Funkcionalno programiranje

Kolekcije

Motivacija za korišćenje kolekcija

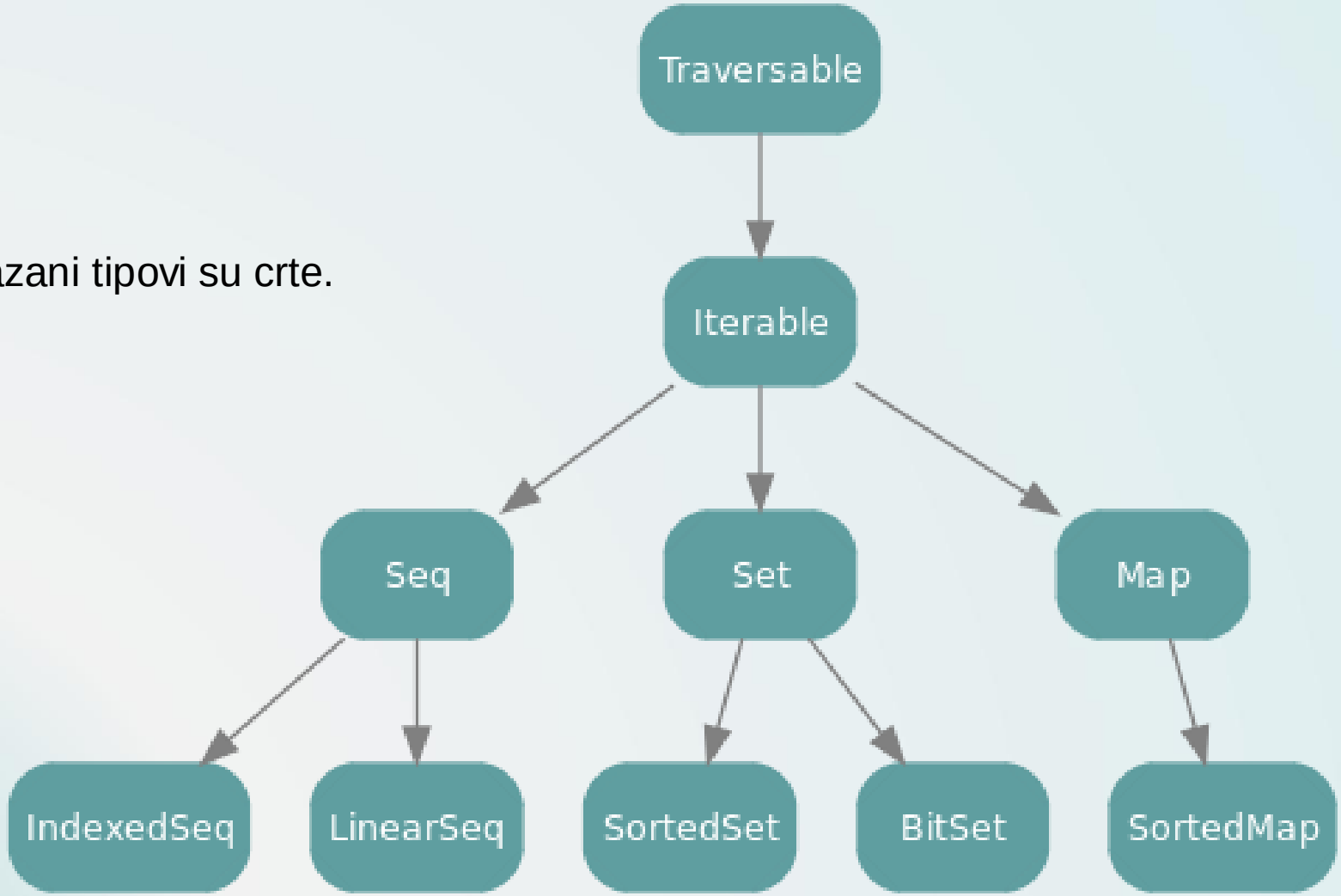
- Efikasnost programera je povećana
- Razumljiv i portabilan kod
- Optimizovana implementacija
- Proverena ispravnost i pouzdanost

