Tüübiparameetrid (lihtne vorm)

- Kandilistes sulgudes, komadega eraldatult.
- Kasutatakse sõnu: geneeriline ja polümorfism.
- Meetodi või klassi nime järel.

```
class A[T](val x:T) {
  def g[U](f: T => U): U = f(x)
}
val a = new A[Int](5)
println(a.g[String](x => x.toString+"#")) // trükib 5#
```

Tüübiparameetrid (lihtne vorm)

- Kandilistes sulgudes, komadega eraldatult.
- Kasutatakse sõnu: geneeriline ja polümorfism.
- Meetodi või klassi nime järel.

```
class A[T](val x:T) {
  def g[U](f: T => U): U = f(x)
}
val a = new A[Int](5)
println(a.g[String](x => x.toString+"#")) // trükib 5#
```

- Püütakse tuletada

```
val a = new A(5)
println(a.g(x => x.toString+"#")) // trükib 5#
```

Parameetrite kitsendamine

- ülevalt: `T <: U`

```
trait Koduloom
trait Kass extends Koduloom
trait Koer extends Koduloom
class LoomaWrapper[L <: Koduloom](p: L) {
  def loom: L = p
}
```

Parameetrite kitsendamine

- ülevalt: `T <: U`

```
trait Koduloom
trait Kass extends Koduloom
trait Koer extends Koduloom
class LoomaWrapper[L <: Koduloom](p: L) {
  def loom: L = p
}
```

- alt: `T >: U`

```
trait Järjend[B] {
  def lisaAlgusesse[U >: B](e: U): Järjend[U]
}
val kassid : Järjend[Kass] = ???
val loomad : Järjend[Koduloom] = kassid.lisaAlgusesse[Koduloom](???)
```

Parameetrite kitsendamine

- ülevalt: `T <: U`

```
trait Koduloom
trait Kass extends Koduloom
trait Koer extends Koduloom
class LoomaWrapper[L <: Koduloom](p: L) {
  def loom: L = p
}
```

- alt: `T >: U`

```
trait Järjend[B] {
  def lisaAlgusesse[U >: B](e: U): Järjend[U]
}
val kassid : Järjend[Kass] = ???
val loomad : Järjend[Koduloom] = kassid.lisaAlgusesse[Koduloom](???)
```

- implitsiitselt teisendatav tüüp: `T <% U`
- `T : M` — leidub implitsiitne väärtus `M[T]`

Klasside variantsus

Oletame, et `U <: V` ehk `U` ülemklass on `V`.

	Klassi definitsioon	Tähendus
kovariantsus	`C[+T]`	`C[U] <: C[V]`
kontravariantsus	`C[-T]`	`C[U] >: C[V]`
invariantsus	`C[T]`	`C[U]` ja `C[V]` pole seotud

Klasside variantsus

Oletame, et `U <: V` ehk `U` ülemklass on `V`.

	Klassi definitsioon	Tähendus
kovariantsus	`C[+T]`	`C[U] <: C[V]`
kontravariantsus	`C[-T]`	`C[U] >: C[V]`
invariantsus	`C[T]`	`C[U]` ja `C[V]` pole seotud

- Listid on kovariantsed (`List[+A]`).
  - `List[Koer]` saame kasutada kui nõutakse `List[Koduloom]`

Klasside variantsus

Oletame, et `U <: V` ehk `U` ülemklass on `V`.

	Klassi definitsioon	Tähendus
kovariantsus	`C[+T]`	`C[U] <: C[V]`
kontravariantsus	`C[-T]`	`C[U] >: C[V]`
invariantsus	`C[T]`	`C[U]` ja `C[V]` pole seotud

- Listid on kovariantsed (`List[+A]`).
  - `List[Koer]` saame kasutada kui nõutakse `List[Koduloom]`
- Printerid on kontravariantsed.
  - `Printer[Koduloom]` saame kasutada kui nõutakse `Printer[Kass]`

```
class Printer[-A] {
  def print(p: A): Unit
}
```

Meetodi nähtavus

võtmesõna	**nähtavus**
`private[this]`	ainult sama objekt
`private`	sama klassi objektid
`protected`	lisaks alamklassidest
`private[paketiNimi]`	nimetaud paketist
	(vaikeväärtus) kõik

Objektide võrdsus

```
class Any {
  final def ==(that: Any): Boolean
  def equals(that: Any): Boolean
  ...
}
class AnyRef extends Any {
  final def eq(that: Any): Boolean
  def equals(that: Any): Boolean = this eq that
  ...
}
```

Objektide võrdsus

```
class Any {
  final def ==(that: Any): Boolean
  def equals(that: Any): Boolean
  ...
}
class AnyRef extends Any {
  final def eq(that: Any): Boolean
  def equals(that: Any): Boolean = this eq that
  ...
}
```

`AnyVal`ide puhul:

- Nii `==` kui `equals` võrdlevad sisuliselt.

