Современные технологии программирования
Лабораторные работы 9-10

Функциональное программирование на языке Scala. Система сборки sbt. Среда разработки Intellij IDEA. Объекты, классы, трэйты.

Для разработки программ на языке Scala и выполнения лабораторных работ потребуются следующие программные средства:

1. Установленная виртуальная машина java (JRE или JDK от Oracle версии 1.7):

```
$ java -version
java version "1.7.0_09"
Java(TM) SE Runtime Environment (build 1.7.0_09-b05)
Java HotSpot(TM) 64-Bit Server VM (build 23.5-b02, mixed mode)
```

2. Установленная Scala:

```
$ scala -version
Scala code runner version 2.10.0 -- Copyright 2002-2012, LAMP/EPFL
```

Установка scala происходит также как и java: скачивается архив с последней стабильной версией с сайта http://www.scala-lang.org/downloads, распаковывается в любую директорию и в переменную окружения PATH добавляется путь на подкаталог bin этой директории. В Ubuntu для этого редактируется файл /etc/enviroment:

```
PATH="/srv/server/jdk1.7.0_09/bin:/srv/server/scala-2.10.0/bin:/usr/local/sbin:/usr/local/bin:..."
JAVA_HOME="/srv/server/jdk1.7.0_09"
JDK_HOME="/srv/server/jdk1.7.0_09"
SCALA_HOME="/srv/server/scala-2.10.0"
CLASSPATH=".:/srv/server/jdk1.7.0_09/bin"
```

Для информирования различных других программ о местонахождении java и scala также бывает полезно задать переменные окружения JAVA_HOME, JDK_HOME и SCALA_HOME.

После выполнения этих шагов Вам будет уже доступен интерактивный интерпретатор Scala (REPL), с помощью которого, используя любой текстовый редактор с подсветкой синтаксиса языка Scala (geany, gedit, ...), можно разрабатывать небольшие программы на этом языке.

3. Система сборки sbt.

По некоторым причинам для Scala полноценное использование IDE (NetBeans, Intellij IDEA, Eclipse) так, как это было для Java, - невозможно.

Для всех этих сред существуют плагины для поддержки подсветки синтаксиса языка и автодополнения, однако компиляция и сборка проходят в них крайне медленно, да и полноценная отладка кода также невозможна. Поэтому лучшим решением

является использование для компиляции и сборки проекта системы sbt — simple build tool, основанной на репозиториях maven.

Установка: скачиваем архив с ресурса: http://www.scala-sbt.org/release/docs/Getting-Started/Setup.html, распаковываем в любой каталог (фактически из архива нужны только два файла: **sbt-launch.jar** и запускающий скрипт **sbt**) и добавляем путь на этот каталог в переменную окружения PATH. После установки команда sbt должна вызываться из любого каталога файловой системы.

4. Среда разработки Intellij IDEA.

Для более сложных проектов, содержащих множество каталогов и файлов, а также для фреймворка play 2, будем использовать IDE Intellij IDEA community edition (свободное программное обеспечение). В основном, для ввода текстов программ и быстрой навигации по файлам проекта.

Установка: скачиваем с ресурса http://www.jetbrains.com/idea/download/index.html архив с последней версией Intellij IDEA community edition, распаковываем в некоторый каталог и размещаем на рабочем столе запускающий скрипт, типа:

```
#!/bin/sh
cd /home/george/Soft/idea-12/bin
sh idea.sh
```

Для поддержки Scala и sbt нужно установить необходимые плагины. Делается это в самой программе Intellij IDEA: Главное меню → File → Settings → Plugins → Browse repositories → ввод в строке поиска scala → выбор Scala и SBT. После выбора произойдет скачивание выбранных плагинов и их установка.

Язык Scala

Scala - статически типизированный объектно-ориентированный язык программирования, на котором можно создавать приложения для jvm (виртуальной java-машины). Поддерживает как императивный так и функциональный стили программирования. Компилятор (scalac) для jvm написан на java Мартином Одерски (Швейцария, Лозанна). Существует также компилятор для платформы .NET.

На Scala одну и ту же программу можно написать совершенно по-разному. Рассмотрим, как пример, следующую задачу:

f(n) = сумме всех натуральных чисел меньших n, делящихся на 3 или на 5. Написать программу для вычисления n → f(n).

