



# Основы программирования на Java SE 11

Практические работы



## Оглавление

---

Оглавление .....	3
Упражнение 1. Компиляция и запуск Java приложений .....	5
Раздел 1. Использование SDK.....	6
Раздел 2. Использование среды разработки .....	9
Упражнение 2. Написание Java кода с использованием среды разработки .....	15
Раздел 1. Использование cheat sheet.....	16
Раздел 2. Использование scrapbook.....	21
Упражнение 3. Создание классов.....	34
Раздел 1. Создание класса OnlineMedia .....	35
Раздел 2. Создание класса DigitalVideoDisc.....	37
Раздел 3. Создание класса Order .....	43
Раздел 4. Создание класса DataFromProperties для считывания данных об экземплярах класса DigitalVideoDisc.....	47
Раздел 5. Реализация основной логики приложения OnlineMedia. ....	50
Раздел 6. Импорт в проект файла параметров и запуск приложения.	51
Упражнение 4. Отладка Java приложений .....	55
Раздел 1. Создание логики приложения .....	56
Раздел 2. Использование отладчика .....	63
Раздел 3. Пошаговый проход по коду.....	66
Раздел 4. Отладка условия выхода .....	71
Упражнение 5. Наследование и рефакторинг .....	81
Раздел 1. Создание класса OnlineMedia .....	82
Раздел 2. Создание класса Media.....	87
Раздел 3. Удаление лишних полей и методов классов Book и DigitalVideoDisc.....	89
Раздел 4. Обновление существующих классов для работы с новой моделью классов .....	92
Упражнение 6. Коллекции и дженерики .....	96
Раздел 1. Создание класса Track .....	97

Раздел 2. Создание класса CompactDisc .....	99
Раздел 3. Обновление существующих классов .....	102
Раздел 4. Джениерики: создание класса Library .....	104
Раздел 5. Адаптация классов под новую модель с использованием дженериков.....	106
Упражнение 7. Интерфейсы и сортировка.....	109
Раздел 1. Создание интерфейса Playable.....	110
Раздел 2. Реализация интерфейса Playable .....	112
Раздел 3. Реализация интерфейса Comparable .....	114
Упражнение 8. Потоки .....	117
Раздел 1. Создание класса MemoryDaemon .....	118
Упражнение 9. Исключения.....	122
Раздел 1. Создание класса PlayerException.....	123
Раздел 2. Реализация логики генерации и обработки исключений ...	124
Упражнение 10. JavaBeans .....	128
Раздел 1. Создание связанного свойства cost.....	129
Раздел 2. Создание свойства с ограничением category .....	133
Упражнение 11. Сериализация .....	139
Раздел 1. Объявление классов сериализуемыми .....	140
Раздел 2. Создание класса для сериализации и его тестирование ...	143
Раздел 3. Использование нового API для ввода/вывода .....	146
Упражнение 12. JUnit.....	148
Раздел 1. Генерация документации.....	149
Раздел 2. Создание тест кейсов .....	152
Раздел 3. Создание тест кейсов для обработки исключительных ситуаций .....	158
Раздел 4. Запуск тестов .....	161
Упражнение 13. Лямбда-выражения .....	167
Раздел 1. Создание класса с лямбда-выражениями.....	168

## Упражнение 1. Компиляция и запуск Java приложений

---

### О чём это упражнение:

В этой лабораторной работе вы научитесь компилировать и запускать Java приложения. Эта задача будет решаться с помощью инструментов командной строки и среды разработки.

### Что вы должны будете сделать:

- Использовать программы javac и java для компиляции и запуска приложения
- Использовать проекцию Java в Eclipse
- Запускать приложение из Eclipse

## Раздел 1. Использование SDK

В этой части упражнения вы используете Java SDK, предварительно установленный на виртуальной машине, для компиляции и запуска приложения.

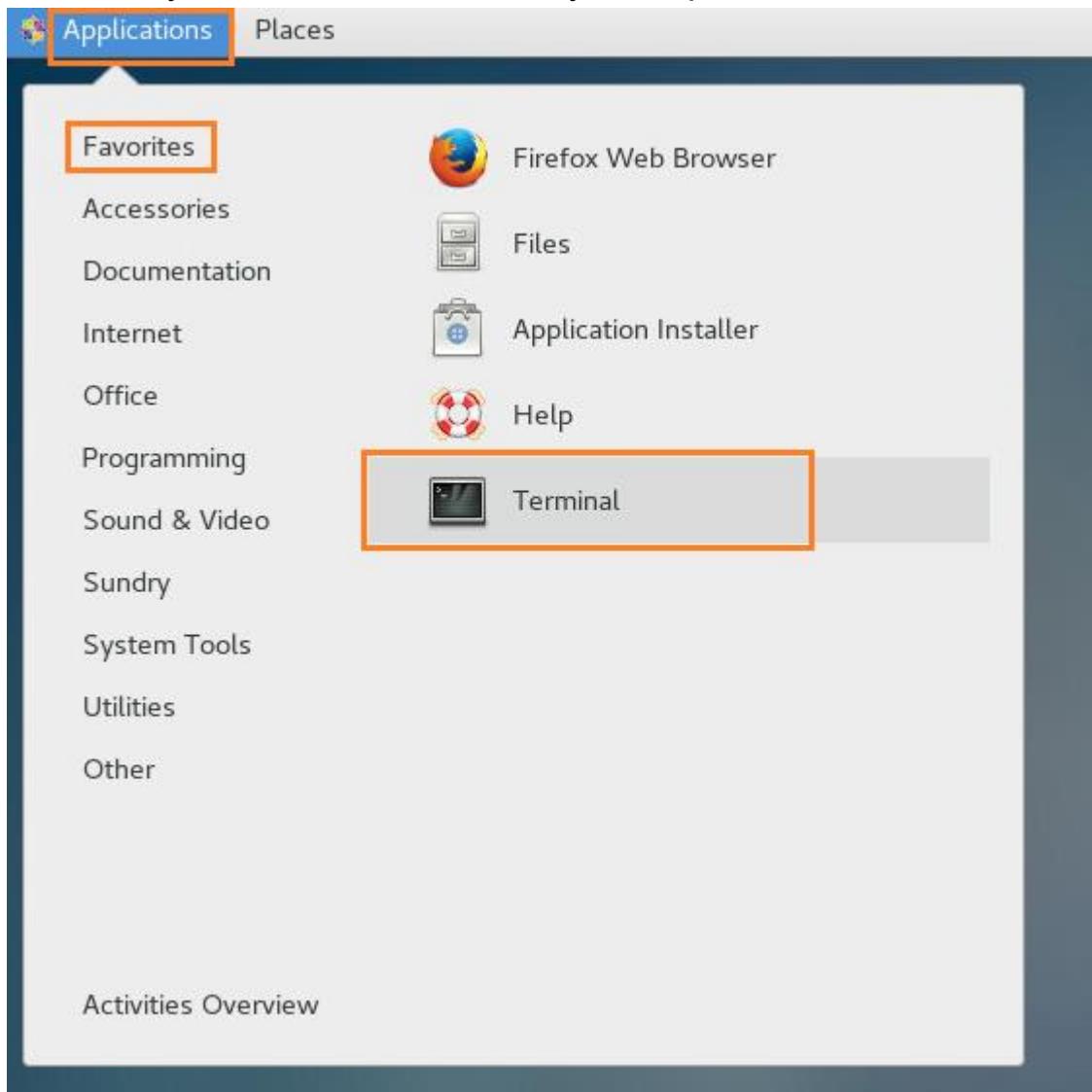
1. Запустите терминал для работы с интерпретатором и компилятором Java: **java** и **javac**.

- a. Реквизиты для доступа к виртуальной машине:

Учетная запись: **root**

Пароль: **web1sphere**

- b. Воспользуйтесь ссылкой для запуска терминала в меню: **Terminal**



2. Проверьте наличие необходимых файлов

- a. Запустите без аргументов:

**java**

Убедитесь, что запустился исполняемый файл **java**. В этом случае

в терминале появляется руководство по аргументам для запуска. Если интерпретатор возвращает сообщение об ошибке в связи с невозможностью найти соответствующий файл, проверьте отсутствие ошибки при вводе команды или обратитесь к инструктору.

Это необходимо, чтобы убедиться в возможности вызова java из произвольного каталога. Это возможно благодаря тому, что при установке java в Linux, исполняемые файлы java попадают в каталоги, прописанные в переменной PATH. Исполняемые файлы из этих каталогов можно запускать без указания полного пути до файла. Значение переменной можно посмотреть, вызвав команду:

**echo \$PATH**

Помимо способа, показанного выше, подсказку по все аргументам для запуска java можно получить, выполнив команду:

**java -h**

b. Запустите компилятор java без аргументов:

**javac**

Это позволит оценить возможность вызова компилятора из произвольного места файловой системы. По умолчанию команда возвращает список аргументов для запуска.

c. Узнайте версию компилятора:

**javac -version**

3. Скомпилируйте класс **DeveloperChoice**.

a. В терминале перейдите в каталог

**/root/labfiles/SimpleJava/mypackage** и посмотрите список файлов в нем:

**cd /root/labfiles/SimpleJava/mypackage**

**ls**

b. С помощью текстового редактора **gedit** откройте файл

**DeveloperChoice.java** из этого каталога:

**gedit DeveloperChoice.java**

c. После того, как изучите логику этого приложения, закройте текстовый редактор **gedit**.

d. Скомпилируйте файл:

**javac DeveloperChoice.java**

e. Еще раз посмотрите список файлов в каталоге:

**ls**

Убедитесь, что появился новый файл **DeveloperChoice.class**.

- f. Запустите скомпилированный файл:

```
java –classpath /root/labfiles/SimpleJava/  
mypackage.DeveloperChoice
```

Аргумент **–classpath** нужен для указания пути до класса.

**mypackage.DeveloperChoice** – полное название класса класса с учетом пакета.

- g. Введите **1** и нажмите **Enter**. Это завершит исполнение программы.

- h. Откройте исходный файл **DeveloperChoice.java** с помощью

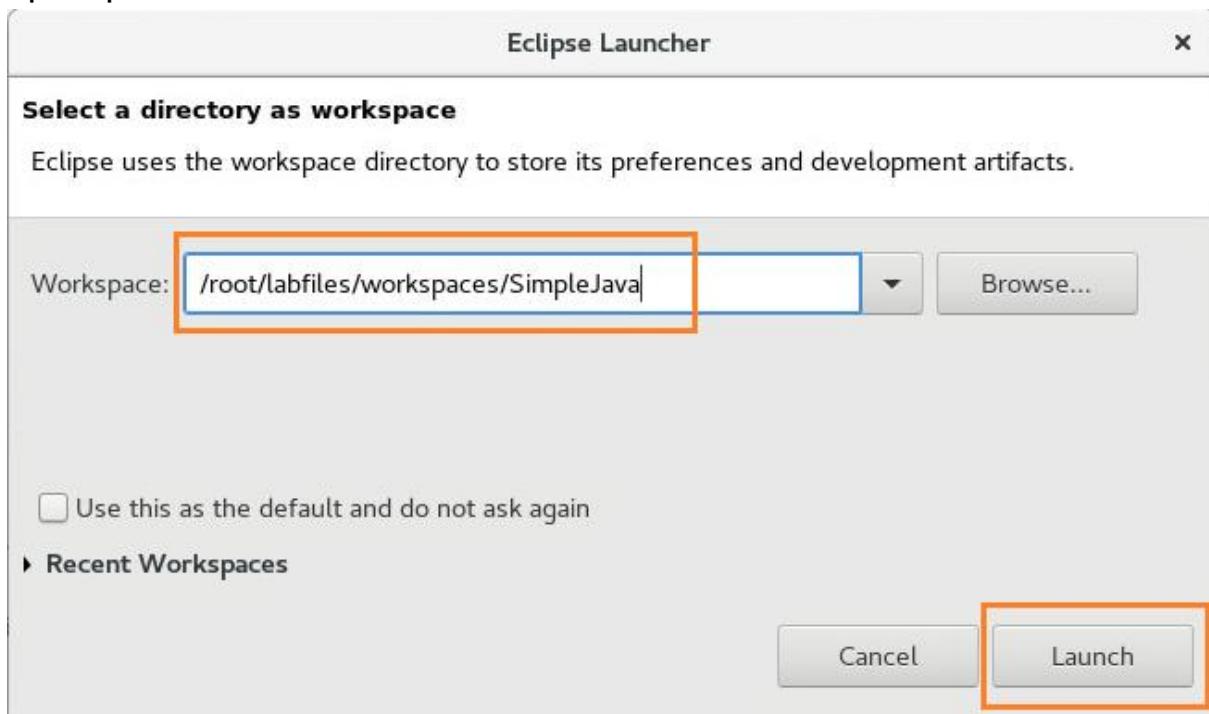
текстового редактора **gedit** и скорректируйте код программы для изменения выводимых сообщений.

- i. Перекомпилируйте класс и перезапустите его, чтобы убедиться в корректности сделанных изменений.

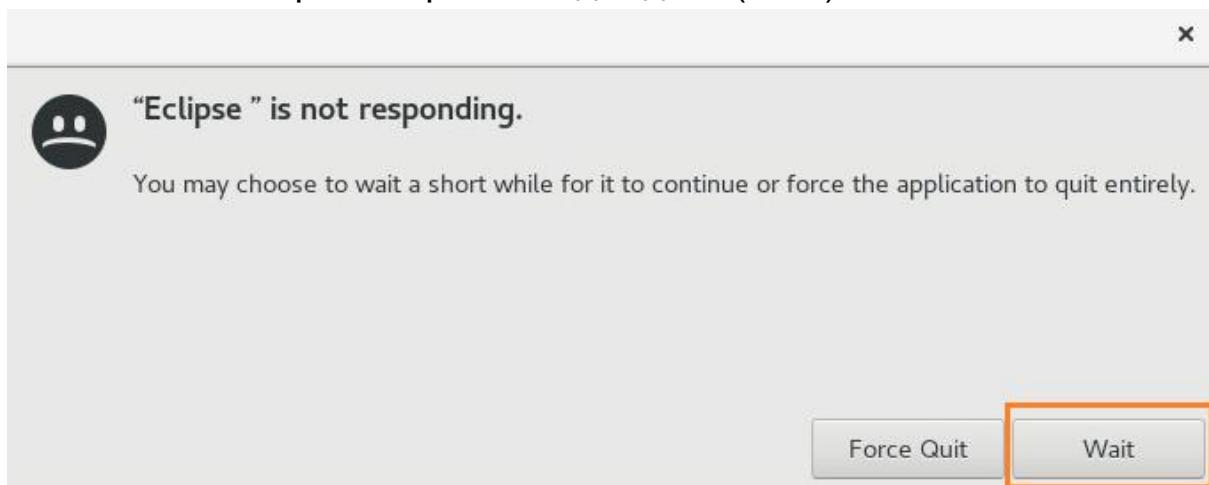
## Раздел 2. Использование среды разработки

В этой части упражнения вы используете среду разработки Eclipse для решения аналогичной задачи.

1. Запустите **Eclipse** с помощью ярлыка на рабочем столе виртуальной машины.
2. После запуска укажите/введите каталог **/root/labfiles/workspaces/SimpleJava** в качестве рабочего пространства и нажмите **Launch**.



В случае появления здесь и далее сообщения о том, что Eclipse не отвечает – выберите вариант подождать (**Wait**).

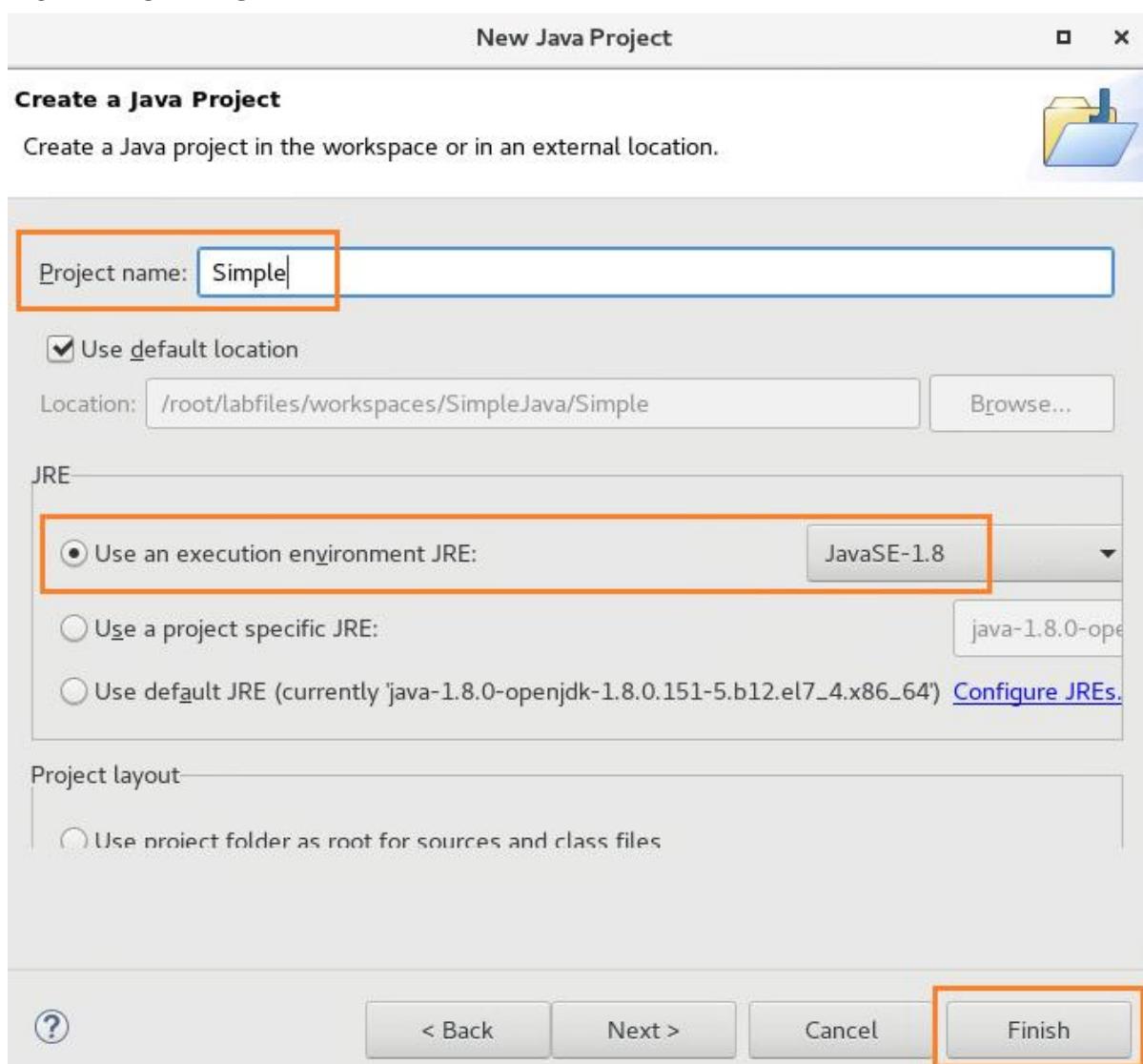


3. После открытия рабочей области **закройте** страницу **Welcome**, нажав по символу X у соответствующей вкладки:



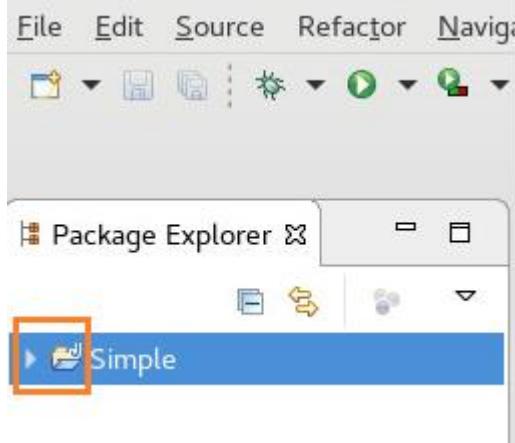
4. Создайте проект **Simple**

- Перейдите в меню: **File -> New -> Other**, далее выберите **Java Project**. Нажмите **Next**.
- Укажите название **Simple**, значения остальных полей оставьте заполненными по умолчанию.
- Нажмите **Finish**.



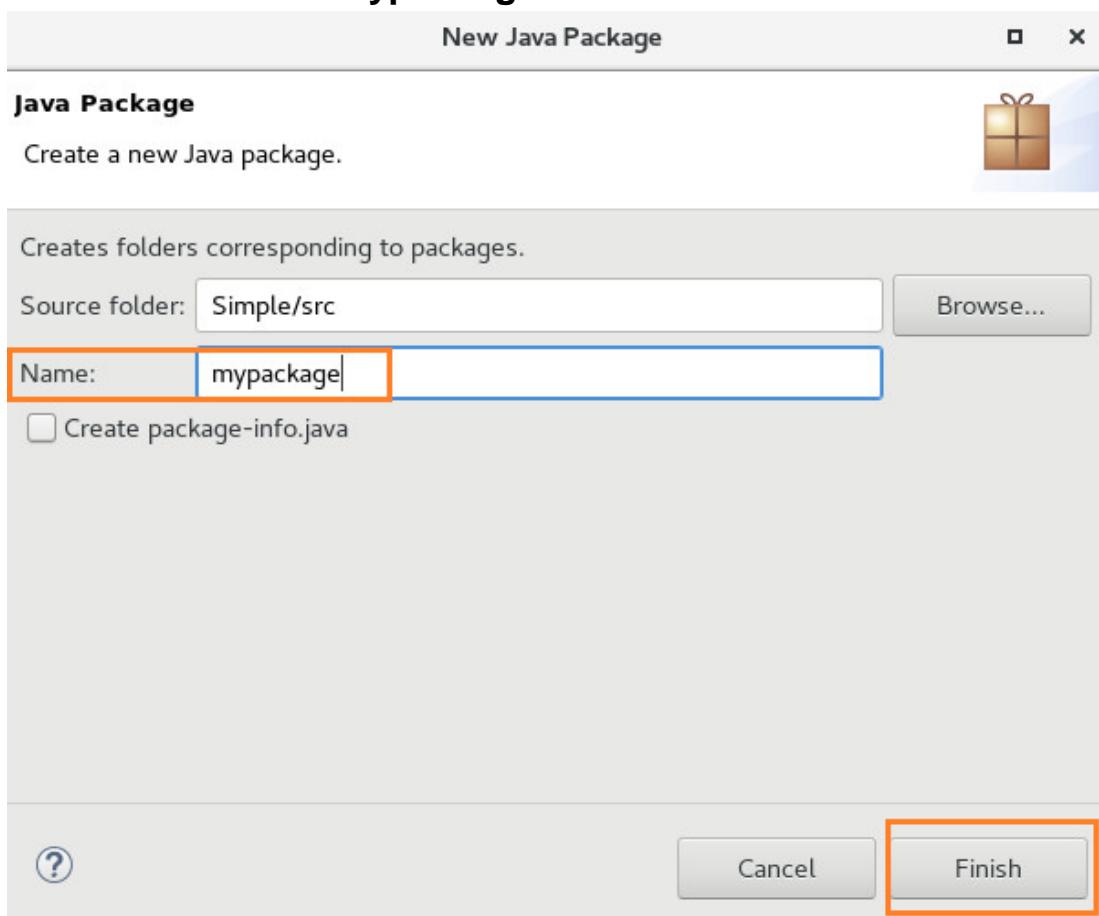
На вопрос об открытии перспективы для Java разработки ответьте согласием: **Open Perspective**.

- d. Новый проект появляется в дереве навигации (представление **Package Explorer**): обратите внимание на значок J рядом с ним:



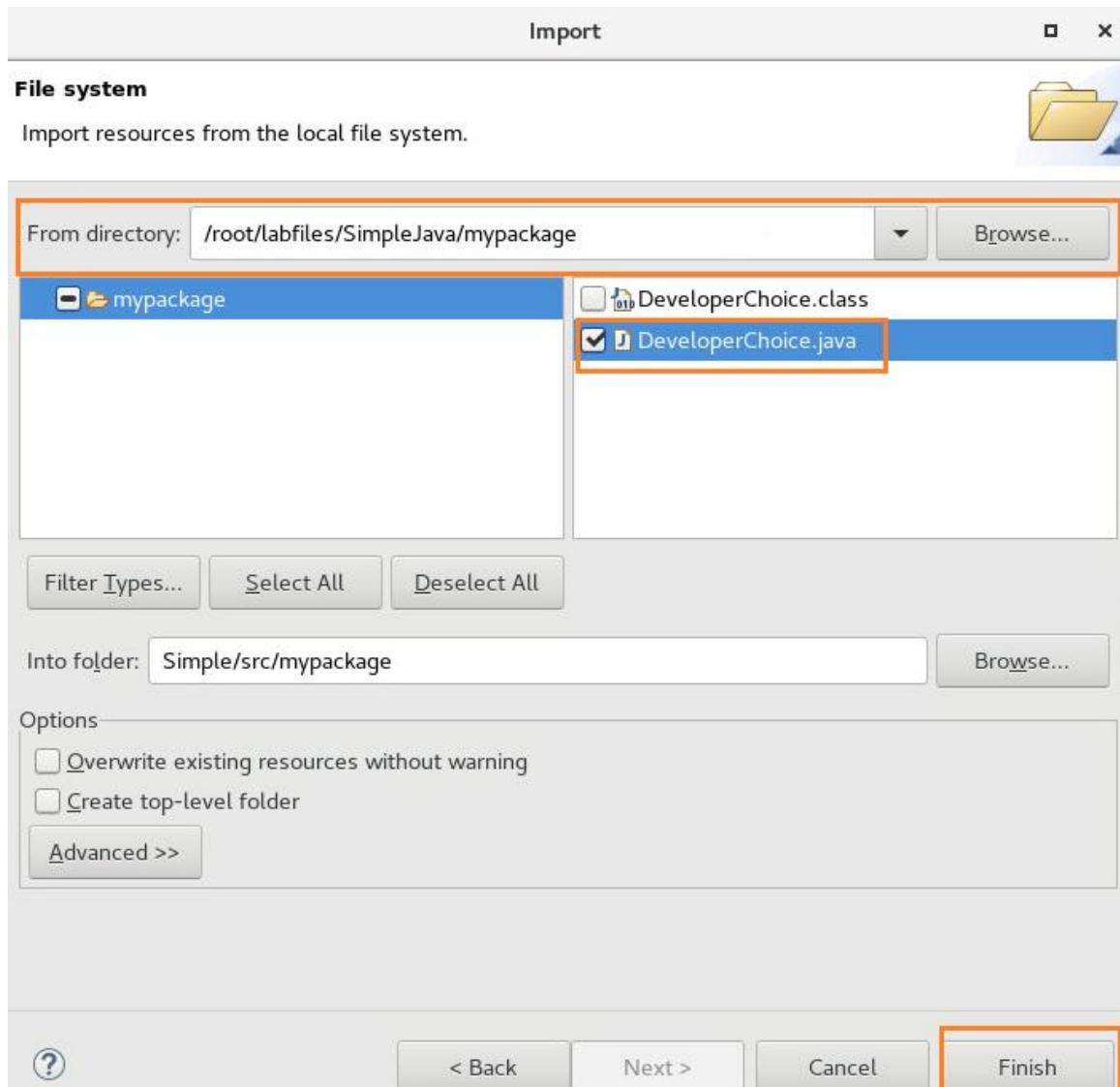
## 5. Создайте в проекте пакет **mypackage**

- В представлении **Package Explorer** разверните проект **Simple**.
- Правой кнопкой мыши нажмите по каталогу **src** и выберите в контекстном меню **New -> Package**.
- Укажите название **mypackage** и нажмите **Finish**.



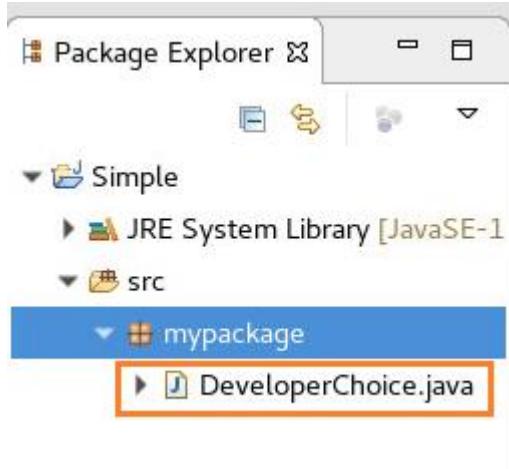
## 6. Импортируйте файл в проект

- a. В представлении **Package Explorer** правой кнопкой нажмите по только что созданному пакету и выберите в контекстном меню **Import**.
- b. В секции **General** выберите **File System** и нажмите **Next**.
- c. Нажмите **Browse** рядом с полем **From directory** и выберите каталог **/root/labfiles/SimpleJava/mypackage**
- d. Выберите только файл **DeveloperChoice.java**.



- e. Нажмите **Finish**.

- f. Импортированный файл появляется в представлении **Package Explorer**, если раскрыть пакет **mypackage**.



7. Вы можете отредактировать исходный код программы.

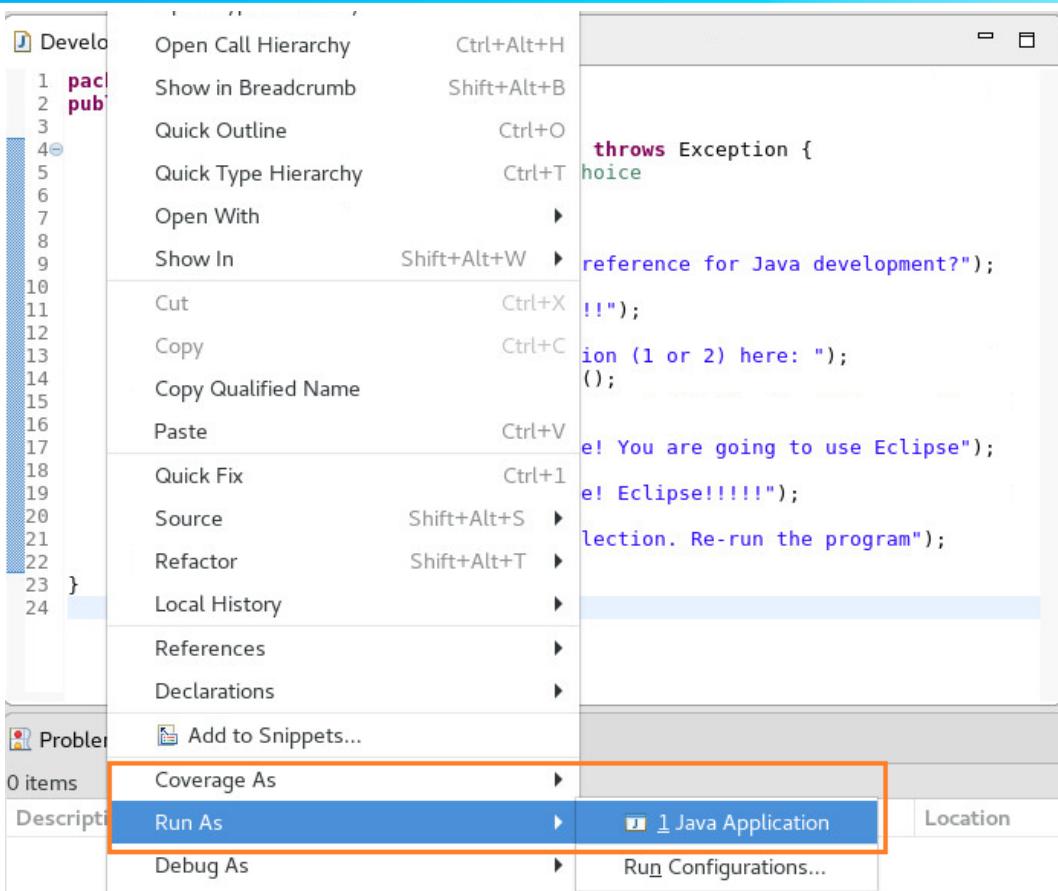
- a. Дважды щелкните левой кнопкой мыши по файлу **DeveloperChoice.java**.

```
1 package mypackage;
2 public class DeveloperChoice {
3
4     public static void main(String[] args) throws Exception {
5         // A variable to keep the user's choice
6         char yourChoice;
7
8         // The choices available
9         System.out.println("What is your preference for Java development?");
10        System.out.println("1 - Eclipse");
11        System.out.println("2 - Eclipse!!!!");
12
13        System.out.print("Type your selection (1 or 2) here: ");
14        yourChoice = (char) System.in.read();
15
16        if (yourChoice == '1')
17            System.out.println("Good choice! You are going to use Eclipse");
18        else if (yourChoice == '2')
19            System.out.println("Good choice! Eclipse!!!!!");
20        else
21            System.out.println("Invalid selection. Re-run the program");
22    }
23 }
24 }
```

- b. Внесите изменения с помощью редактора и сохраните их, нажав **Ctrl+S**.

8. Запустите приложение **DeveloperChoice.java**

- a. Нажмите правой кнопкой мыши в любом месте редактора, где открыт файл **DeveloperChoice.java**, выберите **Run As > Java Application**.



b. Ниже откроется представление **Console**, где отображается вывод приложения и где можно вводить данные.

The screenshot shows the Eclipse Console view. At the top, there are tabs: Problems, Javadoc, Declaration, and Console. The Console tab is selected. The console window displays the following text:  
DeveloperChoice [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-  
What is your preference for Java development?  
1 - Eclipse  
2 - Eclipse!!!!  
Type your selection (1 or 2) here: 1|

c. Введите **1**, нажмите **Enter**, завершив работу приложения.

**Конец упражнения**

## Упражнение 2. Написание Java кода с использованием среды разработки

---

### О чём это упражнение:

В этой лабораторной работе вы воспользуетесь некоторыми базовыми функциями среды разработки для упрощения Java разработки.

### Что вы должны будете сделать:

- Создать проект
- Импортировать код в проект
- Использовать scrapbook
- Создать переменные
- Создать массивы и использовать их в циклах

## **Раздел 1. Использование cheat sheet**

В данном разделе создается приложение с помощью пошаговой инструкции, доступной в среде разработки.

1. Откройте новое рабочее пространство
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/workspaces/SimpleJava2** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
2. Откройте шпаргалку (cheat sheet) **Hello world**.
  - a. В главном меню выберите **Help -> Cheat Sheets**.

b. Выберите вариант **Create a Hello world Application.**



c. Нажмите **OK**.

3. Следуя инструкции из шпаргалки, напишите приложение Hello world.

