,2,3,4).withFilter(f).map(g)
```

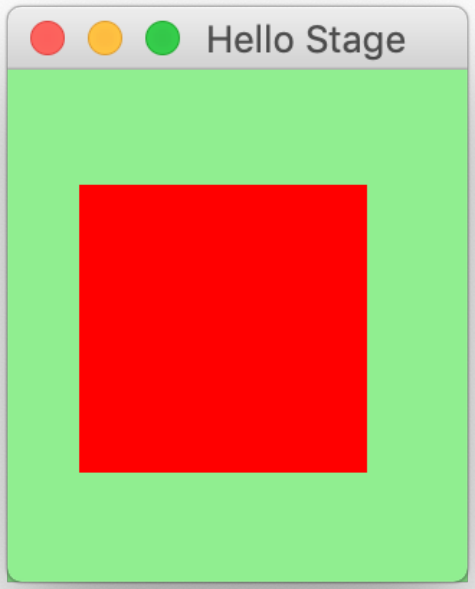
WithFilter

```
abstract class WithFilter[+A, +CC[_]] {
  def map[B](f: A => B): CC[B]
  def flatMap[B](f: A => MyIterable[B]): CC[B]
  def foreach[U](f: A => U): Unit
  def withFilter(q: A => Boolean): WithFilter[A, CC]
}

trait IterableOps[+A, +CC[_], +C] extends Any {
  ...
  def withFilter(q: A => Boolean): WithFilter[A, CC]
}
```

- Sama eesmärgiga on ka vaated (`View`).
  - parem implementatsioon 2.13-s

ScalaFX

```
object HelloStageDemo extends JFXApp {
  stage = new PrimaryStage {
    title = "Hello Stage"
    width = 160
    height = 200
    scene = new Scene {
      fill = LightGreen
      content = new Rectangle {
        x = 25
        y = 40
        width = 100
        height = 100
        fill = Red
      }
    }
  }
}
```



- `build.sbt`-sse lisada:

```
libraryDependencies += "org.scalafx" %% "scalafx" % "12.0.2-R18"

// Determine OS version of JavaFX binaries
lazy val osName = System.getProperty("os.name") match {
  case n if n.startsWith("Linux") => "linux"
  case n if n.startsWith("Mac")  => "mac"
  case n if n.startsWith("Windows") => "win"
  case _ => throw new Exception("Unknown platform!")
}

// Add dependency on JavaFX libraries, OS dependent
lazy val javaFXModules = Seq("base", "controls", "fxml",
                "graphics", "media", "swing", "web")
libraryDependencies ++= javaFXModules.map( m =>
  "org.openjfx" % s"javafx-$m" % "12.0.2" classifier osName
)
```

- Programm laiendab `JFXApp` trait-i.
- Kood kirjutada konstruktorisse, mitte `main`-i v.m.s.
- Kasutab `x_=(...)` meetodeid, näiteks

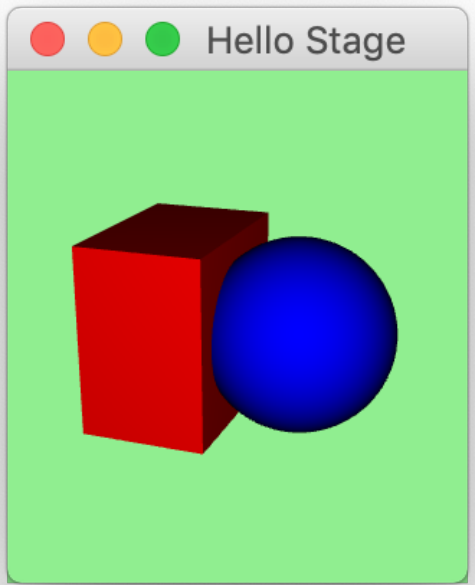
```
title = "Hello Stage"   ==>   title_=("Hello Stage")
```

Arhitektuur

- Stage – Aken ise
- Scene – Akna sisu
- Node – Akna element
  - Text
  - Shape (Line, Rectangle, . . .)
  - Chart (Bar, Pie, Line, Area, Bubble, Scatter)
  - Pane (VBox, HBox, StackPane, TilePane,GridPane . . .)
  - Control (Button, Lable, TreeView, TextArea . . .)
  - Canvas
  - ImageView
  - WebView

Shape3d

```
content = Seq(
  new Box() {
    transforms = Seq(new Translate(-1, 0, 0))
    material = new PhongMaterial(Color.Red)
    height = 3; width = 2; depth = 3
  },
  new Sphere() {
    radius = 1.5
    transforms =
      Seq(new Translate(1, 0, 0))
    material =
      new PhongMaterial(Color.Blue)
  }
)