Scala kolekcije

- Jednostavne za korišćenje
- Nepromenljive i promenljive kolekcije
- Statički tipizirane, pouzdane
- Brze, optimizovane operacije
- Univerzalan način korišćenja
- Paralelne kolekcije
- https://docs.scala-lang.org/overviews/core/architecture-of-scala-collections.html

Striktno neparalelne kolekcije

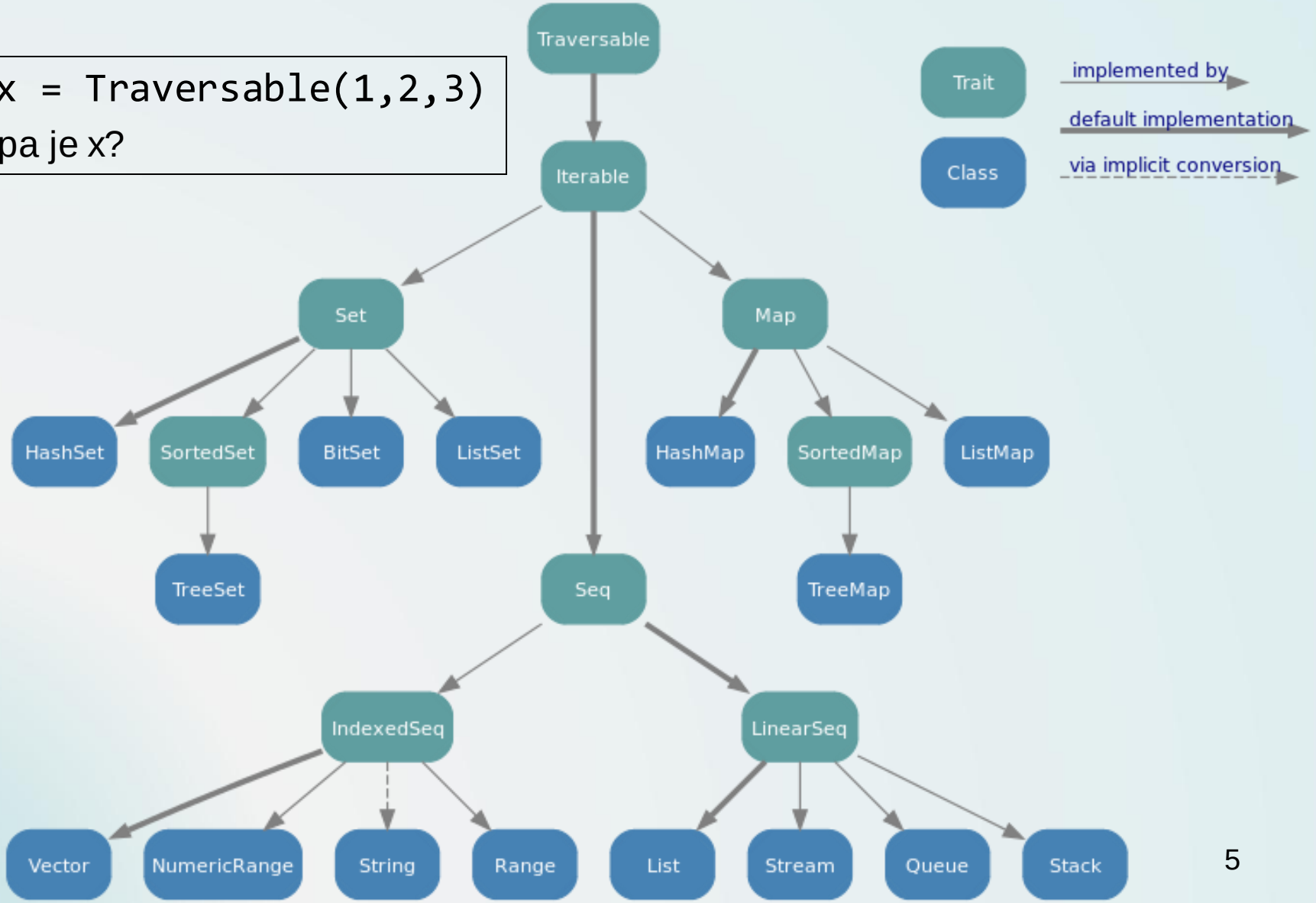
Svi prikazani tipovi su crte.



