Современные технологии программирования
Лабораторная работа 12

Разработка приложений с оконным графическим интерфейсом на языке Scala. Цель: научиться создавать GUI приложения на Scala.

При использовании sbt есть следующие возможности интегрировать уже имеющийся код на Scala в приложение с оконным интерфейсом.

1. Приложение с оконным интерфейсом (основной проект) создавать на java независимо от проекта sbt. Код на Scala в проекте sbt собрать в jar-библиотеку (плагин assembly), которую подключить обычным образом к основному проекту (так было в лабораторной работе 11).
2. Приложение с оконным интерфейсом (основной проект) создавать на java внутри проекта sbt (используя каталог src/main/java). В этом случае sbt будет компилировать и собирать весь код как на Java так и на Scala.
3. Всё приложение, включая оконный интерфейс, создавать на scala в проекте sbt, но использовать в scala обычные java-классы библиотеки Swing.
4. То же что и в пункте 3, только для создания GUI использовать пакет scala.swing.

Самый интересный вариант, естественно, последний, так как только при нем мы, используя возможности языка Scala, можем упростить создание оконного интерфейса на основе библиотеки Swing.

Для того чтобы продемонстрировать отличия 3 и 4 пунктов приведем пример использования внутри scala обычных java-классов библиотеки Swing.

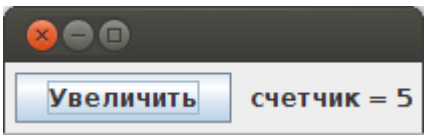
```
import javax.swing._
import java.awt.BorderLayout
import java.awt.event._

object GUI {
  var counter = 1
  val button = new JButton("Увеличить")
  val label = new JLabel(" счетчик = 0")

  def makeAction(action: (ActionEvent) => Unit) =
    new ActionListener {
      override def actionPerformed(event: ActionEvent) { action(event) }
    }
  button.addActionListener(makeAction((event: ActionEvent) => counter += 1))
  button.addActionListener(makeAction((event: ActionEvent) => label.setText(" счетчик = "+counter)))
  button.addActionListener(makeAction((event: ActionEvent) => if (counter > 9) sys.exit(0)))

  val frame = new JFrame
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
  val panel = new JPanel
  panel.add(button)
  panel.add(label)
  frame.add(panel, BorderLayout.EAST)
  frame.pack
  def main(args: Array[String]) = frame.setVisible(true)
}
```

Хотя в этом примере мы и используем некоторые возможности Scala (добавляем обработчики нажатия кнопки button как анонимные функции), однако, в остальном, используются обычные java-классы компонент библиотеки Swing. Все методы этих классов доступны внутри Scala - их можно вызывать так, как если бы мы писали программу на java.

Сохранив код данного примера в файл gui.scala, мы можем вызвать приложение командой:

```
$ scala gui.scala
```

или же сначала скомпилировать его при помощи scalac.



Приведем теперь код этого же приложения на языке Scala с использованием пакета scala.swing.

```
import scala.swing._

val frame = new MainFrame {
  var counter = 0
  val label = new Label(" счетчик = 0")
  contents = new FlowPanel {
    contents += new Button(Action("Увеличить"){
      counter += 1
      if (counter > 9) sys.exit(0) else label.text = " счетчик = "+counter
    })
    contents += label
  }
}

frame.visible = true
```

Если сохранить код данного примера в файл gui-swing.scala, то мы сможем вызвать приложение командой:

```
$ scala gui-swing.scala
```

Работает оно точно также как и предыдущее. Отметим, что теперь в коде отсутствует метод main. Однако скомпилировать код командой

```
$ scalac gui-swing.scala
```

нам не удастся как раз из-за отсутствия объекта и метода main в нем. Если же заменить последнюю строку на

```
  def main(args: Array[String]) = frame.visible = true
```

и поместить весь код внутрь объекта, то компиляция пройдет успешно.

Из приведенного кода видно, что в scala.swing разработаны специальные классы и специальный синтаксис для использования компонент библиотеки Swing. MainFrame - класс главного окна, внутри которого JFrame. contents - это панель содержимого, куда добавляются другие компоненты, Label, Button - классы визуальных компонент (внутри них JLabel и JButton), Action - класс обработчика события, FlowPanel - менеджер размещения.