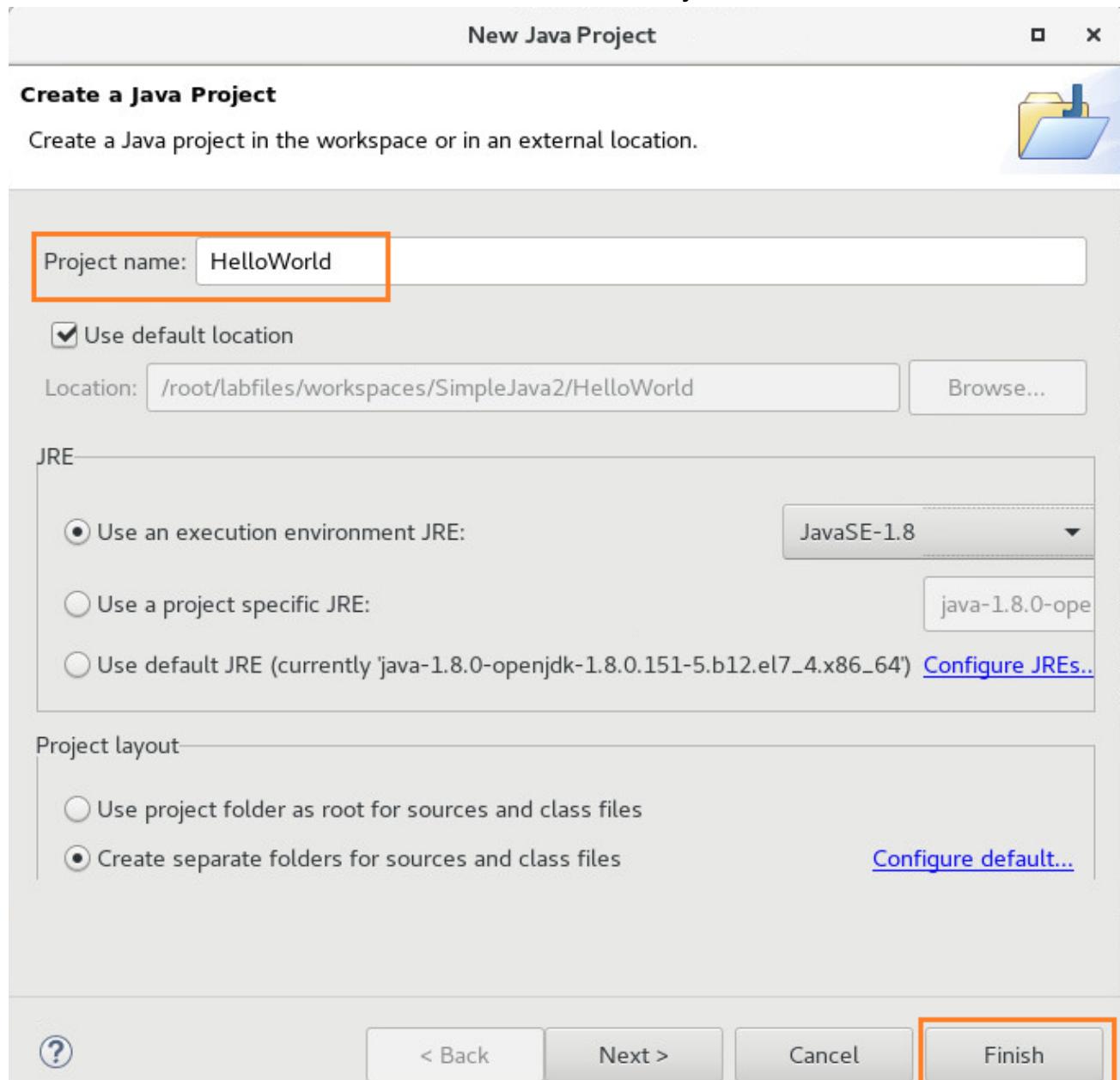
a. Прочтайте начало инструкции (**Introduction**) в соответствующем представлении справа и нажмите **Click to Begin**.



- b. Характерная галочка появляется рядом с каждым выполненным шагом.



- c. Откройте перспективу **Java**, следуя инструкциям из раздела **Open the Java Perspective**. При нажатии по кнопке **Click to perform** действие будет выполнено за вас. После завершения шага нажмите кнопку **Click when complete**. Это пометит текущий шаг, как выполненный, и приведет вас к следующей части инструкции.
- d. Запустите мастер создания нового проекта в соответствии с шагом **Create a Java project**.
- e. Укажите название проекта **HelloWorld** в качестве имени проекта. Остальные поля оставьте со значениями по умолчанию.



- f. Нажмите **Finish**.

- g. Пометьте соответствующий шаг, как выполненный (**Click when complete**).  
4. В соответствии со следующим шагом инструкции создайте класс **HelloWorld**.

a. Укажите следующие опции при создании нового класса:

**Source Folder:**

**HelloWorld/src**

**Package:**

**com.ibm.samples**

**Name:**

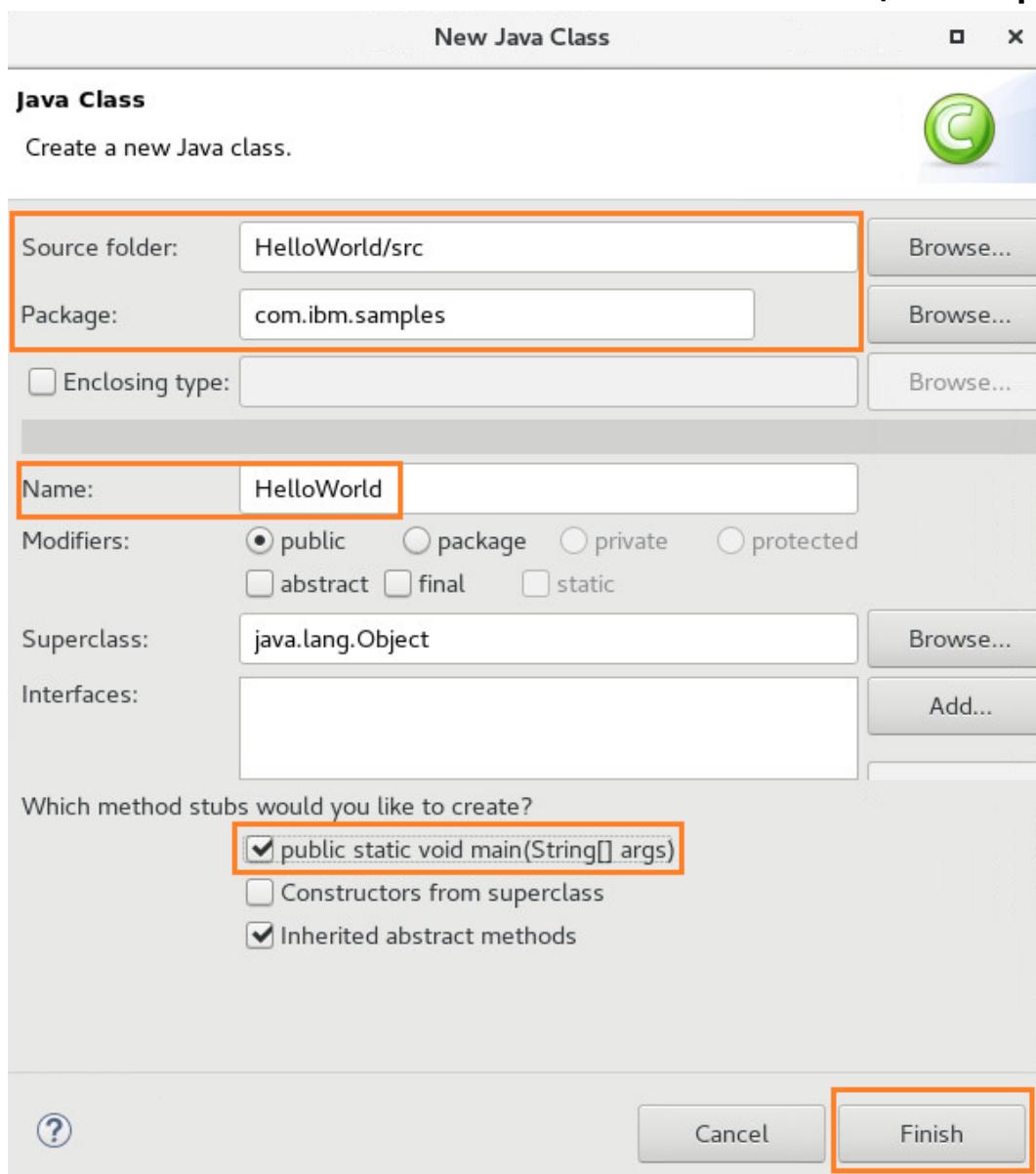
**HelloWorld**

**public static void main(String[] args):**

**опция выбрана**

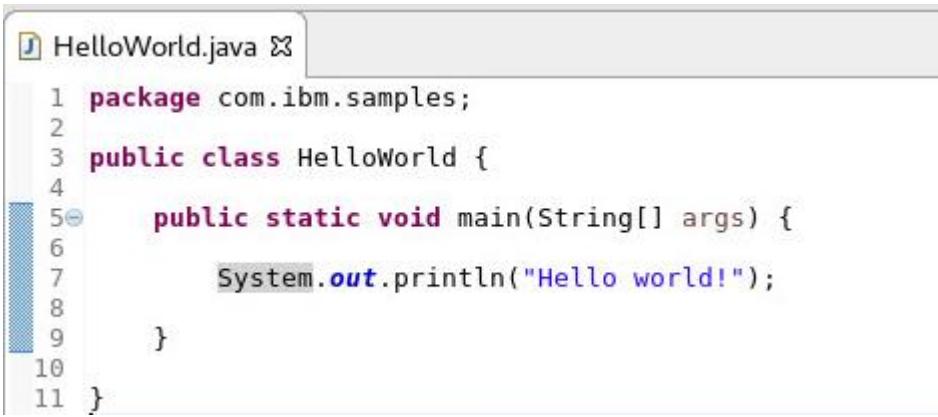
**Inherited Abstract Methods:**

**опция выбрана**



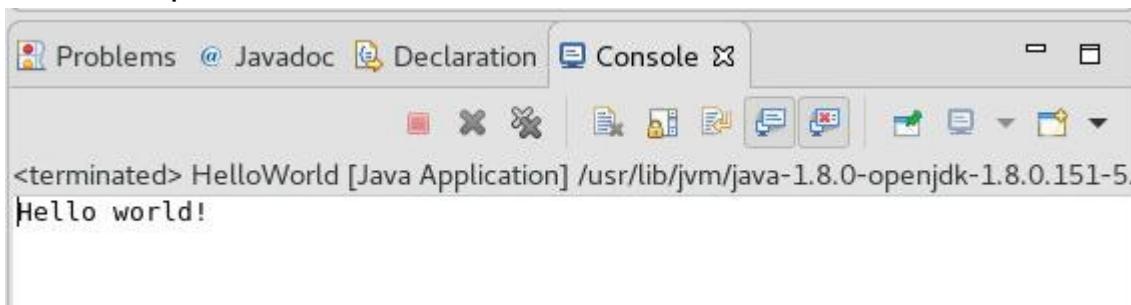
b. Нажмите **Finish**.

- c. Пометьте соответствующий шаг, как выполненный (**Click when complete**).
5. Добавьте в свое приложение логику для вывода в консоль константы с помощью следующей части шпаргалки.
  - a. Должен получиться следующий код:



```
1 package com.ibm.samples;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6
7         System.out.println("Hello world!");
8     }
9 }
10
11 }
```

- b. Нажмите **Click when Complete** по завершению.
6. Запустите приложение
  - a. Выполните инструкции из раздела **Run your Java application**.
  - b. Если при выполнении инструкции возникает предупреждение о необходимости сохранить исходный код класса **HelloWorld** – сделайте это: **Ctrl + s**.
  - c. После запуска откроется представление **Console**, где выводится листинг приложения:

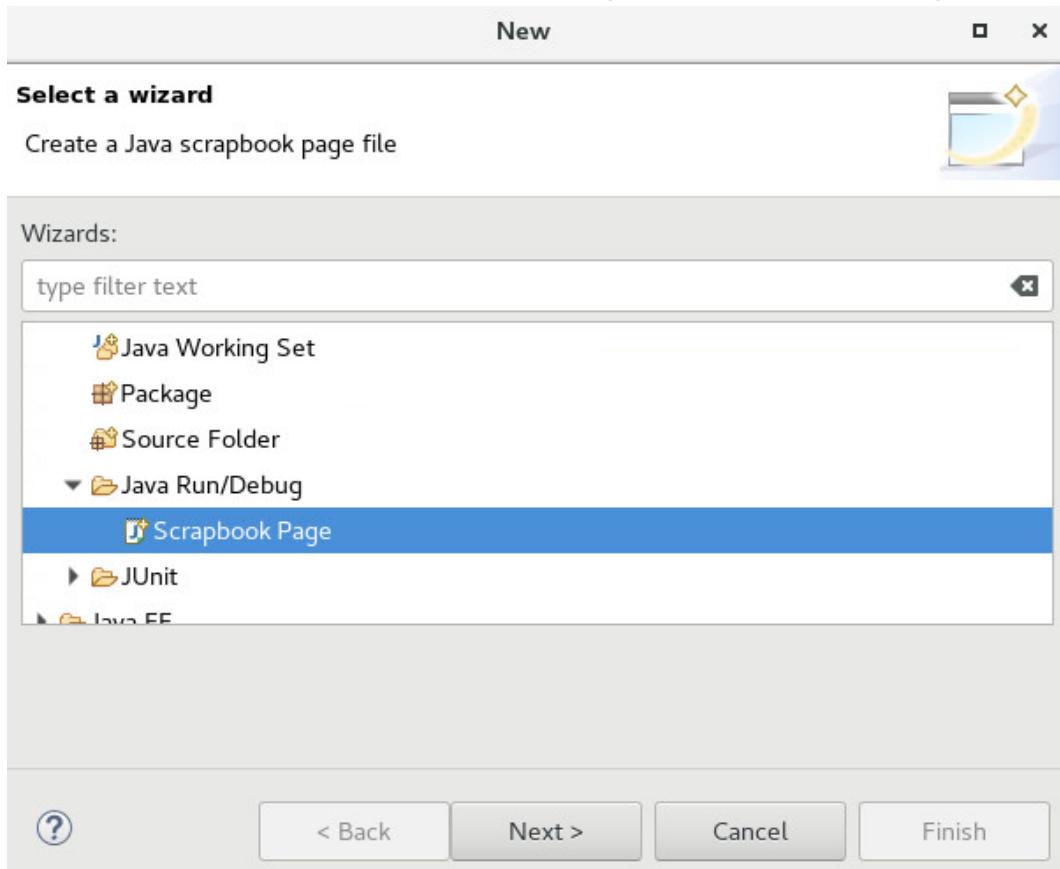


- d. Пометьте последний шаг, как выполненный.
- e. Закройте представление **Cheat Sheet**.

## Раздел 2. Использование scrapbook

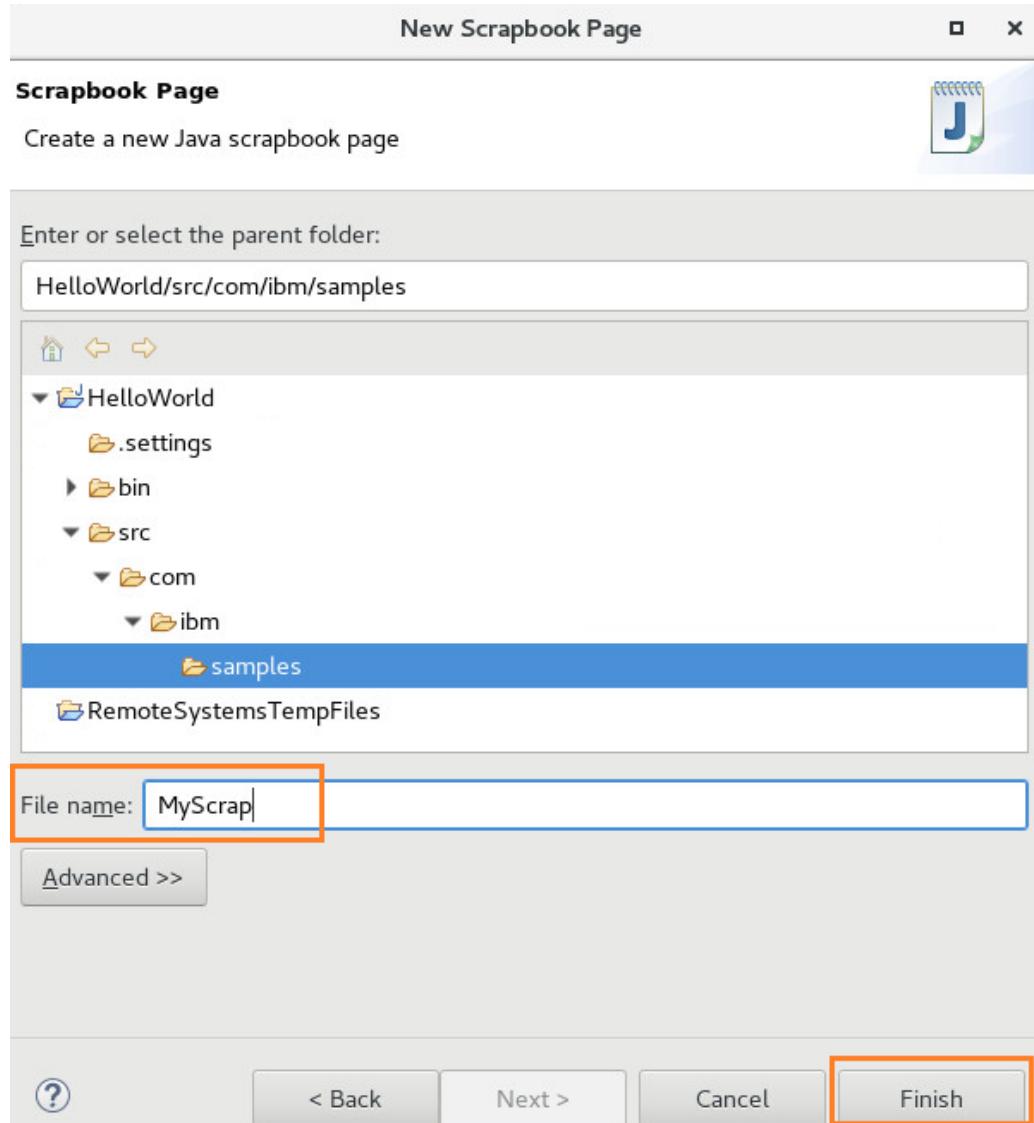
В данном разделе рассматривается вспомогательный инструмент для разработчика – **scrapbook** или альбом.

1. В существующем проекте, в созданном пакете сделайте новый альбом для экспериментов с вашим Java кодом, прежде чем добавлять соответствующие строки в ваши классы.
  - a. В представлении **Package Explorer** раскройте проект **HelloWorld**.
  - b. Правой кнопкой мыши нажмите по пакету **com.ibm.samples**. Выберите в контекстном меню **New -> Other**. Открывается диалог создания нового ресурса.
  - c. Выберите **Java -> Java Run/Debug -> Scrapbook Page**.

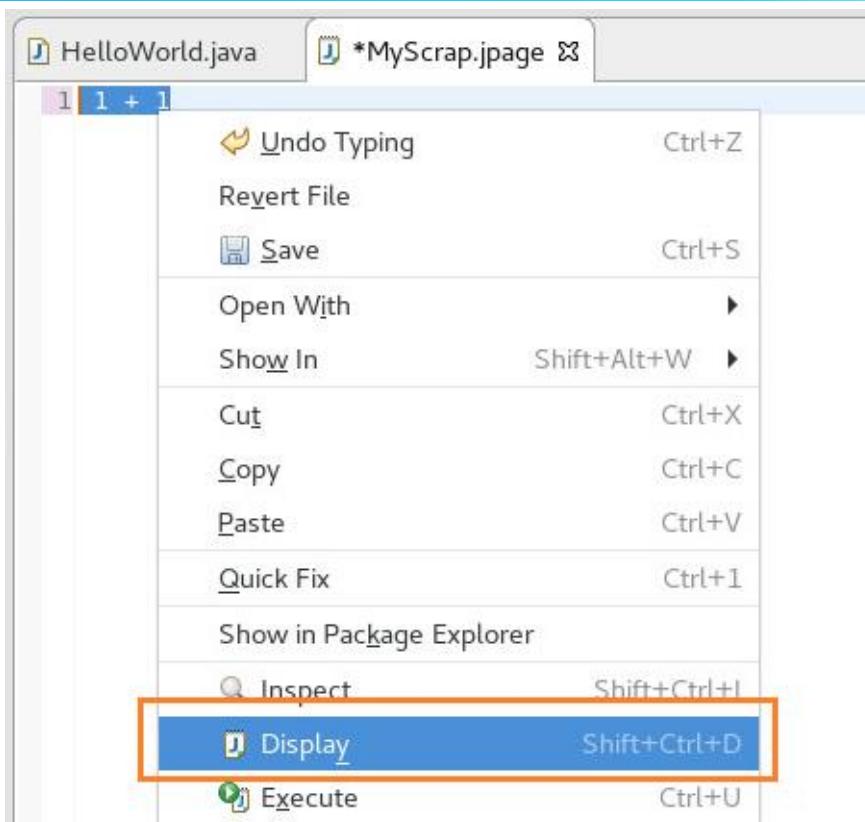


- d. Нажмите **Next**.

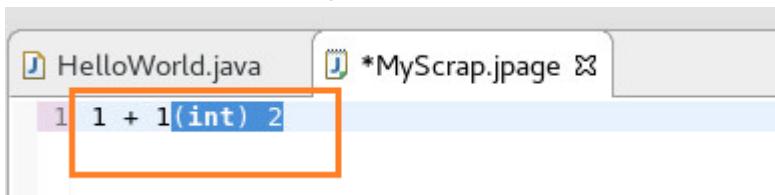
e. Введите **MyScrap** в поле **File name**.



- f. Нажмите **Finish**. Альбом автоматически открывается в редакторе. Суффикс **.jpage** автоматически добавляется к имени файла.
2. Используйте функцию **Display**.
- В редакторе, где открыт альбом, введите:  
**1 + 1**
  - Выделите текст, введенный только что, нажмите правую кнопку мыши и выберите **Display**.



с. Отображается результат: **(int) 2**



Это говорит о том, что результатом такого вычисления будет целое число – 2.

д. Повторите процедуру для следующих строчек:

**1.2f + 2.1f**

**1.2d + 2.1d**

**1.2 + 2.1**

В первом случае результат – **float**, из-за символа f, который был добавлен после численных констант. Во втором – **double**, из-за символа d.

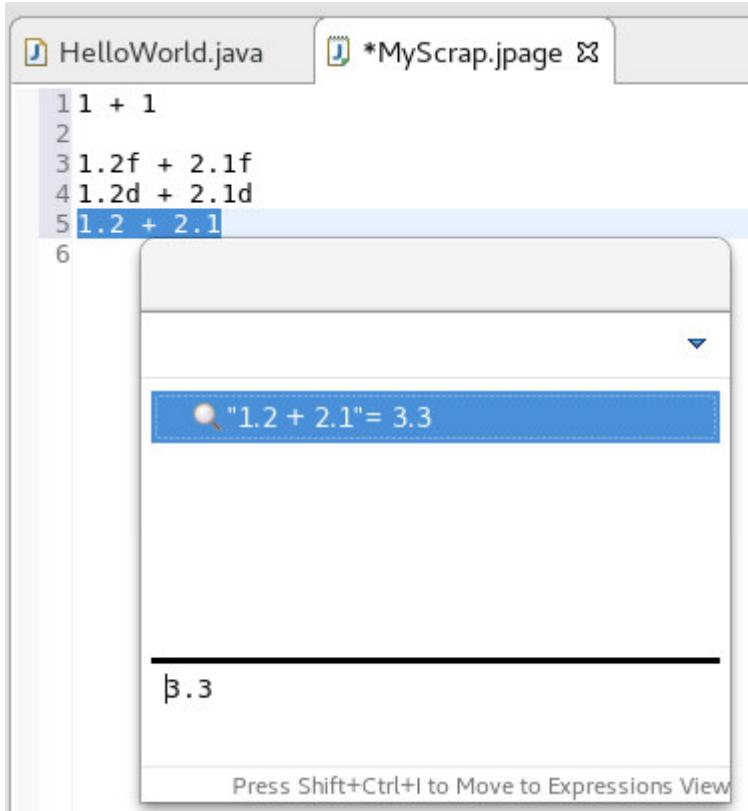
```
1 1 + 1(int) 2
2
3 1.2f + 2.1f(float) 3.3
4 1.2d + 2.1d(double) 3.3
5 1.2 + 2.1(double) 3.3|
```

- е. Почему в третьем случае результат типа **double**?
3. Используйте функцию **Inspect**. Она дает возможность увидеть результат вычислений или во всплывающем окне, или в отдельном представлении **Expressions**.
- а. Удалите из альбома все результаты работы функции **Display**, оставив только введенные вами строчки с выражениями.

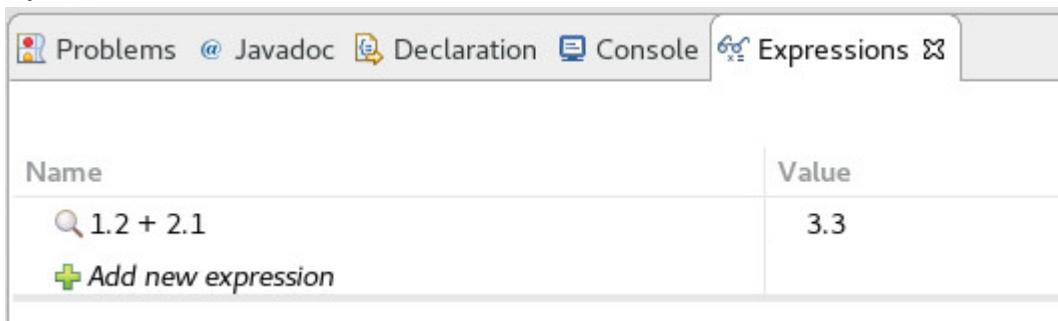
```
1 1 + 1
2
3 1.2f + 2.1f
4 1.2d + 2.1d
5 1.2 + 2.1
6|
```

- б. Выделите каждую из четырех строчек, нажмите по ней правой кнопкой мыши и выберите **Inspect** в контекстном меню.

с. Результат вычисления отображается во всплывающем окне.



д. Нажмите **Ctrl + Shift + I** для отображения результата в отдельном представлении.



е. Удалите из альбома все строки, очистив его.

ф. Объявите новую переменную x:

**int x = 0;**

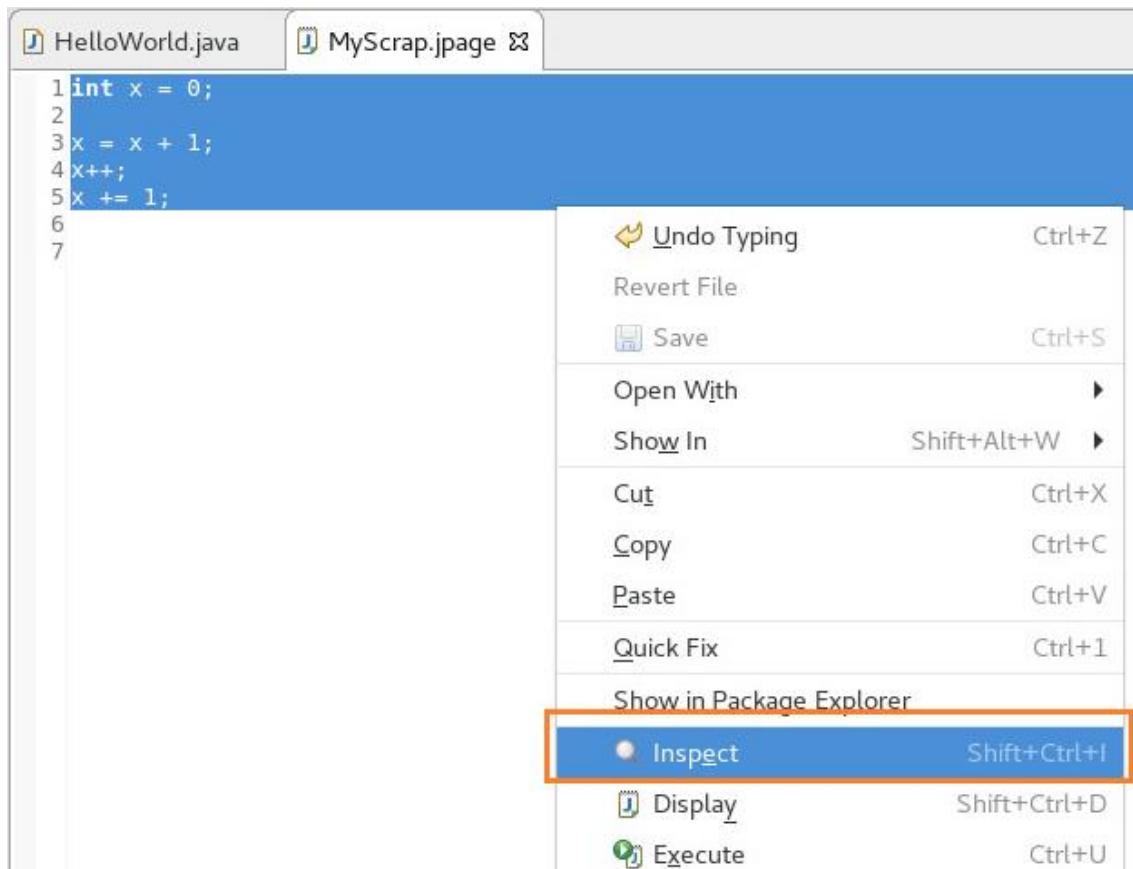
г. Увеличьте значение переменной на 1 тремя различными способами:

**x = x + 1;**

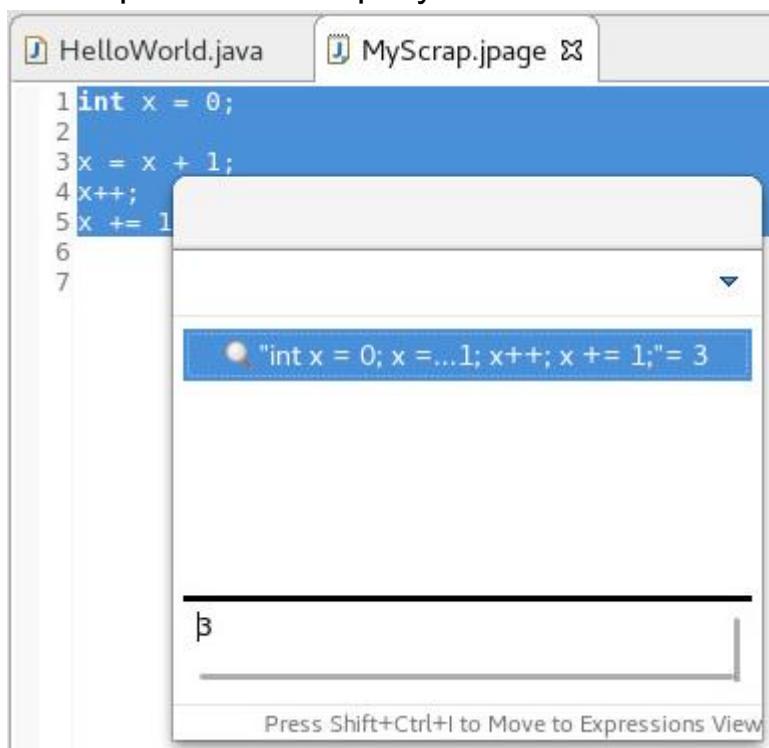
**x++;**

**x += 1;**

h. Выделите все 4 строчки, нажмите по ним правой кнопкой мыши и выберите **Inspect** в контекстном меню.



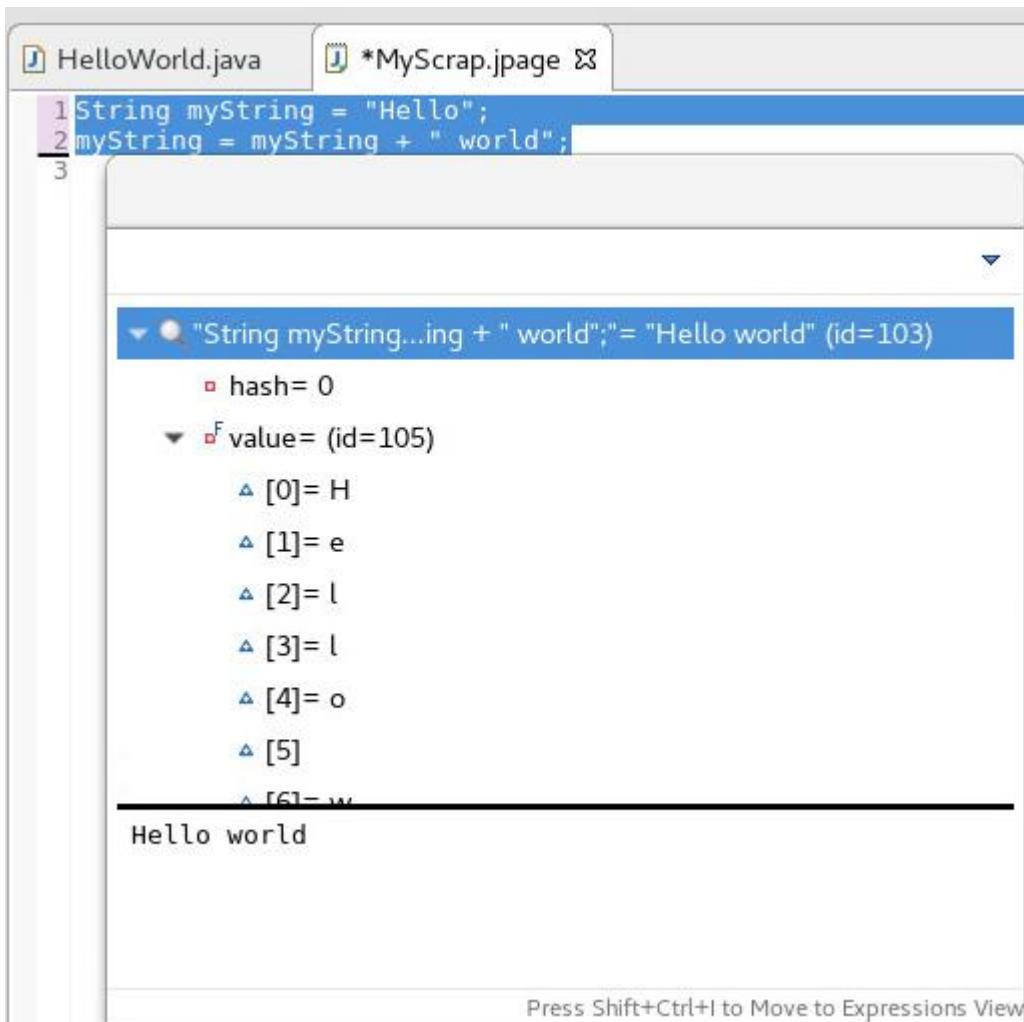
i. З отображается как результат вычисления во всплывающем окне.