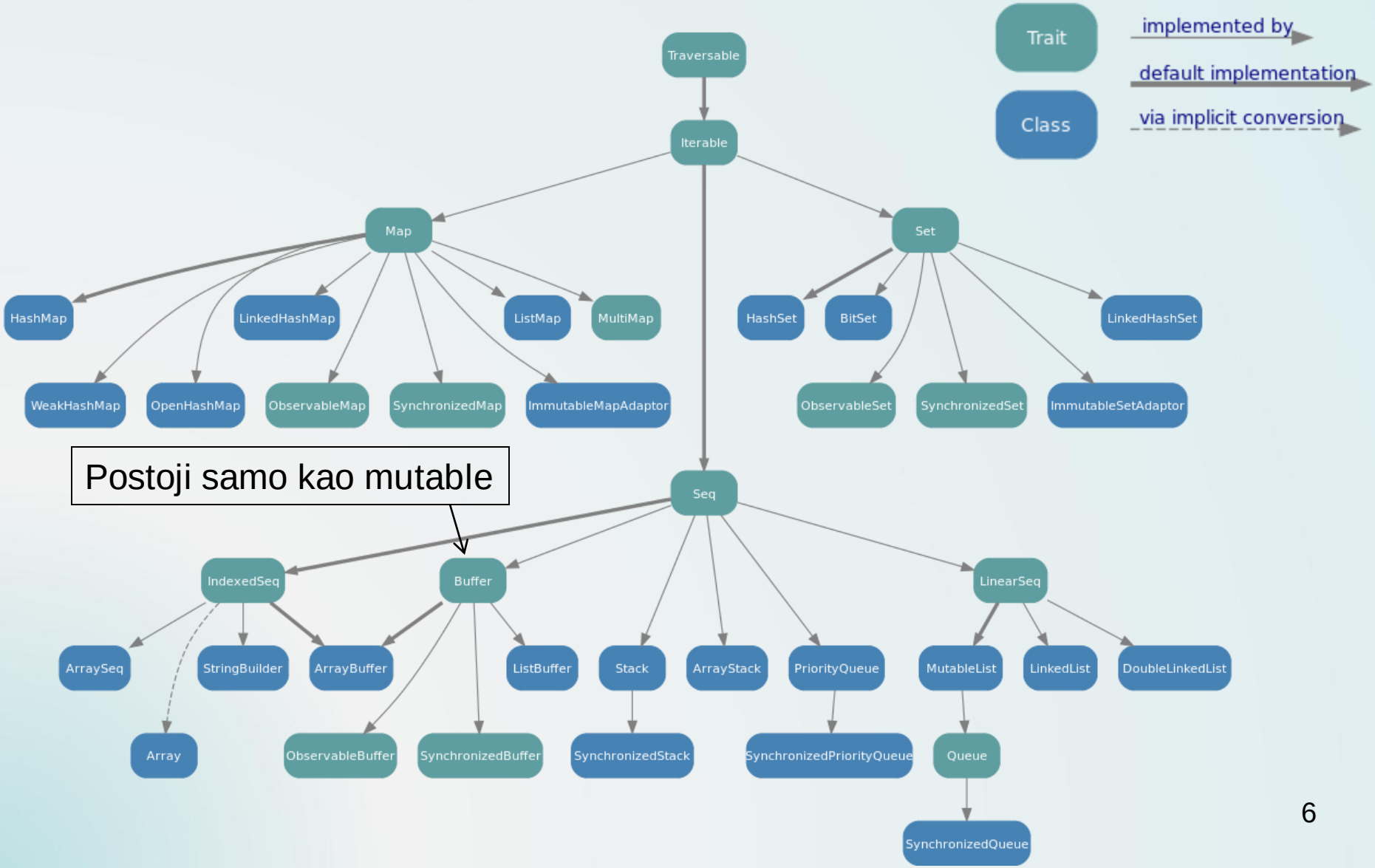
Nepromenljive kolekcije

```
var x = Traversable(1,2,3)
```

Kog tipa je x?