`AnyRef`ide puhul:

- `eq` võrdleb viitasid
- `equals` tuleks üle laadida sisulise võrsusega, vaikimisi sama mis `eq`.
  - Ärge unustage ka `hashCode` üle defineerida!
- `x == y` $\Longleftrightarrow$ `if (x eq null) y eq null else x equals y`

Objektide võrdsus

```
class Any {
  final def ==(that: Any): Boolean
  def equals(that: Any): Boolean
  ...
}
class AnyRef extends Any {
  final def eq(that: Any): Boolean
  def equals(that: Any): Boolean = this eq that
  ...
}
```

`AnyVal`ide puhul:

- Nii `==` kui `equals` võrdlevad sisuliselt.

`AnyRef`ide puhul:

- `eq` võrdleb viitasid
- `equals` tuleks üle laadida sisulise võrsusega, vaikimisi sama mis `eq`.
  - Ärge unustage ka `hashCode` üle defineerida!
- `x == y` $\Longleftrightarrow$ `if (x eq null) y eq null else x equals y`

Mustrisobitus

- Saab kasutada lausena:

```
var sign = ...
val c: Char = ...
c match {
  case '+' => sign = 1
  case '-' => sign = -1
  case _  => sign = 0
}
```

Mustrisobitus

- Saab kasutada lausena:

```
var sign = ...
val c: Char = ...
c match {
  case '+' => sign = 1
  case '-' => sign = -1
  case _  => sign = 0
}
```

- Saab kasutada avaldisena:

```
val c: Char = ...
val sign =
  c match {
    case '+' => 1
    case '-' => -1
    case _  => 0
  }
```

Mustrisobitus

- Saab kasutada lausena:

```
var sign = ...
val c: Char = ...
c match {
  case '+' => sign = 1
  case '-' => sign = -1
  case _  => sign = 0
}
```

- Saab kasutada avaldisena:

```
val c: Char = ...
val sign =
  c match {
    case '+' => 1
    case '-' => -1
    case _  => 0
  }
```

- Nägime, et saab sobitada literaalide ning alakriipsuga.

Mustrisobitus

- Mustrid võivad sisaldada valvureid ja muutujaid:

```
var sign = 0
var s = ""
StdIn.readChar() match {
  case '+' => sign = 1
  case '-' => sign = -1
  case c if Character.isDigit(c) => s :+= c
  case _  => sign = 0
}
```

- Mustrid üle ennikute, listide ja `case`-klasside:

```
list match {
  case Nil => 0
  case x :: y :: xs => x + y
}
tuple match {
  case (x, 1) => x
  case (x, y) => x/y
}
```

Mustrisobitus

- Mustrisobitus üle tüüpide:

```
val a : Any = ???
a match {
  case x: Int    => x
  case x: String => x.length
  case _         => 1
}
```

- Kuna geneerikute (v.a. massiivid) tüübid kustutatakse, ei soovitata mustrisobitust üle nende teha.

```
Map(1 -> 'x', 2 -> 'c') match {
  case x: Map[Int,String] => println(1) // sobitub
  case _                  => println(2)
}
```

Mustrisobitus

- Mustrid kasutamine väljaspool **`case`**-avaldist:

```
val (x, y) = (x, y)
for ((k,v) <- paarid) ...
for ((k,v) <- paarid if v != 0) ...
```

Mustrisobitus

- Mustrid kasutamine väljaspool **case**-avaldist:

```
val (x, y) = (x, y)
for ((k,v) <- paarid) ...
for ((k,v) <- paarid if v != 0) ...
```

- Saame ise mustreid luua:

```
object at {
  def unapply(arg: Email): Option[(String, String)] =
    Some(arg.n, arg.h)
}
class Email(val n: String, val h: String)
val mail = new Email("kalmera", "ut.ee")

mail match {
  case n at "ut.ee" => "kohalik"
  case _            => "mujalt"
}
```

Mustrisobitus

- Suvaline arv argumente mustris:

```
object set {
  def unapplySeq[T](arg: Set[T]): Option[Seq[T]] =
    Some(arg.toSeq)
}
val s = Set(1, 2, 3)

s match {
  case set(x, y, z) => "kolm"
  case _            => "????"
}
```