1. Решение на java:

```
int f (int n) {
  int s = 0;
  for (int i=1; i<n; i++) {
    if ((i%3==0)||(i%5==0)) s+=i;
  }
  return s;
}
```

2. Решение на Scala (императивный стиль — как в java):

```
def f(n: Int) = {
  var s = 0
  for (i <- 1 to n-1) {
    if ((i%3==0)||(i%5==0)) s+=i
  }
  s
}
```

3. Решение на Scala (функциональный стиль):

```
def f(n: Int) = (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
```

4. Решение на Scala (задание функции как объекта):

```
val f =  (n: Int) => (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
```

Для того чтобы протестировать пример на java придется дописать еще метод main:

```
public class Test {
  static int f (int n) {
    int s = 0;
    for (int i=1; i<n; i++) {
      if ((i%3==0)||(i%5==0)) s+=i;
    }
    return s;
  }

  public static void main(String[] args) {
    if (args.length>0) System.out.println(f(Integer.parseInt(args[0])));
    else System.out.println("Не задан аргумент");
  }
}
```

Примеры же на Scala можно легко протестировать в интерактивном интерпретаторе Scala (REPL), просто скопировав и вставив в него исходный код функции.

```
$ scala
Welcome to Scala version 2.10.0 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_09).
Type in expressions to have them evaluated.
Type :help for more information.

scala> def f(n: Int) = (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
f: (n: Int)Int

scala> f(999)
res0: Int = 232169

scala>
```

Интерактивный интерпретатор обладает возможностью автодополнения кода для используемых классов Scala и Java и помогает быстро проверить как работает тот или иной метод, без необходимости чтения документации.

Для использования программы вне интерпретатора, необходимо, как и в java создать объект, реализующий метод main.

```
object Test {
  def f(n: Int) = (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum

  def main(args: Array[String]) = {
    if (args.length>0) println(f(args(0).toInt))
    else println("Не задан аргумент")
  }
}
```

Сохраним этот текст в файл с именем script.scala. Теперь мы можем его использовать просто как скрипт, выполняя команду типа:

```
$ scala script.scala 999
```

Но можем также его скомпилировать для jvm:

```
$ scalac script.scala
```

Правда запустить программу на выполнение командой (как для java):

```
$ java Test 999
```

не удастся, поскольку библиотека Scala (scala-library.jar) не содержится в classpath, однако, если мы добавим ее в classpath, то все запустится:

```
$ java -cp .:/srv/server/scala-2.10.0/lib/scala-library.jar Test 999
232169
```

Из этого примера уже можно заметить снятие в синтаксисе Scala многих ограничений языка Java и также некоторые существенные различия:

1. Необязательность точки с запятой в конце строк
2. Имя объекта с методом main не обязано совпадать с именем файла
3. Необязательность задания типа переменной, если он может быть вычислен компилятором из контекста
4. Необязательность директивы return для возвращаемого значения
5. Необязательность круглых скобок у метода без аргументов (sum = sum())
6. Возможность объявить не класс, а объект устраняет директивы static
7. Типы объявляются после переменной или функции через двоеточие (n: Int)
8. Элементы массива(списка) выбираются при помощи круглых, а не квадратных скобок (args(0))
9. Типизация массива(списка) задается квадратными, а не угловыми скобками (Array[String])
10. Функции определяются директивой def, изменяемые переменные var,

неизменяемые val (тем самым отпадает необходимость в директиве final)

11. При задании функций между определением и реализацией используется знак = как и при задании переменных

Отметим еще, что в Scala нет примитивных типов и числа сразу являются объектами (как строки), поэтому арифметическая операция +, например, это обычный метод у объекта-числа:

```
scala> val a = 2; val b = 3
a: Int = 2
b: Int = 3

scala> a + b
res17: Int = 5

scala> a.+(b)
res18: Int = 5

scala>
```

Сравнение == объектов в Scala осуществляется по их внутреннему содержанию, а не по указателям, как в java:

```
scala> val x = "abc"; val y = "abc"
x: String = abc
y: String = abc

scala> x == y
res32: Boolean = true

scala> val l = List(1, 2, 3, 2, 1)
l: List[Int] = List(1, 2, 3, 2, 1)

scala> l == l.reverse
res33: Boolean = true

scala>
```

Для значительной части стандартных операций Java в Scala имеются их упрощенные эквиваленты, например так можно ввести строку в консоли и преобразовать ее к целому:

```
scala> val x = readLine
x: String = 12345

scala> x.toInt
res2: Int = 12345

scala>
```

При этом, если какая-то функциональность отсутствует в Scala, то Вы всегда можете использовать любые стандартные классы java, например,

```
scala> def date = {
     |   import java.util.Date
     |   import java.text.SimpleDateFormat
     |   (new SimpleDateFormat("dd.MM.yyyy k:m:s")).format(new Date)
     | }
date: String

scala> date
res10: String = 06.01.2013 17:42:52

scala>
```

Обратите внимание на то, что import можно делать прямо внутри функции. Кстати, вместо * для импортирования всех классов пакета используется символ _

```
import java.util._
```

На Scala легко начать писать программы в императивном стиле, если Вы знаете Java. И даже в этом случае Вы уже получите значительную свободу от ограничений языка Java. Однако главная задача - научиться использовать функциональные возможности языка Scala. Проанализируем, например, наше решение:

```
def f(n: Int) = (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
```