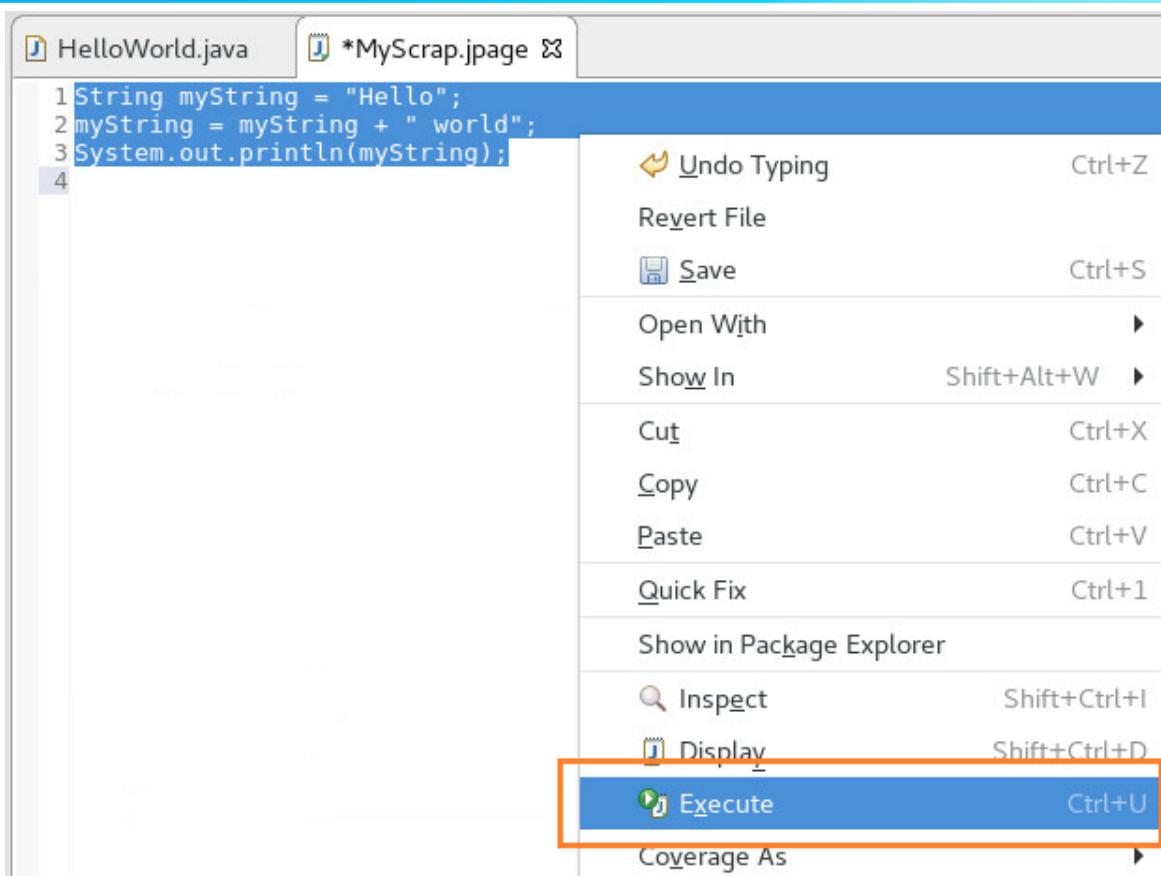
4. Создайте строковую переменную, изменив ее с помощью конкатенации, и оцените результат.
- а. Очистите альбом.
  - б. Объявите строковую переменную и присвойте ей значение:  
**String myString = "Hello";**
  - с. Дополните эту строчку новым словом. Для конкатенации используйте оператор +  
**myString = myString + " world";**  
Обратите внимание на пробел перед словом world.
  - д. Выделите две написанные строчки, нажмите по ним правой кнопкой мыши и выберите в контекстном меню **Inspect**.
  - е. Раскройте при необходимости строчку **String myString...**



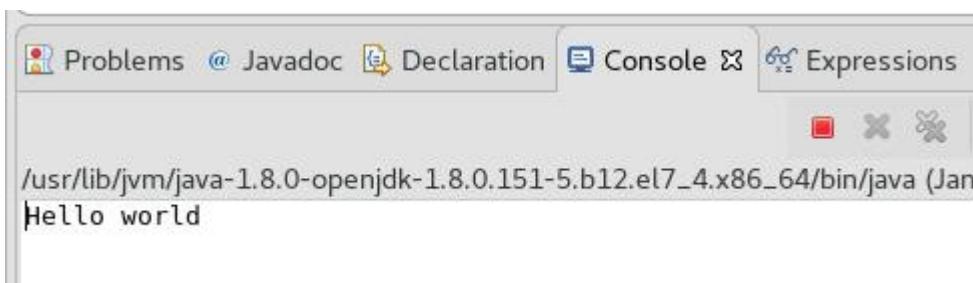
- ф. Обратите внимание на секцию **value** (ее можно раскрыть), там посимвольно отображается содержимое строковой переменной.



5. В пакете **java.lang** есть класс **System**. В нем есть поле **out** типа **PrintStream**. Оно может быть использовано для вывода данных в поток стандартного вывода – обычно это дисплей консоли Java.
- В альбоме после двух строчек, записанных в предыдущем пункте, добавьте строку для вывода переменной **myString** в консоль:  
**System.out.println(myString);**
  - Выделите все три строчки из альбома, нажмите по ним правой кнопкой мыши, выберите в контекстном меню **Execute**.



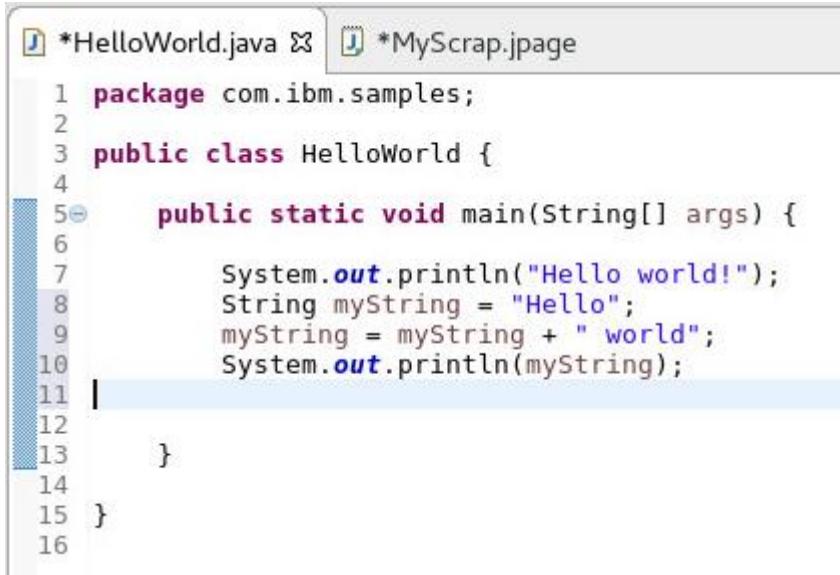
с. Результат исполнения кода отображается в представлении **Console**.



6. Добавьте код из вашего альбома в метод **main** класса **HelloWorld**.

- Скопируйте содержимое вашего альбома (все три строчки) в буфер обмена.
- Откройте файл **HelloWorld.java** в редакторе, если он еще не открыт, или переключитесь на соответствующую вкладку, если он уже открыт.
- Вставьте скопированный код после строчки:  
**System.out.println("Hello World!");**

d. Ваш класс должен выглядеть следующим образом:



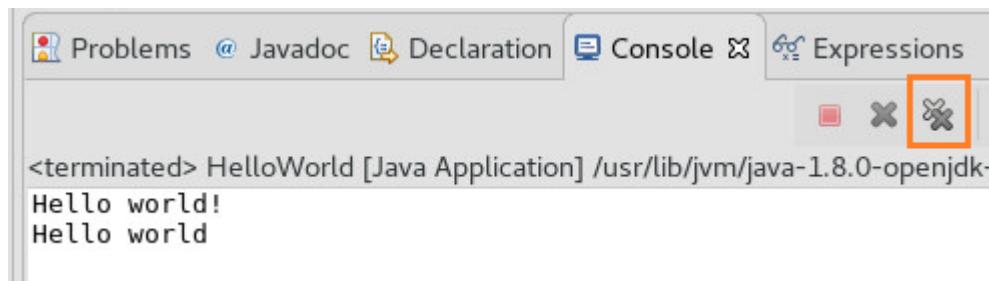
```
1 package com.ibm.samples;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6
7         System.out.println("Hello world!");
8         String myString = "Hello";
9         myString = myString + " world";
10        System.out.println(myString);
11    }
12
13 }
14
15 }
16
```

e. Сохраните сделанные изменения.

f. Запустите **HelloWorld.java** через контекстное меню: **Run As -> Java Application**.

g. Результат будет отображаться в консоли.

7. Очистите вывод от предыдущего запуска приложения в консоли с помощью кнопки **Remove All Terminated Launches**.



```
<terminated> HelloWorld [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-  
Hello world!  
Hello world
```

8. Создайте массив целых чисел в альбоме и перенесите этот код в класс **HelloWorld**.

a. Перейдите на вкладку с альбомом **MyScrap.jpage**.

b. Удалите текущее содержимое альбома.

c. Объявите массив целых чисел и заполните его значениями:

```
int[] numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

d. Ниже, добавьте строки для вывода в консоль элементов массива в цикле for:

```
for (int i = 0; i < 10; i++) {  
    System.out.println(numbers[i]);
```

}

e. Выделите все строки в альбоме, нажмите по ним правой кнопкой мыши и выберите **Execute**.

f. Числа из массива должны отобразиться в консоли:



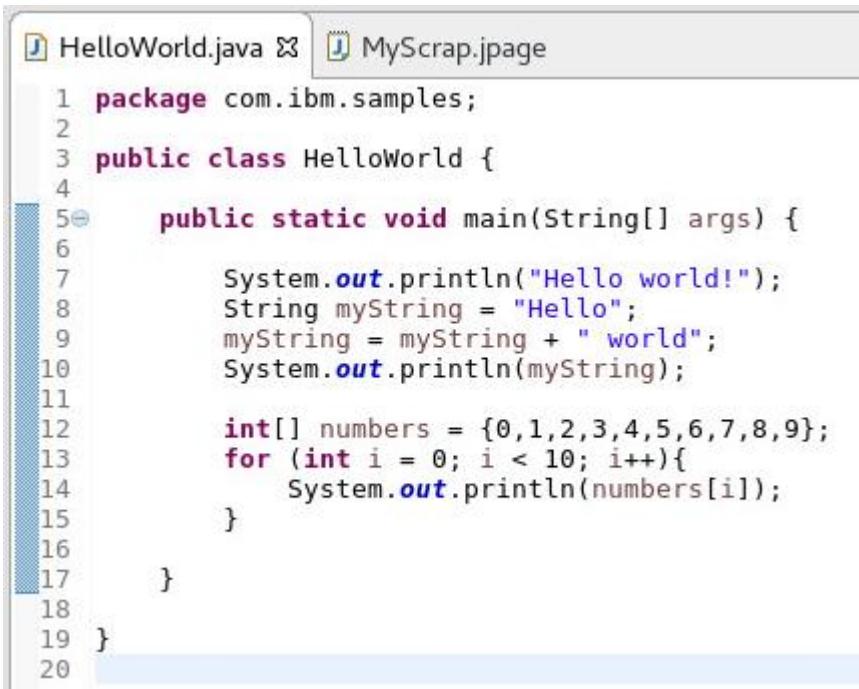
The screenshot shows a Java IDE interface with several tabs at the top: Problems, Javadoc, Declaration, Console, and Expressions. The Console tab is active, displaying the command '/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-5.b12.el7\_4.x86\_64/bin/java' followed by a series of numbers from 0 to 9, each on a new line. Below the numbers is some standard Java code.

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

g. Скопируйте код из альбома в метод **main** класса **HelloWorld** после строчки:

```
System.out.println(myString);
```

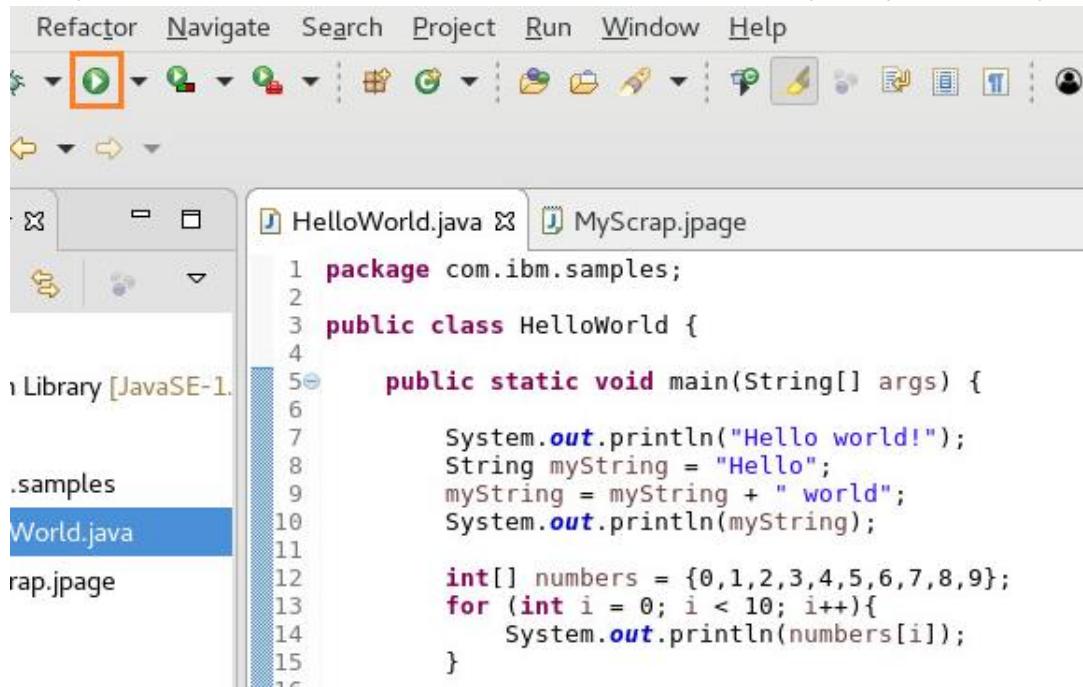
h. Код должен выглядеть следующим образом:



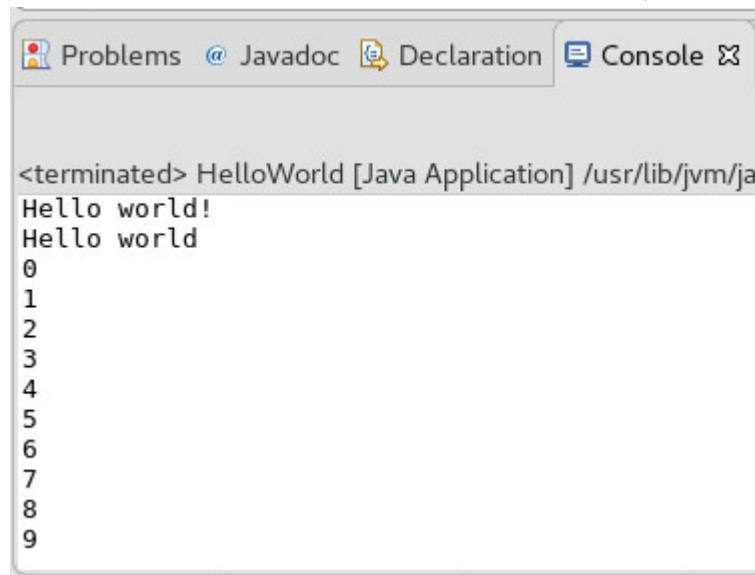
The screenshot shows a Java IDE interface with two tabs: HelloWorld.java and MyScrap.java. The HelloWorld.java tab is active, displaying the following Java code:

```
1 package com.ibm.samples;  
2  
3 public class HelloWorld {  
4  
5     public static void main(String[] args) {  
6  
7         System.out.println("Hello world!");  
8         String myString = "Hello";  
9         myString = myString + " world";  
10        System.out.println(myString);  
11  
12        int[] numbers = {0,1,2,3,4,5,6,7,8,9};  
13        for (int i = 0; i < 10; i++){  
14            System.out.println(numbers[i]);  
15        }  
16    }  
17}  
18}  
19}
```

- i. Запустите **HelloWorld.java**, нажав соответствующую кнопку **Run**:



- j. В консоли отображается соответствующий листинг:



**Конец упражнения**

## Упражнение 3. Создание классов

---

### О чём это упражнение:

В этой лабораторной работе вы создадите классы, необходимые для работы системы **OnlineMedia**, с помощью которой можно создавать заказы на различные товары: CD, DVD, книги.

Диаграмму показывающую особенности создаваемых классов и взаимосвязи между ними можно посмотреть в файле  
</root/labfiles/Visualization/BuildingClassesAfterDiagram.png>

Информация о каждой потенциальной позиции в заказе хранится в файле **media.properties**. Класс **Order** может восприниматься как карточка покупателя магазина, которая описывает товары в заказе. Класс **OnlineMedia**, в частности его метод **main** используется для работы со всеми остальными частями системы.

### Что вы должны будете сделать:

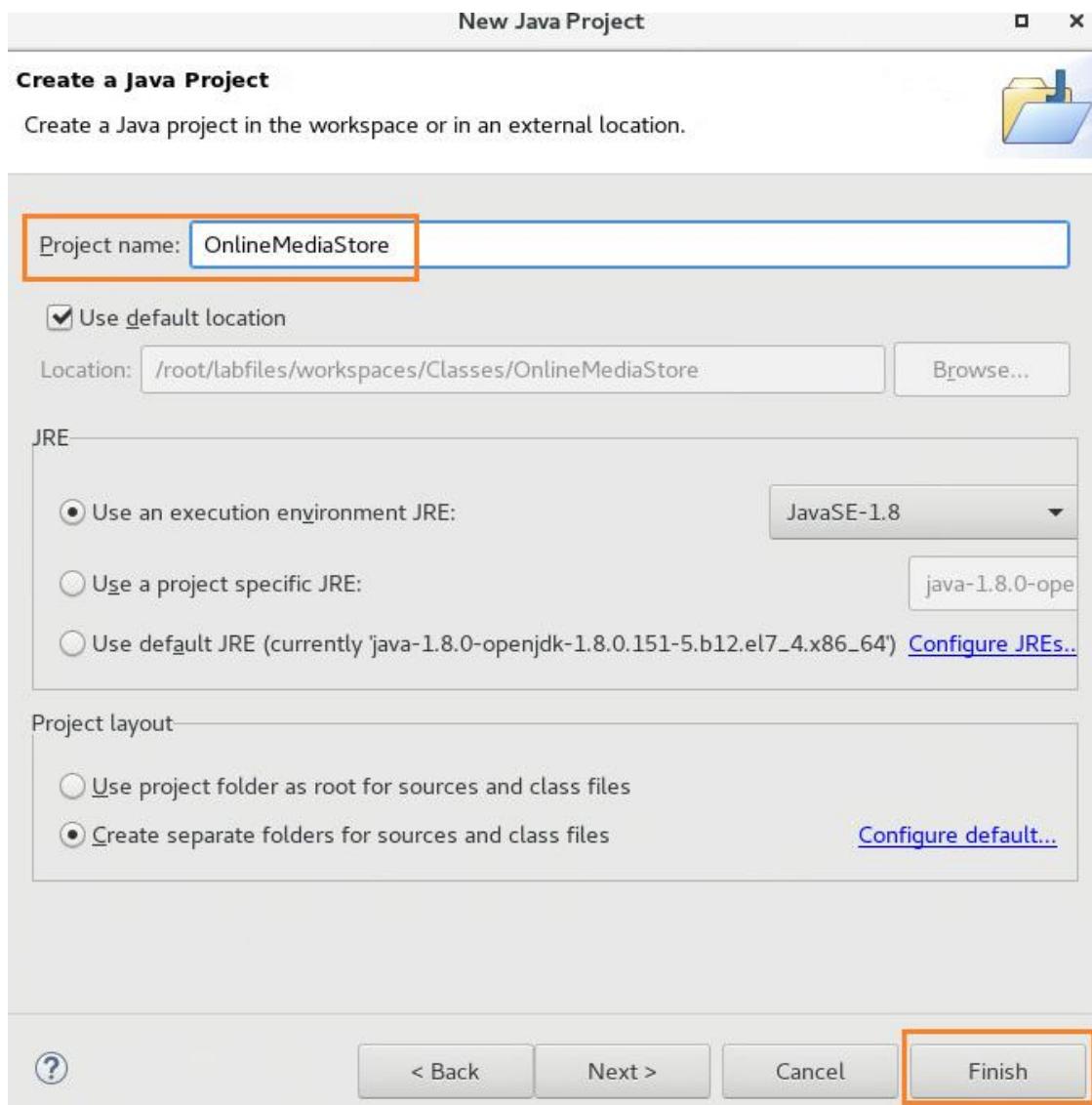
- Создать и модифицировать Java классы
- Создавать переменные, являющиеся экземплярами описанных классов
- Создавать методы
- Использовать массивы

## **Раздел 1. Создание класса *OnlineMedia***

В данном разделе разрабатывается класс, куда будет добавлен код для создания заказов и добавления туда новых позиций.

1. Откройте новое рабочее пространство
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/workspaces/Classes** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
2. Создайте проект **OnlineMediaStore**
  - a. Перейдите в меню: **File -> New -> Other**, далее выберите **Java Project**. Нажмите **Next**.
  - b. Укажите название **OnlineMediaStore**, значения остальных полей оставьте заполненными по умолчанию.

## с. Нажмите **Finish**.



На вопрос об открытии перспективы для Java разработки ответьте согласием: **Open Perspective**.

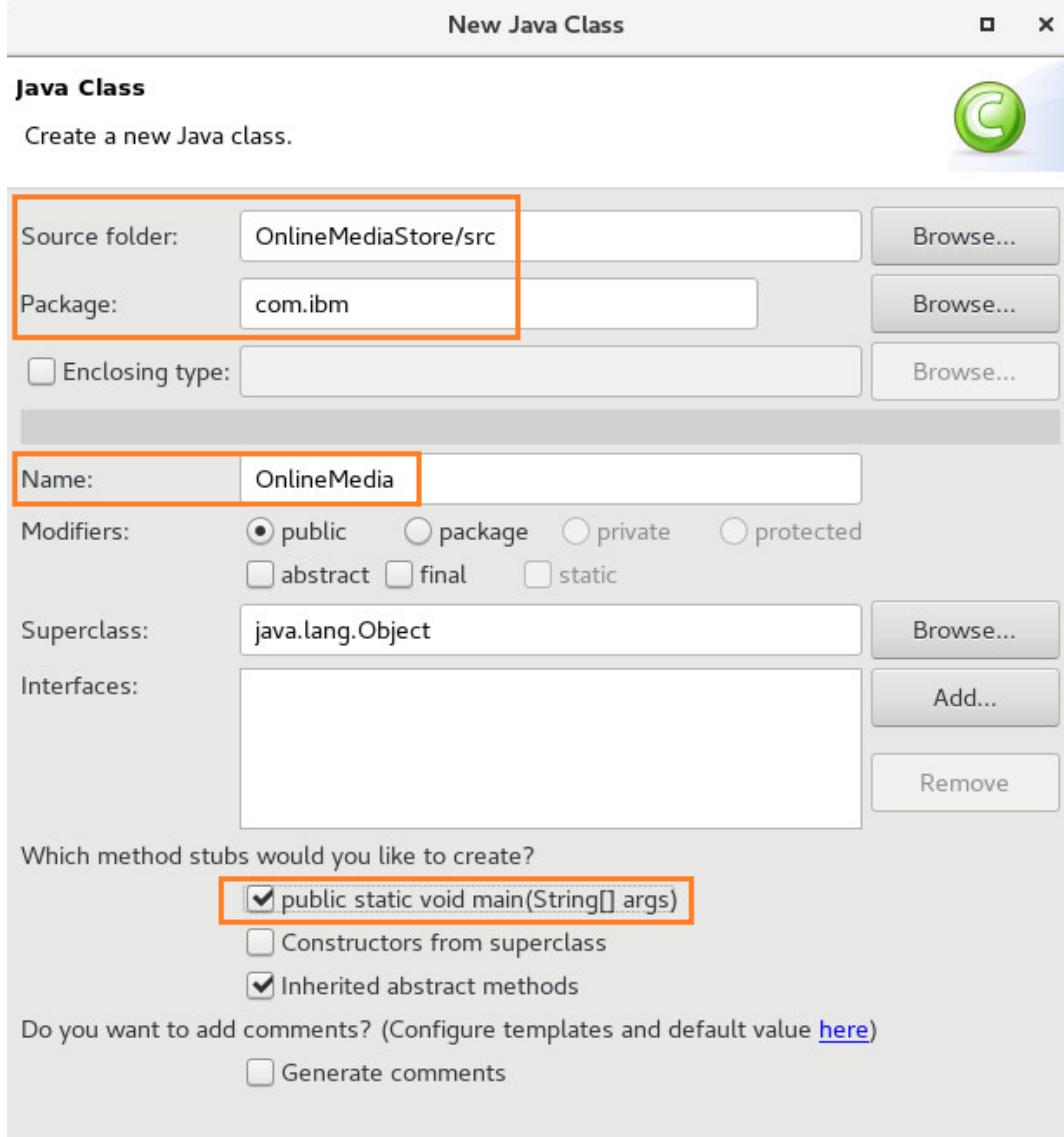
## 3. Создайте класс **OnlineMedia**.

- Из главного меню перейдите: **File -> New -> Class**.

Также можно нажать правой кнопкой мыши по проекту **OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.

- В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore/src**.
- В названии пакета введите **com.ibm**.
- Укажите **OnlineMedia** как имя класса.
- Установите опцию **public static void main(String[] args)**. Это добавит в ваш класс метод **main** для того, чтобы запускать этот класс.

f. Остальные значения не меняйте:



g. Нажмите **Finish**.

## Раздел 2. Создание класса *DigitalVideoDisc*

В данном разделе создается класс, моделирующий DVD диск: например, экземпляры этого класса будут хранить информацию о свойствах DVD диска.

1. В представлении **Package Explorer** нажмите правой кнопкой по пакету **com.ibm** и выберите в контекстном меню **New -> Class**.
2. Укажите **DigitalVideoDisc** как название класса.
3. Остальные значения не меняйте. В этот раз опция **public static void main(String[] args)** не будет выбираться, так как создаваемый класс

никогда не будет запускаться самостоятельно.

## Java Class

Create a new Java class.



Source folder:

Package:

Enclosing type:

Name:

Modifiers:  public  package  private  protected  
 abstract  final  static

Superclass:

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)  
 Constructors from superclass  
 Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
 Generate comments

4. Нажмите **Finish**. Новый файл **DigitalVideoDisc.java** открывается в редакторе.

а. Добавьте следующий код для описания атрибутов класса после его объявления:

```
private String title;  
private String category;  
private String director;  
private int length;  
private float cost;
```

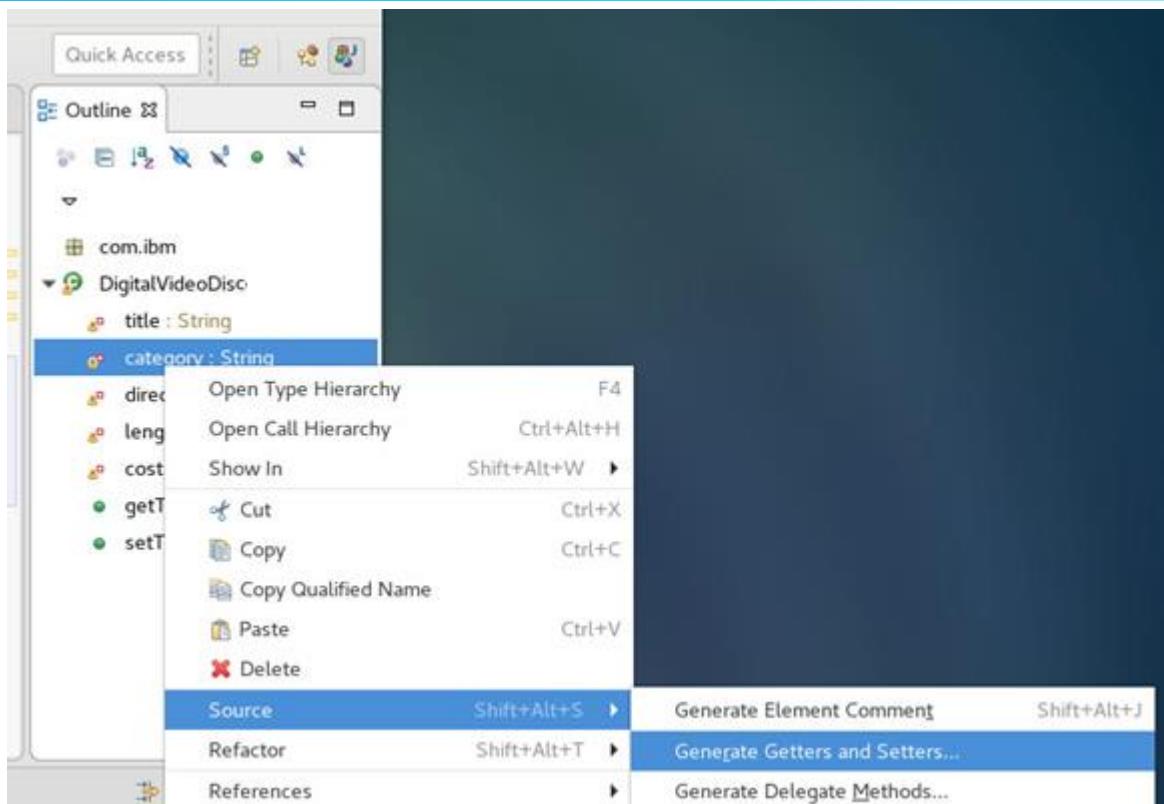
```
1 package com.ibm;
2
3 public class DigitalVideoDisc {
4
5     private String title;
6     private String category;
7     private String director;
8     private int length;
9     private float cost;
10
11 }
12
```

- b. После конструктора в коде класса следует добавить методы **getter** и **setter** для всех полей класса. Начните с поля **title**.

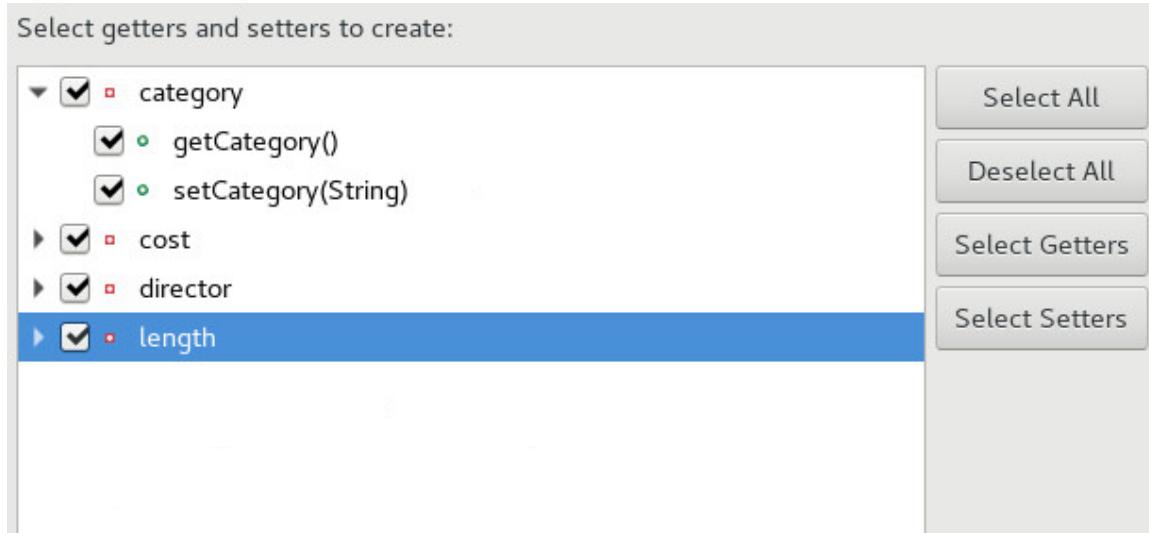
```
/***
 * Returns the title
 */
public String getTitle() {
    return title;
}
/***
 * Sets the title
 */
public void setTitle(String title) {
    this.title = title;
}
```

5. Далее следует сделать аналогичные методы для всех оставшихся полей. Eclipse позволяет автоматизировать этот процесс.

- a. Перейдите в представление **Outline**. Здесь отображаются все поля и методы класса.  
b. Нажмите правой кнопкой мыши по полю **category**. Выберите **Source -> Generate Getters and Setters** в контекстном меню.



- с. В открывшемся диалоге нажмите кнопку **Select All** чтобы выбрать все поля класса, для которых нужно сгенерировать методы.



- д. Нажмите **OK**.

- е. В редакторе появится следующий код:

```
public String getCategory() {  
    return category;  
}
```

```
public void setCategory(String category) {  
    this.category = category;  
}  
  
public float getCost() {  
    return cost;  
}  
  
public void setCost(float cost) {  
    this.cost = cost;  
}  
  
public String getDirector() {  
    return director;  
}  
  
public void setDirector(String director) {  
    this.director = director;  
}  
  
public int getLength() {  
    return length;  
}  
  
public void setLength(int length) {  
    this.length = length;  
}
```

Факт наличия/отсутствия комментариев зависит от используемой версии среды разработки.

- f. Убедитесь, что в представлении Outline появились сгенерированные методы.



g. Сохраните сделанные изменения через меню: **File -> Save**. Если в коде есть ошибки, то Eclipse сообщает о них в виде красных маячков на соответствующих строках кода. Объяснения ошибок можно увидеть в представлении **Problems**.

The screenshot shows the Eclipse IDE interface. In the top-left, there are two tabs: 'OnlineMedia.java' and 'DigitalVideoDisc.java'. The code editor displays Java code for a class 'DigitalVideoDisc'. A specific line of code, 'private string title;', is highlighted with a red box. The code editor has a light blue background for the code area. To the right of the code editor is the 'Outline' view, which lists various classes and methods. Below the code editor is the 'Problems' view, which shows a table of errors. The table has columns: Description, Resource, Path, Location, and Type. The 'Description' column shows three error items, each starting with a red exclamation mark icon followed by the message 'string cannot be resolved to a type'. The 'Resource' column shows 'DigitalVideoDisl', the 'Path' column shows '/OnlineMediaStore/s', and the 'Location' column shows 'line 5' for the first error. All three errors are categorized as 'Java Problem'. The entire 'Problems' view area is also highlighted with a red box.

Description	Resource	Path	Location	Type
✖ Errors (3 items)				
✖ string cannot be resolved to a type	DigitalVideoDisl	/OnlineMediaStore/s	line 5	Java Problem
✖ string cannot be resolved to a type	DigitalVideoDisl	/OnlineMediaStore/s	line 12	Java Problem
✖ string cannot be resolved to a type	DigitalVideoDisl	/OnlineMediaStore/s	line 15	Java Problem

## Раздел 3. Создание класса *Order*

Этот класс будет хранить информацию о списке DVD дисков и методы для редактирования списка.

1. В представлении **Package Explorer** нажмите правой кнопкой по пакету **com.ibm** и выберите в контекстном меню **New -> Class**.
2. Укажите **Order** как название класса.

3. Убедитесь, что выбран правильный каталог и пакет. Остальные значения не меняйте.

## Java Class

Create a new Java class.



Source folder:

Package:

Enclosing type:

Name:

Modifiers:  public  package  private  protected  
 abstract  final  static

Superclass:

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)  
 Constructors from superclass  
 Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
 Generate comments

4. Нажмите **Finish**. Новый файл **Order.java** открывается в редакторе.  
5. Добавьте в класс атрибут **itemsOrdered** с массивом объектов **DigitalVideoDisc**.

- a. Объявите соответствующее приватное поле в классе:

```
private DigitalVideoDisc itemsOrdered[] = new
DigitalVideoDisc[10];
```



```
1 package com.ibm;
2
3 public class Order {
4
5     private DigitalVideoDisc itemsOrdered[] = new DigitalVideoDisc[10];
6
7 }
```

6. Добавьте атрибут **qtyOrdered** для хранения количества позиций в заказе.

а. Начальное значение поля **0**:

```
private int qtyOrdered = 0;
```

б. Сделайте методы типа **getter** и **setter** для этого поля. Можно реализовать их самостоятельно или сгенерировать с помощью Eclipse.

