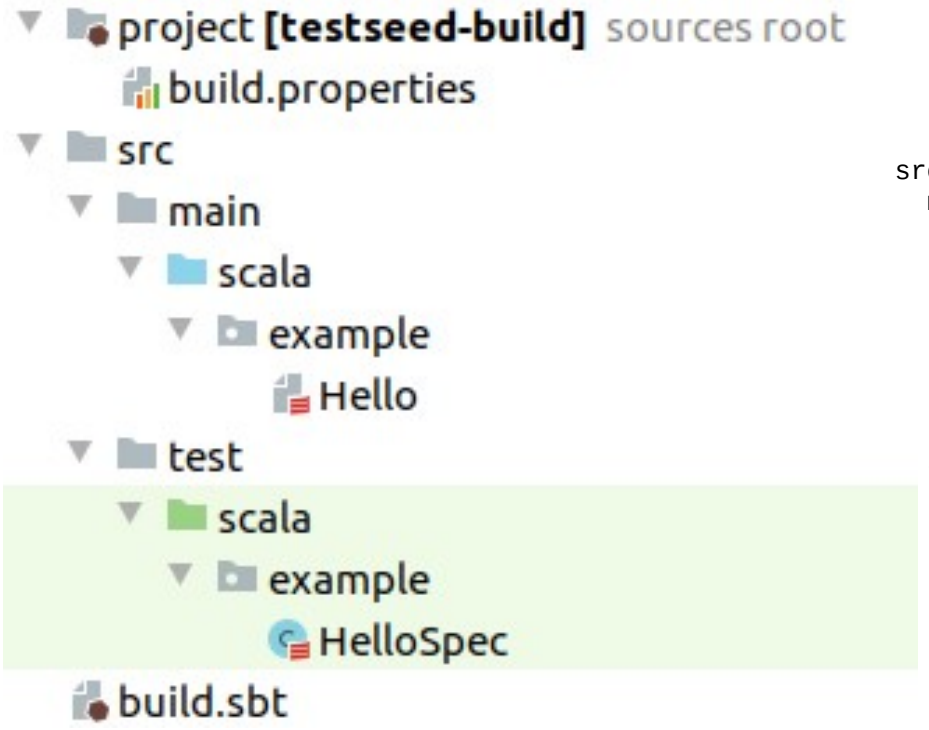
scala build tool (sbt)

- Система автоматической сборки для проектов, написанных на языках Scala и Java
- Сайт: https://www.scala-sbt.org/index.html

Создание небольшого sbt приложения

- sbt new scala/scala-seed.g8




```
src/
  main/
    resources/
       <files to include in main jar here>
    scala/
       <main Scala sources>
    scala-2.12/
       <main Scala 2.12 specific sources>
    java/
       <main Java sources>
  test/
    resources
       <files to include in test jar here>
    scala/
       <test Scala sources>
    scala-2.12/
       <test Scala 2.12 specific sources>
    java/
       <test Java sources>
```

build.properties

- Вы можете указать версию sbt для проекта. sbt.version=1.6.2
- Если требуемая версия недоступна локально, программа запуска sbt загрузит ее для вас. Если этого файла нет, программа запуска sbt выберет произвольную версию (не рекомендуется)

build.sbt

```
ThisBuild / scalaVersion     := "2.13.7"
ThisBuild / version          := "0.1.0-SNAPSHOT"
ThisBuild / organization     := "com.example"
ThisBuild / organizationName := "example"

lazy val root = (project in file("."))
  .settings(
    name := "testseed",
    libraryDependencies ++= Seq(
      "org.scalatest" % "scalatest_2.13" % "3.2.9" % Test
    )
  )
```

На данном этапе нас будет интересовать работа с зависимостями.

Зависимости библиотеки можно добавить двумя способами:
0) неуправляемые зависимости – это jar-файлы, помещенные в каталог lib
1) управляемые зависимости настраиваются и автоматически загружаются из репозиториев (дальше будем рассматривать этот случай)

Объявление зависимостей выглядит следующим образом:

```
libraryDependencies += groupID % artifactID % revision % configuration
```

Моментик:
При использовании %% sbt добавит версию Scala вашего проекта к имени требуемого артефакта.