Promenljive kolekcije



Konzistencija kolekcija

- Sve kolekcije se mogu stvoriti upotrebom iste sintakse:
  - ime tipa kolekcije, a u zagradama: elementi

- Primeri:
  - `Traversable(1,2,3)`
  - `Iterable("x", "y", "z")`
  - `Buffer(x,y,z)`
  - `...`

- Sve nasleđuju API iz `Traversable`, ali metode su nadjačane i specijalizovane da vraćaju vlastiti tip

```
scala> List(1,2,3) map (_ + 1)
       res0: List[Int] = List(2,3,4)
  // a ne res0: Traversable[Int] ...
```

Paralelne kolekcije

- Napravljena nova hijerarhija, po uzoru na neparalelne
- Prefiks imena: Par
- `ParIterable, ParSeq, ParMap, ParSet`
- Ne postoji `ParTraversable`
  (zato što postoji specifičniji podtip: `ParIterable`)
- Postoje mutable i immutable implementacije

Tipovi sa prefiksom Gen

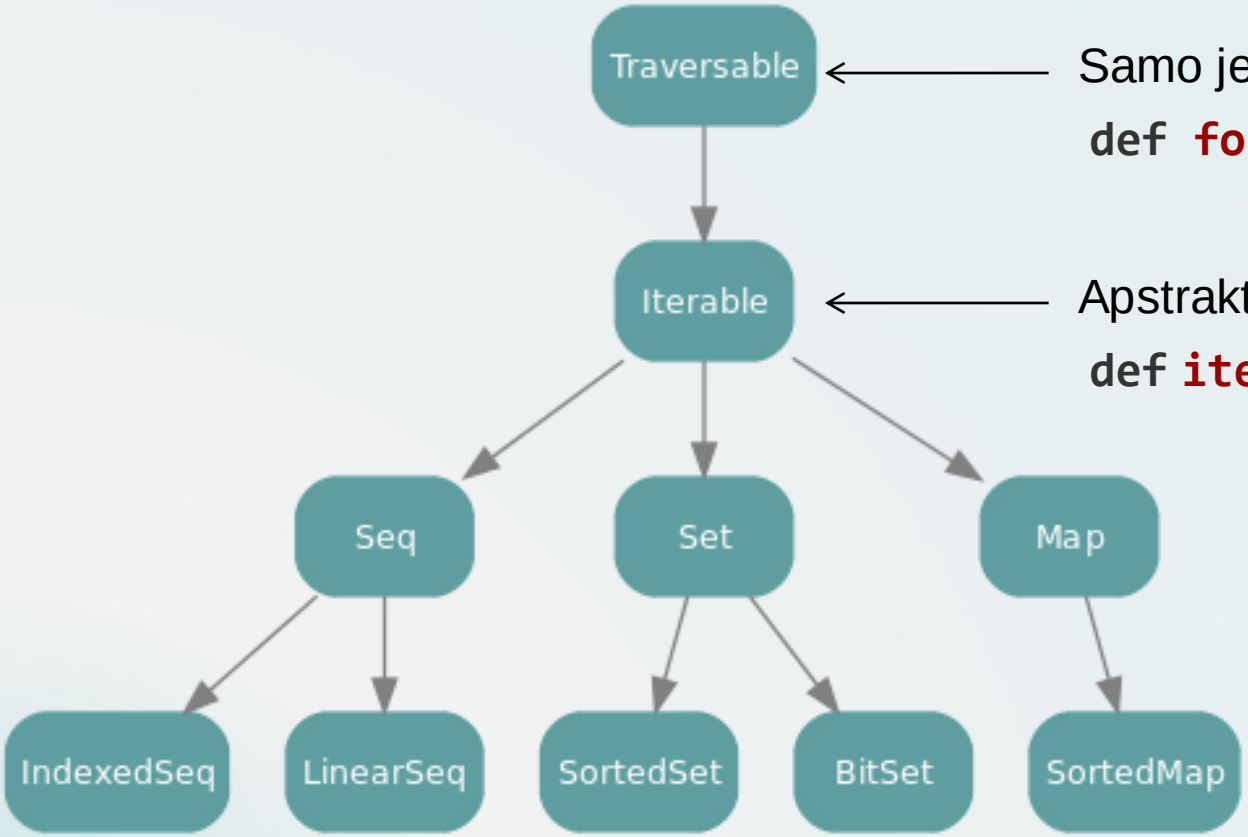
- `GenTraversable, GenIterable, GenSeq, GenMap, GenSet`
- Preslikana hijerarhija serijskih i paralelnih kolekcija
  - To su njihovi roditeljski tipovi
- To znači da
  - metoda koja uzima Seq ne može da primi paralelnu kolekciju
  - metoda koja uzima GenSeq može da operiše nad paralelnim i serijskim kolekcijama