У каждого класса существует множество дополнительных полезных полей и методов, например у MainFrame есть title, size, centerOnScreen, у Button есть icon и так далее (подробнее см. в API-документации по пакету scala.swing). Добавим в код эти параметры и приведем его к виду:

```
import scala.swing._
import javax.swing.ImageIcon

object Counter {
  val frame = new MainFrame {
    title = "Счетчик до 9"
    var counter = 0
    val label = new Label(" счетчик = 0")
    val button = new Button(Action("Вычислить") {
      counter += 1
      if (counter > 9) sys.exit(0) else label.text = " счетчик = "+counter
    })
    button.icon = new ImageIcon(getClass.getResource("/play.png"))
    contents = new FlowPanel {
      contents += button
      contents += label
    }
    size = new Dimension(300,80)
    centerOnScreen
  }
  def main(args: Array[String]) = frame.visible = true
}
```

Сохраним изменения в файл counter.scala и перенесем его каталог src/main/scala вновь созданного проекта sbt, а в каталог src/main/resources скопируем файл пиктограммы play.png. Файлы build.sbt, plugins.sbt и build.properties можно скопировать из проекта лабораторной работы 11. При этом, файл build.sbt отредактируем и приведем к виду:

```
import AssemblyKeys._

name := "lab12"

version := "1.0"

scalaVersion := "2.10.0"

libraryDependencies += "org.scala-lang" % "scala-swing" % "2.10.0"

assemblySettings
```

Если все сделали правильно, то после запуска sbt и удовлетворения всех зависимостей, команды compile и run будут компилировать проект и запускать его на выполнение.

При закрытии окна программы в sbt могут появляться сообщения об ошибках. Не стоит обращать на них внимание, так как после сборки программы плагином assembly при ее работе в операционной системе никаких ошибок уже не будет.

Приведем еще примеры использования scala.swing. В следующем примере демонстрируется использование меню, чтение текста из файла, редактирование и сохранение текста.

```
import scala.swing._

object Menu {
 val ta = new TextArea
 def openFile = {
   val ch = new FileChooser
   if (ch.showOpenDialog(null)==FileChooser.Result.Approve) {
     val source = scala.io.Source.fromFile(ch.selectedFile)
     ta.text = source.mkString
     source.close
   }
 }
 def saveFile = {
   val ch = new FileChooser
   if (ch.showSaveDialog(null)==FileChooser.Result.Approve) {
     val pw = new java.io.PrintWriter(ch.selectedFile)
     pw.print(ta.text)
     pw.close
   }
 }

 val frame = new MainFrame {
   title = "Окно приложения"
   contents = ta
   menuBar = new MenuBar {
     contents += new Menu("Файл") {
       contents += new MenuItem(Action("Открыть") {
         openFile
       })
       contents += new MenuItem(Action("Сохранить") {
         saveFile
       })
       contents += new Separator
       contents += new MenuItem(Action("Выйти") {
         sys.exit(0)
       })
     }
   }
   size = new Dimension(500,500)
   centerOnScreen
 }

 def main(args: Array[String]) = frame.visible = true
}
```

Еще один пример демонстрирует использование GridPanel, BorderPanel и BoxPanel:

```
import scala.swing._

object Layouts {

  val ta = new TextArea
  ta.border = Swing.EmptyBorder(10, 10, 10, 10)
  val frame = new MainFrame {
    title = "Окно приложения"
    import BorderPanel.Position._
    contents = new BorderPanel {
      layout += new GridPanel(2,2) {
        contents += new Label("Текст ")
        contents += new Button("Кнопка")
        contents += new TextField
        contents += new ComboBox(List("Это", "просто", "тест"))
      } -> East
      layout += new BoxPanel(Orientation.Horizontal) {
        contents += new Label("Текст ")
        contents += new Button("Кнопка")
        contents += new TextField
        contents += new ComboBox(List("Это", "просто", "тест"))
      } -> North
      layout += ta -> Center
    }
    size = new Dimension(500,200)
    centerOnScreen
  }

  def main(args: Array[String]) = frame.visible = true
}
```