Например:

```
"org.scalatest" %% "scalatest" % "3.2.9" % Test – вытянет scalatest_2.13 версии 3.2.9
```

Некоторые из команд sbt

Command	Description
`clean`	Deletes all generated files (in the `target` directory).
`compile`	Compiles the main sources (in `src/main/scala` and `src/main/java` directories).
`test`	Compiles and runs all tests.
`console`	Starts the Scala interpreter with a classpath including the compiled sources and all dependencies. To return to sbt, type `:quit`, Ctrl+D (Unix), or Ctrl+Z (Windows).
`run <argument>*`	Runs the main class for the project in the same virtual machine as sbt.
`package`	Creates a jar file containing the files in `src/main/resources` and the classes compiled from `src/main/scala` and `src/main/java`.
`help <command>`	Displays detailed help for the specified command. If no command is provided, displays brief descriptions of all commands.
`reload`	Reloads the build definition (`build.sbt`, `project/*.scala`, `project/*.sbt` files). Needed if you change the build definition.

Еще нас будет интересовать команда testOnly – позволяющая запускать только определенные тесты.

sbt "testOnly example.HelloSpec"

Specs2

https://etorreborre.github.io/specs2/

Для подключения зависимости нужно прописать в build.sbt:

libraryDependencies += "org.specs2" %% "specs2-core" % "4.13.3" % *Test – версии выше 4.14.1 используют Scala 3*

В зависимости от функций spec2, которые вы хотите использовать в дальнейшем, вам потребуется добавить больше зависимостей в вашу сборку. Некоторые из либ, которяе могут пригодится:

Name	Functionality
`specs2-matcher-extra`	for the optional ***specs2*** matchers
`specs2-scalacheck`	to use ScalaCheck properties in specifications
`specs2-mock`	to use Mockito matchers
`specs2-analysis`	to use the package dependencies matcher
`specs2-gwt`	to write given/when/then specifications
`specs2-html`	to export specifications as html
`specs2-form`	to create html form-like specifications
`specs2-junit`	to run specifications as JUnit tests

Unit Specification

```
import org.specs2.mutable._

class UnitHiSpec extends Specification {

  "The 'Hello world' string" should {


    "contain 11 characters" in {
      "Hello world" must have size(11)
    }


    "start with 'Hello'" in {
      "Hello world" must startWith("Hello")
    }


    "end with 'world'" in {
      "Hello world" must endWith("world")
    }


  }

}
```

Спецификация в Spec2 начинается со строки, описывающей тестируемый класс, а заканчивается методом should. В блоке should находится одно или несколько описаний тестов. Каждый тест из себя представляет строковое описание проверяемого действия и блок тестового кода.

Все тесты Specs2 являются асинхронными и каждый выполняется в своем потоке

Обратите внимание, что эта спецификация импортирует:
import org.specs2.mutable(!).Specification

Acceptance Specification

```
import org.specs2.Specification

class AcceptanceHiSpec extends Specification {

  def is =
    s2"""
    Hi specification
    where 'Hello world' contain 11 characters  $e1
    where 'Hello world' start with 'Hello'     $e2
    where 'Hello world' end with 'world'       $e3
      """

  def e1: MatchResult[String] = "Hello world" must have size 11

  def e2: MatchResult[String] = "Hello world" must startWith("Hello")

  def e3: MatchResult[String] = "Hello world" must endWith("world")

}
```

Обратите внимание, что эта спецификация импортирует:
import org.specs2.Specification

Методом, запускающим весь тест для класса, является метод is. Метод возвращает объект, содержащий все тестовые примеры

Matchers

```
def e3: MatchResult[String] = "Hello world" must endWith("world")
```

Для того, чтобы указать ожидаемое поведение тестируемого блока, используются сопоставители.