Interoperabilnost kolekcija

- Svaka serijska kolekcija može biti pretvorena u paralelnu pozivom metode `par`
- Svaka paralelna kolekcija može biti pretvorena u serijsku pozivom metode `seq`
- Konverzija se ignoriše ako se metoda `par` pozove nad paralelnom, odnosno `seq` nad serijskom kolekcijom

Detalji tipova

Zašto i `Traversable` i `Iterable`?



Samo jedna apstraktna metoda:

```
def foreach[U](f: Elem => U)
```

Apstraktna metoda iterator:

```
def iterator: Iterator[A]
```

Traversable

Poglavlje 24

- `def foreach[U](f: Elem => U)`
- Namena: prolazak kroz kolekciju, primena funkcije f nad svakim elementom
  - bitan samo bočni efekat funkcije
  - rezultat funkcije se ignoriše
- `Traversable` definiše oko 40 metoda koje se mogu efikasno implementirati preko `foreach`
  - `forall, count, find, foldLeft, filter, remove...`

- Postoji crta `TraversableOnce`
- Nije povezana sa crtom `Traversable`
- Koriste je kolekcije i iteratori (koji nisu kolekcije)

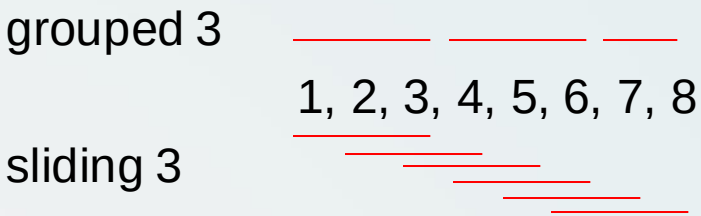
Iterable

- Deklariše apstraktnu metodu `iterator` koja vraća iterator za pristup elementima kolekcije
- Metoda `foreach` u ovoj crti je implementirana uz upotrebu metode `iterator`
  - Potklase je mogu nadjačati u cilju veće efikasnosti

```
def foreach[U](f: Elem => U): Unit = {
    val it = iterator
    while (it.hasNext) f(it.next())
}
```

Iterable

- Metode `grouped` i `sliding`
  - vraćaju iteratore
  - ovi iteratori ne vraćaju pojedinačne elemente, već sekvence




```
scala> val xs = List(1, 2, 3, 4, 5)
xs: List[Int] = List(1, 2, 3, 4, 5)
scala> val git = xs grouped 3
git: Iterator[List[Int]] = non-empty iterator
scala> git.next()
git: List[Int] = List(1, 2, 3)
scala> git.next()
git: List[Int] = List(4, 5)
```

Kako izabrati odgovarajuću kolekciju?