```
/**  
 * Returns the qtyOrdered  
 */  
public int getQtyOrdered() {  
    return qtyOrdered;  
}  
  
/**  
 * Sets the qtyOrdered  
 */  
public void setQtyOrdered(int qtyOrdered) {  
    this.qtyOrdered = qtyOrdered;  
}
```

7. Реализуйте метод **addDigitalVideoDisc()**.

а. Этот метод будет использоваться для добавления новой позиции в заказ.

б. Добавьте в класс следующий код:

```
public void addDigitalVideoDisc (DigitalVideoDisc  
disc) {  
    int qty = getQtyOrdered();  
    // check that the order is not already full  
    if (qty < 10) {  
        // add the disc to the order  
        itemsOrdered[qty] = disc;  
        // increment the number of discs ordered  
        setQtyOrdered(qty + 1);  
    }  
}
```

8. Реализуйте метод **removeDigitalVideoDisc()**.

а. Это метод будет использоваться для удаления позиций из заказа.

b. Добавьте следующий код в класс:

```
public void removeDigitalVideoDisc (DigitalVideoDisc disc) {  
    int qty = getQtyOrdered();  
    // check that the order is not empty  
    if (qty > 0) {  
        // loop through the discs in the order  
        for (int i = 0; i < qty; i++) {  
            // if you have found the correct disc in  
            // the order  
            if (disc.equals(itemsOrdered[i])) {  
                // remove the disc from the order  
                itemsOrdered[i] = null;  
                // decrement the number of discs  
                // ordered  
                setQtyOrdered(qty - 1);  
                // exit the loop;  
                break;  
            }  
        }  
    }  
}
```

9. Реализуйте метод **totalCost()**.

- Этот метод будет возвращать сумму всего заказа, проходя по массиву и добавляя сумму позиции к общей сумме заказа.
- Вы можете использовать следующий код:

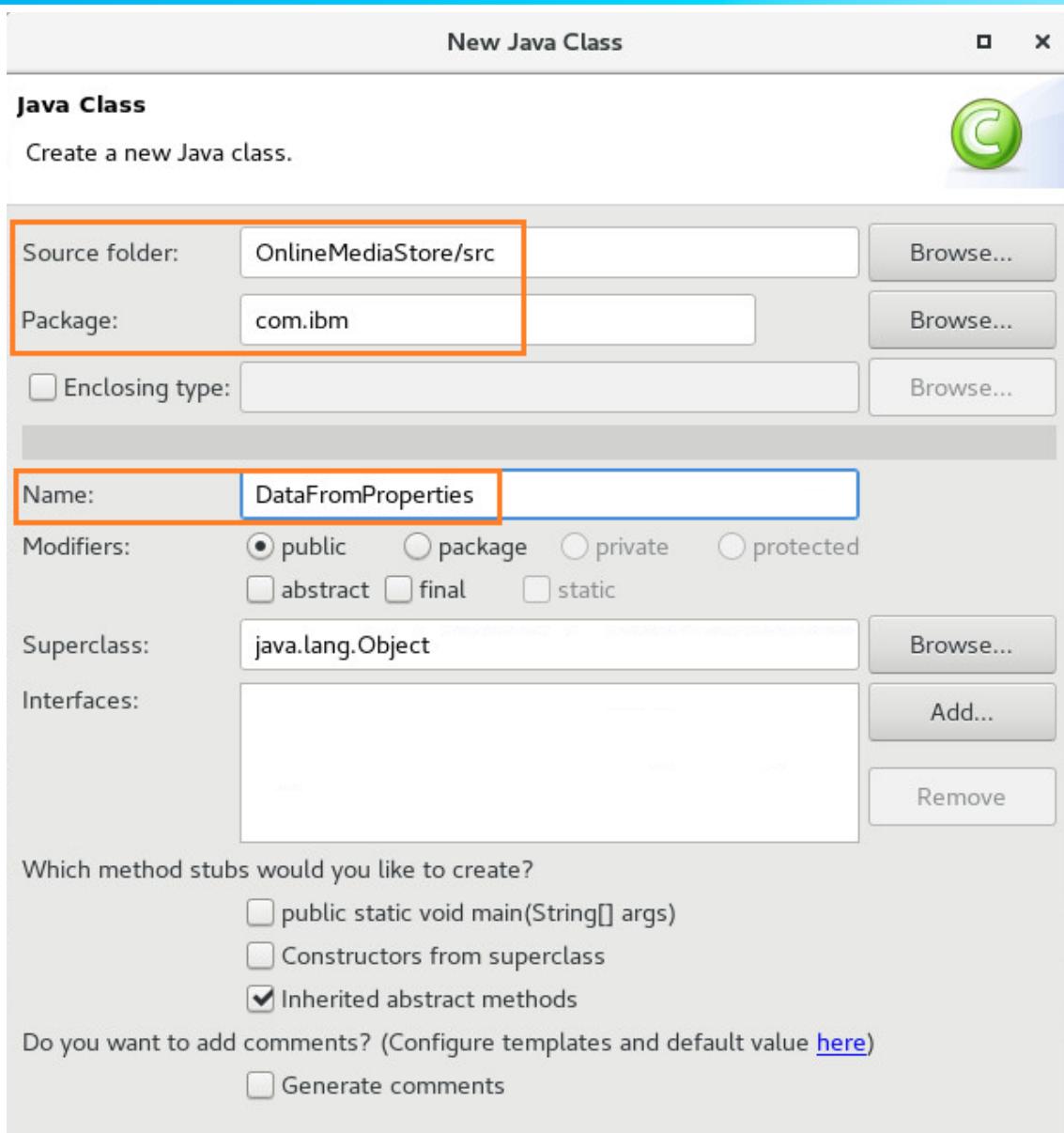
```
public float totalCost() {  
  
    int qty = getQtyOrdered();  
    float total = 0;  
  
    // loop through the discs in the order  
    for (int i = 0; i < qty; i++) {  
        total = total + itemsOrdered[i].getCost();  
    }  
    return total;  
}
```

10. Сохраните сделанные в классе изменения, при наличии ошибок, устранитте их.

**Раздел 4. Создание класса *DataFromProperties* для считывания данных об экземплярах класса *DigitalVideoDisc*.**

Этот класс будет использоваться для считывания данных о дисках из файла параметров.

1. Создайте новый класс **DataFromProperties**.
  - a. В представлении **Package Explorer** раскройте пакет **OnlineMediaStore -> com.ibm**.
  - b. Нажмите по этому пакету правой кнопкой мыши и выберите **New - > Class** в контекстном меню.
  - c. Введите название класса **DataFromProperties**. Остальные значения оставьте без изменений.



d. Нажмите **Finish**.

## 2. Реализуйте метод **addADvd()**.

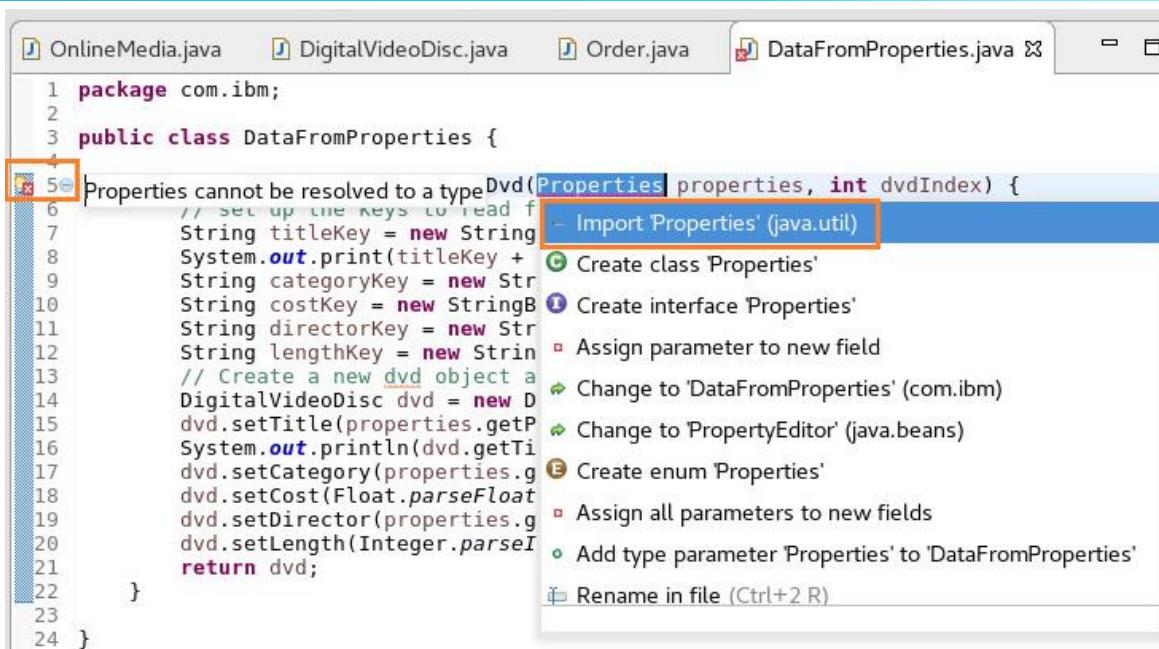
a. Этот метод позволяет получить из файла параметров атрибуты некоторого диска по его индексу.

b. Вы можете использовать следующий код:

```
public DigitalVideoDisc addADvd(Properties properties, int dvdIndex) {  
    String titleKey = new  
    StringBuffer().append("dvd").append(dvdIndex).append  
    (" .title").toString();  
    System.out.print(titleKey + " = ");
```

```
        String categoryKey = new
StringBuffer().append("dvd").append(dvdIndex).append
(".category").toString();
        String costKey = new
StringBuffer().append("dvd").append(dvdIndex).append
(".cost").toString();
        String directorKey = new
StringBuffer().append("dvd").append(dvdIndex).append
(".director").toString();
        String lengthKey = new
StringBuffer().append("dvd").append(dvdIndex).append
(".length").toString();
        // Create a new dvd object and set the fields
        DigitalVideoDisc dvd = new DigitalVideoDisc();
        dvd.setTitle(properties.getProperty(titleKey));
        System.out.println(dvd.getTitle());
        dvd.setCategory(properties.getProperty(categoryKe
y));
        dvd.setCost(Float.parseFloat(properties.getProper
ty(costKey)));
        dvd.setDirector(properties.getProperty(directorKe
y));
        dvd.setLength(Integer.parseInt(properties.getProp
erty(lengthKey)));
        return dvd;
    }
```

- c. После написания этого метода, среда разработки говорит об ошибке. Тип **Properties** не определен для данного класса. Для использования соответствующий класс нужно импортировать.
- d. Нажмите по лампочке слева от кода, сигнализирующей об ошибке. Выберите **Import ‘Properties’ (java.util)**.



е. Стручку с импортом этого класса появляется в начале класса.

ф. Сохраните сделанные изменения.

## Раздел 5. Реализация основной логики приложения **OnlineMedia**.

В основном классе приложения **OnlineMedia** в методе **main** сначала нужно создать экземпляр класса **Order**, потом создать экземпляры дисков, которые нужно добавить в заказ.

1. Реализуйте логику метода **main**.

а. Откройте файл **OnlineMedia.java**.

б. Метод **main** уже есть в этом классе, но его реализация пустая.

с. Замените ее на следующую реализацию:

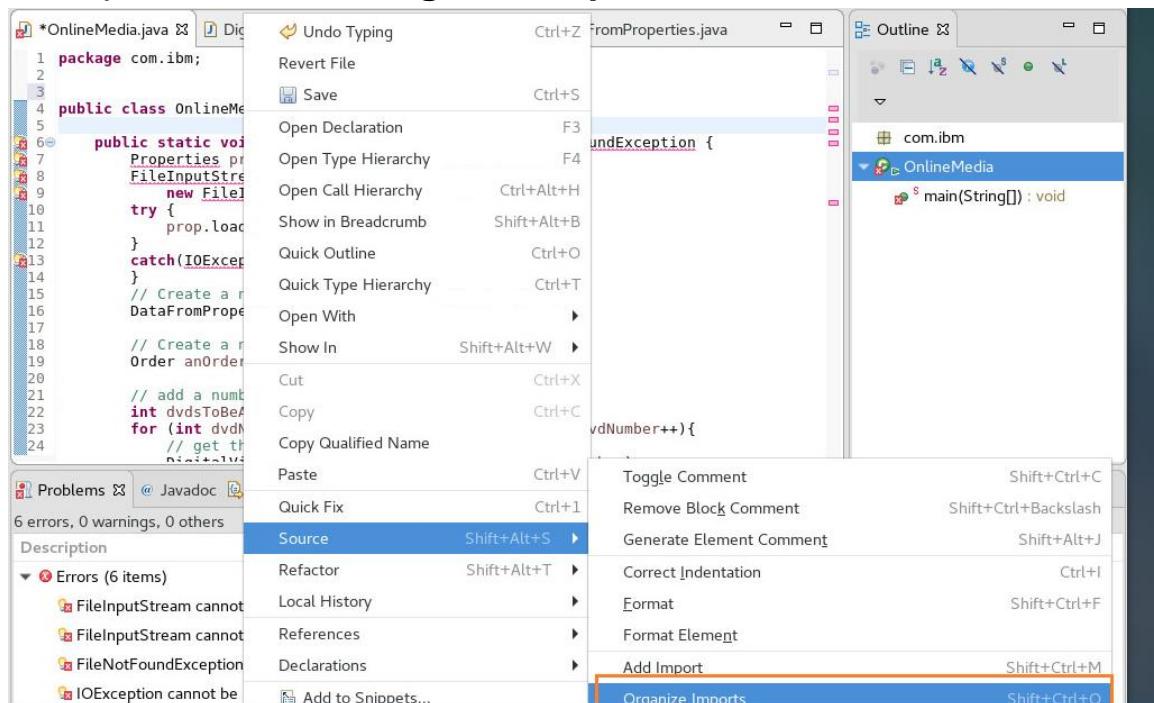
```
public static void main(String[] args) throws
FileNotFoundException {
    Properties prop = new Properties();
    FileInputStream fis =
        new FileInputStream("media.properties");
    try {
        prop.load(fis);
    }
    catch(IOException e) {
    }
    // Create a new object for getting property data
```

```
DataFromProperties data = new  
DataFromProperties();  
// Create a new Order object  
Order anOrder = new Order();  
int dvdsToBeAdded = 3;  
for (int dvdNumber=1; dvdNumber < dvdsToBeAdded;  
dvdNumber++) {  
    DigitalVideoDisc dvd = dvd.addADvd(prop,  
dvdNumber);  
    anOrder.addDigitalVideoDisc(dvd);  
}  
System.out.print("Total Cost is: ");  
System.out.println(anOrder.totalCost());  
}
```

2. Импортируйте необходимые классы для исправления ошибки в коде.

а. Правой кнопкой мыши нажмите в любом месте редактора.

Выберите **Source -> Organize Imports**.

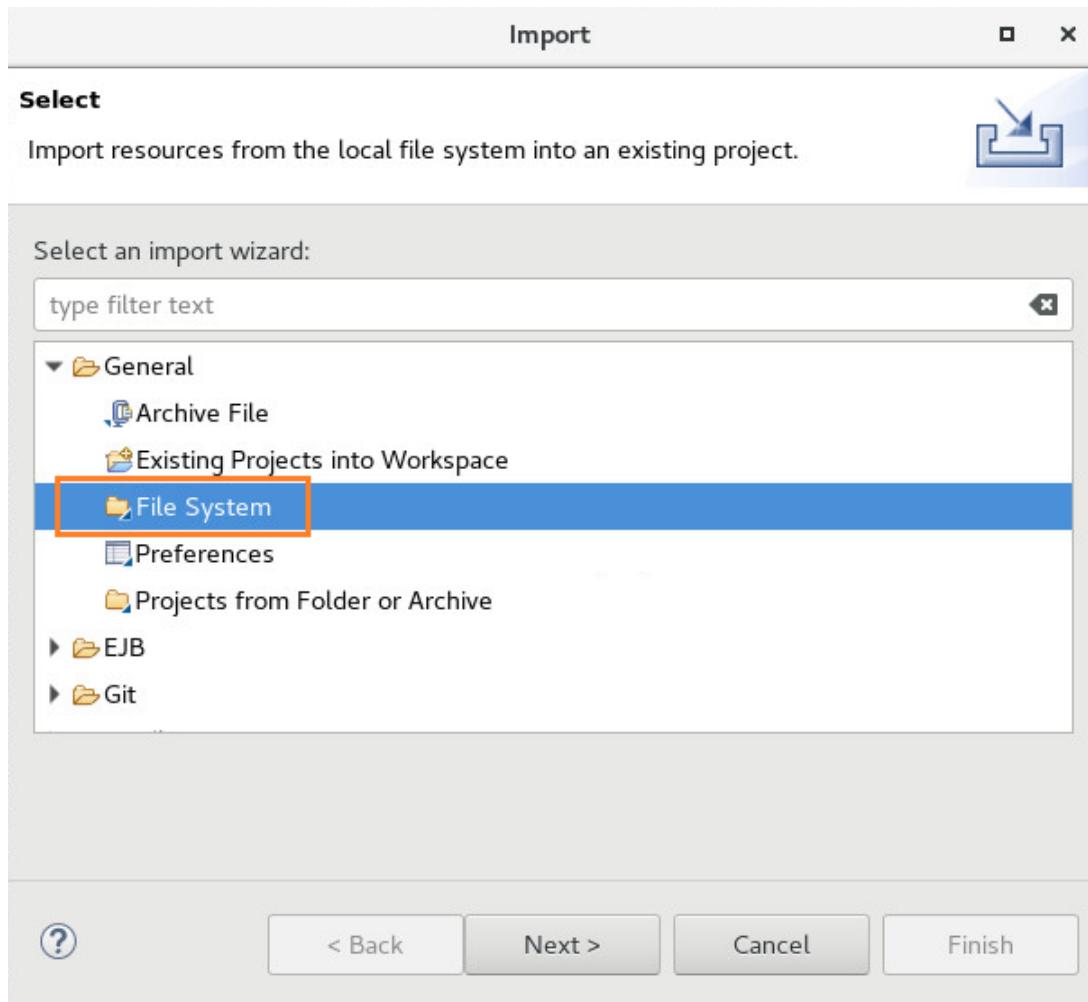


б. Сохраните сделанные изменения.

## **Раздел 6. Импорт в проект файла параметров и запуск приложения.**

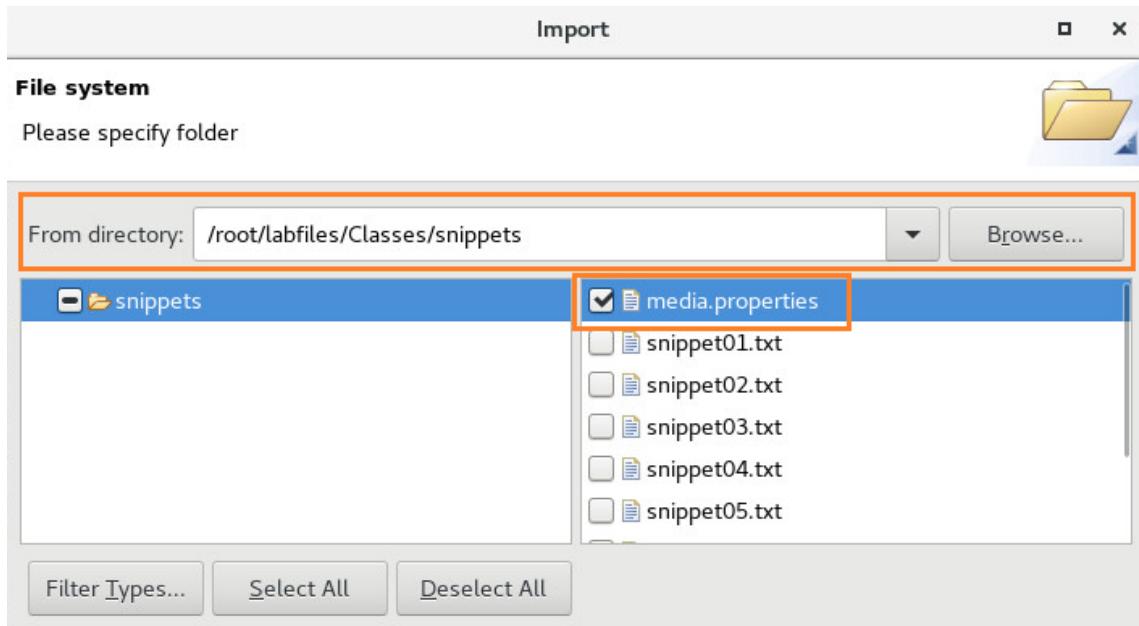
1. Импортируйте файл **media.properties** в проект.

- В главном меню выберите **File -> Import**.
- Выберите **General -> File System**.



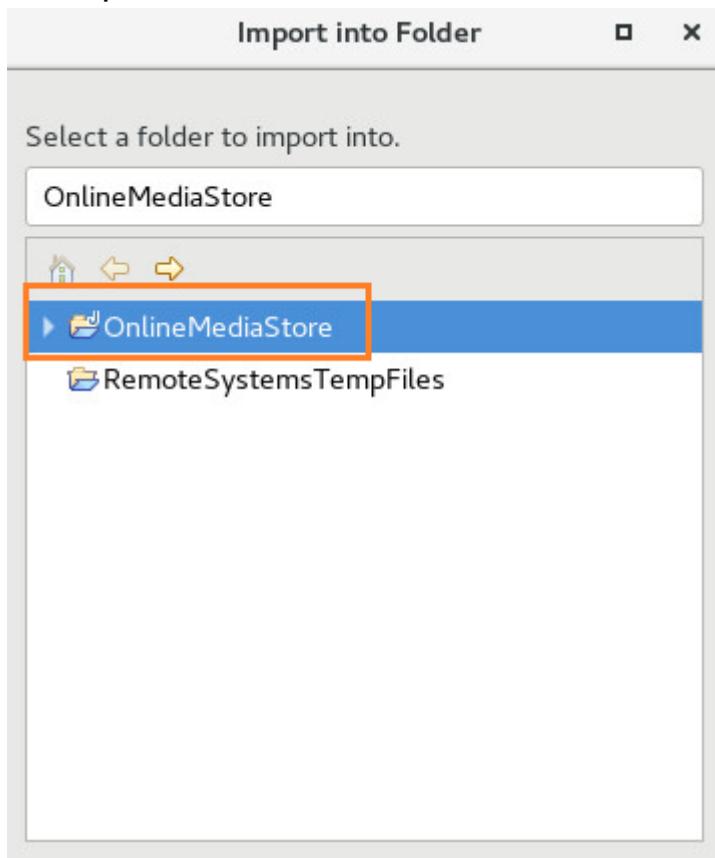
- Нажмите **Next**.
- Нажмите **Browse** рядом с полем **From directory** и выберите каталог **/root/labfiles/Classes/snippets**

е. Выберите только файл **media.properties**.



f. Нажмите **Browse** рядом с полем **Into folder**.

g. Выберите **OnlineMediaStore** и нажмите **OK**.

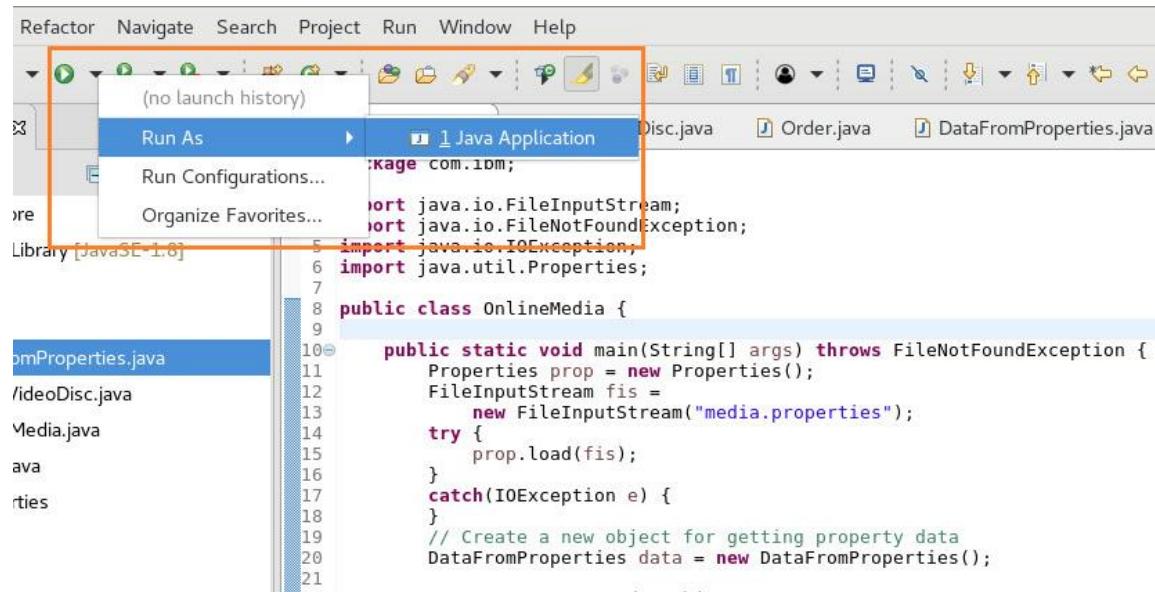


h. Нажмите **Finish**.

i. Файл параметров попадает в проект.

2. Запустите приложение.

- Откройте в редакторе файл **OnlineMedia.java**.
- На панели сверху рядом с кнопкой **Run** нажмите стрелку, выберите **Run As -> Java Application**.



```
Refactor Navigate Search Project Run Window Help
Run As Java Application Disc.java Order.java DataFromProperties.java
Run Configurations...
Organize Favorites...
Library [JavaSE-1.8]
comProperties.java
VideoDisc.java
Media.java
ava
ties
public class OnlineMedia {
    public static void main(String[] args) throws FileNotFoundException {
        Properties prop = new Properties();
        FileInputStream fis =
            new FileInputStream("media.properties");
        try {
            prop.load(fis);
        }
        catch(IOException e) {
        }
        // Create a new object for getting property data
        DataFromProperties data = new DataFromProperties();
    }
}
```

- Результат открывается в представлении **Console**.



```
Problems @ Javadoc Declaration Console
<terminated> OnlineMedia [Java Application] /usr/lib/jvm/java-
dvd1.title = IBM Dance Party
dvd2.title = IBM Kids Sing-along
dvd3.title = IBM Smarter Planet
Total Cost is: 63.89
```

**Конец упражнения**

## Упражнение 4. Отладка Java приложений

---

### О чём это упражнение:

В процессе разработки постоянно возникает необходимость в выполнении некоторых манипуляций, связанных с отладкой приложения. Среда разработки предоставляет соответствующие возможности. Отладка приложений возможна с помощью JVM, находящейся локально на машине разработчика или развернутой удаленно. Вы можете использовать точки останова, пошаговое исполнение, просматривать и менять значения переменных.

В этой лабораторной работе создается приложение, выводящее строки в консоль. Предполагается, что пользователь введет слово `Quit`, для ее завершения, но программа не работает как должно, вместо этого попадая в бесконечный цикл. Предлагается решить эту проблему с помощью указанных инструментов отладки.

### Что вы должны будете сделать:

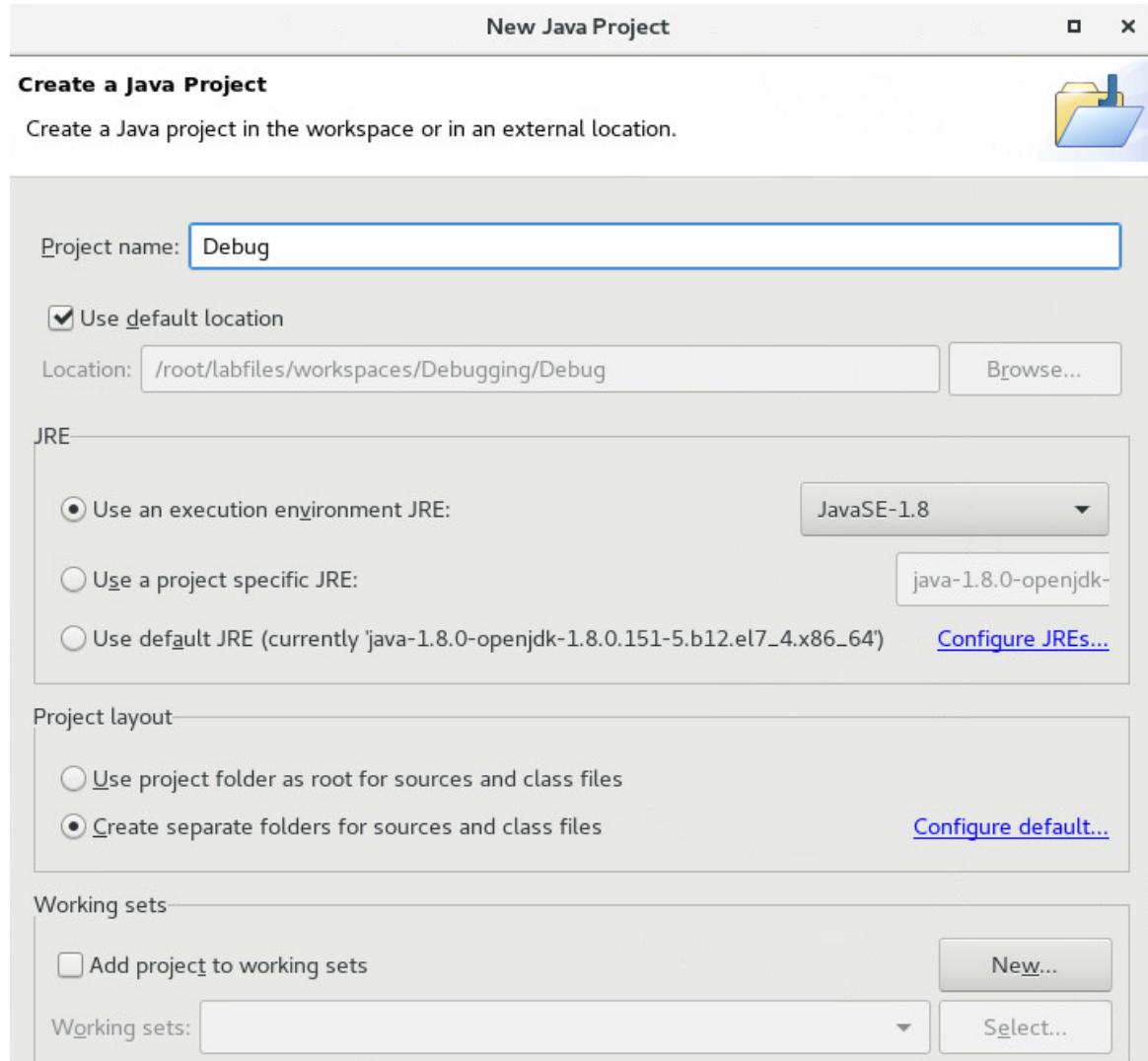
- Запустить приложение в режиме отладки
- Устанавливать точки останова, просматривать и менять значения переменных
- Исполнять код в пошаговом режиме для отладки

## **Раздел 1. Создание логики приложения**

В этом разделе создается новый проект, с главным модулем, анализирующим ввод пользователя, и если он совпадает со словом **Quit**, то приложение должно завершаться. В этом коде допускается умышленная ошибка.

1. Откройте новое рабочее пространство.
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/workspaces/Debugging** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
2. Создайте проект **Debug**
  - a. Перейдите в меню: **File -> New -> Other**, далее выберите **Java Project**. Нажмите **Next**.
  - b. Укажите название **Debug**, значения остальных полей оставьте заполненными по умолчанию.

с. Нажмите **Finish**.

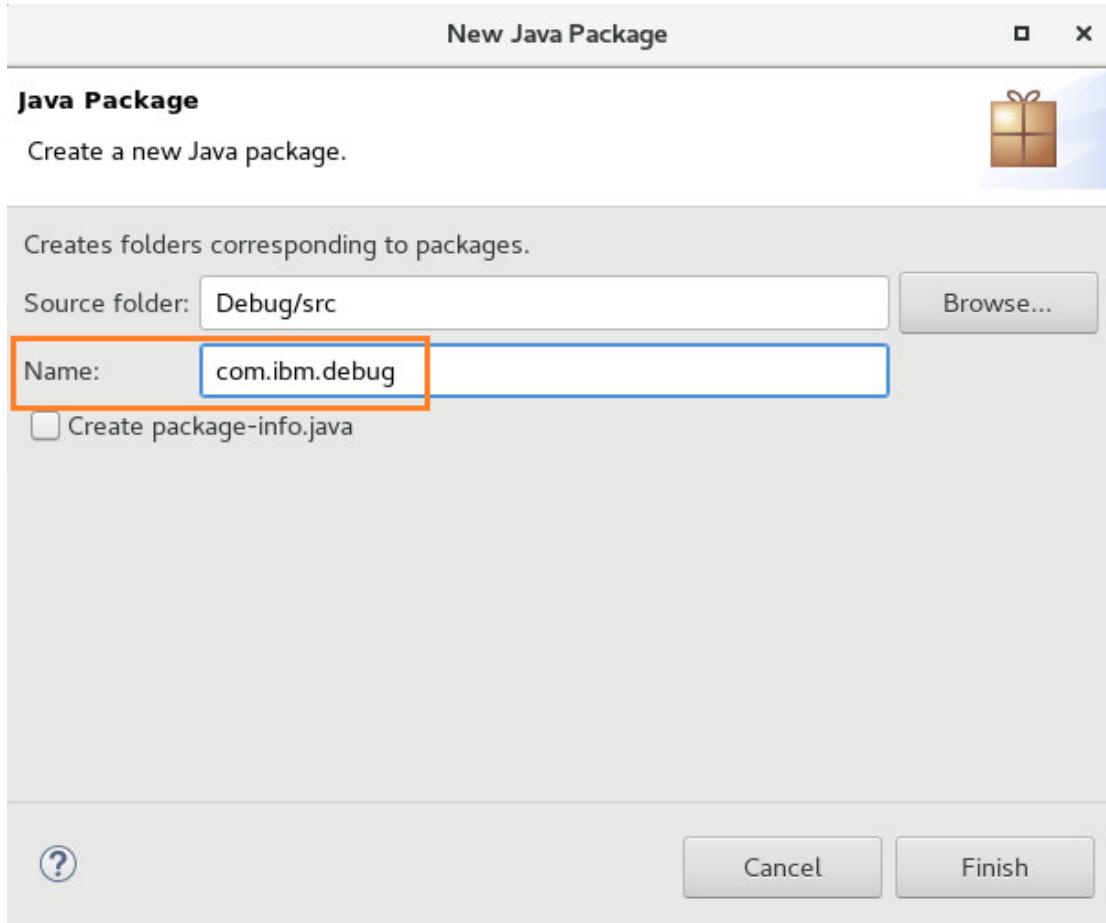


На вопрос об открытии перспективы для Java разработки ответьте согласием: **Open Perspective**.

3. Создайте пакет **com.ibm.debug**.

а. В представлении **Package Explorer** нажмите правой кнопкой мыши по проекту **Debug** и выберите в контекстном меню **New -> Package**.

b. Укажите имя **com.ibm.debug**.



c. Нажмите **Finish**.

4. Создайте класс **ExitCondition**.

a. Из главного меню перейдите: **File -> New -> Class**.

Также можно нажать правой кнопкой мыши по проекту **Debug**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.

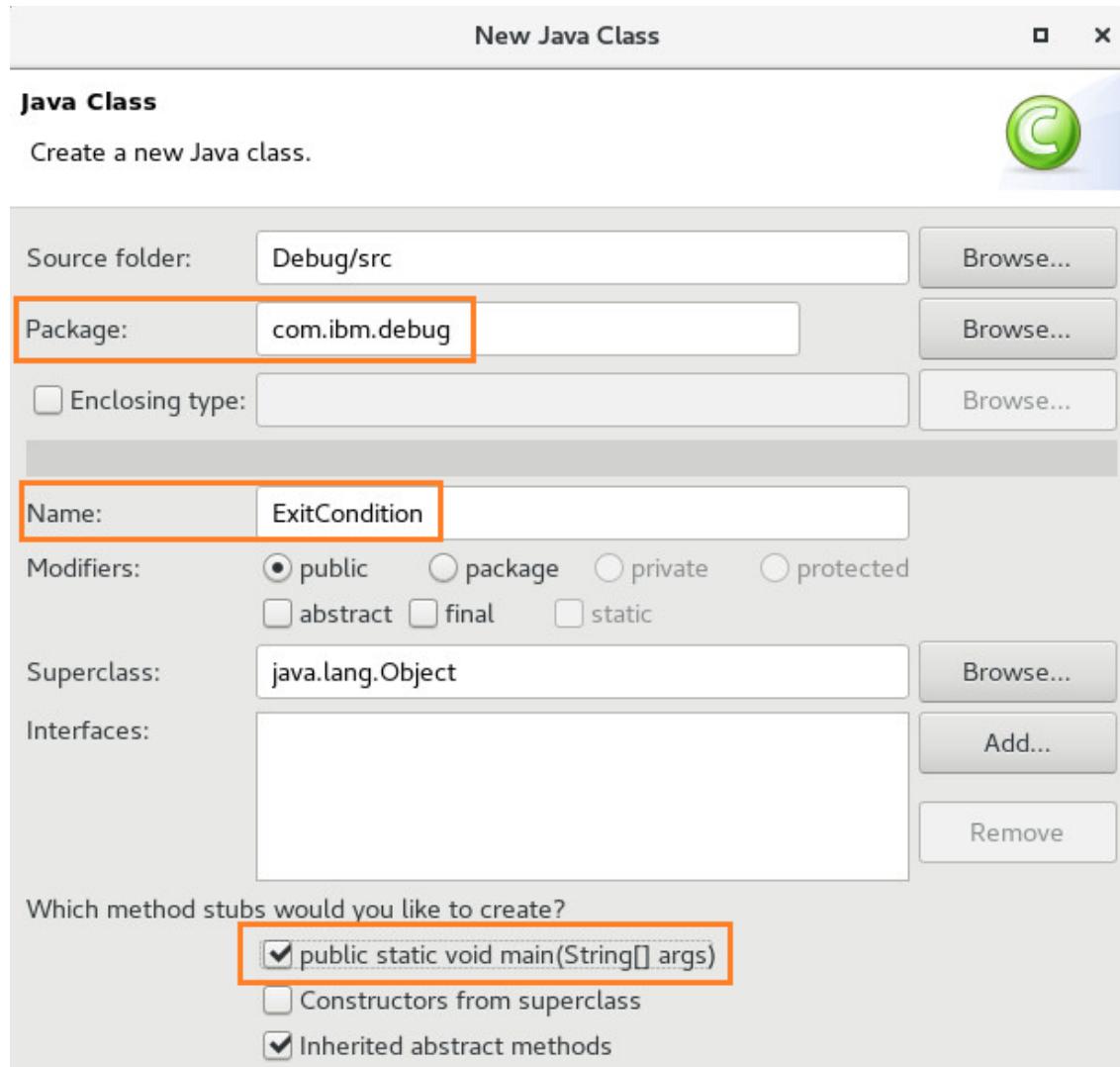
b. В поле **Source Folder** по умолчанию выбрано **Debug/src**.

c. В названии пакета укажите **com.ibm.debug**.

d. Укажите **ExitCondition** как имя класса.

e. Установите опцию **public static void main(String[] args)**. Это добавит в ваш класс метод **main** для того, чтобы запускать этот класс.

f. Остальные значения не меняйте:



g. Нажмите **Finish**.

5. Реализуйте логику класса **ExitCondition**.

a. После указания пакета и до объявления класса введите следующую строку:

```
import java.io.*;
```

Это нужно для импорта классов, связанных с работой системы ввода/вывода.

```
1 package com.ibm.debug;
2
3 import java.io.*;
4
5 public class ExitCondition {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9
10    }
11
12 }
13
```

- b. Используйте следующую реализацию для метода **main**, заменив ей существующую.

```
public static void main(String[] args) throws
IOException {
    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
    StringBuffer s;
    StringBuffer exiting = new StringBuffer("Quit");
    while (true) {
        System.out.println("Enter any string. To
quit, type Quit");
        s = new StringBuffer(br.readLine());
        if (s.equals(exiting)) {
            System.out.println("Goodbye");
            System.exit(0);
        } else {
            System.out.println("You typed \\" + s +
"\\"!");
        }
    }
}
```

```
ExitCondition.java
1 package com.ibm.debug;
2
3 import java.io.*;
4
5 public class ExitCondition {
6
7     public static void main(String[] args) throws IOException {
8         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
9         StringBuffer s;
10        StringBuffer exiting = new StringBuffer("Quit");
11        while (true) {
12            System.out.println("Enter any string. To quit, type Quit");
13            s = new StringBuffer(br.readLine());
14            if (s.equals(exiting)) {
15                System.out.println("Goodbye");
16                System.exit(0);
17            } else {
18                System.out.println("You typed " + s + "!");
19            }
20        }
21    }
22}
23
24
```

c. Сохраните сделанные изменения, при необходимости исправьте возникшие ошибки.

6. Запустите приложение **ExitCondition**.

- a. Нажмите кнопку **Run**, при открытом в редакторе файле **ExitCondition**.
- b. Если представление **Console** не открывается автоматически, его можно открыть через меню: **Window -> Show View -> Console**.
- c. В консоли отображается приветствие

```
Problems @ Javadoc Declaration Console
ExitCondition [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0
Enter any string. To quit, type Quit
```

- d. Введите в консоли **line 1** и нажмите **Enter**. Это **не должно** приводить к завершению приложения.

```
Problems @ Javadoc Declaration Console
ExitCondition [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-
Enter any string. To quit, type Quit
line 1
You typed "line 1"!
Enter any string. To quit, type Quit
```

- e. Введите в консоли **Quit** и нажмите **Enter**. Это **должно** приводить к завершению приложения. Но в текущей логике приложения не приводит.

```
ExitCondition [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-
Enter any string. To quit, type Quit
line 1
You typed "line 1"!
Enter any string. To quit, type Quit
Quit
You typed "Quit"!
Enter any string. To quit, type Quit
```

- f. Для того, чтобы решить эту проблему (почему программа не завершается при вводе **Quit**) необходимо воспользоваться инструментами для отладки.  
g. Завершите работу приложения, нажав кнопку **Terminate**.

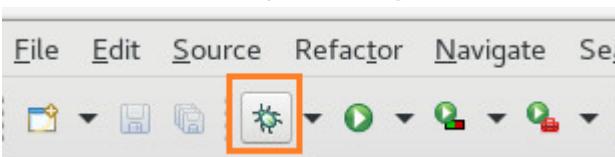
```
1 package com.ibm.debug;
2
3 import java.io.*;
4
5 public class ExitCondition {
6
7     public static void main(String[] args) throws IOException {
8         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
9         StringBuffer s;
10        StringBuffer exiting = new StringBuffer("Quit");
11        while (true) {
12            System.out.println("Enter any string. To quit, type Quit");
13            s = new StringBuffer(br.readLine());
14            if (s.equals(exiting)) {
15                System.out.println("Goodbye");
16                System.exit(0);
17            } else {
18                System.out.println("You typed \"" + s + "\"!");
19            }
20        }
21    }
}
```

ExitCondition [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-5.b12.el7\_4.x86\_64/bin/java Jan 1  
Enter any string. To quit, type Quit  
line 1  
You typed "line 1"!  
Enter any string. To quit, type Quit  
Quit  
You typed "Quit"!  
Enter any string. To quit, type Quit

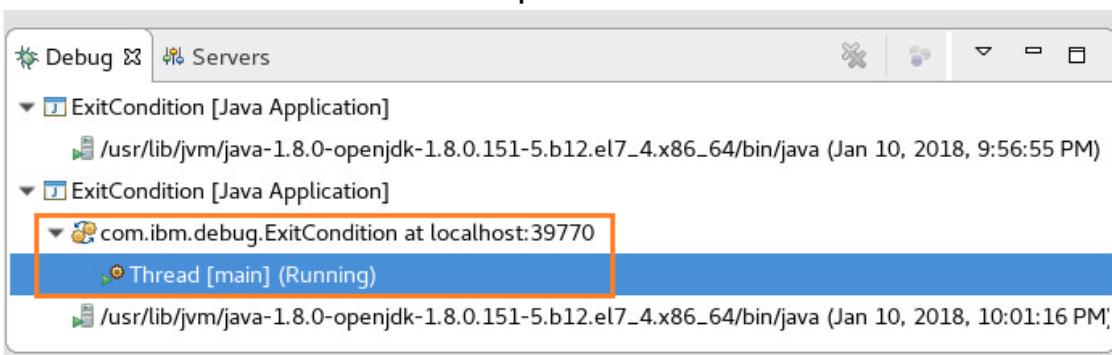
## Раздел 2. Использование отладчика

В этом разделе с использованием различных инструментов для отладки вы пытаетесь решить проблему, поставленную выше.

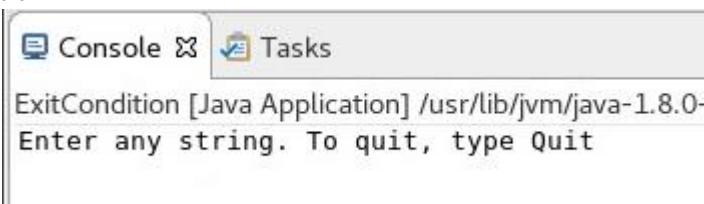
1. Запустите приложение в режиме отладки.
  - a. В представлении **Package Explorer** раскройте **Debug -> com.ibm.debug**.
  - b. Выберите файл **ExitCondition.java**.
  - c. Нажмите кнопку **Debug** в панели инструментов.



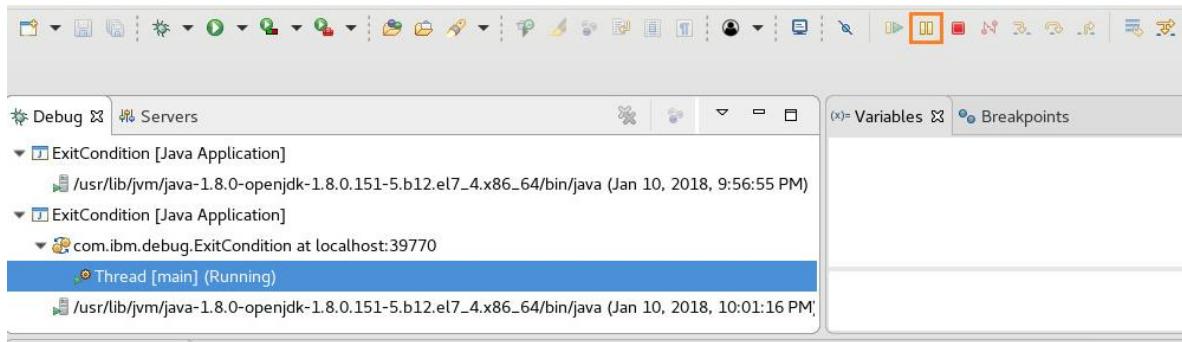
- d. Если среда разработки предлагает открыть перспективу (проекцию) **Debug**, согласитесь с этим – **Open Perspective**. Если предложение не появляется, откройте эту перспективу вручную: **Window -> Perspective -> Open Perspective -> Debug**.
- e. Открывается перспектива **Debug**. В представлении **Debug** показывается список потоков, исполняемых в данный момент. Сейчас исполняется только приложение **ExitCondition**.



- f. В редакторе можно установить точки останова (**breakpoint**) и исполнять код в пошаговом режиме.
- g. В консоли отображается вывод приложения и можно вводить данные.

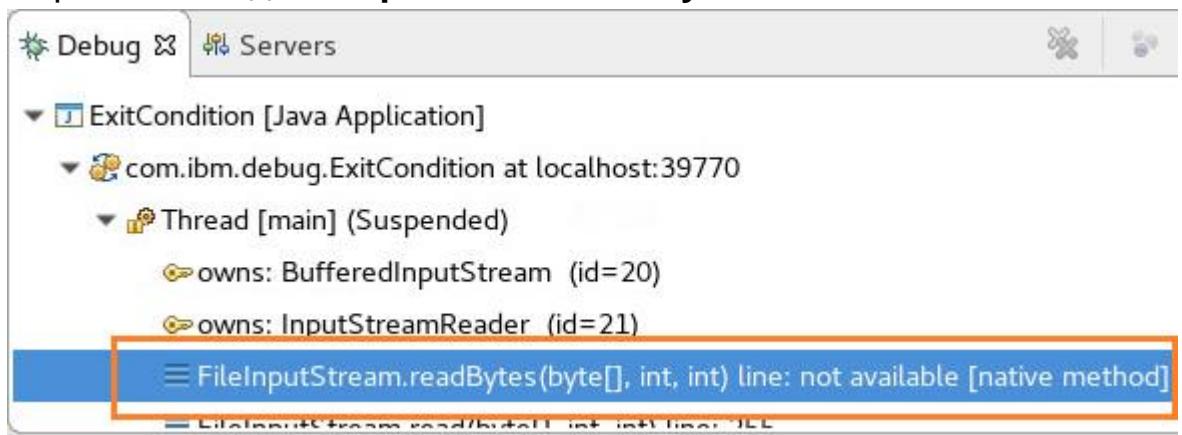


- h. Введите данные, например, **Quit** и нажмите **Enter**.
2. Приостановите исполнение приложения.
- В представлении **Debug** выберите поток **Thread [main]**.
  - Нажмите кнопку **Suspend**.



Отладчик приостанавливает основной поток и в редакторе открывается класс, метод которого исполнялся при выполнении этой операции. В данном случае программа ожидала ввода пользователя, и работал метод **readBytes()** в классе **FileInputStream**.

3. Посмотрите верхний метод стэка трейса.
- В представлении **Debug** раскройте thread main:
  - Верхний метод **FileInputStream.readBytes**.



4. Посмотрите нижний метод стэка трейса.
- В представлении **Debug** выберите нижний метод **ExitCondition.main(String[])**.

b. Он автоматически открывается в редакторе.

The screenshot shows the IntelliJ IDEA interface with the 'Debug' tool window open. The stack trace at the top lists several method calls, with the last one, 'ExitCondition.main(String[]) line: 13', highlighted by a red box. Below the stack trace, the code editor displays 'ExitCondition.java'. Line 13 of the code is also highlighted with a red box: 

```
13 |     s = new StringBuffer(br.readLine());
```

 This indicates that the debugger has stopped execution at this specific line of code.

5. Посмотрите значения переменных приложения.

- Убедитесь, что метод **ExitCondition.main(String[])** выбран в представлении **Debug**.
- Посмотрите первую переменную **args**. Эта переменная пустая, так как запуская приложение, вы не передавали ему на вход аргументов.

The screenshot shows the 'Variables' tool window in the IntelliJ IDEA debugger. It lists the current variable values:

Name	Value
args	String[0] (id=31)
br	BufferedReader (id=30)
exitting	StringBuffer (id=36)
[[ ]]	

- Посмотрите значения других переменных: **br** и **exitting**. Для любых типов данных кроме примитивных типов и строк требуется воспользоваться кнопкой **+**.
- Раскройте значение **exitting -> value**. Там вы увидите значение, зафиксированное в коде приложения.

(x)= Variables		Breakpoints
Name		Value
▼ ▲ value		(id=42)
▲ [0]		Q
▲ [1]		u
▲ [2]		i
▲ [3]		t
▲ [4]		
[Q, u, i, t,		

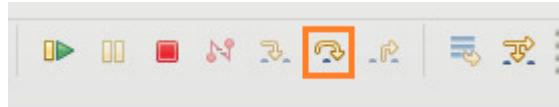
### Раздел 3. Пошаговый проход по коду

В этом разделе вы познакомитесь с функциями пошаговой отладки. Пошаговая отладка может выполняться с использованием 4 базовых функций:

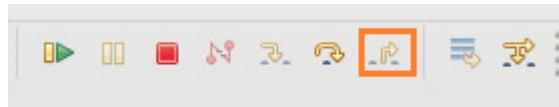
**Step Into** – позволяет выполнять приложение по шагам с погружением на новые уровни вложенности. То есть, отладчик **переходит** в класс/функцию каждого вызываемого метода.



**Step Over** – позволяет выполнять приложение по шагам без погружения на новые уровни вложенности. То есть, отладчик **не переходит** в класс/функцию каждого вызываемого метода.

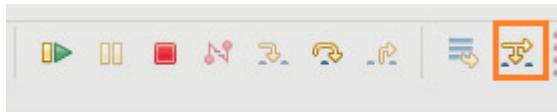


**Step Return** – позволяет отладчику **вернуться на более высокий уровень** вложенности, приостанавливая отладку на текущем уровне.



**Use Step Filters/Step Debug** – позволяетходить на те уровни вложенности, для которых есть исходные коды. То есть, если какой-то класс есть только в скомпилированном виде, то отладчик не будет переходить на этот уровень вложенности, что в большинстве случаев не

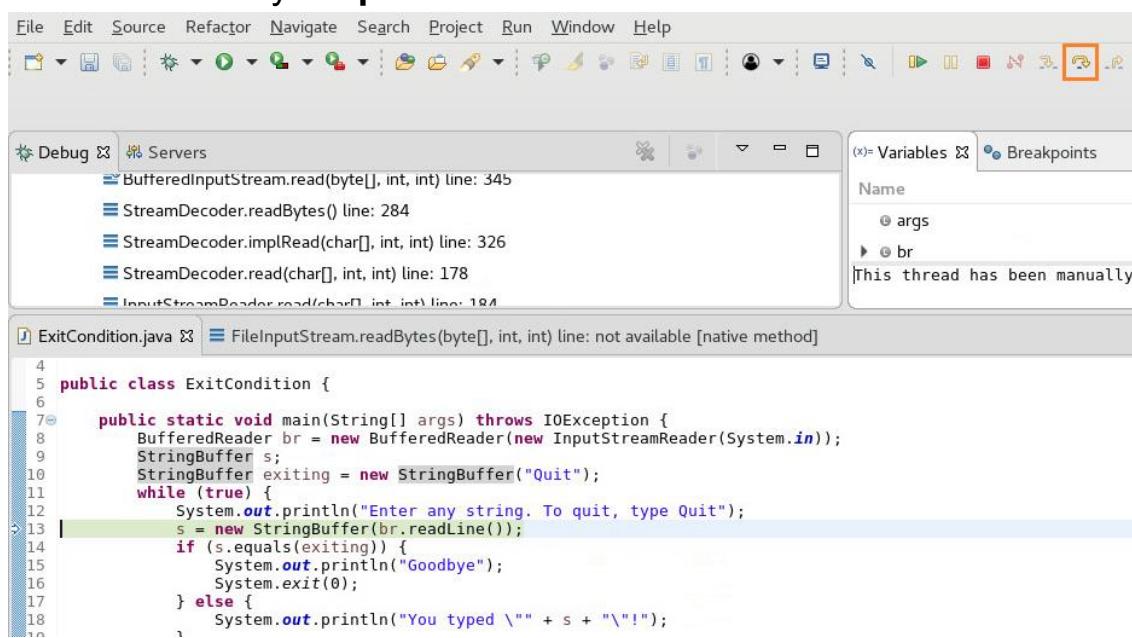
имеет смысла.



1. Продолжите выполнение программы.

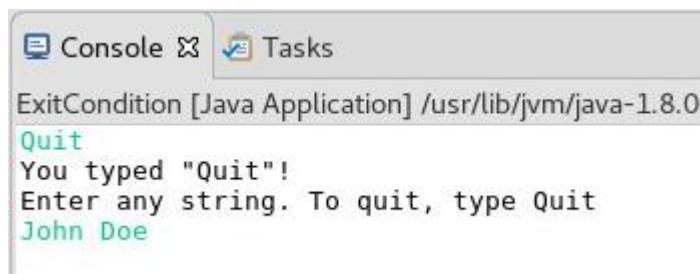
- a. Убедитесь, что программа заморожена при исполнении строчки
- ```
s = new StringBuffer(br.readLine());
```

b. Нажмите кнопку **Step Over**.



c. Перейдите в представление **Console**.

- d. После приветствия **Enter any string** введите **John Doe** и нажмите **Enter**.



- e. Посмотрите переменную **br** и раскройте блок **cb**. Там отобразятся введенные данные.

| Name                            | Value                  |
|---------------------------------|------------------------|
| br                              | BufferedReader (id=30) |
| cb                              | (id=44)                |
| [0..99]                         |                        |
| [0]                             | J                      |
| [1]                             | o                      |
| [2]                             | h                      |
| [3]                             | n                      |
| [4]                             |                        |
| [5]                             |                        |
| [6]                             |                        |
| [7]                             |                        |
| [8]                             |                        |
| [9]                             |                        |
| java.io.BufferedReader@7e774085 |                        |

Функция **Step Into** заблокирована, так как в распоряжении среды разработки нет исходных кодов класса BufferedReader.

f. С помощью кнопки **Step Over** доберитесь до строчки

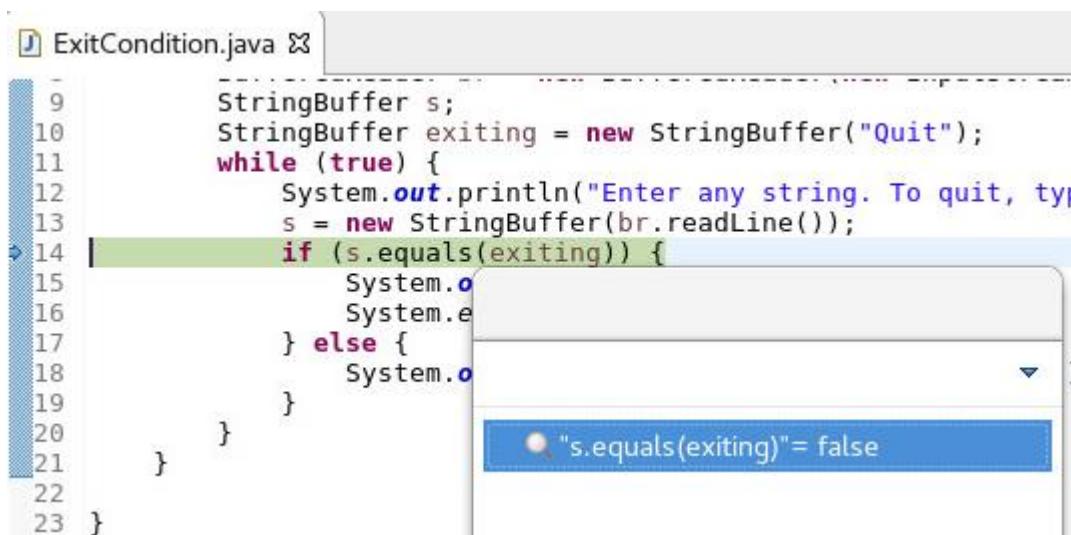
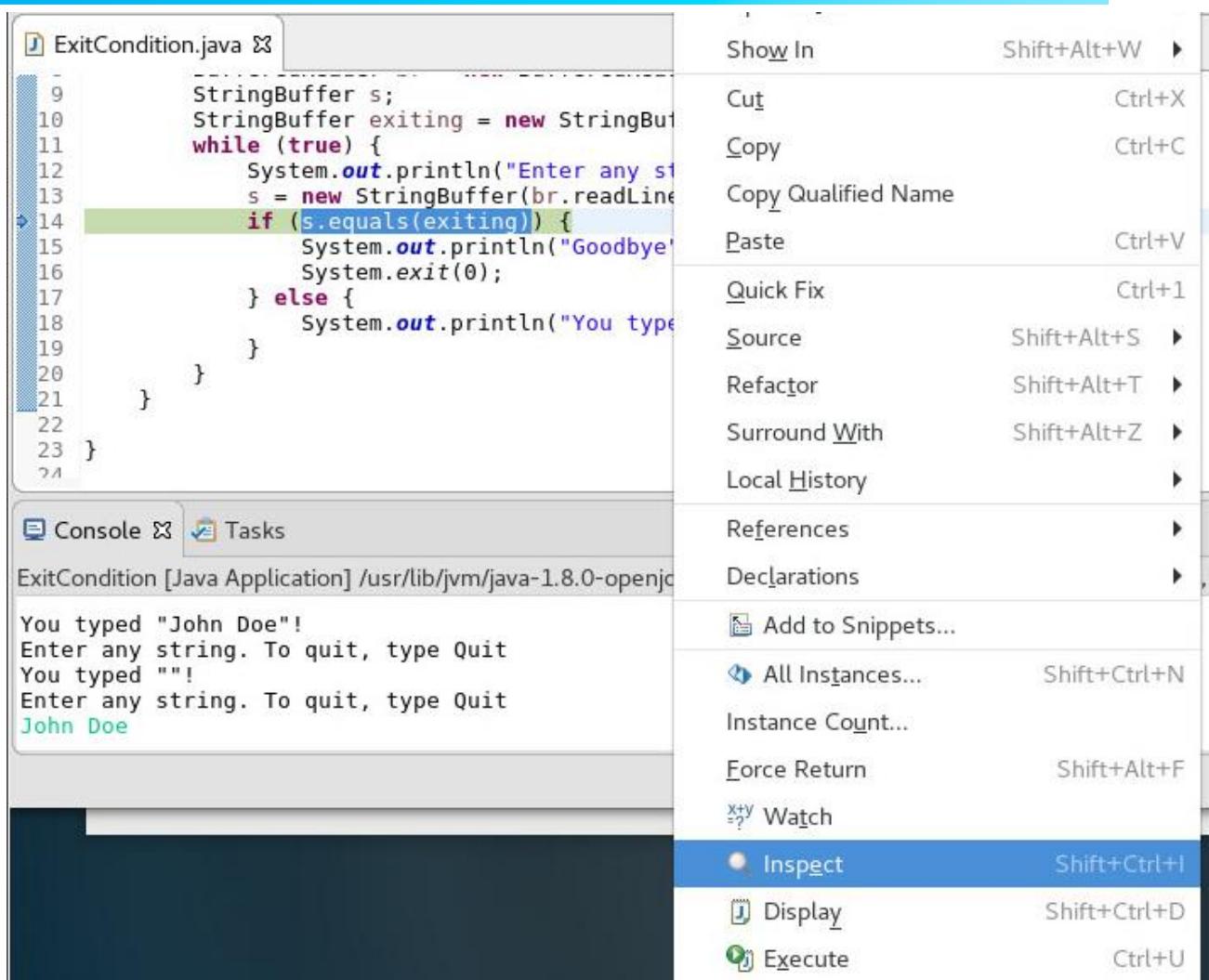
```
if (s.equals(exiting))
```

g. Сравните значения переменных **s** и **exiting**. В данный момент в одной хранится значение **John Doe**, а в другой **Quit**.

| Name    | Value                  |
|---------|------------------------|
| br      | BufferedReader (id=30) |
| cb      | (id=44)                |
| [0..99] |                        |
| [0]     | J                      |
| [1]     | o                      |
| [2]     | h                      |
| [3]     | n                      |
| [4]     |                        |
| [5]     | D                      |
| [6]     | o                      |
| [7]     | e                      |
| [8]     | \n                     |
| [9]     |                        |

| Name                               | Value                  |
|------------------------------------|------------------------|
| ▶ <code>readLine()</code> returned | "John Doe" (id=68)     |
| ● args                             | String[0] (id=31)      |
| ▶ ● br                             | BufferedReader (id=30) |
| ▼ ● exiting                        | StringBuffer (id=36)   |
| △ count                            | 4                      |
| ▶ □ <code>toStringCache</code>     | (id=73)                |
| ▼ + value                          | (id=42)                |
| △ [0]                              | Q                      |
| △ [1]                              | u                      |
| △ [2]                              | i                      |
| △ [3]                              | t                      |
| △ [4]                              |                        |
| △ [5]                              |                        |

h. С учетом этого факта условный переход разрешается отрицательно. Убедитесь с помощью функции **Inspect**, что условие разрешается в значение **false**.



- Правой кнопкой мыши нажмите по строке  
`s = new StringBuffer(br.readLine());`

Выберите в контекстном меню **Run to Line**.

- j. В консоли отобразятся еще раз введенные данные и появится начальное приветствие.

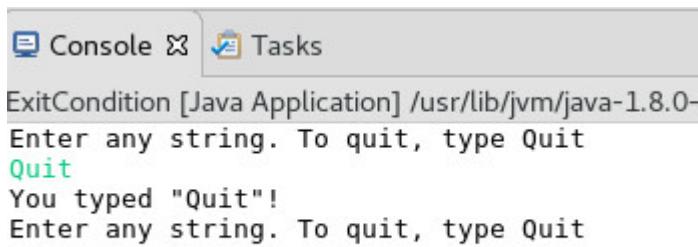
#### **Раздел 4. Отладка условия выхода**

В этом разделе вы решите проблему с программой, которая не завершается при соответствующем вводе.

1. Выясните, почему программа не завершается.
  - a. Перезапустите приложение в режиме Debug.
  - b. Установите breakpoint на строчке  
**s = New StringBuffer(br.readLine());**

Для этого нужно нажать в этой строке правой кнопкой мыши в области левее номера данной строки и выбрать **Toggle Breakpoint**;

- c. Перейдите в представление **Console**.
- d. Введите **Quit** и нажмите **Enter**.



```
Console Tasks
ExitCondition [Java Application] /usr/lib/jvm/java-1.8.0-
Enter any string. To quit, type Quit
Quit
You typed "Quit"!
Enter any string. To quit, type Quit
```

- e. Нажмите **Step Over**. Это приведет код к условному переходу, где проверяется условие выхода.

f. Сравните значения переменных **s** и **exitting**. Они обе равны **Quit**.

| Name     | Value                  |
|----------|------------------------|
| args     | String[0] (id=27)      |
| br       | BufferedReader (id=26) |
| cb       | (id=40)                |
| [0...99] |                        |
| [0]      | Q                      |
| [1]      | u                      |
| [2]      | i                      |
| [3]      | t                      |
| [4]      | \n                     |

| Name          | Value                  |
|---------------|------------------------|
| args          | String[0] (id=27)      |
| br            | BufferedReader (id=26) |
| exitting      | StringBuffer (id=34)   |
| count         | 4                      |
| toStringCache | null                   |
| value         | (id=43)                |
| [0]           | Q                      |
| [1]           | u                      |
| [2]           | i                      |
| [3]           | t                      |
| [4]           |                        |

2. С учетом того, что значения идентичны, логичным выглядел бы положительный результат данного условного перехода. Убедитесь, что это не так.
- а. Выделите подстроку **s.equals(exitting)** и посмотрите ее значение с помощью **Inspect**.