Лучше всего это сделать в интерактивном интерпретаторе Scala:

```
scala> val n = 10
n: Int = 10

scala> 1 to n-1
res19: scala.collection.immutable.Range.Inclusive = Range(1, 2, 3, 4, 5, 6, 7, 8, 9)

scala> (1 to n-1).filter(x => (x%3==0) || (x%5==0))
res20: scala.collection.immutable.IndexedSeq[Int] = Vector(3, 5, 6, 9)

scala> (1 to n-1).filter(x => (x%3==0) || (x%5==0)).sum
res21: Int = 23

scala>
```

При n = 10 мы видим, что `1 to n-1` возвращает тип `Range(1, 2, 3, 4, 5, 6, 7, 8, 9)` - диапазон чисел от 1 до 9. У Range есть метод filter, которому в качестве аргумента передается анонимная функция `x => (x%3==0) || (x%5==0)` и этот метод выбирает (фильтрует) только те элементы, на которых эта функция возвращает true. В результате мы получаем `Vector(3, 5, 6, 9)` , у которого есть метод sum, возвращающий сумму элементов вектора.

Как научиться писать программы в функциональном стиле на Scala? Прежде всего следует изучить возможности самого языка, в котором по сравнению с Java добавлено значительное число уже готовых функций для этого (таких как filter, map, exists, count, find, …).

С другой стороны этого может оказаться недостаточно, нужно научиться мыслить по-другому и использовать абстракции. Нужно привыкнуть к тому, что функции — это обычные объекты типа (x: Int, y: Int) => x + y и мы можем писать функции, которые возвращают объекты-функции.

Как правило, если нам нужно написать метод в функциональном стиле и в языке нет готовых функций для этого, то следует искать возможность использования рекурсии. Рекомендуется внимательно изучить примеры главы «Функции первого класса» из книги Мартина Одерски «Scala в примерах»:

http://ru.wikibooks.org/wiki/Scala_%D0%B2_%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D1%80%D0%B0%D1%85

Пробуйте, исследуйте, экпериментируйте в REPL! При выполнении работ и для начального обучения языку Scala также рекомендуется использовать интернет-ресурсы и ссылки, приведенные на них:

http://www.rsdn.ru/article/scala/scala.xml

http://twitter.github.com/scala_school/ru/

Лабораторная работа 9

Общее описание задания и рекомендации (для всех вариантов)

Для каждого из заданий привести два решения на языке Scala: первое должно быть написано в императивном стиле, а второе — в функциональном стиле программирования. Решение в функциональном стиле пошагово проанализировать в интерактивном интерпретаторе Scala (REPL). В отчете привести скриншоты (или текст из REPL). Для второго задания подготовить не менее пяти исходных списков для его демонстрации и проверки и дать возможность выбора варианта списка в интерфейсе программы.
Программы - решения скомпилировать для jvm, в отчете привести примеры их запуска из командной строки и работы. Требование к оформлению кода: краткость, одинаковые отступы (рекомендуется два пробела), подробные рекомендации здесь:

http://twitter.github.com/effectivescala/index-ru.html#%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5

Не используйте код, найденный в интернет, который не понимаете - лучше придумайте свое решение. Решение должно быть логичным и понятным.

Варианты заданий

Вариант 1

1. f(m,n) - сумма всех натуральных чисел от m до n включительно, в десятичной записи которых нет одинаковых цифр.

2. Список содержит целые числа, а также другие списки, такие же как и первоначальный. Получить список, содержащий только целые числа из всех вложенных списков. Пример:

   f(List(List(1, 1), 2, List(3, List(5, 8)))) = List(1, 1, 2, 3, 5, 8)

Вариант 2

1. f(n) - сумма квадратов всех простых делителей натурального числа n.
2. Список содержит элементы одного, но любого типа. Среди элементов есть повторяющиеся. Получить список, содержащий каждый элемент только один раз (удалить все повторы).

Вариант 3

1. f(n) - сумма цифр наибольшего простого делителя натурального числа n.
2. Список содержит элементы одного, но любого типа. Получить список, содержащий каждый имеющийся элемент старого списка k раз подряд. Число k задается при выполнении программы.

Вариант 4

1. f(m,n) - наименьшее общее кратное натуральных чисел m и n.
2. Список содержит элементы одного, но любого типа. Получить список, из элементов исходного, удаляя каждый k-й элемент. Число k задается при выполнении программы.

Вариант 5

1. f(n,k) - число размещений из n по k. Факториал не использовать.
2. Список содержит элементы одного, но любого типа. Получить новый список, перемещая циклически каждый элемент на k позиций влево (при перемещении на одну позицию первый элемент становится последним, второй первым и так далее). Число k задается при выполнении программы. Если k отрицательное, то перемещение происходит вправо.

Вариант 6

1. f(n) - наибольшее совершенное число не превосходящее n. Совершенным называется натуральное число n равное сумме своих делителей, меньших n, например 6 = 1 + 2 + 3 (f(6) = 6, f(7) = 6, ...).
2. Список содержит элементы одного, но любого типа. Получить два списка из элементов исходного, выбирая в первый элементы с четными индексами, а во второй с нечетными.