Nepromenljive sekvencijalne kolekcije

	IndexedSeq	LinearSeq	Description
`List`		✓	A singly linked list. Suited for recursive algorithms that work by splitting the head from the remainder of the list.
`Queue`		✓	A first-in, first-out data structure.
`Range`	✓		A range of integer values.
`Stack`		✓	A last-in, first-out data structure.
`Stream`		✓	Similar to `List`, but it's lazy and persistent. Good for a large or infinite sequence, similar to a Haskell `List`.
`String`	✓		Can be treated as an immutable, indexed sequence of characters.
`Vector`	✓		The "go to" immutable, indexed sequence. The Scaladoc describes it as, "Implemented as a set of nested arrays that's efficient at splitting and joining."

Kako izabrati odgovarajuću kolekciju?

Promenljive sekvencijalne kolekcije (1/2)

	IndexedSeq	LinearSeq	Description
`Array`	✓		Backed by a Java array, its elements are mutable, but it can't change in size.
`ArrayBuffer`	✓		The "go to" class for a mutable, sequential collection. The amortized cost for appending elements is constant.

Kako izabrati odgovarajuću kolekciju?

Promenljive sekvencijalne kolekcije (2/2)

	IndexedSeq	LinearSeq	Description
`ArrayStack`	✓		A last-in, first-out data structure. Prefer over `Stack` when performance is important.
`DoubleLinkedList`		✓	Like a singly linked list, but with a `prev` method as well. The documentation states, "The additional links make element removal very fast."
`LinkedList`		✓	A mutable, singly linked list.
`ListBuffer`		✓	Like an `ArrayBuffer`, but backed by a list. The documentation states, "If you plan to convert the buffer to a list, use `ListBuffer` instead of `ArrayBuffer`." Offers constant-time prepend and append; most other operations are linear.
`MutableList`		✓	A mutable, singly linked list with constant-time append.
`Queue`		✓	A first-in, first-out data structure.
`Stack`		✓	A last-in, first-out data structure. (The documentation suggests that an `ArrayStack` is slightly more efficient.)
`StringBuilder`	✓		Used to build strings, as in a loop. Like the Java `StringBuilder`.

Kako izabrati odgovarajuću kolekciju?

Mape, promenljive i nepromenljive

	Immutable	Mutable	Description
`HashMap`	✓	✓	The immutable version "implements maps using a hash trie"; the mutable version "implements maps using a hashtable."
`LinkedHashMap`		✓	"Implements mutable maps using a hashtable." Returns elements by the order in which they were inserted.
`ListMap`	✓	✓	A map implemented using a list data structure. Returns elements in the opposite order by which they were inserted, as though each element is inserted at the head of the map.
`Map`	✓	✓	The base map, with both mutable and immutable implementations.
`SortedMap`	✓		A base trait that stores its keys in sorted order. (Creating a variable as a `SortedMap` currently returns a `TreeMap`.)
`TreeMap`	✓		An immutable, sorted map, implemented as a red-black tree.
`WeakHashMap`		✓	A hash map with weak references, it's a wrapper around `java.util.WeakHashMap`.

Kako izabrati odgovarajuću kolekciju?

Skupovi, promenljivi i nepromenljivi

	Immutable	Mutable	
`BitSet`	✓	✓	A set of "non-negative integers represented as variable-size arrays of bits packed into 64-bit words." Used to save memory when you have a set of integers.
`HashSet`	✓	✓	The immutable version "implements sets using a hash trie"; the mutable version "implements sets using a hashtable."
`LinkedHashSet`		✓	A mutable set implemented using a hashtable. Returns elements in the order in which they were inserted.
`ListSet`	✓		A set implemented using a list structure.
`TreeSet`	✓	✓	The immutable version "implements immutable sets using a tree." The mutable version is a mutable `SortedSet` with "an immutable AVL Tree as underlying data structure."
`Set`	✓	✓	Generic base traits, with both mutable and immutable implementations.
`SortedSet`	✓	✓	A base trait. (Creating a variable as a `SortedSet` returns a `TreeSet`.)