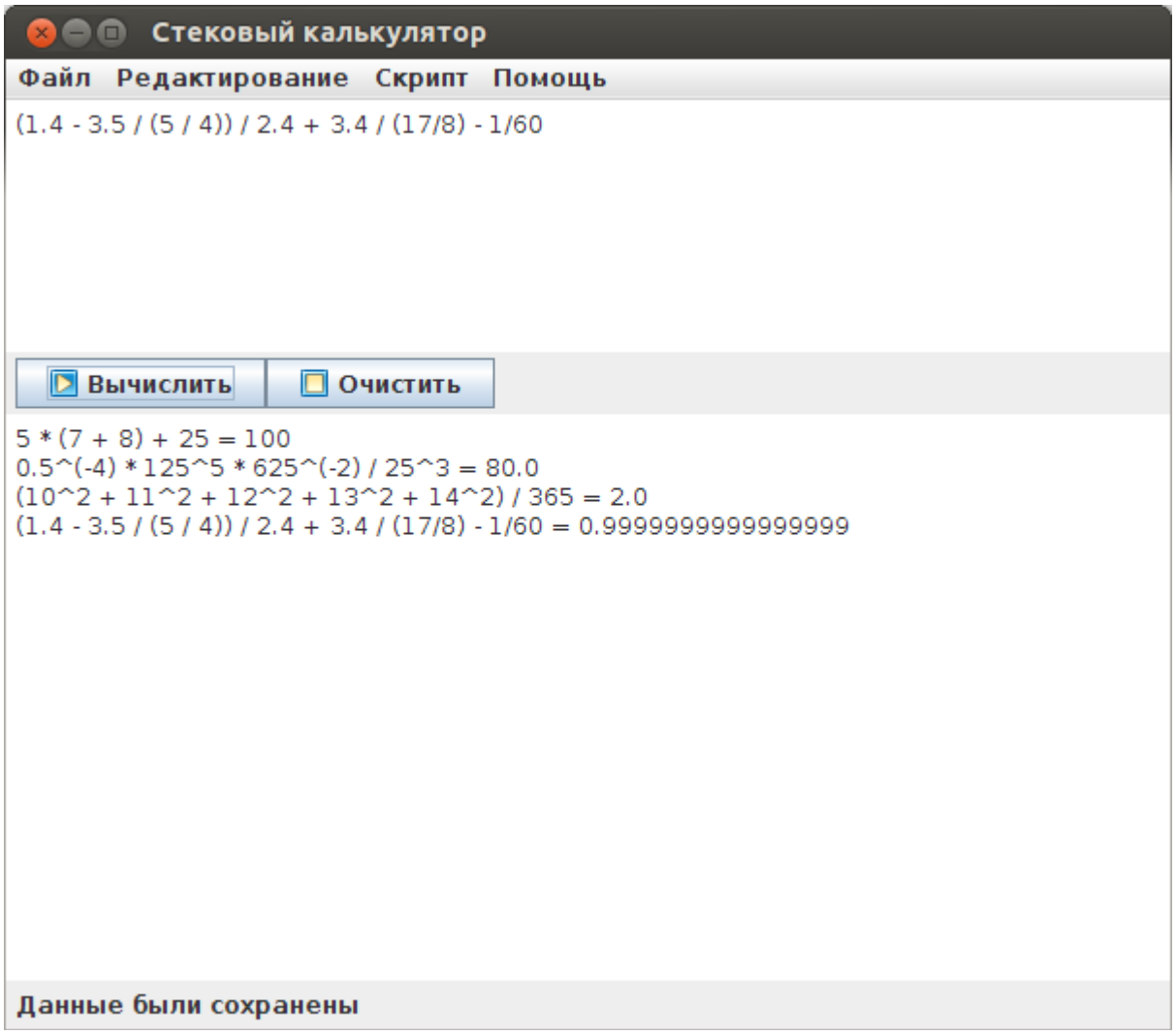
Варианты заданий лабораторной работы

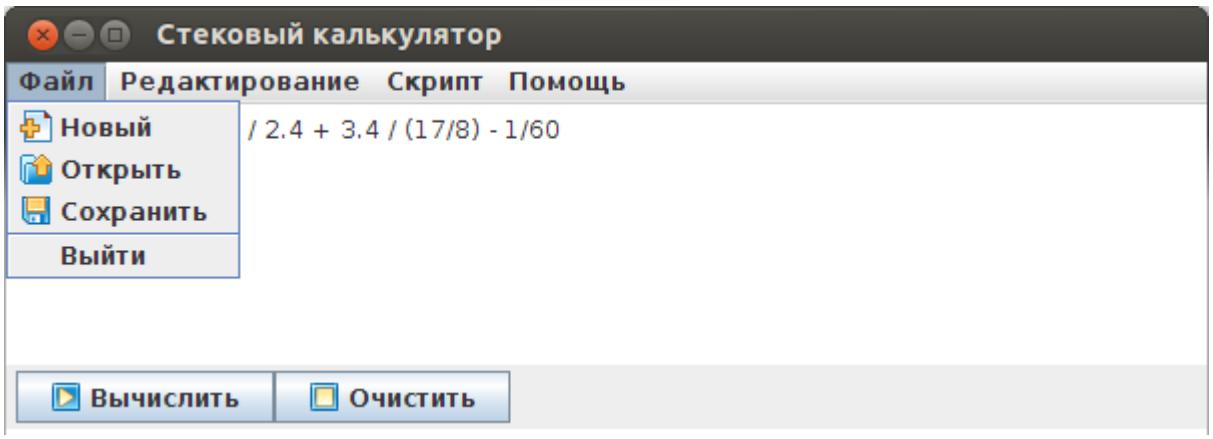
Общая часть задания для всех вариантов.