Вариант 7

1. f(n) - количество точек с целыми координатами, находящихся в круге с центром в начале координат и радиусом n (n — натуральное число).
2. Список содержит элементы одного, но любого типа. Получить список, из элементов исходного, выбирав с m-го по n-й элемент (m<=n). Числа m и n задается при выполнении программы.

Вариант 8

1. f(n) - сумма всех четных членов ряда Фибоначчи, не превышающих n. Ряд Фибоначчи: 1,1,2,3,5,8,13,21,34,55,89,... (каждый член, начиная с третьего равен сумме двух предшествующих).
2. Элементами главного списка является другие списки. Отсортировать главный список, расположив внутренние списки по возрастанию количества их элементов.

Вариант 9

1. f(n,k) - количество чисел от 1 до n включительно, сумма квадратов цифр которых равна k.
2. Список содержит целые числа и строки. Получить два списка из элементов исходного, выбирая в первый числа, а во второй строки.

Вариант 10

1. f(n,d) - суммарное количество использований цифры d (0,1,2,3,4,5,6,7,8,9) для записи всех чисел от 1 до n включительно.
2. Список содержит элементы одного, но любого типа. Получить список, из элементов старого, выбрав из него случайно k элементов. Число k задается при выполнении программы.

Вариант 11

1. f(n) - наибольшее из чисел от 1 до n включительно, обладающее свойством: сумма цифр n в некоторой степени > 1 равна самому числу n. Пример: $512 = 8^3$
2. Список в качестве элементов содержит кортежи типа: (n, s), где n — целые числа, а s — строки. Получить два списка из элементов исходного, выбирая в первый числа, а во второй строки из кортежей.

Использование sbt и Intellij IDEA

Для выполнения следующей лабораторной работы нам нужно научиться пользоваться системой сборки sbt и средой разработки Intellij IDEA. Для создания каталога проекта и входа в него, выполним команды:

```
$ mkdir lab10
$ cd lab10
```

Теперь в этом каталоге нам нужно создать текстовый файл build.sbt, содержащий параметры проекта. Его можно создать в любом текстовом редакторе, но можно это сделать и при помощи самого sbt, последовательно выполнив команды:

```
$ sbt

> set name := "lab10"
> set version := "1.0"
> set scalaVersion := "2.10.0"
> session save
> exit
```

Проверим содержимое build.sbt:

```
$ cat build.sbt
name := "lab10"

version := "1.0"

scalaVersion := "2.10.0"
```

Именно так все и должно быть — пустые строки обязательны (мы могли бы создать этот файл сами в редакторе текстов). Заметим, что в нашем каталоге появились два подкаталога **project** и **target**. В подкаталоге project создадим еще два текстовых файла **build.properties** и **plugins.sbt** со следующим содержанием:

build.properties

```
sbt.version=0.12.1
```

plugins.sbt

```
resolvers += Resolver.url("artifactory", url("http://scalasbt.artifactoryonline.com/scalasbt/sbt-plugin-releases"))
(Resolver.ivyStylePatterns)

addSbtPlugin("com.github.mpeltonen" % "sbt-idea" % "1.2.0")
```

plugins.sbt содержит две непустые строки (пустая строка между ними обязательна). Он нужен для использования различных плагинов sbt. В данном случае мы подключаем плагин для формирования необходимых файлов для Intellij IDEA (чтобы наш проект можно было открывать в ней).

Все эти файлы можно было бы и не создавать, а скопировать из предыдущего проекта (если бы он был).

Нам нужно будет вручную создать структуру каталогов проекта для sbt:

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

Разумеется нам пока не все каталоги нужны, поэтому выполняем следующие команды:

```
$ mkdir src
$ mkdir src/main
$ mkdir src/main/scala
```

Когда у нас появятся тесты, мы создадим каталог для тестов (src/test/scala), если потребуется загружать данные из ресурсов создадим каталог для ресурсов (src/main/resources). На java мы писать ничего не собираемся поэтому каталоги для кода на java не создаем вообще. Впрочем, все что нужно можно будет добавить позднее.