camera = new PerspectiveCamera(true) {
  transforms =
    Seq(new Rotate(-20, Rotate.YAxis), new Rotate(-20, Rotate.XAxis),
      new Translate(0, 0, -15))
}
```



Kõrvalepõige: suhtlus kasutajaga

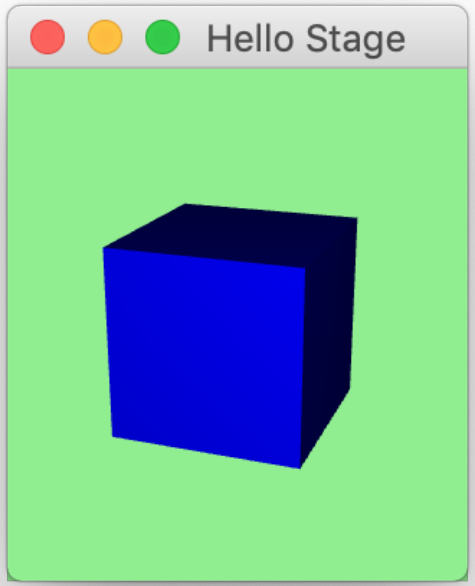
- Küsitlus
  - `if (tingimus) then tee_midagi()`
  - \+ sobiv, kui küsitleda harva
  - \- ei sobi, kui vaja kiiresti reageerid

- Sündmus
  - `override def onEvent(...) { ... }`
  - `handlers +={ ... }`
  - *Observer pattern*: registreerin vaatleja muutuva väärtuse A juurde, mis muudab väärtust B.
  - \+ sobiv, kui vaja kiiresti reageerida
  - \- tihti unustatakse *handler*ite eemaldamine

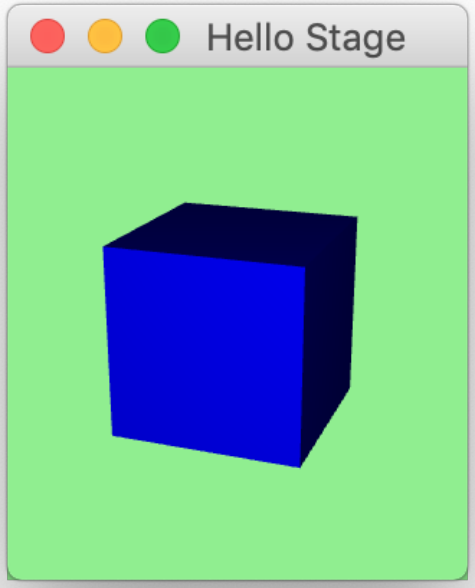
- Functional Reactive Programming (FRP)
  - `area <== base * height / 2`
  - `prop <== when(cond) choose(value1) otherwise(value2)`
  - FRP: registreerin B juurde seose, et ta võtaks väärtuse A-st.
  - \+ sobiv, kui vaja kiiresti reageerida
  - \+ sobiv lihtsate tingimuste jaoks
  - \+ pole vaja *handler*eid eemaldada
  - \- ei sobi tsükliliste sõltuvuste korral

Events

```
content =
  new Box() {
    material =
      new PhongMaterial(Color.Red)
    onMouseClicked = { _ =>
      material =
        new PhongMaterial(Color.Blue)
    }
    height = 3
    width = 3
    depth = 3
  }
```

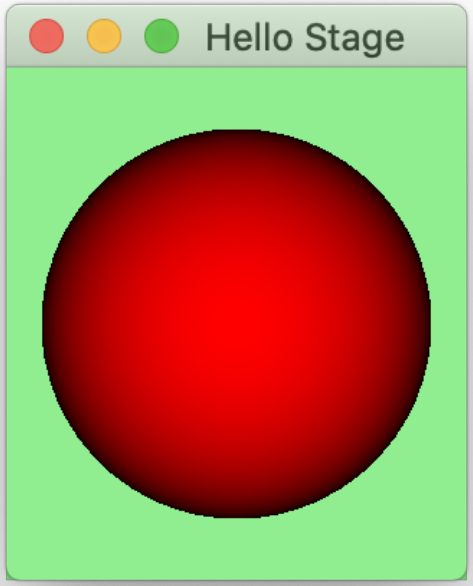


Property



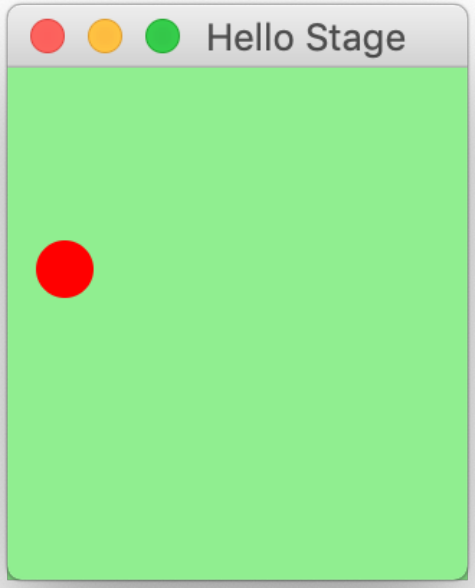