```
9     StringBuffer s;
10    StringBuffer exiting = new StringBuffer("Quit");
11    while (true) {
12        System.out.println("Enter any string. To quit, type 'Quit'");
13        s = new StringBuffer(br.readLine());
14        if (s.equals(exiting)) {
```

if (s.equals(exiting)) {

```
15            System.out.println("You entered " + s);
16            System.exit(0);
17        } else {
18            System.out.println("You entered " + s);
19        }
20    }
21 }
22 }
```

- b. С помощью кнопки **Step Over**, убедитесь, что вы попадаете в секцию **else**.
3. Так почему же условие не разрешается положительно?
- Выделите в редакторе выражение **s.equals(exiting)**.
  - Попробуйте с нажатой кнопкой **Ctrl** нажать по методу **equals**. Это должно привести к попытке открыть исходный код этого метода. Он не открывается, так как в данной версии JDK нет исходных файлов для данного класса. Но следует обратить внимание на сам класс. Это **Object**.

ExitCondition.java Object.class

**Class File Editor**

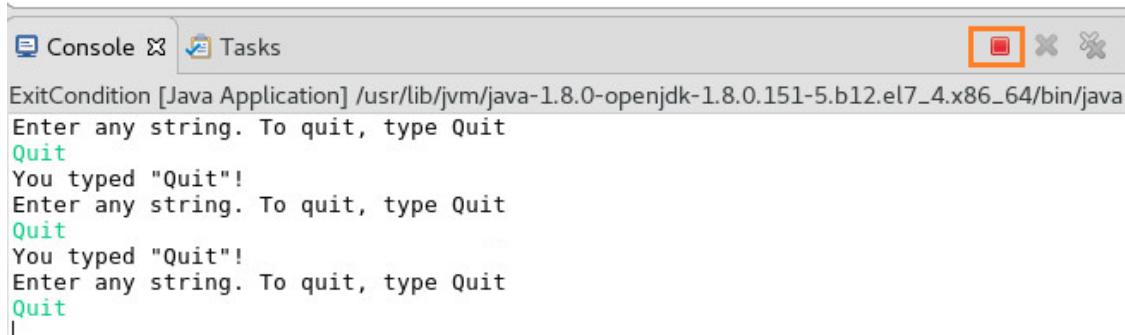
**Source not found**

The JAR file /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-5.b12.el7\_4.x86\_64/jre/lib/rt.jar has no source attachment.  
You can attach the source by clicking Attach Source below:

А для этого класса метод **equals** предполагает сравнение ссылок на оба объекта. Так как в нашем случае речь идет о разных объектах, ссылки на них отличаются. Соответственно, метод работает не так, как предполагалось изначально.

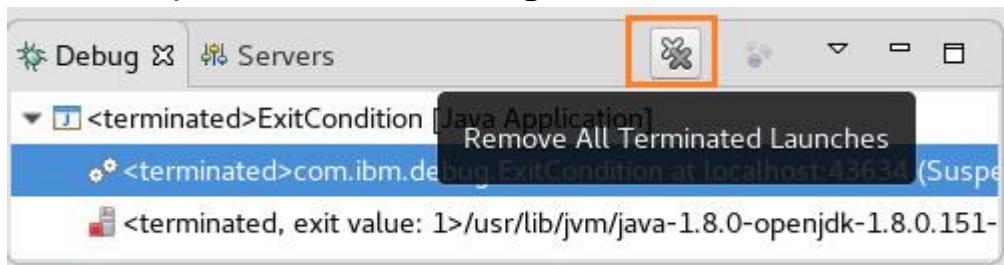
4. Исправьте код приложения.

a. Остановите приложение, нажав кнопку **Terminate**.

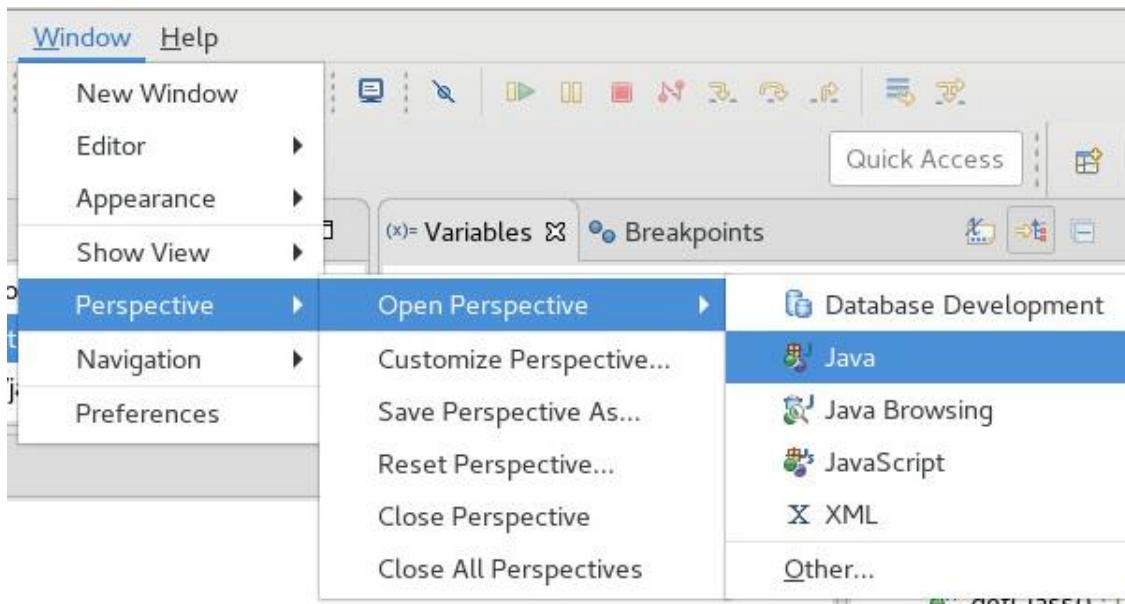


```
Console Tasks
ExitCondition [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-5.b12.el7_4.x86_64/bin/java
Enter any string. To quit, type Quit
Quit
You typed "Quit"!
Enter any string. To quit, type Quit
Quit
You typed "Quit"!
Enter any string. To quit, type Quit
Quit
```

b. Нажмите кнопку **Remove All Terminated Launches**. Это нужно для очистки представления **Debug**.



c. Перейдите в перспективу **Java**. **Window -> Perspective -> Open Perspective -> Java**.



d. Откройте в редакторе файл **ExitCondition.java**.

e. Установите курсор в самом начале файла.

f. Выберите в меню **Edit -> Find/Replace**.

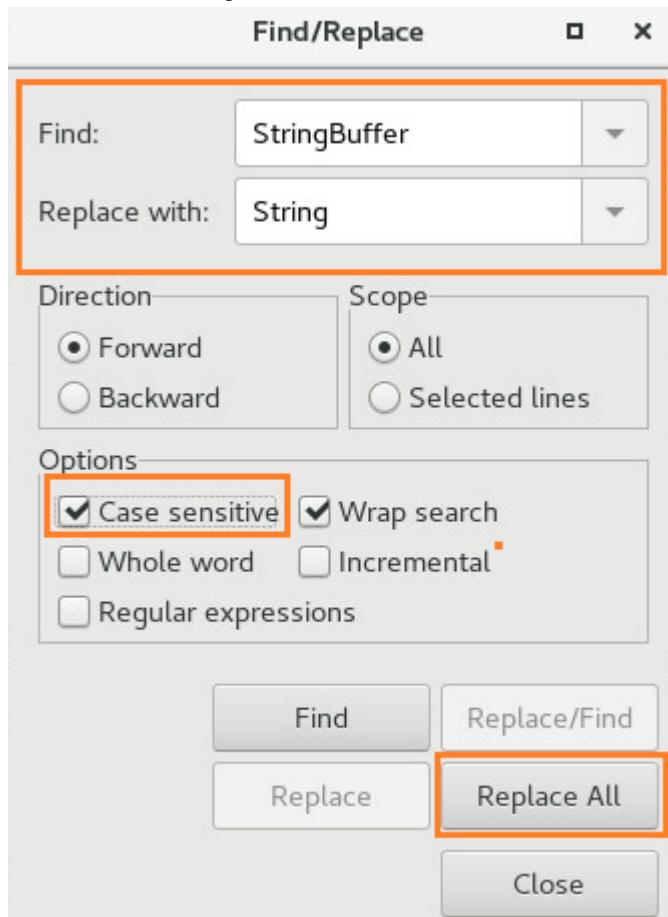
g. Заполните поля для поиска/замены следующим образом:

**Find:** **StringBuffer**

**Replace with:** **String**

**Case Sensitive:** **выбрано**

h. Нажмите **Replace All**.



i. Нажмите **Close**.

5. Сохраните сделанные изменения.

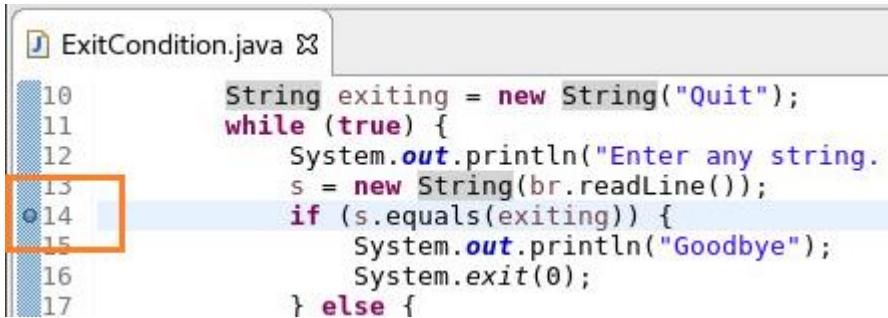
6. Установите точку останова на строке `if (s.equals(exiting))`.

а. Установите курсор на этой строчке.

б. Выберите в главном меню: **Run -> Toggle Line Breakpoint**.

с. Рядом с данной строчкой в редакторе появляется голубая точка, символизирующая breakpoint.

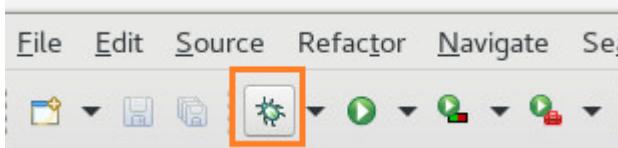
Аналогичным образом уберите другие breakpoints, если они есть в этом классе.



```
10     String exiting = new String("Quit");
11     while (true) {
12         System.out.println("Enter any string.");
13         s = new String(br.readLine());
14         if (s.equals(exiting)) {
15             System.out.println("Goodbye");
16             System.exit(0);
17         } else {
```

7. Протестируйте новую версию приложения.

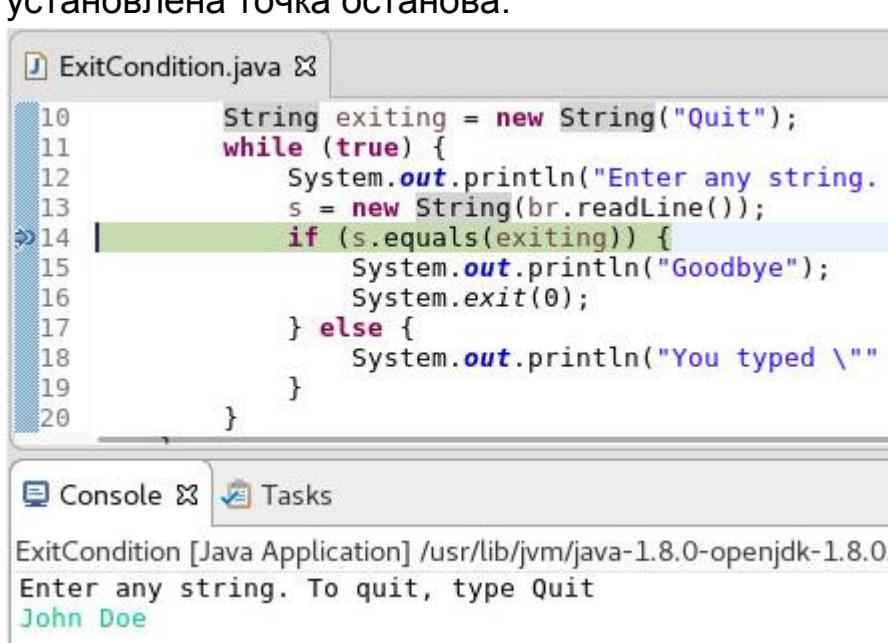
a. Нажмите кнопку **Debug** в панели инструментов.



b. Откройте перспективу **Debug**. **Window -> Perspective -> Open Perspective -> Debug**.

c. В консоли введите **John Doe** и нажмите **Enter**.

Исполнение программы останавливается на строке, где была установлена точка останова.

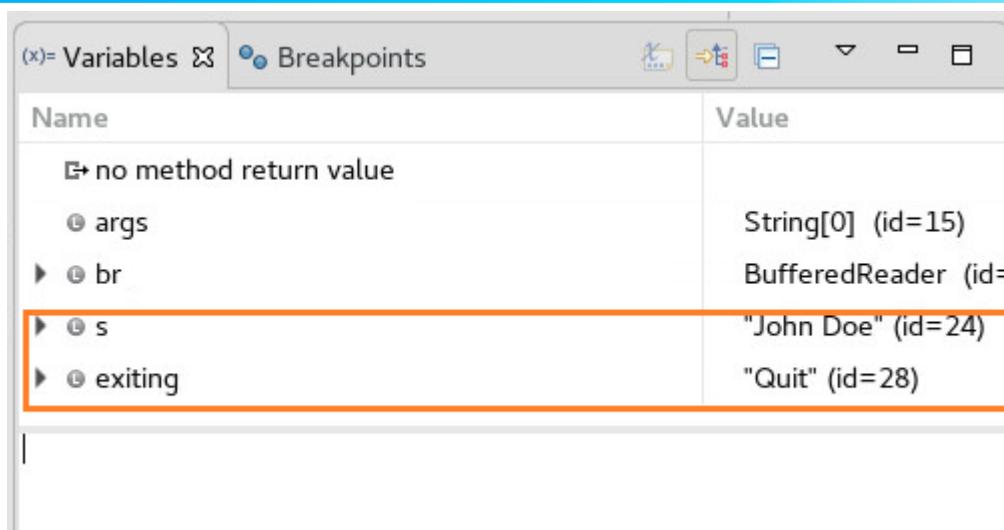


```
10     String exiting = new String("Quit");
11     while (true) {
12         System.out.println("Enter any string.");
13         s = new String(br.readLine());
14         if (s.equals(exiting)) {
15             System.out.println("Goodbye");
16             System.exit(0);
17         } else {
18             System.out.println("You typed \'"
19         }
20     }
```

Console

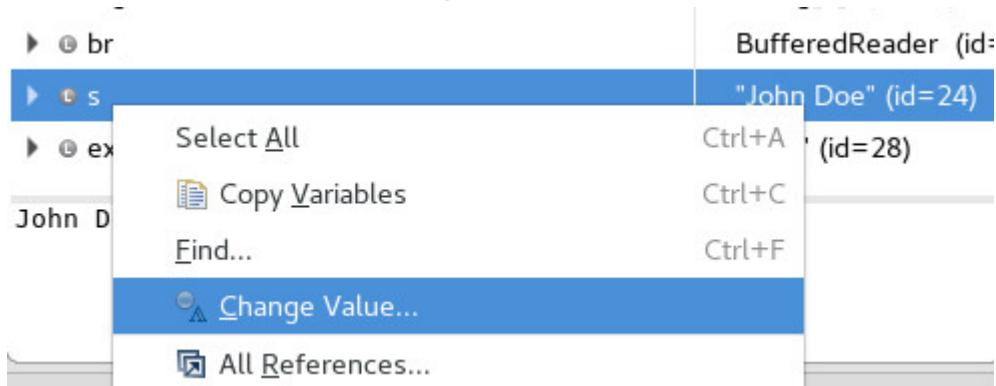
```
ExitCondition [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.
Enter any string. To quit, type Quit
John Doe
```

d. Строки **s** и **exiting** отображаются в секции **Variables** уже без дополнительных шагов. Это происходит из-за использования класса **String**.

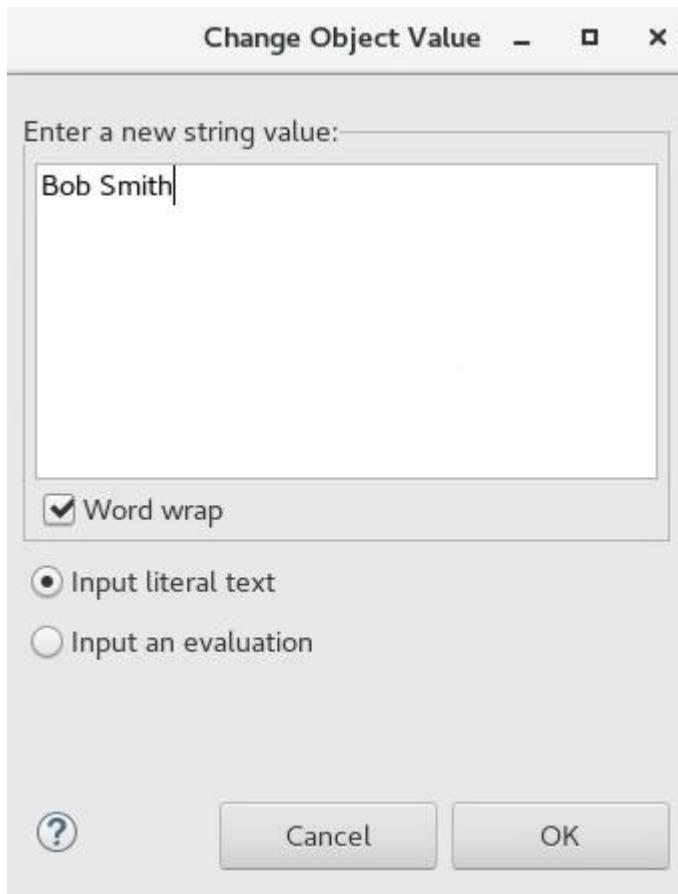


8. Измените значение переменной с помощью отладчика.

- В представлении **Variables** выберите **s**.
- Правой кнопкой мыши вызовите контекстное меню.
- Выберите вариант **Change Value**.

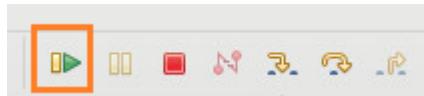


- Ведите **Bob Smith** и нажмите **OK**. Значение переменной поменяется.



| Name                     | Value               |
|--------------------------|---------------------|
| ↳ no method return value |                     |
| ↳ args                   | String[0] (id=15)   |
| ↳ br                     | BufferedReader (id= |
| ↳ s                      | "Bob Smith" (id=30) |
| ↳ exiting                | "Quit" (id=28)      |
| Bob Smith                |                     |

е. Нажмите кнопку **Resume**.



9. Проверьте работу программы при вводе **Quit**.

- В консоли введите **Quit**.
- Исполнение замораживается на точке останова.
- Сравните значения переменных **s** и **exiting**.

d. Выделите подстроку `s.equals(exiting)` и посмотрите ее значение с помощью **Inspect**.

The screenshot shows an IDE interface with two tabs: 'ExitCondition.java' and 'Console'. In the code editor, line 14 is highlighted with a green selection bar, containing the code: `s.equals(exiting)`. A tooltip window titled 'Inspect' is open over this line, displaying the value: `"s.equals(exiting)" = true`. In the 'Console' tab, the application's output is visible, showing the user input 'John Doe' and the program's response 'You typed "Bob Smith"!'. The status bar at the bottom right of the IDE says 'Press Shift+Ctrl+I to Move to Expressions View'.

e. Нажмите **Step Over**.

f. Убедитесь, что теперь условный переход предсказуемо разрешился положительно.

The screenshot shows the IDE with the code editor open. Line 15 is highlighted with a green selection bar, containing the code: `System.out.println("Goodbye"); System.exit(0);`. This indicates that the previous step (Step Over) has been completed, and the execution flow has moved to the else block of the if statement. The rest of the code in the loop is shown below.

g. Нажмите кнопку **Resume**. Убедитесь, что программа завершается и появляется сообщение **Goodbye**.

The screenshot shows the 'Console' tab with the following output:  
<terminated> ExitCondition [Java Application] /usr/lib/jvm/java-  
Enter any string. To quit, type Quit  
John Doe  
You typed "Bob Smith"!  
Enter any string. To quit, type Quit  
Quit  
Goodbye

h. Перейдите в перспективу **Java**. **Window -> Perspective -> Open Perspective -> Java**.

**Конец упражнения**

## Упражнение 5. Наследование и рефакторинг

---

### О чём это упражнение:

В этой лабораторной работе рассматриваются возможности, связанные с наследованием классов. Потом делается акцент на функциях Eclipse, связанных с рефакторингом.

В этом упражнении создается новый класс **Book**. Очевидно, что многие его атрибуты будут совпадать с атрибутами класса **DigitalVideoDisc**. В связи с этим организуется наследование: создается базовый класс **Media**, а классы **Book** и **DigitalVideoDisc** будут наследовать его. В классе **Order** элементами заказы теперь могут быть любые экземпляры **Media**.

Диаграмму показывающие особенности создаваемых классов и взаимосвязи между ними можно посмотреть в файле  
</root/labfiles/Visualization/InheritanceRefactoringAfterDiagram.png>

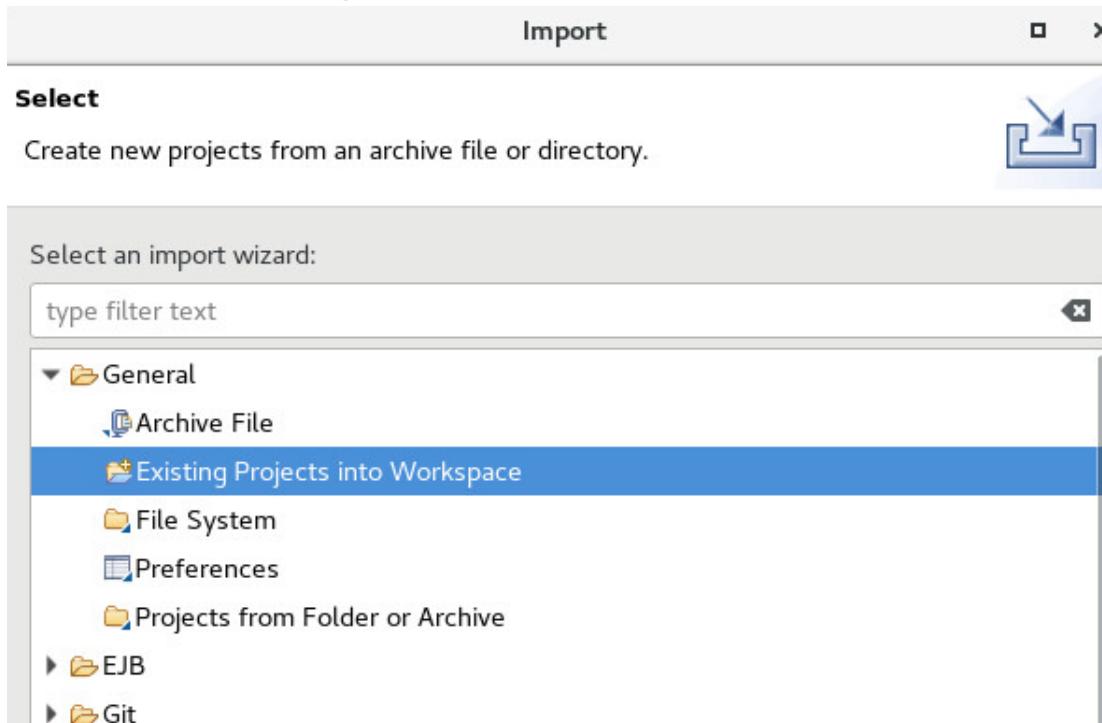
### Что вы должны будете сделать:

- Использовать наследование
- Использовать коллекции
- Воспользоваться функциями для рефакторинга

## Раздел 1. Создание класса *OnlineMedia*

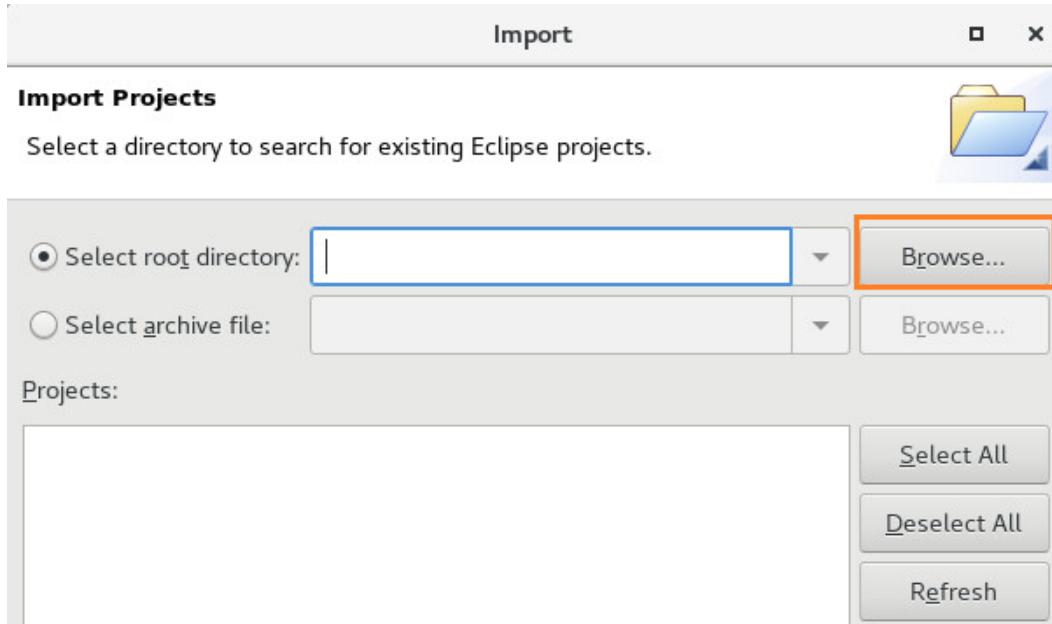
В данном разделе создается класс, где будет код для создания заказов и добавления туда новых позиций.

1. Откройте новое рабочее пространство
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/workspaces/Inheritance** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
2. Импортируйте проекты в рабочее пространство из каталога **/root/labfiles/Inheritance/import/OnlineMediaStore**
  - a. В меню выберите **File -> Import**.
  - b. **General -> Existing Project into the Workspace**



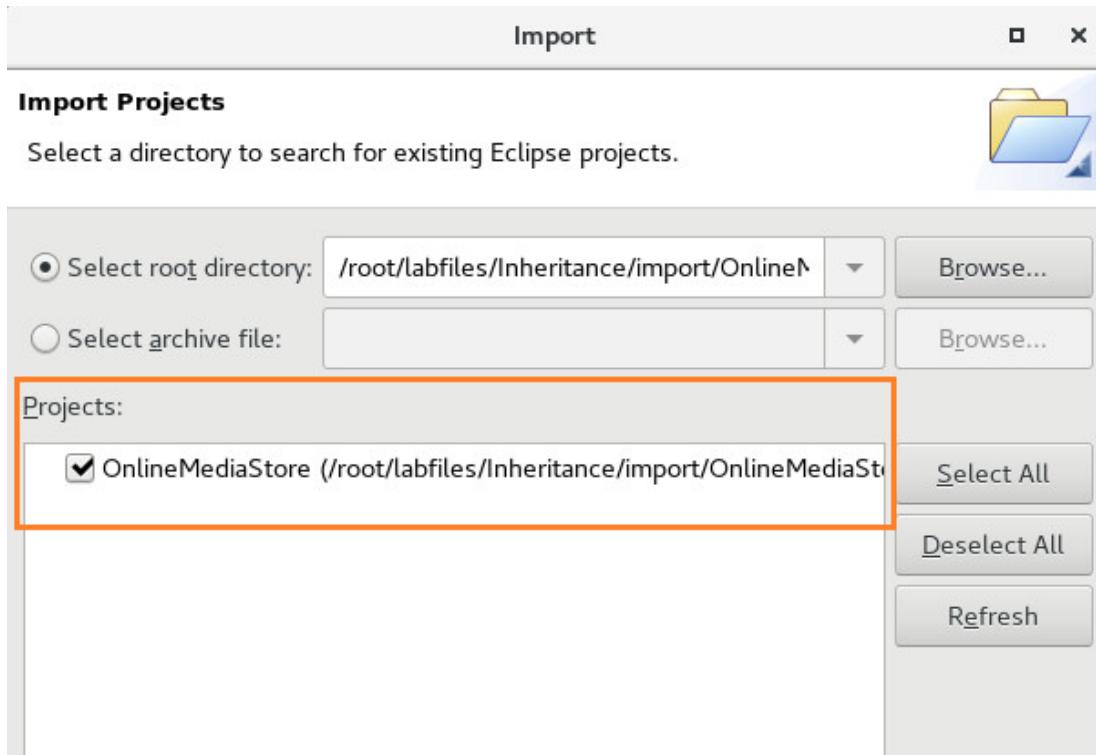
- c. Нажмите **Next**.

d. Нажмите **Browse** рядом с полем **Select root directory**.



e. Выберите каталог, указанный выше.

f. Выберите все проекты и нажмите **Finish**.



g. После этого шага в новом рабочем пространстве появятся классы, созданные в предыдущих упражнениях.

h. Перейдите в перспективу **Java**.

3. Создайте класс **Book**.

a. Из главного меню перейдите: **File -> New -> Class.**

Также можно нажать правой кнопкой мыши по проекту

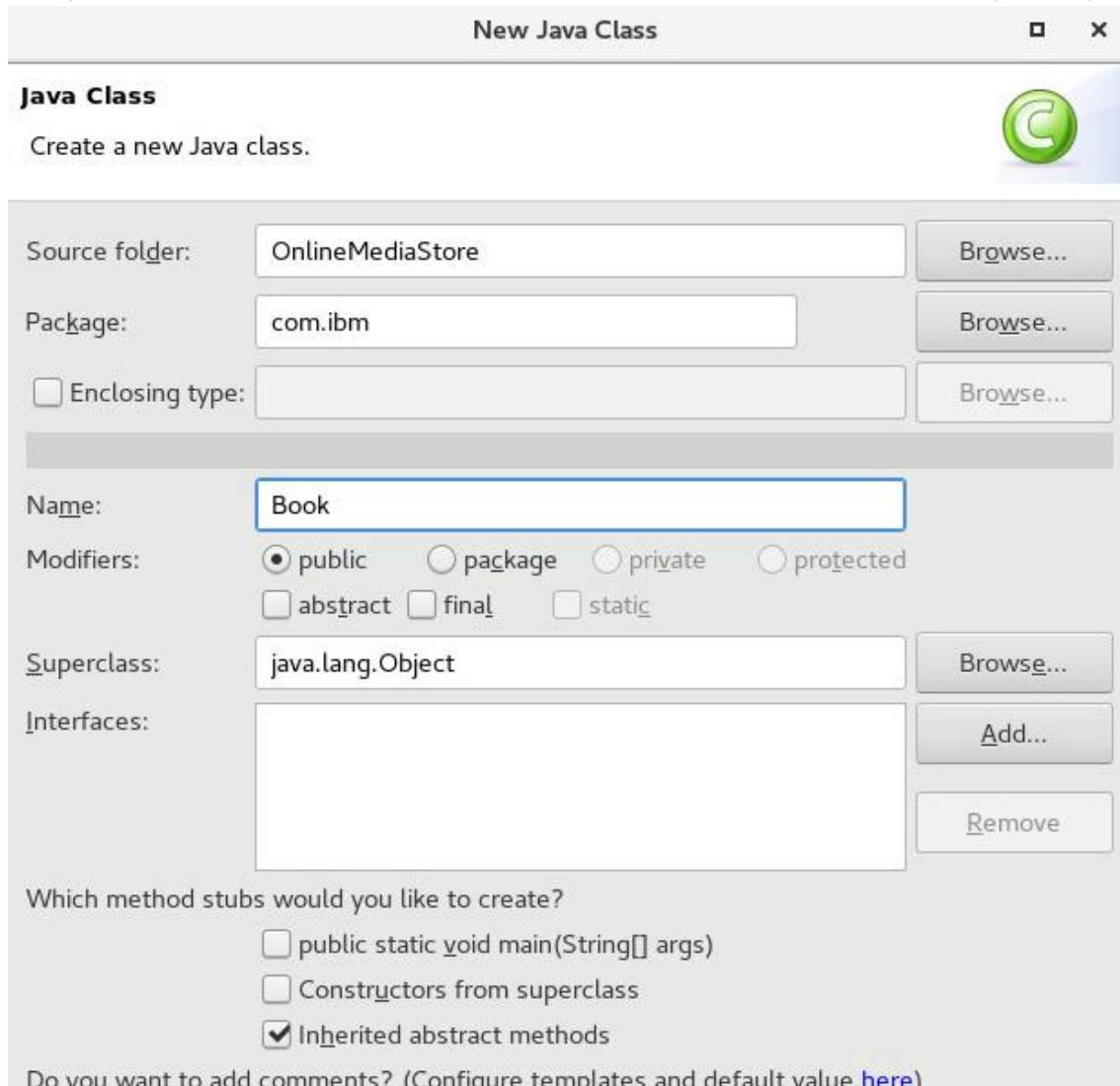
**OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.

b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.

c. В названии пакета укажите **com.ibm**.

d. Укажите **Book** как имя класса.

e. Не устанавливайте опцию **public static void main(String[] args)**.



f. Нажмите **Finish**.

4. Опишите класс **Book**.

а. Добавьте в класс поля и методы в соответствии с листингом, представленном ниже:

```
package com.ibm;
```

```
public class Book {
```

```
private String title;
private String category;
private float cost;
private java.util.ArrayList authors = new
java.util.ArrayList();

public Book() {
    super();
    // TODO Auto-generated constructor stub
}
public String getCategory() {
    return category;
}
public void setCategory(String category) {
    this.category = category;
}
public float getCost() {
    return cost;
}
public void setCost(float cost) {
    this.cost = cost;
}
public String getTitle() {
    return title;
}
public void setTitle(String title) {
    this.title = title;
}
}
```

- b. Вместо написания методов вручную, можно попросить Eclipse сгенерировать их.

5. Реализуйте методы **addAuthor()** и **removeAuthor()**.

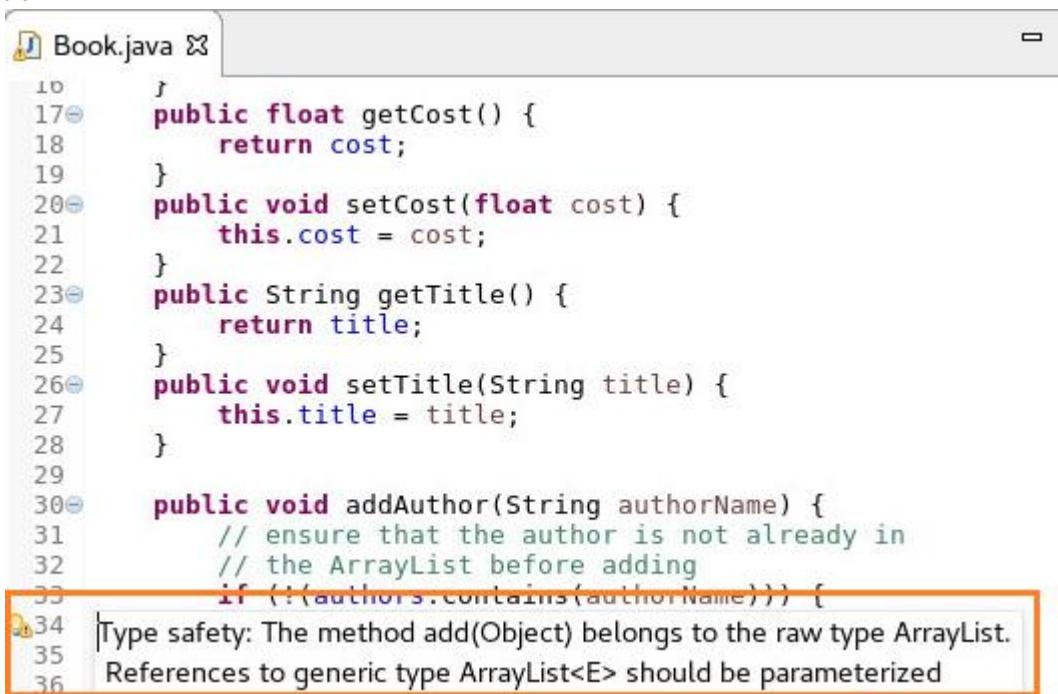
- a. Добавьте в класс **Book** следующий код:

```
public void addAuthor(String authorName) {
    // ensure that the author is not already in
    // the ArrayList before adding
    if (!(authors.contains(authorName))) {
        authors.add(authorName);
```