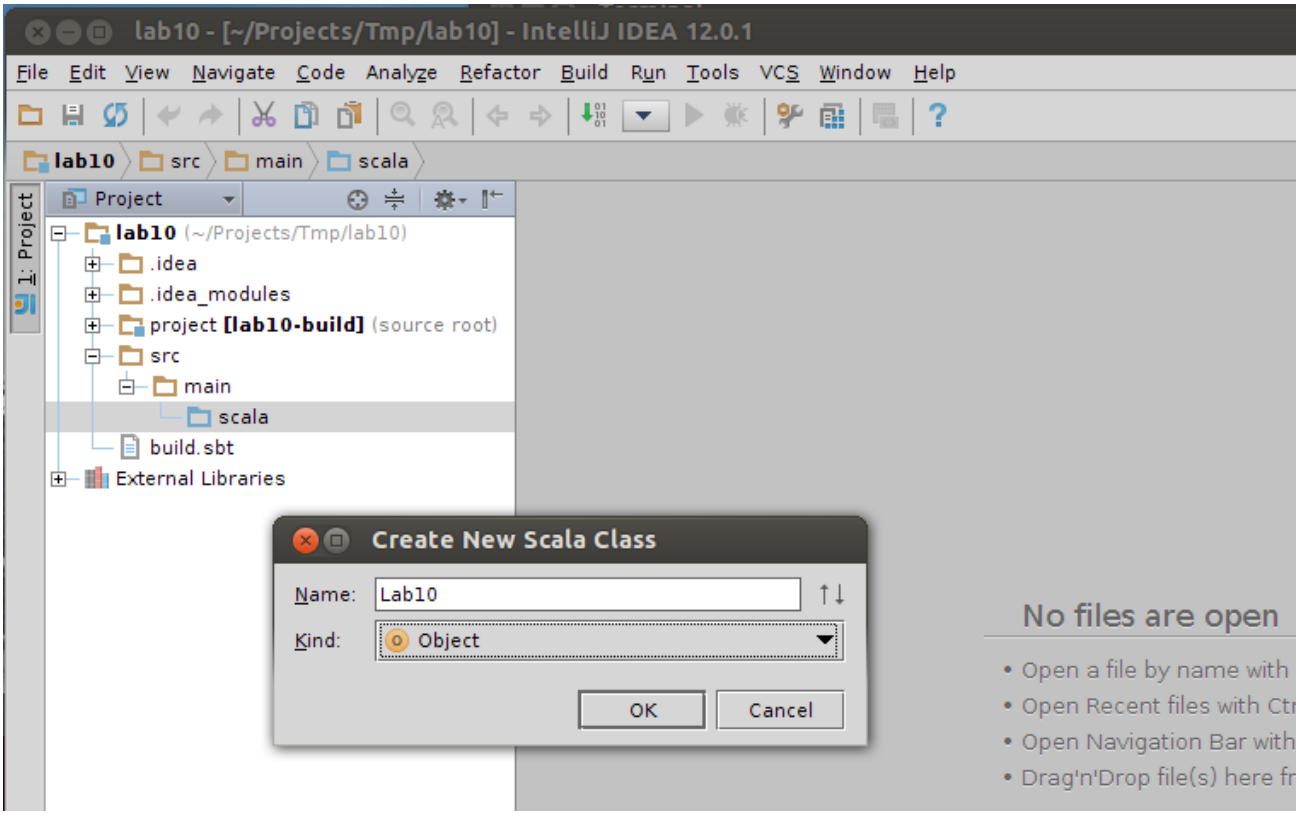
После этого, вызовем sbt и выполним команду gen-idea:

```
$ sbt

> gen-idea
```

Интернет должен быть доступен, так как sbt будет пытаться скачивать из репозиториев необходимые для проекта ресурсы. В результате выполнения команды будут сгенерированы файлы для открытия проекта в Intellij IDEA. После того как все завершится и мы снова увидим приглашение >, не выходя из sbt вызываем Intellij IDEA и открываем каталог нашего проекта в ней (в меню выбираем open и выбираем каталог lab10).

После открытия проекта, нажимаем на закладку Project (если она закрыта) и выбираем на ней каталог scala, а затем правой кнопкой мыши вызываем контекстное меню и выбираем new → Scala Class. В появившемся окне вводим Lab10 и выбираем из списка Object, а не Class.



Удаляем комментарии и вводим def main(args: Array[String]) = println("Привет из Scala!"):

```
object Lab10 {
  def main(args: Array[String]) = println("Привет из Scala!")
}
```

Сохраняем (Ctrl-S), переключаемся в окно консоли sbt и последовательно выполняем команды:

```
> compile
> run
```

Для того чтобы упаковать нашу программу в jar-архив нужно выполнить команду

```
> package
```

Архив lab10_2.10-1.0.jar создается в подкаталоге target/scala-2.10. Другие команды sbt приведены в документации на сайте проекта:

http://www.scala-sbt.org/release/docs/Getting-Started/Running.html

Там же можно найти команды для вызова команд из истории и другую полезную информацию. Таким образом, ввод текстов программ (а также создание, удаление, редактирование новых и старых файлов проекта, например, изменение версии в build.sbt или добавление плагина в project/plugins.sbt) происходит в Intellij IDEA, а компиляция и сборка проекта осуществляется при помощи команд sbt.

