ОБОБЩЁННЫЕ ТИПЫ

ОБОБЩЁННЫЙ ТИП

```
final case class NamedInt(name: String, value: Int)

final case class NamedDouble(name: String, value: Double)
```

ОБОБЩЁННЫЙ ТИП

```
final case class NamedInt(name: String, value: Int)

final case class NamedDouble(name: String, value: Double)
```

```
final case class Named[A](name: String, value: A)

Named[Int]
Named[Double]
```

ССЫЛКИ НА ТИП

```
final case class Named[A](name: String, value: A){
  def withName(newName: String): Named[A] =
    Named(newName, value)
}
```

- После этого мы можем конструировать значения Named, указывая или не указывая тип, параметр типа может выводиться автоматически

ССЫЛКИ НА ТИП

```
final case class Named[A](name: String, value: A){
  def toMap: Map[String, A] = Map(name -> value)
}
```

- Можем ссылаться на другие обобщённые типы

ССЫЛКИ НА ТИП

```
final case class Named[A](name: String, value: A){
  def mapValue[B](f: A => B): Named[B] =
    Named(name, f(value))
}
```

- Класс также может иметь обобщённые методы, тогда в теле и параметрах мы можем ссылаться на параметры типа как класса, так и метода

АБСТРАКТНЫЕ ТИПЫ

```
trait Named[A]{
    def name: String
    def value: A
    def modify(f: A => A): Named[A]
}
```

АБСТРАКТНЫЕ ТИПЫ

```
trait Named[T]{
    def name: String
    def value: T
    def modify(f: A => A): Named[A]
}

def namedInt(n: String, v: Int) =
    new Named[Int]{
      def name = n
      def value = v
      def modify(f: A => A) = namedInt(f(v))
    }
```

- При реализации типы будут проверены

АБСТРАКТНЫЕ ТИПЫ

```
trait Named[A]{
    def name: String
    def value: A
}
case class NamedList[A](name: String, value: List[A]])
 extends Named[List[A]]
```

- При наследовании вы можете можете передавать параметры типов

НЕСКОЛЬКО ПАРАМЕТРОВ

```
final case class Dict[K, V](items: List[(K, V)])
```

- Класс может иметь несколько параметров типов

ВЕРХНЯЯ ГРАНИЦА

```
final case class Dict[I <: Item](items: List[I])

def dict[I <: Item](items: I*): Dict[I] =
  Dict(items.toList)

trait Item {
  def key: String
  def value: String
}
```

- Параметры типов могут иметь ограничение "сверху"
  Означает, что тип должен быть подтипом указанного типа

НИЖНЯЯ ГРАНИЦА

```
final case class Dict[I <: Item](items: List[I]){
  def +:[J >: I](item: J): Dict[J]  =
    Dict(item :: items)
}

trait Item {
  def key: String
  def value: String
}
```

- Параметры типов могут иметь ограничение "снизу"
  Означает, что тип должен быть надтипом указанного типа

ССЫЛКИ

```
case class Dict[K, V, I <: Item[K, V]](items: List[I])

trait Item[K, V] {
  def key: K
  def value: V
}
```

- Параметры типов могут ссылаться друг на друга в ограничениях

АБСТРАКТНЫЕ ТИПЫ

```
trait Comparable[A <: Comparable[A]]{
  def compare(x: A): Int
}
```

- И даже на самого себя

В этом разделе мы изучили обобщённые типы

В следующем практика