В примере оператор must применяет значение к сопоставителю endWith, созданному с ожидаемым значением. Таких сопоставителей множество. О них читаем в доке =)

Matcher	Comment
`beTypedEqualTo`	typed equality. `a must beTypedEqualTo(b)` will not work if `a` and `b` don't have compatible types
`be_===`	synonym for `beTypedEqualTo`
`a ==== b`	synonym for `a must beTypedEqualTo(b)`
`a must_=== b`	similar to `a must_== b` but will not typecheck if `a` and `b` don't have the same type
`be_==~`	check if `(a: A) == conv(b: B)` when there is an implicit conversion `conv` from `B` to `A`
`beTheSameAs`	reference equality: check if `a eq b` (`a must be(b)` also works)
`be`	`a must be(b)` : synonym for `beTheSameAs`
`beTrue, beFalse`	shortcuts for Boolean equality

Matcher	Comment
`1 must beEqualTo(1)`	the normal way
`1 must be_==(1)`	with a symbol
`1 must_== 1`	my favorite!
`1 mustEqual 1`	if you dislike underscores
`1 should_== 1`	for `should` lovers
`1 === 1`	the ultimate shortcut
`1 must be equalTo(1)`	with a literate style

ption
f a string matches a regular expression
ut for `beMatching("(.|\\s)*"+s+"(.|\\s)*")`
f some groups are found in a string
the length of a string

`have size`	check the size of a string (seen as an `Iterable[Char]`)
`be empty`	check if a string is empty
`beEqualTo(b).ignoreCase`	check if 2 strings are equal regardless of casing
`beEqualTo(b).ignoreSpace`	check if 2 strings are equal when you `replaceAll("\\s", "")`
`beEqualTo(b).trimmed`	check if 2 strings are equal when trimmed
`beEqualTo(b).ignoreSpace.ignoreCase`	you can compose them
`contain(b)`	check if a string contains another one
`startWith(b)`	check if a string starts with another one
`endWith(b)`	check if a string ends with another one

Текущего хватит, чтобы написать примитичный тест.

В целом в бибилиотечке много всего интересного - идем читать документацию

ScalaCheck

Использует подход property-based тестирования, который помогает проверить, соответствует ли тестируемая функция заданному свойству.

Тестовые данные генерируются автоматически и в отличие от обычных тестов, нет необходимости задавать явно все тестовые примеры.

ScalaCheck сам по себе: `"org.scalacheck" %% "scalacheck" % "1.14.3" % Test`

ScalaCheck для specs2: `"org.specs2" %% "specs2-scalacheck" % specs2Version % Test`

Specs2 + ScalaCheck

```
import org.specs2.ScalaCheck
import org.specs2.mutable.Specification

class MinimalSpec2ChechSpec extends Specification with ScalaCheck {
  "MinimalSpec2ChechSpec" should {
    "correct work" in {
      prop { (a: Int) => {
        a + a == 2 * a
      }}
    }
  }
}
```

Чтобы объявить свойства ScalaCheck, вам сначала нужно расширить трейт org.specs2.ScalaCheck.

Затем вы можете передать функции, возвращающие любой результат (логический, результат, MatchResult или ScalaCheck Prop) в метод prop

Для работы требуются неявные экземпляры Arbitrary[T] для каждого типа генерируемого параметра. Часть их написана уже за нас, некоторые нужно будет описывать самому. =)

```
sealed trait Led
case object Red extends Led
case object Green extends Led
case object Blue extends Led

class LedSpec2ChechSpec extends Specification with ScalaCheck {

  val ledGenerator: Gen[Led] = Gen.oneOf(Red, Green, Blue)
  implicit val arbitraryLed: Arbitrary[Led] = Arbitrary(ledGenerator)

  "LedSpec2ChechSpec" should {
    "correct work" in {
      prop { (a: Led) => a mustEqual Red }
    }
  }

}
```

Specs2 + ScalaCheck

```
Falsified after 0 passed tests.
> ARG_0: Green
The seed is lwJOqFTCMhKjb4VSrD2xzHWtUnPNMMKuLkJD6hQIDyN=

> Green != Red
Expected :Red
Actual   :Green
```

При падении теста в консоль будет выведен seed, использовав который, можно будет воспроизвести тестовый набора данных и найти ошибку.

```
testOnly *MySpec -- ex myTest scalacheck.seed lwJOqFTCMhKjb4VSrD2xzHWtUnPNMMKuLkJD6hQIDyN=
```

```
prop ((i: Int) => i % 2 == 0).setSeed("lwJOqFTCMhKjb4VSrD2xzHWtUnPNMMKuLkJD6hQIDyN=")
```