Объектно-ориентированное программирование в Scala

В Scala существуют объекты, классы и трэйты и отсутствуют интерфейсы (их заменяют трэйты). Мы можем объявить объект и это будет означать, что он будет создан без директивы new в единственном экземпляре и все его методы и поля будут существовать пока работает программа.

```
object Lab10 {
  def main(args: Array[String]) = println("Привет из Scala!")
}
```

Мы можем объявить класс, тогда он живет по обычным правилам (новые экземпляры создаются при помощи new), однако уже в объявлении мы можем передать параметры и тем самым определить его конструктор по-умолчанию:

```
scala> class Person(val name: String, val lastname: String)
defined class Person

scala> val person = new  Person("Вася","Иванов")
person: Person = Person@651293a0

scala> person.name
res6: String = Вася

scala>
```

Заметим, что если нам нужно изменять переменные, то мы должны использовать var вместо val. Как обычно, следует переопределить метод toString для изменения строки вывода объекта:

```
scala> class Person(val name: String, val lastname: String) {
    | override def toString = name+" "+lastname
    | }
defined class Person

scala> val person = new  Person("Вася","Иванов")
person: Person = Вася Иванов

scala>
```

Классы могут быть абстрактными, если в них есть нереализованные методы. Как обычно, это объявляется при помощи директивы abstract у класса (у метода ничего объявлять не нужно).

Трэйты могут содержать как нереализованные так и реализованные методы:

```
scala> trait Figure {
    |   def name = "фигура"
    |   def area: Double
    | }
defined trait Figure

scala>
```

Классы трэйты и объекты могут наследоваться от трэйтов и классов при помощи директивы extends (implements не используется). Так называемые кэйс-классы

предваряются директивой case и в них автоматически создаются: метод toString, геттеры (без определения val), хэши для сравнения объектов.

```
scala> case class Person(name: String, lastname: String)
defined class Person

scala> val person = Person("Вася", "Иванов")
person: Person = Person(Вася,Иванов)

scala> person.name
res10: String = Вася

scala>
```

Обратите внимание на то, что объекты таких классов создаются без слова new.

Лабораторная работа 10

Общее описание задания и рекомендации (для всех вариантов)

Для того чтобы получить представление о разработке на языке Scala предлагается реализовать на этом языке основной функционал лабораторной работы 2 из прошлого семестра (каждый выполняет свой вариант).

Задача здесь совсем не в том чтобы повторить свою работу, преобразуя код из одного языка в другой, а в том чтобы спроектировать все по-другому, упростить программу как можно сильнее, используя возможности языка Scala.

Вы можете свободно использовать трэйты, абстрактные классы или case-классы - в проектировании нет никаких ограничений. Главный критерий: краткость, простота и чтобы программа выполняла то, что она у Вас делала (или должна была делать) в java.

Разработку вести в Intellij IDEA, компиляцию и сборку выполнять в sbt. Как результат последнюю версию программы собрать при помощи плагина Assembly и продемонстрировать ее запуск в jvm независимо от Scala.

В отчете привести для иллюстрации скриншоты кода во время разработки, примеры запуска из командной строки и работы программы. Для всех более менее сложных для понимания функций привести объяснения того как они работают.

Демонстрационный пример

Продемонстрируем как это можно сделать на примере преобразования java кода, приведенного в методическом пособии по лабораторной работе 2.

Задача была следующая. Множества точек на плоскости: точка, отрезок, окружность, эллипс (сдвиг, растяжение).

Мы создадим всего два файла. Lab10.scala и curve.scala. Оба будут находиться в подкаталоге src/main/scala. Пакет назовем lab10 (в Scala имена пакетов не обязаны

совпадать с именем каталога). Итак, содержимое файла Lab10.scala:

```
package lab10

object Lab10 {

  val list = List(
    new Point(Point2D(3,5)),
    new Circle(Point2D(1,2),5),
    new Ellipse(Point2D(1,1),5,8),
    new Segment(Point2D(1,1),Point2D(5,8)),
    new Point(Point2D(2,2)),
    new Segment(Point2D(0,0),Point2D(3,5)),
    new Ellipse(Point2D(0,0),3,4),
    new Circle(Point2D(0,0),5),
    new Ellipse(Point2D(-1,2),5,3),
    new Circle(Point2D(1,1),4)
  )

  def main(args: Array[String]): Unit = {
    var key = 1
    while (key>0) {
      println
      println("Выберите тип сортировки (выйти - 0):")
      println("--------------------------------------------------")
      println("1 - по возрастанию id")
      println("2 - по убыванию id")
      println("3 - по возрастанию имени")
      println("4 - по убыванию имени")
      println("5 - по возрастанию удаленности от начала координат")
      println("6 - по убыванию удаленности от начала координат")
      println("--------------------------------------------------")
      try { key = readInt } catch {case _ : Throwable => key = 0}
      list.sortWith((a,b) => key match {
        case 2 => a.id>b.id
        case 3 => a.name<b.name
        case 4 => a.name>b.name
        case 5 => a.center.mod<b.center.mod
        case 6 => a.center.mod>b.center.mod
        case _ => a.id<b.id
      }).map(e => println(e))
    }
  }
}
```

В переменной list создается список тех же объектов, что были в лабораторной работе 2. Класс Point2D мы рассмотрим позднее. Здесь обратите внимание на сортировку.