Baferi

- Omogućavaju efikasnu izmenu sadržaja, umetanje elemenata, brisanje elemenata, dodavanje na kraj
  - `+=` i `++=` - dodavanje na kraj
  - `+=:` i `++=:` - dodavanje na početak

- Često se koriste `ListBuffer` i `ArrayBuffer`
  - brza konverzija u `List` odnosno `Array`

Tokovi (streams)

- Nalik na listu, ali se elementi određuju (računaju) po potrebi (call-by-need)
- Zato tok može da bude neograničenog kapaciteta
- Osim toga, tokovi imaju iste performanse kao i liste
- Tokovi se stvaraju pomoću operatora `#::`

```
scala> val str = 1 #:: 2 #:: 3 #:: Stream.empty
str: scala.collection.immutable.Stream[Int] = Stream(1, ?)
// prikazana samo glava, rep još nije izračunat.
// toString vodi računa da ne izazove računanje
```

Tokovi (streams)

```
scala> def fibFrom(a: Int, b: Int): Stream[Int] =
   a #:: fibFrom(b, a + b)
fibFrom: (a: Int, b: Int)Stream[Int]


scala> val fibs = fibFrom(1, 1).take(7)
fibs: scala.collection.immutable.Stream[Int] = Stream(1, ?)
scala> fibs.toList
res23: List[Int] = List(1, 1, 2, 3, 5, 8, 13)
```

Pogledi (views)

- Od svih konkretnih kolekcija, jedino tokovi obavljaju lenjo (lazy) računanje vrednosti elemenata
- Moguće je uspostaviti pogled (`view`) nad kolekcijom
- Pogled je nepromenjena kolekcija, ali sve transformacije nad podacima se lenjo računaju
- Vraćanje iz pogleda u striktnu kolekciju metodom `force`

```
scala> val v = Vector(1 to 10: _*)
v: scala.collection.immutable.Vector[Int] =
Vector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
scala> v map (_ + 1) map (_ * 2)
res5: scala.collection.immutable.Vector[Int] =
Vector(4, 6, 8, 10, 12, 14, 16, 18, 20, 22)
```

Pogledi (views)

```
scala> val vv = v.view
vv: scala.collection.SeqView[Int,Vector[Int]] =
SeqView(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> vv map (_ + 1)
res13: scala.collection.SeqView[Int,Seq[_]] = SeqViewM(...)

scala> res13 map (_ * 2)
res14: scala.collection.SeqView[Int,Seq[_]] = SeqViewMM(...)

scala> res14.force
res15: Seq[Int] = Vector(4, 6, 8, 10, 12, 14, 16, 18, 20, 22)
```

Performanse

	head	tail	apply	update	prepend	append	insert
immutable							
`List`	C	C	L	L	C	L	-
`Stream`	C	C	L	L	C	L	-
`Vector`	eC	eC	eC	eC	eC	eC	-
`Stack`	C	C	L	L	C	L	-
`Queue`	aC	aC	L	L	L	C	-
`Range`	C	C	C	-	-	-	-
`String`	C	L	C	L	L	L	-
mutable							
`ArrayBuffer`	C	L	C	C	L	aC	L
`ListBuffer`	C	L	L	L	C	C	L
`StringBuilder`	C	L	C	C	L	aC	L
`MutableList`	C	L	L	L	C	C	L
`Queue`	C	L	L	L	C	C	L
`ArraySeq`	C	L	C	C	-	-	-
`Stack`	C	L	L	L	C	L	L
`ArrayStack`	C	L	C	C	aC	L	L
`Array`	C	L	C	C	-	-	-

	lookup	add	remove	min
immutable				
`HashSet/HashMap`	eC	eC	eC	L
`TreeSet/TreeMap`	Log	Log	Log	Log
`BitSet`	C	L	L	eC[a]
`ListMap`	L	L	L	L
mutable				
`HashSet/HashMap`	eC	eC	eC	L
`WeakHashMap`	eC	eC	eC	L
`BitSet`	C	aC	C	eC[a]