```
        }
    }

    public void removeAuthor(String authorName) {
        // ensure that the author is present in the
        // ArrayList before removing
        if ((authors.contains(authorName))) {
            authors.remove(authorName);
        }
    }
}
```

б. Среда разработки указывает на потенциальную ошибку с типами данных:



The screenshot shows a code editor window titled "Book.java". The code defines a class with several methods: getCost(), setCost(float cost), getTitle(), setTitle(String title), and addAuthor(String authorName). The addAuthor method contains a check to ensure the author is not already in the authors list. A red box highlights the line of code that performs this check: "if (!authors.contains(authorName)) {". Below this line, a tooltip displays the error message: "Type safety: The method add(Object) belongs to the raw type ArrayList. References to generic type ArrayList<E> should be parameterized".

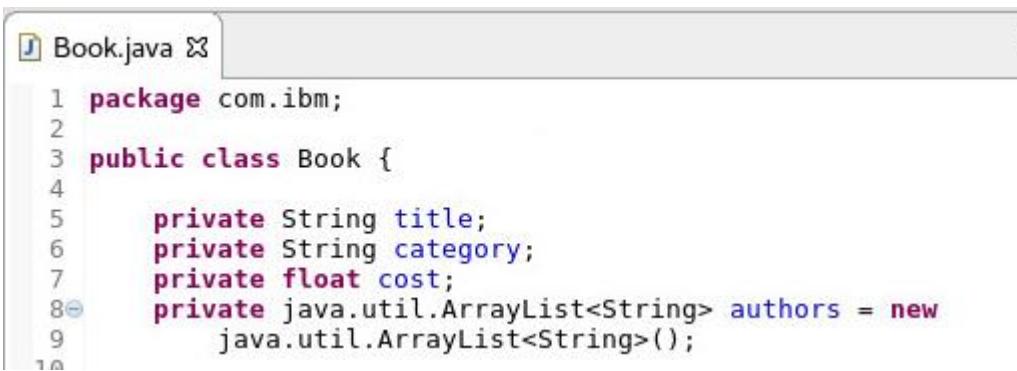
```
16
17    }
18
19    public float getCost() {
20        return cost;
21    }
22
23    public void setCost(float cost) {
24        this.cost = cost;
25    }
26
27    public String getTitle() {
28        return title;
29    }
30
31    public void setTitle(String title) {
32        this.title = title;
33    }
34
35    public void addAuthor(String authorName) {
36        // ensure that the author is not already in
        // the ArrayList before adding
        if (!authors.contains(authorName)) {

```

Type safety: The method add(Object) belongs to the raw type ArrayList.  
References to generic type ArrayList<E> should be parameterized

с. Для исправления этой ситуации нужно подчеркнуть, что в поле **authors** будет храниться объект **ArrayList** с массивом элементов конкретного типа, в данном случае **String**. Исправить это нужно в двух местах: при объявлении переменной и при инстанцировании этой переменной:

```
private java.util.ArrayList<String> authors = new
        java.util.ArrayList<String>();
```



```
1 package com.ibm;
2
3 public class Book {
4
5     private String title;
6     private String category;
7     private float cost;
8     private java.util.ArrayList<String> authors = new
9         java.util.ArrayList<String>();
10}
```

- d. Предупреждение должно исчезнуть.
- e. Сохраните сделанные изменения.

## Раздел 2. Создание класса *Media*

В данном разделе создается класс, который станет родителем (суперклассом) для **Book** и **DigitalVideoDisc**. В него будут перенесены все общие атрибуты и методы.

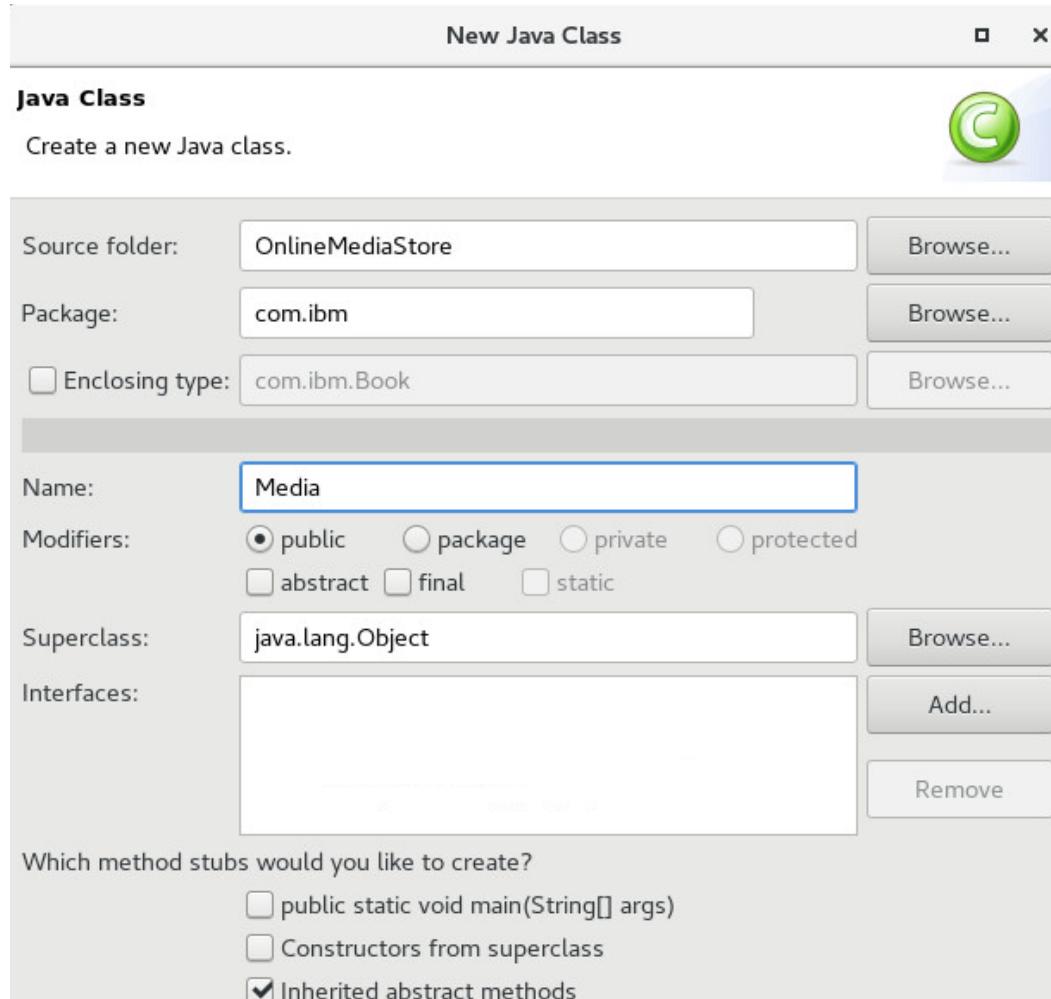
### 1. Создайте класс **Media**.

- a. Из главного меню перейдите: **File -> New -> Class**.

Также можно нажать правой кнопкой мыши по проекту **OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.

- b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.
- c. В названии пакета укажите **com.ibm**.
- d. Укажите **Media** как имя класса.

e. Не устанавливайте опцию **public static void main(String[] args)**.



f. Нажмите **Finish**.

2. Перенесите в новый класс поля и методы, которые являются идентичными для классов **Book** и **DigitalVideoDisc**.

a. Вы можете воспользоваться следующим кодом:

```
package com.ibm;

public class Media {
    private String title;
    private String category;
    private float cost;
    public Media() {
        super();
        // TODO Auto-generated constructor stub
    }
    public String getCategory() {
        return category;
    }
}
```

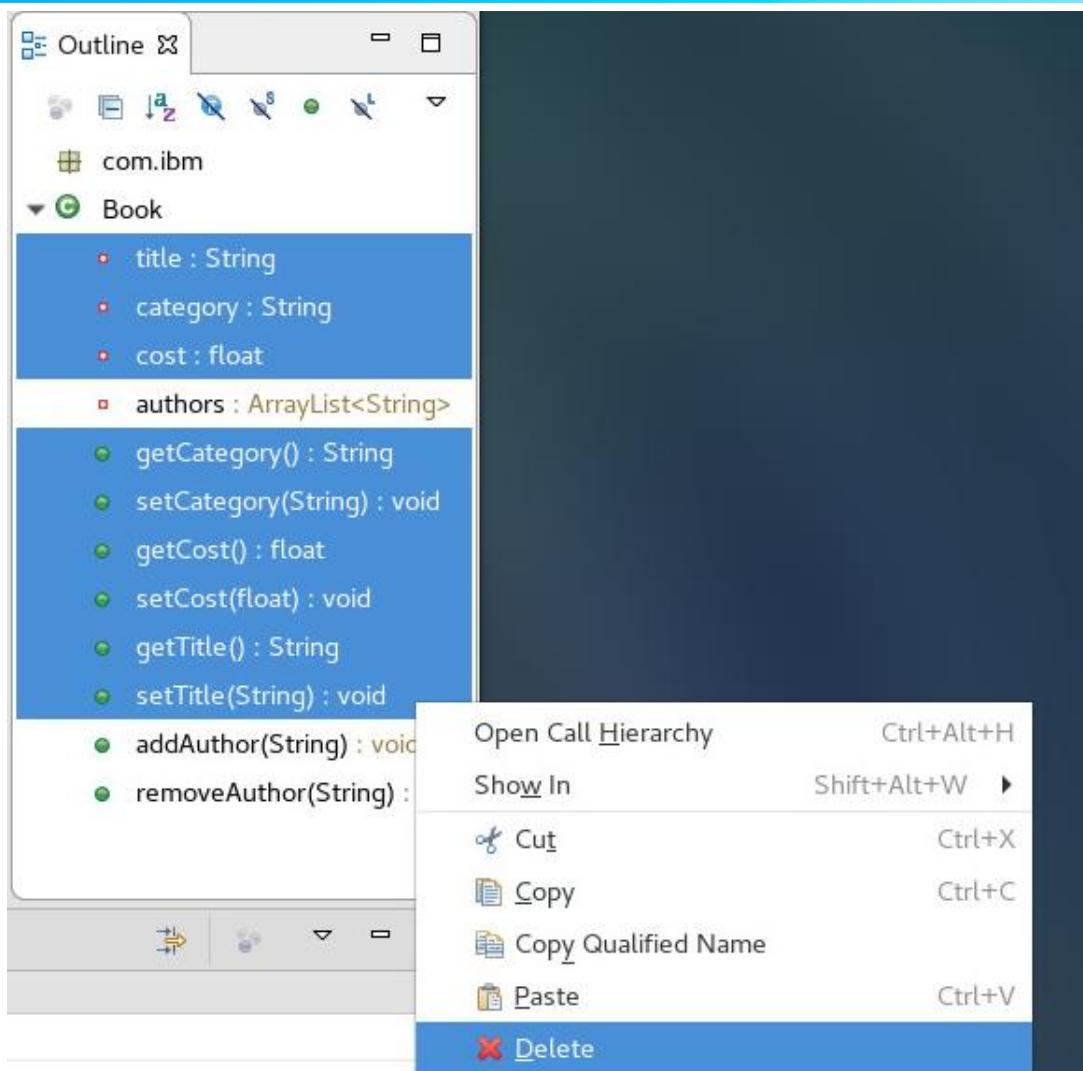
```
    }
    public void setCategory(String category) {
        this.category = category;
    }
    public float getCost() {
        return cost;
    }
    public void setCost(float cost) {
        this.cost = cost;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
}
```

b. Сохраните сделанные изменения.

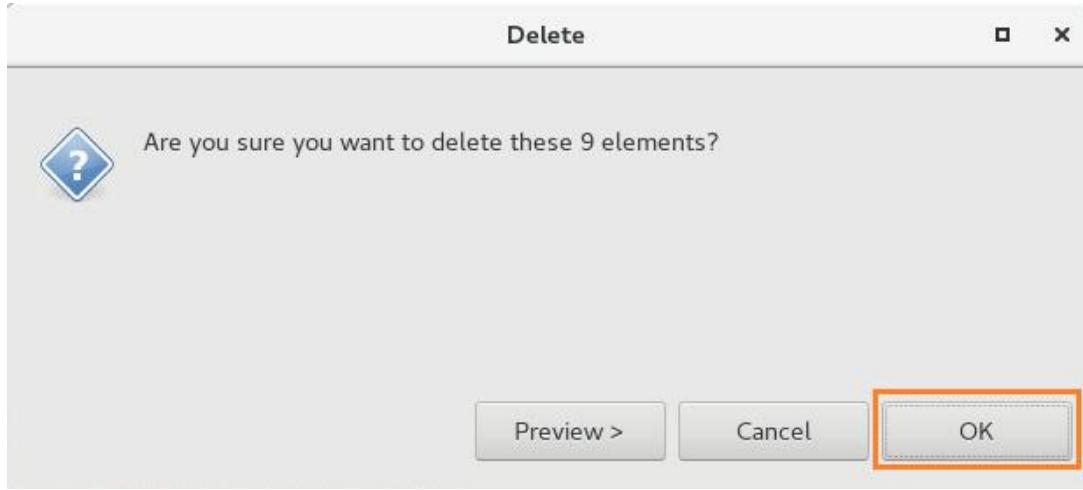
## **Раздел 3. Удаление лишних полей и методов классов Book и DigitalVideoDisc**

В данном разделе классы **Book** и **DigitalVideoDisc** адаптируются с учетом наследования от класса **Media**. Исключаются все поля и методы, которые были перенесены в класс **Media**.

1. Удалите лишние поля и методы.
  - a. Откройте в редакторе файл **Book.java**.
  - b. Перейдите в представление **Outline**.
  - c. Выберите поля **title**, **category**, **cost** и методы для установки/получения значений этих полей. Выбрать все сразу можно, зажав **Ctrl**.
  - d. Нажмите по выбранным полям правой кнопкой мыши и выберите **Delete**.



e. Подтвердите ваш выбор – **OK**.



- f. **Сохраните** сделанные изменения.  
2. Проделайте аналогичные манипуляции с классом **DigitalVideoDisc**.  
3. **Сохраните** все сделанные изменения.

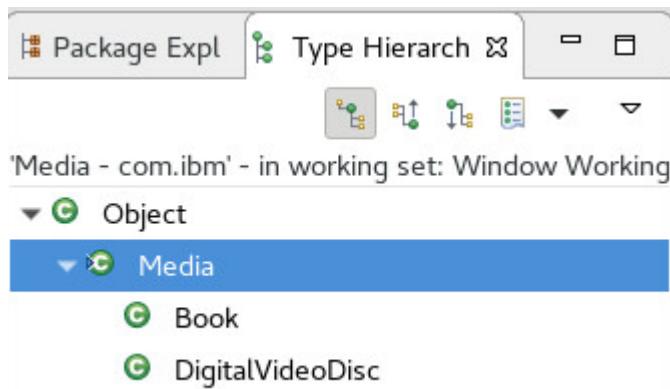
4. В классе **Book** остается только поле **authors**, а в классе **DigitalVideoDisc** остаются **director** и **length**.
5. При этом возникают ошибки в классах **DataFromProperties** и **Order** из-за удаленных полей/методов. Пока их можно игнорировать, позже они будут исправлены.
6. Измените объявления классов **Book** и **DigitalVideoDisc** для объявления наследования.
  - a. В строке для объявления класса перед открывающимися фигурными кавычками добавьте объявление наследования. Стока для класса **Book** будет выглядеть следующим образом:

```
public class Book extends Media {
```



```
1 package com.ibm;
2
3 public class Book extends Media{
4
5     private java.util.ArrayList<String> authors = new
6         java.util.ArrayList<String>();
7 }
```

- b. Эти слова говорят о необходимости наследовать для данных классов поля и методы класса **Media**.
- c. Повторите процедуру для **DigitalVideoDisc**.
7. Просмотрите иерархию созданных классов.
  - a. В представлении **Package Explorer** разверните **OnlineMediaStore** -> **com.ibm**.
  - b. Нажмите правой кнопкой мыши по файлу **Media.java**. Выберите **Open Type Hierarchy**. Открывается представление **Hierarchy**. По его содержимому можно сделать вывод, что этот класс является наследником от класса **Object** и родителем для классов **Book** и **DigitalVideoDisc**.



## **Раздел 4. Обновление существующих классов для работы с новой моделью классов**

В данном разделе классы **Order** и **OnlineMedia** меняются для того, чтобы отразить изменения, сделанные в классах в предыдущем разделе. Класс **Order** теперь будет описывать не список дисков, а список экземпляров **Media**. То есть, в нем могут быть как диски, так и книги. Соответственно, нужно поменять все методы, связанные с этим атрибутом класса.  
**OnlineMedia** адаптируется таким образом, чтобы вызывать новые методы класса **Order**.

1. Удалите лишние поля и методы из класса **Order**.
  - a. Откройте в редакторе файл **Order.java**.
  - b. Перейдите в представление **Outline**.
  - c. Выберите поле **itemsOrdered** и методы для работы с ним: **addDigitalVideoDisc()** и **removeDigitalVideoDisc()**. Выбрать все сразу можно, зажав **Ctrl**.
  - d. Нажмите по выбранным полям правой кнопкой мыши и выберите **Delete**.

Подтвердите ваш выбор – **OK**.

e. **Сохраните** сделанные изменения.
2. Аналогичным образом удалите поле **qtyOrdered** и методы **getQtyOrdered()** и **setQtyOrdered()**.
3. Повторно введите в класс поле **itemsOrdered**.
  - a. **Объявите** его следующим образом:

```
private ArrayList<Media> itemsOrdered = new  
ArrayList<Media>(3);
```

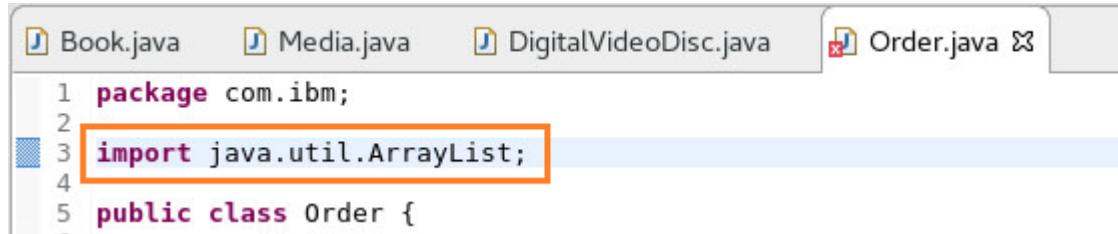
В отличие от предыдущей реализации здесь используется структура **ArrayList** вместо массива и используется тип **Media** вместо **DigitalVideoDisk**.



```
Book.java Media.java DigitalVideoDisc.java Order.java
1 package com.ibm;
2
3 public class Order {
4
5     private ArrayList<Media> itemsOrdered = new ArrayList<Media>(3);
6
7     public Order() {
8         // TODO Auto-generated constructor stub
9     }
}
```

- b. Для исправления ошибки, связанной с неизвестным типом добавьте импорт нужного класса в начало файла:

```
import java.util.ArrayList;
```



```
Book.java Media.java DigitalVideoDisc.java Order.java
1 package com.ibm;
2
3 import java.util.ArrayList;
4
5 public class Order {
```

- c. Сохраните сделанные изменения.

#### 4. Реализуйте методы **addMedia()** и **removeMedia()**.

- a. Вы можете воспользоваться следующим кодом:

```
public void addMedia(Media mediaItem) {
    // ensure that the Media item is not already
    // in the ArrayList before adding
    if (!(itemsOrdered.contains(mediaItem))) {
        itemsOrdered.add(mediaItem);
    }
}

public void removeMedia(Media mediaItem) {
    // ensure that the Media item is present
    // in the ArrayList before removing
    if (itemsOrdered.contains(mediaItem)) {
        itemsOrdered.remove(mediaItem);
    }
}
```

```
}
```

## 5. Измените метод **totalCost()**.

а. Замените текущую реализацию метода на следующую:

```
public float totalCost() {  
    // store the running total of the items in the  
    // order  
    float total = 0;  
    for (java.util.Iterator<Media> iter =  
        itemsOrdered.iterator();  
        iter.hasNext();) {  
        total += iter.next().getCost();  
    }  
    return total;  
}
```

б. Сохраните сделанные изменения.

## 6. Измените метод **main** класс **OnlineMedia**.

а. Откройте в редакторе класс **OnlineMedia**.

б. Замените текущую реализацию метода **main** на новую. Так чтобы использовать метод **addMedia**. И чтобы в заказ попали бы не только диски , но и книжка, которая создается в самом методе **main**.

Вы можете воспользоваться следующим кодом:

```
public static void main(String[] args) throws  
FileNotFoundException {  
    Properties prop = new Properties();  
    FileInputStream fis =  
        new FileInputStream("media.properties");  
    try {  
        prop.load(fis);  
    } catch (IOException e) {  
    }  
    // Create a new object for getting property data  
    DataFromProperties data = new  
    DataFromProperties();  
    // Create a new Order object  
    Order anOrder = new Order();  
    // add a number of dvds to the order
```

```
int dvdsToBeAdded = 3;
for (int dvdNumber=1; dvdNumber <= dvdsToBeAdded;
dvdNumber++) {
    // get the dvd data from the properties file
    DigitalVideoDisc dvd = data.addADvd(prop,
dvdNumber);
    // add the dvd to the order
    anOrder.addMedia(dvd);
}
// add a book to the order
Book book = new Book();
book.setTitle("Java Programming");
book.setCategory("Java");
book.setCost(69.99f);
book.addAuthor("Joe Wiggleworth");
book.addAuthor("Paula McMillan");
System.out.println("Book title = " +
book.getTitle());
anOrder.addMedia(book);
System.out.print("Total Cost of the Order is: ");
System.out.println(anOrder.totalCost());
}
```

- c. **Сохраните** сделанные изменения. При необходимости исправьте ошибки.
- d. **Запустите** вашу программу.
- e. В представлении **Console** открывается следующий листинг:



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the following text:

```
<terminated> OnlineMedia [Java Application] /usr/lib/jvm/java-1.8.0-
|dvd1.title = IBM Dance Party
|dvd2.title = IBM Kids Sing-along
|dvd3.title = IBM Smarter Planet
Book title = Java Programming
Total Cost of the Order is: 133.88
```

## Упражнение 6. Коллекции и дженерики

---

### О чём это упражнение:

В этой лабораторной работе рассматриваются коллекции и дженерики.

Создается новый класс **CompactDisc**, который расширяет класс **Media**. Он наследует все родительские поля и методы, обладая своими собственными полями **artist** (строка) и **ArrayList** типа **Track**, который называется **tracks**. У класса **Track** есть поля **title** и **length**.

Диаграмму показывающую особенности создаваемых классов и взаимосвязи между ними можно посмотреть в файле  
**/root/labfiles/Visualization/CollectionsGenericsAfterDiagram.png**

### Что вы должны будете сделать:

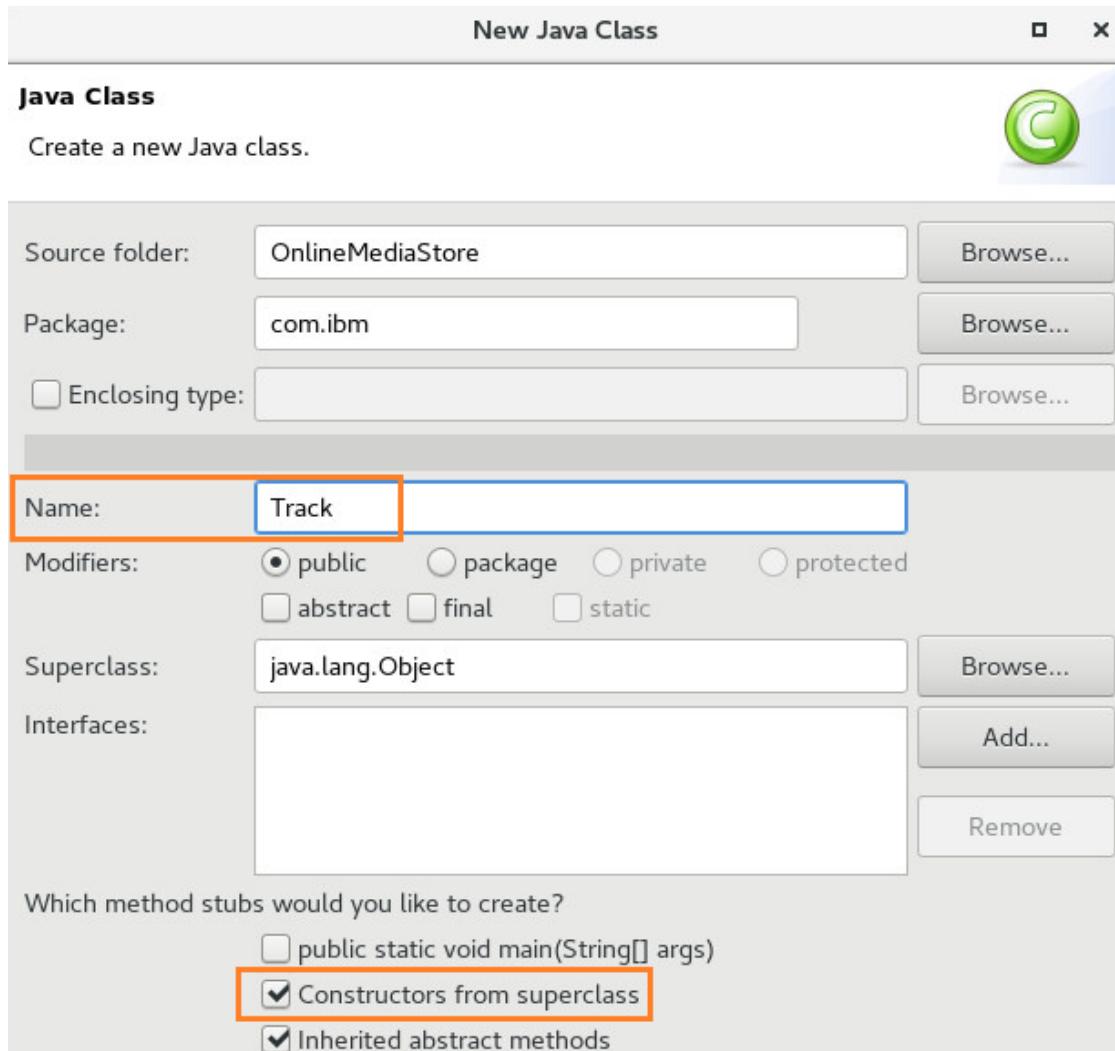
- Использовать коллекции
- Использовать дженерики

## **Раздел 1. Создание класса Track**

В данном разделе создается класс, для объектов описания композиций с CD.

1. Откройте новое рабочее пространство
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/workspaces/Collection** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
2. Импортируйте проекты в рабочее пространство из каталога **/root/labfiles/Collection/import/OnlineMediaStore**. Если вы забыли последовательность действий, то можно воспользоваться инструкцией из начала прошлого упражнения.
3. Перейдите в перспективу **Java**.
4. Создайте класс **Track**.
  - a. Из главного меню перейдите: **File -> New -> Class**.  
Также можно нажать правой кнопкой мыши по проекту **OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.
  - b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.
  - c. В названии пакета укажите **com.ibm**.
  - d. Укажите **Track** как имя класса.

## e. Constructors from superclass – опция выбрана.



f. Нажмите **Finish**.

5. Добавьте поля и базовые методы для класса **Track**.

a. Сделайте два атрибута: строка – **title**, целое число – **length**.

b. Вы можете воспользоваться следующим кодом:

```
package com.ibm;
```

```
public class Track {  
    private String title;  
    private int length;  
    public Track() {  
        // TODO Auto-generated constructor stub  
    }  
    public int getLength() {  
        return length;  
    }  
}
```

```
public void setLength(int length) {  
    this.length = length;  
}  
public String getTitle() {  
    return title;  
}  
public void setTitle(String title) {  
    this.title = title;  
}  
}
```

c. Сохраните сделанные изменения.

## **Раздел 2. Создание класса *CompactDisc***

В данном разделе создается класс для работы с CD. Он будет содержать информацию об исполнителе (**artist**), продолжительности (**length**) и списке композиций (**tracks**).

1. Создайте класс **CompactDisc**.

a. Из главного меню перейдите: **File -> New -> Class**.

Также можно нажать правой кнопкой мыши по проекту **OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.

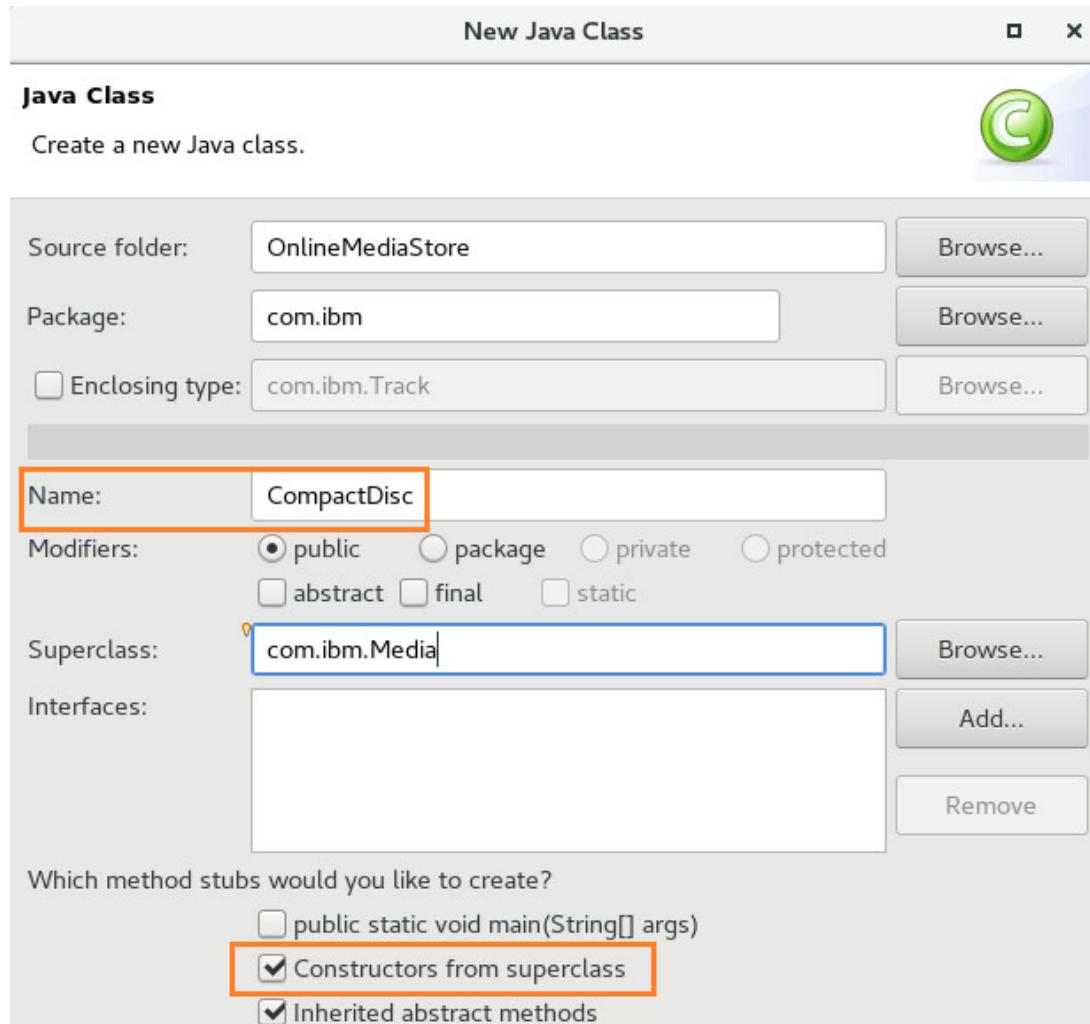
b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.

c. В названии пакета укажите **com.ibm**.

d. Укажите **CompactDisc** как имя класса.

e. Укажите **com.ibm.Media** как имя суперкласса.

## f. Constructors from superclass – опция выбрана.



## g. Нажмите Finish.

2. Введите поля и методы класса.

a. Сделайте указанные выше поля и методы для считывания/установки значений полей **artist** и **tracks**.

b. Вы можете использовать следующий код:

```
public class CompactDisc {  
    private String artist;  
    private int length;  
    private ArrayList<Track> tracks = new  
    ArrayList<Track>();  
  
    public CompactDisc() {  
        // TODO Auto-generated constructor stub  
    }  
    public String getArtist() {  
        return artist;
```

```
    }
    public void setArtist(String artist) {
        this.artist = artist;
    }
    public ArrayList<Track> getTracks() {
        return tracks;
    }
    public void setTracks(ArrayList<Track> tracks) {
        this.tracks = tracks;
    }
}
```

с. Организуйте импорт класса **ArrayList** в ваш класс: правая кнопка мыши в редакторе, **Source -> Organize Imports**.

3. Реализуйте методы для добавления и удаления треков на диск.

а. Вы можете использовать следующий код:

```
public void addTrack (Track track1) {
    // ensure that the track is not already in the
    // ArrayList before adding
    if (!(tracks.contains(track1))) {
        tracks.add(track1);
    }
}
public void removeTrack(Track track1) {
    // ensure that the track is present in the
    // ArrayList before removing
    if ((tracks.contains(track1))) {
        tracks.remove(track1);
    }
}
```

4. Реализуйте метод для получения продолжительности проигрывания диска – **getLength()**. Так как у каждого трека есть своя продолжительность, то для оценки продолжительности всего диска необходимо пройти в цикле по каждому из них и посчитать сумму.

а. Вы можете использовать следующий код:

```
public int getLength() {
    int total = 0;
    for (Track track : tracks) {
```

```
        total += track.getLength();
    }
    return total;
}
```

b. Сохраните сделанные изменения.

## **Раздел 3. Обновление существующих классов**

В данном разделе существующие классы адаптируются под работу с CD. В классе **DataFromProperties** будет реализовано считывание данных о диске и треках для него из файла параметров. В основном классе **OrderMedia** создается набор треков, связываемых с CD, который в свою очередь попадает в заказ.

1. Добавьте в класс **DataFromProperties** метод для считывания параметров CD из файла параметров по образу и подобию того, как это было сделано для DVD.

a. Вы можете использовать следующий код:

```
public CompactDisc addACd(Properties properties, int
numberOfItems) {

    String titleKey = new
StringBuffer().append("cd").append(numberOfItems).ap
pend(".title").toString();
    System.out.print(titleKey + " = ");
    String categoryKey = new
StringBuffer().append("cd").append(numberOfItems).ap
pend(".category").toString();
    String costKey = new
StringBuffer().append("cd").append(numberOfItems).ap
pend(".cost").toString();
    String artistKey = new
StringBuffer().append("cd").append(numberOfItems).ap
pend(".artist").toString();
    // Create a new CompactDisc object
    CompactDisc cd = new CompactDisc();
    cd.setTitle(properties.getProperty(titleKey));
    System.out.println(cd.getTitle());
```

```
        cd.setCategory(properties.getProperty(categoryKey));
    );
    cd.setCost(Float.parseFloat(properties.getProperty(costKey)));
    System.out.print("Cost of CD = ");
    System.out.println(cd.getCost());
    cd.setArtist(properties.getProperty(artistKey));
    return cd;
}
```

**b. Реализуйте метод для считывания информации о треке:**