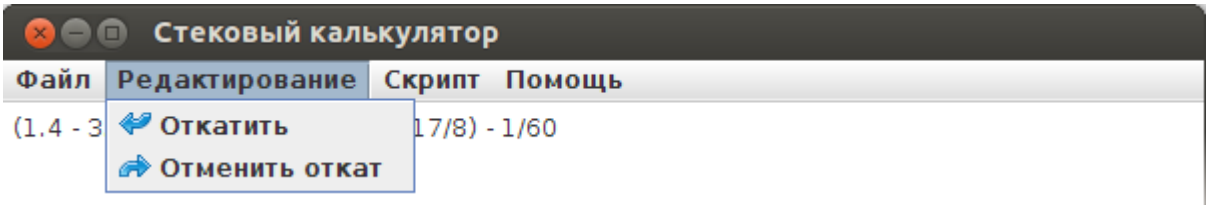
За основу берется проект лабораторной работы 11. Изменения внесенные в Calc.scala должны соответствовать варианту работы 11. Реализованный ранее на java в лабораторной работе 11 оконный интерфейс, следует переписать на scala с использованием пакета scala.swing.

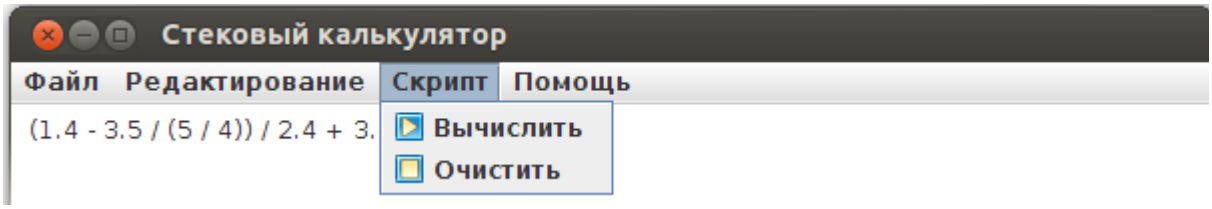
Вся программа, включая оконный интерфейс, должна быть написана только на scala. Она должна компилироваться и собираться в одном проекте sbt.

В оконный интерфейс программы следует добавить главное меню, статус панель и реализовать все добавленные в главном меню функции. Каким должен в итоге получиться интерфейс программы показано на скриншотах ниже.


Стековый калькулятор
Файл Редактирование Скрипт Помощь
(1.4 - 3.5 / (5 / 4)) / 2.4 + 3.4 / (17/8) - 1/60
Вычислить
Очистить
5 * (7 + 8) + 25 = 100
0.5^(-4) * 125^5 * 625^(-2) / 25^3 = 80.0
(10^2 + 11^2 + 12^2 + 13^2 + 14^2) / 365 = 2.0
(1.4 - 3.5 / (5 / 4)) / 2.4 + 3.4 / (17/8) - 1/60 = 0.9999999999999999
Данные были сохранены


Стековый калькулятор
Файл Редактирование Скрипт Помощь
Новый
Открыть
Сохранить
Выйти
/ 2.4 + 3.4 / (17/8) - 1/60
Вычислить
Очистить





Описание функций главного меню.

Меню Файл.
"Новый" — очищает верхнее окно
"Открыть" — загружает текст верхнего и нижнего окна из файла
"Сохранить" — сохраняет текст верхнего и нижнего окна в файл
"Выйти" — выход из программы

Оба текста сохраняются в одном файле. Автор должен придумать как это реализовать. Информацию об ошибках, удачном сохранении и так далее выводить на статус-панели.

Меню Редактирование.
"Откатить" — возвращает в верхнее окно предыдущий текст (тот что был до последнего изменения)
"Отменить откат" — возвращает в верхнее окно текст, который был перед откатом

Последнее изменение текста скрипта можно откатить (затем предпоследнее и так далее), а также отменить последний откат (неограниченное число раз). Для реализации этого рекомендуется использовать два стека: перед действиями изменяющими текст его следует помещать в стек undoStack.push(текст), а при откате текущий текст помещается в другой стек redoStack, а предпоследний выбирается из первого стека undoStack.pop и так далее.

Меню Скрипт.
Функции меню дублируются кнопками.
"Вычислить" — вычисляет выражение и добавляет его в нижнее окно
"Очистить" — очищает нижнее окно

Если произошли ошибки при вычислении выражения, то информация об ошибке выводится на статус-панели и в нижнее окно ничего не добавляется.

Меню Помощь.
Информация об авторе и программе (название, версия и тп).