Мы получаем число key (его вводит пользователь при выполнении key=readInt). У list используется метод sortWith, в который передается анонимная функция (a, b) => Boolean (она должна вернуть true, если a<b и сортируем по возрастанию). В выражении key match мы перебираем все случаи чему может быть равен key и возвращаем соответствующее логическое условие.

По умолчанию считаем key=1 (последний case), поэтому перебираем key = 2,3,4,5,6, а иначе считаем его единицей, что бы не ввели (если введут 0 или вообще не целое число > 0, то выход из цикла while).

После того как list отсортирован мы выводим все его элементы: map(e => println(e)).

Перейдем теперь к содержимому файла curve.scala:

```
package lab10

case class Point2D(var x: Double, var y: Double) {
  def sqr(x: Double) = x*x
  def mod = math.sqrt(sqr(x)+sqr(y))
  def +(p: Point2D) = Point2D(x+p.x,y+p.y)
  def -(p: Point2D) = Point2D(x-p.x,y-p.y)
  def *(k: Double) = Point2D(x*k,y*k)
  def /(k: Double) = Point2D(x/k,y/k)
  def shift(v: Point2D) = { x=x+v.x; y=y+v.y } // сдвиг точки на вектор v
  override def toString = "("+x+","+y+")"
}

trait Prototype {
  val id: Int
  def round(x: Double) = math.round(x*100)*0.01
  def name: String
  def center: Point2D
  def move(p: Point2D): Unit
  def resize(k: Double): Unit
}

class Point(var p: Point2D) extends Prototype{
  val id =  Curve.getId
  def distance(p1: Point2D) = (p-p1).mod
  override def name = "точка"
  override def center = p
  override def toString = "id="+id+", "+name+": "+p
  override def move(p: Point2D) = this.p = p
  override def resize(k: Double) = {}
}

class Segment(var p1: Point2D, var p2: Point2D) extends Prototype{
  val id =  Curve.getId
  def length = (p2-p1).mod
  override def name = "отрезок"
  override def center = (p2-p1)/2
  override def toString = "id="+id+", "+name+": "+p1+"-"+p2+" длина: "+round(length)
  override def move(p: Point2D) = { p1.shift(p-center); p2.shift(p-center) }
  override def resize(k: Double) = { p1 = center-(center-p1)*k; p2 = center+(p2-center)*k }
}

class Circle(var p: Point2D, var r: Double) extends Prototype{
  val id =  Curve.getId
  def area = math.Pi*r*r
  def length = 2*math.Pi*r
  def in(p: Point2D) = (this.p-p).mod<r
  override def name = "окружность"
  override def center = p
  override def toString = "id="+id+", "+name+": центр - "+p+", радиус - "+r+", длина окружности: "+
                    round(length)+" площадь: "+round(area)
  override def move(p: Point2D) = { this.p = p }
  override def resize(k: Double) = r=r*k
}
```

```
class Ellipse(var p: Point2D, var a: Double, var b: Double) extends Prototype{
  val id =  Curve.getId
  def area = math.Pi*a*b
  override def name = "эллипс"
  override def center = p
  override def toString = "id="+id+", "+name+": центр - "+p+", полуоси - "+a+","+b+", площадь: "+round(area)
  override def move(p: Point2D) = { this.p = p }
  override def resize(k: Double) = { a=a*k; b=b*k }
}

object Curve {
  var id = 0
  def getId = {id+=1; id}
}
```

Первый case-класс Point2D описывает вспомогательный объект, содержащий две вещественные координаты (означающий по смыслу двумерную точку или вектор). Он нужен для того чтобы упростить описание и реализацию других классов, поэтому мы наполняем его такими полезными операциями как сложение и вычитание векторов (+ и -) и умножение и деление вектора на число (* и /), вычисления модуля (mod).

Как только мы определили методы + и - мы можем для двух точек p1 и p2 всюду писать p1 + p2 вместо p1.+(p2) и p1 - p2 вместо p1.-(p2). Таково соглашение языка Scala. Учитывая возможность деления на число, можно получить точку — середину отрезка p1 и p2 следующим образом: (p1 + p2)/2

Определив функцию модуля вектора (mod = корень квадратный из суммы квадратов координат), мы можем выразить, например, расстояние между двумя точками p1 и p2 как (p1-p2).mod или (p2-p1).mod.

Все это упрощает запись выражений с точками и способствует лучшему пониманию программы. Еще один метод shift для сдвига точки p на вектор v, позволяет упростить, например, запись метода параллельного переноса отрезка.

Разрабатывая другие классы, можно добавлять нужные методы во вспомогательный класс по мере появления необходимости в них.

Поскольку разные кривые в задании значительно отличаются друг от друга, было принято решение наследовать их не друг от друга, а от одного и того же прототипа, в котором, следует описать все то общее, что им всем присуще.