```
content =
  new Box() {
    material <==
      (when(hover)
        choose new PhongMaterial(Color.Red)
        otherwise new PhongMaterial(Color.Blue))

    height = 3; width = 3; depth = 3
  }
```

Animatsioonid




```
content =
  new Sphere() {
    material = new PhongMaterial(Color.Red)
    radius = 3
    onMouseClicked = { _ =>
      Timeline(at(3 s){radius -> 1}).play()
    }
  }
```

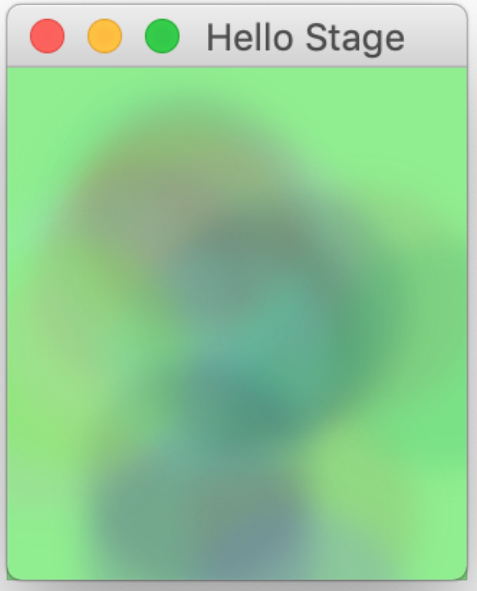
Animatsioonid 2




```
scene = new Scene {
  fill = LightGreen

  private val c = new Circle {
      centerX = 20
      centerY = 70
      radius = 10
      fill = Red
    }

  content = c

  onMouseClicked = { _ =>
    Timeline(at(0.5 s){c.centerY -> (height.value - 5)}).play()
  }
}
```

Animatsioonid 4

```
val cs = for (i <- 0 to 20) yield new Circle {
  centerX = random * 160
  centerY = random * 200
  radius = 40
  fill = color(random, random, random, 0.2)
  effect = new BoxBlur(10,10,3)
  onMouseClicked = handle {
    Timeline(at(3 s) {radius -> 0}).play()
  }
}
new Timeline{
  cycleCount = Timeline.Indefinite
  autoReverse = true
  keyFrames = for (c <- cs) yield at(20 s) {
    Set[KeyValue[_, _ <: Object]](
      c.centerX -> random * 160,
      c.centerY -> random * 200)}
}.play()
```



DelayedInit

Klassid ja objektid, mis pärivad `DelayedInit`, muudetakse nii: `code` $\Longrightarrow$ `delayedInit(code)`. S.t

```
trait C1 extends DelayedInit {
  println("C1 initsialiseerimine")
  def delayedInit(body: => Unit): Unit = {
    println("enne C2 initsialiseerimist")
    body       // C2 initsialiseerimine
    println("peale C2 initsialiseerimist")
  }
}

class C2 extends C1 {
  println("C2 initsialiseerimine")
}

object Test {
  def main(args: Array[String]): Unit = {
    val c = new C2
  }
}
```

App

`DelayedInit` kasutatakse ka trait-i `App` poolt.

```
trait App extends DelayedInit { // mõned detailid eemaldatud
  private val initCode = new ListBuffer[() => Unit]

  override def delayedInit(body: => Unit) {
    initCode += (() => body)
  }

  def main(args: Array[String]) = {
    for (proc <- initCode) proc()
  }
}

object Test extends App {
  val c = new C
}
```

Property

- Erinevad tüüpi omadused: `BooleanProperty`, `DoubleProperty`, `FloatProperty`, `IntegerProperty`, `LongProperty`, `StringProperty` ja `ObjectProperty`.
- Saab ka ise teha:

```
val speed1 = DoubleProperty(55)
val speed2 = new DoubleProperty(this, "speed2", 55)
```

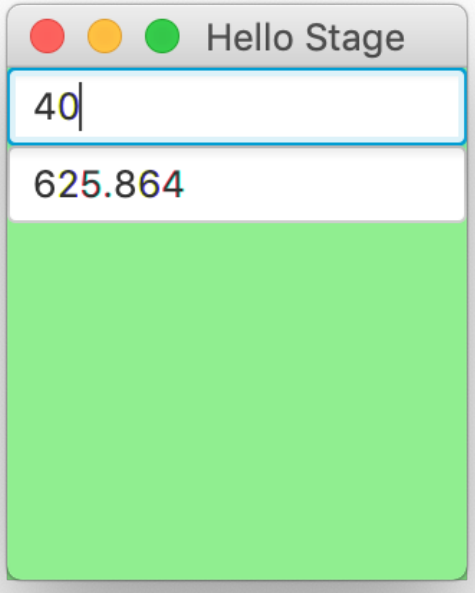
- Seotud sündmuste käsitlemisega:

```
prop.onChange { (source, oldValue, newValue) => doSomething() }
```

- Saab omavahel ühendada: `a <== b`, `a <==> b`

Omadused 2

```
scene = new Scene {
  val iF = new TextField(){
    maxWidth = 160
    text = "1"
  }

  val input1 = createIntegerBinding(
    () => Try(iF.text.value.toInt).getOrElse(0),
    iF.text)

  val outputText = new TextField(){
    maxWidth = 160
    text <== (input1 * 15.6466).asString()
  }

  content = new VBox(iF, outputText)
}
```



ScalaFX - Ensemble