Так появился trait Prototype, в котором одна нереализованная переменная id, четыре нереализованных метода (name, center, move и resize) и один реализованный вспомогательный метод round для округления вещественных чисел до двух знаков после запятой.

Дальнейшее уже несложно. Все классы задания наследуются от Prototype, в каждом из них реализуются по-своему методы объявленные, но нереализованные (абстрактные) у Prototype и добавляются другие методы — для каждого класса свои.

Для получения id создается объект Curve, который с каждым запросом увеличивает значение id на единицу и все объекты при своем создании запрашивают свой id у Curve (метод getId). В списке list все объекты рассматриваются как Prototype.

Использование плагина Assembly

Зависимость при запуске программы от библиотеки Scala можно решить ее копированием в каталог проекта и написанием shell-скрипта с указанием classpath.

Однако кроме Scala в проекте могут использоваться и другие библиотеки (swing, например), поэтому хотелось бы по окончанию разработки получить один jar файл с программой, в который вошли бы все библиотеки, используемые в проекте.

Именно это делает плагин Assembly для системы сборки sbt. Для того чтобы его добавить в проект нужно отредактировать два файла: project/plugins.sbt и build.sbt, приведя их соответственно к виду:

plugins.sbt (добавляем пустую и еще одну строку)

```
resolvers += Resolver.url("artifactory", url("http://scalasbt.artifactoryonline.com/scalasbt/sbt-plugin-releases"))
(Resolver.ivyStylePatterns)


addSbtPlugin("com.github.mpeltonen" % "sbt-idea" % "1.2.0")



addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.8.5")
```

build.sbt (добавляем первую + пустую в начало и пустую + последнюю в конец)

```
import AssemblyKeys._

name := "lab10"

version := "1.0"

scalaVersion := "2.10.0"

assemblySettings
```

Если мы уже находимся в sbt, то вызываем команду reload, затем compile и наконец, assembly:

```
> reload
[info] Loading project definition from /home/george/Projects/Tmp/lab10/project
[info] Set current project to lab10 (in build file:/home/george/Projects/Tmp/lab10/)
> compile
[success] Total time: 0 s, completed 11.01.2013 17:26:56
> assembly
[info] No tests to run for test:test
[info] Including scala-library.jar
[info] Merging 'META-INF/MANIFEST.MF' with strategy 'discard'
[info] SHA-1: WrappedArray(102, -70, -110, 72, -93, -35, 28, 76, -128, 123, -33, 122, 72, -109, 109, -68, 100,
-101, -6, 84)
[info] Packaging /home/george/Projects/Tmp/lab10/target/lab10-assembly-1.0.jar ...
[info] Done packaging.
[success] Total time: 2 s, completed 11.01.2013 17:27:03
>
```

Как видно из сообщений результат сборки находится в target/lab10-assembly-1.0.jar

Вопросы и задания для самопроверки и к защите работ

1. В Scala REPL, не создавая новых переменных, вычислить квадратный корень из 3 и полученный результат возвести в квадрат. На сколько окончательный результат отличается от 3? (подсказка: используйте переменные res).

2. Как заданы переменные res: как val или как var?

3. Что означает 10 max 2? В каком классе определен метод max?

4. Используя BigInt, вычислите 2^{1024}.

5. Как можно получить первый символ строки в Scala? А последний символ?

6. Как работают функции для строк drop, takeRight, dropRight. В чем преимущества и недостатки по отношению к substring?

7. Напишите функцию signum(x): -1 если x<0, 1 если x>0, 0 если x=0.

8. Какое значение и какого типа возвращает пустой блок {} ?

9. Напишите в Scala цикл аналогичный следующему в java:

```
for (int i = 10; i >= 0; i--) System.out.println(i);
```

10. Напишите рекурсивную функцию для вычисления x^n по следующему алгоритму:

$x^n = (x^{(n/2)})^2$, если n четное и положительное

$x^n = x*x^{n-1}$, если n нечетное и положительное

$x^0 = 1$

$x^n = 1/x^{-n}$, если n отрицательное

11. Напишите цикл, разбивающий массив на пары и меняющий соседние элементы в каждой паре. Например, Array(1, 2, 3, 4, 5) преобразуется в Array(2, 1, 4, 3, 5).

12. Дан массив целых чисел. Получить новый массив, содержащий сначала отрицательные элементы и нуль в том порядке как они шли в первоначальном массиве, а затем положительные элементы в том порядке как они шли в первоначальном массиве.

13. Как вычислить среднее значение элементов Array[Double] ?

14. Напишите функцию minmax(values: Array[Int]), которая возвращает пару, содержащую наименьшее и наибольшее значение массива.

15. Напишите функцию lteqgt(values: Array[Int], v: Int), которая возвращает triple, содержащий число значений массива меньше чем v, равное v и больше чем v.

16. Что получится в результате выполнения функции: "Hello".zip("World") ?