```
public Track addATrack(Properties properties, int
trackNumber) {

    String titleKey = new
StringBuffer().append("t").append(trackNumber).append(".title").toString();
    System.out.print(titleKey + " = ");
    String lengthKey = new
StringBuffer().append("t").append(trackNumber).append(".length").toString();
    // create a new Track object and set the fields
    Track t = new Track();
    t.setTitle(properties.getProperty(titleKey));
    System.out.println(t.getTitle());
    t.setLength(Integer.parseInt(properties.getProperty(lengthKey)));
    return t;
}
```

**c. Сохраните** сделанные изменения.

2. Обновите класс **OnlineMedia** так, чтобы создавать в нем набор треков, связывать их с объектов CD, а его добавлять в заказ. Также важно протестировать функцию для проверки общей продолжительности диска.

a. Откройте класс **OnlineMedia** в редакторе.

b. В методе **main** реализуйте код с описанным функционалом. Его можно вставить до строки:

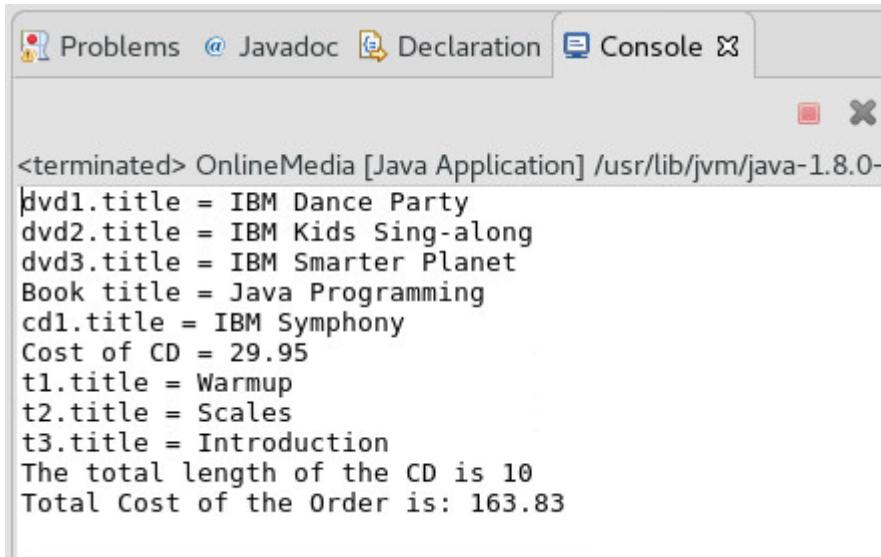
```
System.out.print("Total Cost of the Order is: ");
```

c. Вы можете использовать следующий вариант:

```
// add a cd to the order
int cdNumber = 1;
int tracksToBeAdded = 3;
CompactDisc cd = data.addACd(prop, cdNumber);
for (int i=1; i <= tracksToBeAdded; i++){
    // get the track data from the properties
    file
    Track track = data.addATrack(prop, i);
    cd.addTrack(track);
}
anOrder.addMedia(cd);
System.out.print("The total length of the CD is
");
System.out.println(cd.getLength());
```

d. Сохраните сделанные изменения и исправьте при необходимости возникшие ошибки.

e. Запустите приложение **OnlineMedia** с помощью кнопки **Run**. В представлении **Console** должен появиться следующий листинг:



```
<terminated> OnlineMedia [Java Application] /usr/lib/jvm/java-1.8.0-
jdk1.title = IBM Dance Party
jdk2.title = IBM Kids Sing-along
jdk3.title = IBM Smarter Planet
Book title = Java Programming
cd1.title = IBM Symphony
Cost of CD = 29.95
t1.title = Warmup
t2.title = Scales
t3.title = Introduction
The total length of the CD is 10
Total Cost of the Order is: 163.83
```

## Раздел 4. Джинерики: создание класса *Library*

Методы для добавления и удаления треков в CD, авторов в книгу, позиций в заказ - довольно похожи. В этой части упражнения вы создаете общий набор методов для добавления и удаления элементов

из **ArrayList**, в новом классе **Library**. Тип объекта (трек, строка и т.п.) передается как дженерик параметр для класса.

## 1. Создайте новый класс **Library**.

а. Укажите следующие параметры при создании:

**Name:**

**Library<T>**

**Modifiers:**

**public**

**Constructors from superclass:**

**выбрана**

Create a new Java class.



Source folder: **OnlineMediaStore**

Package: **com.ibm**

Enclosing type: **com.ibm.OnlineMedia**

**Name:** **Library<T>**

Modifiers:  **public**  **package**  **private**  **protected**  
 **abstract**  **final**  **static**

Superclass: **java.lang.Object**

Interfaces:

Which method stubs would you like to create?

**public static void main(String[] args)**  
 **Constructors from superclass**  
 **Inherited abstract methods**

б. Нажмите **Finish**.

2. Добавьте поля в класс **Library**. Параметр для типизации: **mediaType** и **variable** типа **ArrayList<T>** – список элементов указанного типа.

Для работы с этими приватными полями нужно создать соответствующие методы.

а. Вы можете использовать следующий код (сгенерированный конструктор удалять не следует):

```
// Naming type parameter: T = a generic type
```

```
private T mediaType;
private ArrayList<T> variable = new ArrayList<T>();

public T getMediaType() {
    return mediaType;
}

public void setMediaType(T mediaType) {
    this.mediaType = mediaType;
}

public ArrayList<T> getVariable() {
    return variable;
}
```

- b. Организуйте импорт класса **ArrayList** в ваш класс: правая кнопка мыши в редакторе, **Source -> Organize Imports**.
3. Для добавления и удаления новых элементов из списка реализуйте два новых метода **addMedia(T v)** и **removeMedia(T v)**.
- a. Используйте следующий код:

```
public void addMedia(T v) {
    // ensure that the element is not already in the
    // ArrayList before adding
    if (!(variable.contains(v))) {
        variable.add(v);
    }
}

public void removeMedia(T v) {
    // ensure that the element is present in the
    // ArrayList before removing
    if ((variable.contains(v))) {
        variable.remove(v);
    }
}
```

- b. Сохраните сделанные изменения.

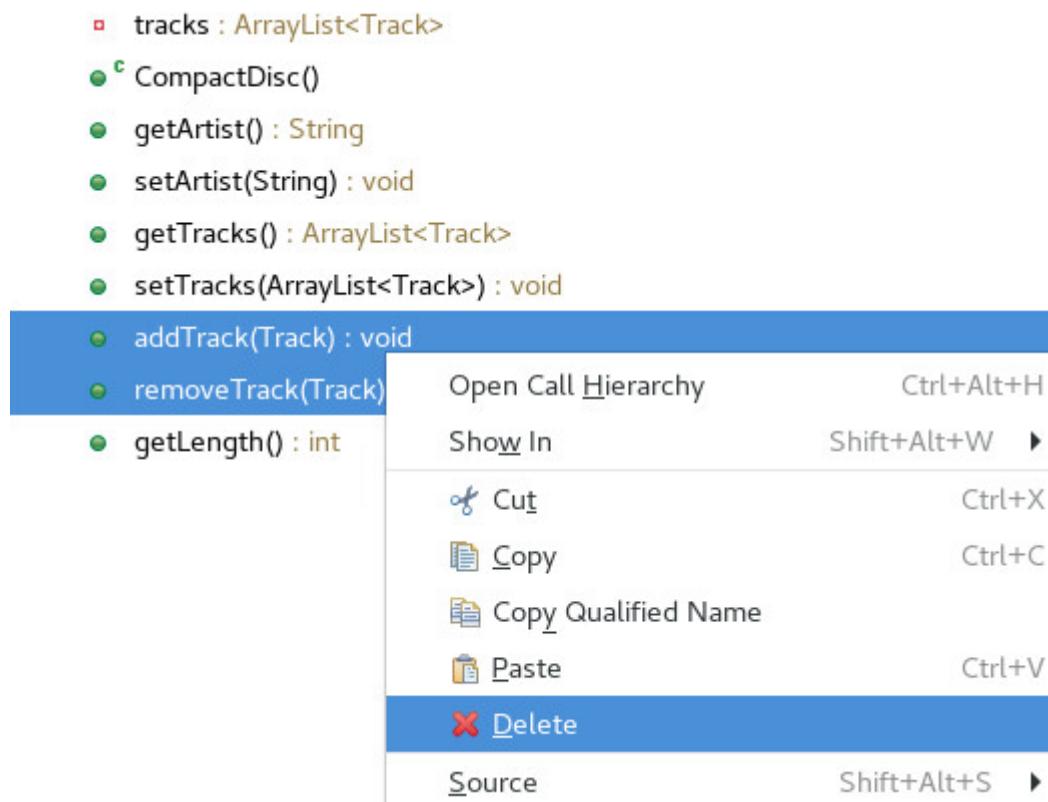
## **Раздел 5. Адаптация классов под новую модель с использованием дженериков**

Методы для добавления и удаления треков в CD, авторов в книгу более не актуальны. Вместо них можно использовать методы из нового класса.

Также необходимо адаптировать главный модуль приложения под работу в новых реалиях.

1. Удалите из класса **CompactDisc** не актуальные методы.

- Откройте в редакторе класс **CompactDisc**.
- Перейдите в представление **Outline**.
- Выберите методы **addTrack** и **removeTrack** и удалите их с помощью функции **Delete** в контекстном меню.



d. Подтвердите сделанный выбор.

e. **Сохраните** сделанные изменения. Возникающая ошибка будет исправлена далее в упражнении.

2. Адаптируйте код **OnlineMedia** так, чтобы использовать там новый метод **addMedia** вместо удаленного **addTrack**.

a. Откройте класс **OnlineMedia** в редакторе.

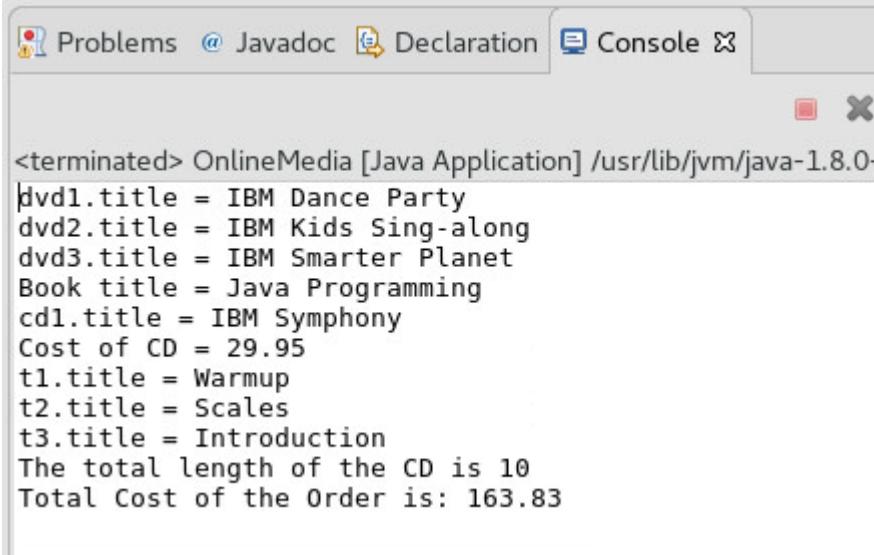
b. Найдите метод **main**.

c. Измените код, руководствуясь примером, приведенным ниже:

```
// add a cd to the order
int cdNumber = 1;
int tracksToBeAdded = 3;
CompactDisc cd = data.addACd(prop, cdNumber);
```

```
Library<Track> mt = new Library<Track>();
Track track = mt.getMediaType();
for (int i=1; i <= tracksToBeAdded; i++) {
    // get the track data from the properties file
    track = data.addATrack(prop, i);
    //cd.addTrack(track);
    mt.addMedia(track);
}
cd.setTracks(mt.getVariable());
anOrder.addMedia(cd);
System.out.print("The total length of the CD is ");
System.out.println(cd.getLength());
```

- d. **Сохраните** сделанные изменения. При необходимости исправьте ошибки.  
e. Запустите приложение с помощью кнопки **Run**. По итогам его работы вы должны увидеть следующий листинг:



The screenshot shows a Java application running in an IDE. The console tab is active, displaying the following output:

```
<terminated> OnlineMedia [Java Application] /usr/lib/jvm/java-1.8.0-
dvd1.title = IBM Dance Party
dvd2.title = IBM Kids Sing-along
dvd3.title = IBM Smarter Planet
Book title = Java Programming
cd1.title = IBM Symphony
Cost of CD = 29.95
t1.title = Warmup
t2.title = Scales
t3.title = Introduction
The total length of the CD is 10
Total Cost of the Order is: 163.83
```

3. Повторите аналогичную процедуру для того, чтобы при работе со списком авторов в книге не использовать специальные методы соответствующего класса, а использовать универсальные методы из класса **Library**.

## Упражнение 7. Интерфейсы и сортировка

---

### О чём это упражнение:

В этой лабораторной работе рассматривается применение интерфейсов в Java, а также использование интерфейса Comparable для сравнения объектов.

В упражнении создается интерфейс **Playable**, который реализуется классами **DigitalVideoDisc**, **CompactDisc** и **Track**.

Для класса Track реализуется интерфейс Comparable.

Диаграмму показывающую особенности создаваемых классов и взаимосвязи между ними можно посмотреть в файле  
**/root/labfiles/Visualization/InterfacesSortingAfterDiagram.png**

### Что вы должны будете сделать:

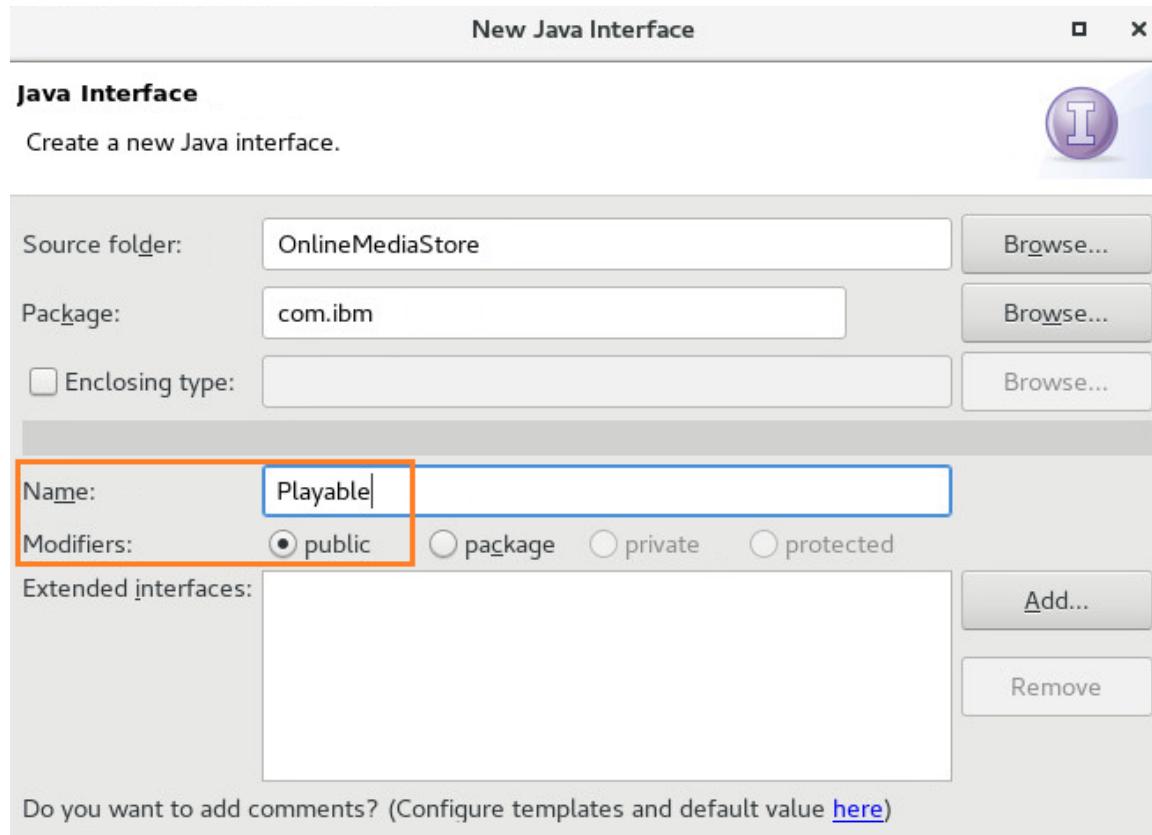
- Создать и реализовать интерфейсы
- Сортировать объекты с использованием интерфейса Comparable

## **Раздел 1. Создание интерфейса *Playable***

В данном разделе создается интерфейс, который далее будет реализован несколькими классами. В этом интерфейсе будет единственный метод **play()**.

1. Откройте новое рабочее пространство
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/workspaces/Interfaces** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
2. Импортируйте проекты в рабочее пространство из каталога **/root/labfiles/Interfaces/import/OnlineMediaStore**. Если вы забыли последовательность действий, то можно воспользоваться инструкцией из начала прошлого упражнения.
3. Перейдите в перспективу **Java**.
4. Создайте интерфейс **Playable**.
  - a. Из главного меню перейдите: **File -> New -> Interface**.  
Также можно нажать правой кнопкой мыши по проекту **OnlineMediaStore**, в контекстном меню выбрать **New -> Interface**. В любом случае откроется мастер создания нового класса.
  - b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.
  - c. В названии пакета укажите **com.ibm**.

d. Укажите **Playable** как имя интерфейса.



e. Нажмите **Finish**.

5. Добавьте в интерфейс определение его метода **play()**.

a. С помощью редактора добавьте в интерфейс следующую строку:  
public void play();

Это определение метода. Реализации у него в самом интерфейсе быть не может. Но у каждого класса, который этот интерфейс реализует, должна быть реализация и для соответствующего метода.

```
1 package com.ibm;
2
3 public interface Playable {
4
5     public void play();
6
7 }
```

b. Сохраните сделанные изменения.

## Раздел 2. Реализация интерфейса **Playable**

В данном разделе реализуется интерфейс для упомянутых выше классов. Для каждого из классов нужно реализовать единственный метод **play()**.

1. Объявите реализацию интерфейса для каждого из классов:

**DigitalVideoDisc**, **CompactDisc** и **Track**.

- В строке объявления класса допишите `implements Playable` сразу после `extends Media` (для **DigitalVideoDisc**, **CompactDisc**) или после `public class Track` (для **Track**).
- Например, для **CompactDisc** это будет выглядеть следующим образом:



```

1 package com.ibm;
2
3 import java.util.ArrayList;
4
5 public class CompactDisc extends Media implements Playable {
6     private String artist;
7     private int length;
8     private ArrayList<Track> tracks = new ArrayList<Track>();
9     public CompactDisc() {
10         // TODO Auto-generated constructor stub
11     }

```

c. Аналогичным образом сделайте с двумя другими классами.

d. Сохраните все сделанные изменения. Сразу после сохранения

Eclipse обращает внимание на ошибку: класс реализует интерфейс, но не реализует его метода. В последующих шагах эта ошибка будет исправлена.

2. Реализуйте метод **play()** для классов **DigitalVideoDisc** и **Track**.

- Для класса **DigitalVideoDisc** можно использовать следующую реализацию метода:

```

public void play() {
    System.out.println("Playing DVD: " +
this.getTitle());
    System.out.println("DVD length: " +
this.getLength());
}

```

b. Похожую реализацию сделайте для метода в классе **Track**.

3. Реализуйте метод **play()** для класса **CompactDisc**. Так как в CD есть информация обо всех треках на нем, можно предусмотреть более интересную реализацию метода, которая будет предусматривать проигрывание всех треков с диска.

а. Можно воспользоваться следующим вариантом:

```
public void play() {  
    System.out.println("Playing CD: " +  
        this.getTitle());  
    System.out.println("CD length:" +  
        this.getLength());  
    for (Track track : tracks) {  
        track.play();  
    }  
}
```

4. Обновите класс **OnlineMedia** для проигрывания дисков перед их добавлением в заказ.

а. Откройте метод **main** класса **OnlineMedia**.

б. Перед добавлением DVD к заказу запустите его проигрывание с помощью строчки:

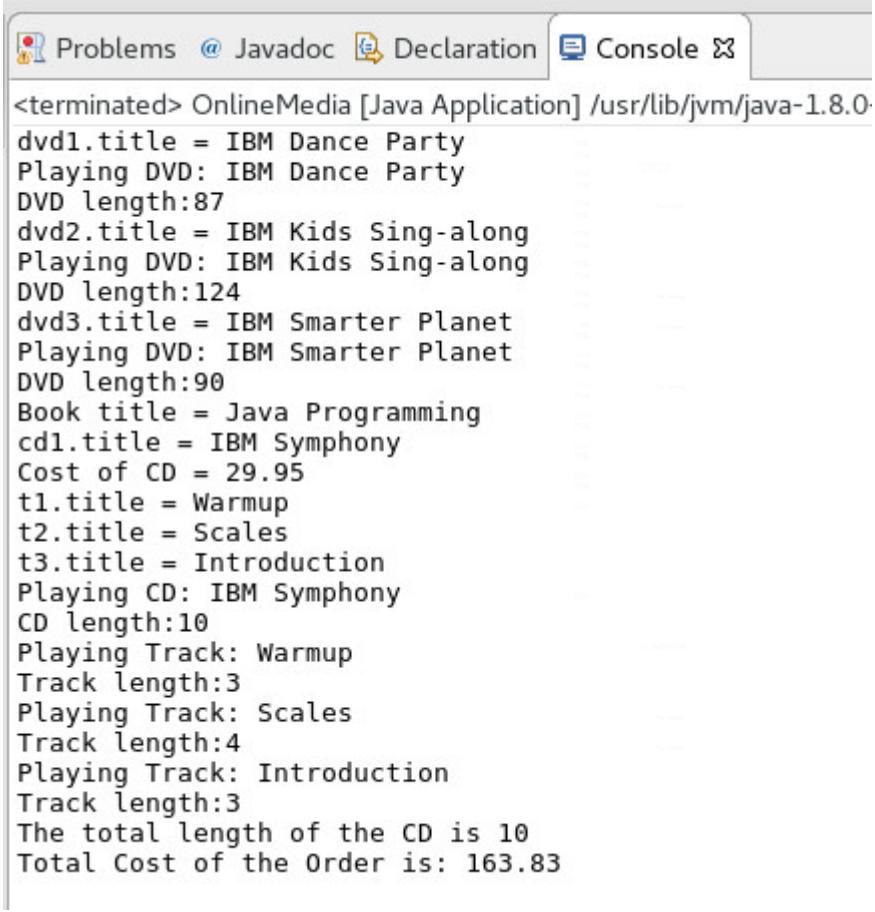
```
dvd.play();
```

с. Аналогичным образом действуйте и для **cd**.

д. Сохраните сделанные изменения и при необходимости исправьте ошибки.

е. Запустите приложение с помощью кнопки **Run**.

f. Вы увидите следующие результаты выполнения в консоли:



The screenshot shows a Java application window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the following output:

```
<terminated> OnlineMedia [Java Application] /usr/lib/jvm/java-1.8.0-
dvd1.title = IBM Dance Party
Playing DVD: IBM Dance Party
DVD length:87
dvd2.title = IBM Kids Sing-along
Playing DVD: IBM Kids Sing-along
DVD length:124
dvd3.title = IBM Smarter Planet
Playing DVD: IBM Smarter Planet
DVD length:90
Book title = Java Programming
cd1.title = IBM Symphony
Cost of CD = 29.95
t1.title = Warmup
t2.title = Scales
t3.title = Introduction
Playing CD: IBM Symphony
CD length:10
Playing Track: Warmup
Track length:3
Playing Track: Scales
Track length:4
Playing Track: Introduction
Track length:3
The total length of the CD is 10
Total Cost of the Order is: 163.83
```

Вывод может отличаться в зависимости от вашей реализации методов **play()**.

### **Раздел 3. Реализация интерфейса Comparable**

В данном разделе реализуется интерфейс **Comparable** для класса **Track**.

Этот интерфейс является частью системного пакета Java и не требует явного импорта. Этот интерфейс параметризован:

```
public interface Comparable<T> {

    public int compareTo(T other)

}
```

С помощью параметра **T** можно описать, с объектом какого класса будет сравниваться объекты класса, реализующего этот интерфейс. Обычно там применяется тот же класс.

1. Добавьте в определение класса реализацию интерфейса:

```
public class Track implements Playable,  
Comparable<Track> {
```

2. Реализуйте метод **compareTo()**. Затем его можно будет использовать для сортировки соответствующих коллекций. Для его реализации можно воспользоваться сравнением одного из полей данного класса. Например, **title**.

a. Воспользуйтесь следующим вариантом:

```
public int compareTo(Track t2) {  
    return  
(this.getTitle() ) .compareTo(t2.getTitle());  
}
```

b. Сохраните сделанные изменения.

3. Проверьте логику работы сравнения треков.

a. Откройте в редакторе класс **OnlineMedia**.

Добавьте строки

```
System.out.println("-----");  
System.out.println ("The tracks in NOT sorted order  
are ");
```

после строки для считывания информации о CD диске из файла.

b.

c. Внесите в метод main логику для сортировки коллекции треков в соответствии с листингом ниже (соответствующий код нужно добавить после кода для проигрывания CD):

```
java.util.Collection<Track> collection =  
cd.getTracks();  
java.util.Collections.sort((java.util.ArrayList<Trac  
k>)collection);  
System.out.println("-----  
----");  
System.out.println ("The tracks in sorted order are:  
");  
// Iterate through the collection and output their  
titles  
// in sorted order  
for (Track trackItem: collection) {  
    System.out.println(trackItem.getTitle());
```

}

d. Запустите программу

e. Убедитесь, что сортировка работает ожидаемо:

```
Playing DVD: IBM Kids Sing-along
DVD length:124
dvd3.title = IBM Smarter Planet
Playing DVD: IBM Smarter Planet
DVD length:90
Book title = Java Programming
cd1.title = IBM Symphony
Cost of CD = 29.95
-----
The tracks in NOT sorted order are:
t1.title = Warmup
t2.title = Scales
t3.title = Introduction
Playing CD: IBM Symphony
CD length:10
Playing Track: Warmup
Track length:3
Playing Track: Scales
Track length:4
Playing Track: Introduction
Track length:3
-----
The tracks in sorted order are:
Introduction
Scales
Warmup
-----
The total length of the CD is 10
Total Cost of the Order is: 163.83
```

4. Измените логику сортировки треков таким образом, чтобы она выполнялась по продолжительности, а не по названию. Проверьте работу этой сортировки.

## Упражнение 8. Потоки

---

### О чём это упражнение:

В этой лабораторной работе рассматривается практический пример создания и использования потока в Java логике.

Создается специальный класс MemoryDaemon, который мониторит утилизацию памяти приложением OnlineMedia во время ее исполнения и выводит уровень утилизации в консоль.

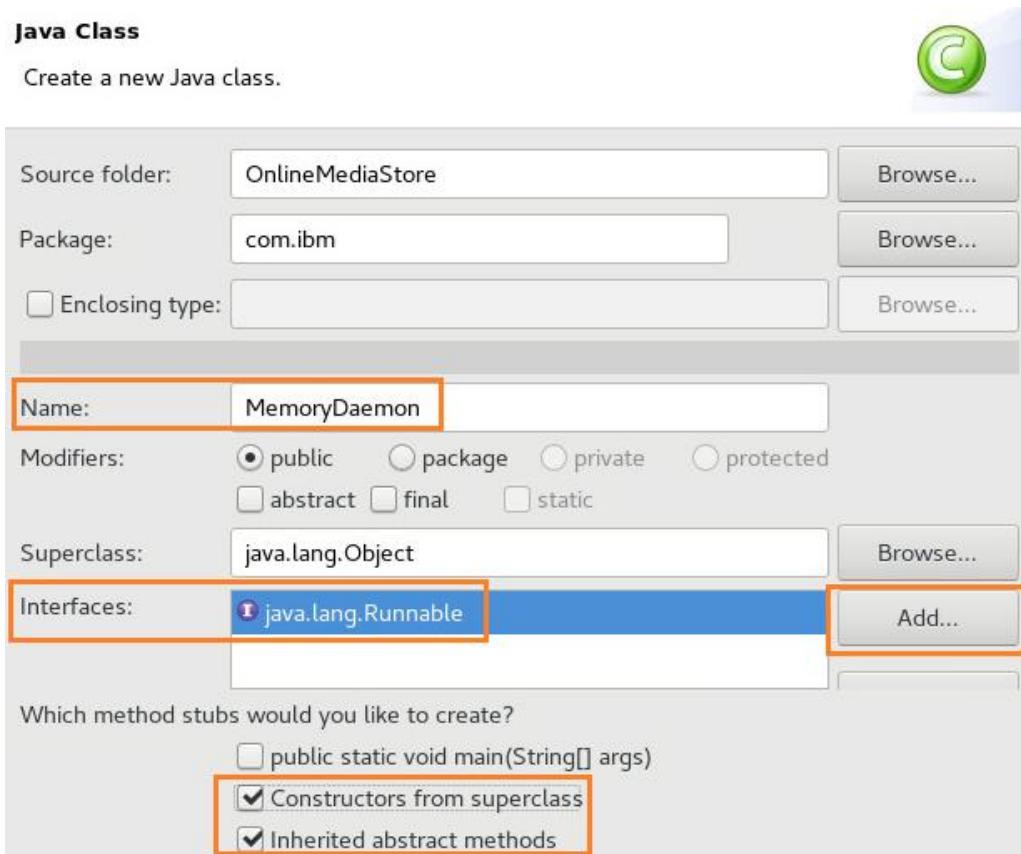
### Что вы должны будете сделать:

- Создавать потоки
- Использовать потоки, работающие в фоновом режиме

## **Раздел 1. Создание класса *MemoryDaemon***

В данном разделе создается класс, запускающий поток в фоновом режиме для мониторинга состояния памяти.

1. Откройте новое рабочее пространство
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/workspaces/Threads** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
2. Импортируйте проекты в рабочее пространство из каталога **/root/labfiles/Threads/import/OnlineMediaStore**. Если вы забыли последовательность действий, то можно воспользоваться инструкцией из начала прошлого упражнения.
3. Перейдите в перспективу **Java**.
4. Создайте класс **MemoryDaemon**.
  - a. Из главного меню перейдите: **File -> New -> Class**.  
Также можно нажать правой кнопкой мыши по проекту **OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.
  - b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.
  - c. В названии пакета укажите **com.ibm**.
  - d. Укажите **MemoryDaemon** как имя класса.
  - e. **Extended interface:** **java.lang.Runnable**.
  - f. **Не устанавливайте** опцию **public static void main(String[] args)**.
  - g. **Установите** опции **Constructors from superclass** и **Inherited abstract methods**.



## h. Нажмите **Finish**.

5. Добавьте в класс поле для хранения уровня утилизации памяти, инициализировав ее значением 0:

```
private long memoryUsed = 0;
```

6. Реализуйте публичные методы **getter** и **setter** для работы с этим полем.

7. Реализуйте метод **run()**, который будет запускаться при старте соответствующего потока. Для контроля за памятью можно использовать методы работы с памятью для объекта Java Runtime.

а. В качестве примера можно использовать следующий код:

```
public void run() {  
    Runtime rt = Runtime.getRuntime();  
    long used;  
  
    while (true) {  
        used = rt.totalMemory() - rt.freeMemory();  
        if (used != getMemoryUsed()) {  
            System.out.println("\tMemory used = " +  
                used);  
        }  
    }  
}
```

```
        setMemoryUsed(used);
    }
}
}
```

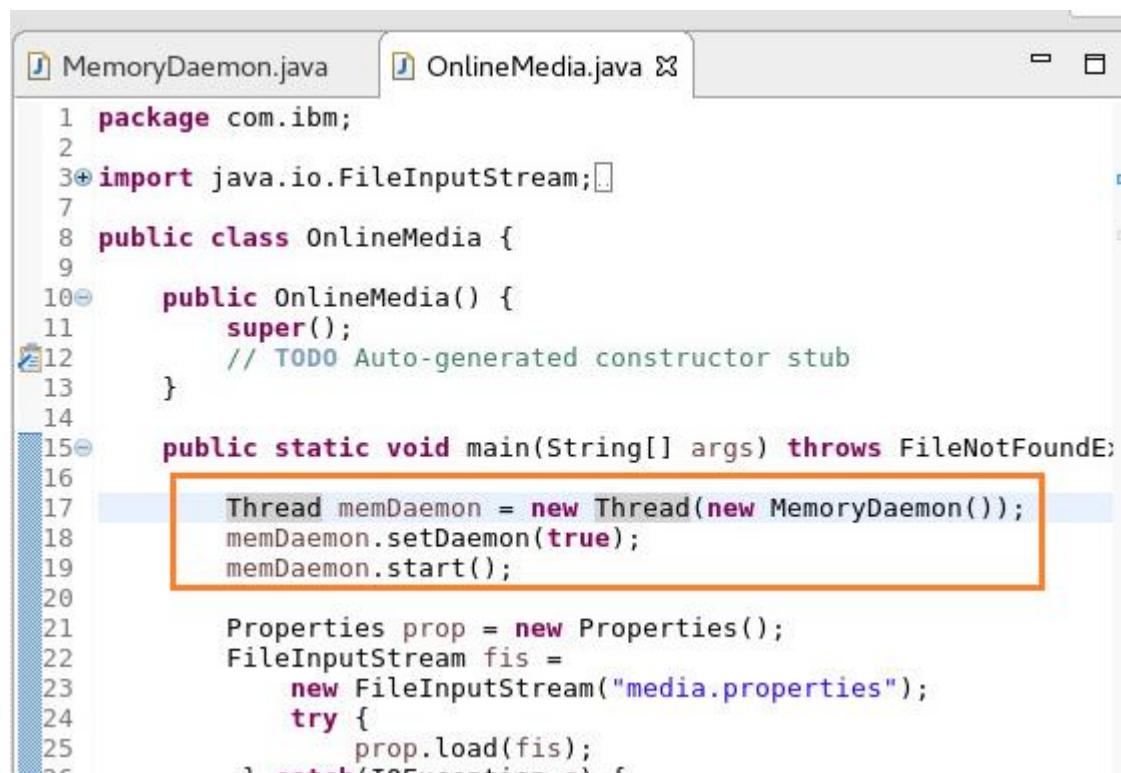
b. **Сохраните** сделанные изменения.

8. Обновите способ запуска программы таким образом, чтобы вспомогательный поток типа **MemoryDaemon** запускался бы вместе с ней и контролировал бы состояние памяти.

a. Откройте класс **OnlineMedia**.

b. В самое начало метода main добавьте следующие строчки:

```
Thread memDaemon = new Thread(new MemoryDaemon());
memDaemon.setDaemon(true);
memDaemon.start();
```



```
1 package com.ibm;
2
3+ import java.io.FileInputStream;[]
4
5 public class OnlineMedia {
6
7     public OnlineMedia() {
8         super();
9         // TODO Auto-generated constructor stub
10    }
11
12
13
14
15+     public static void main(String[] args) throws FileNotFoundException {
16+         Thread memDaemon = new Thread(new MemoryDaemon());
17+         memDaemon.setDaemon(true);
18+         memDaemon.start();
19+
20+
21     Properties prop = new Properties();
22     FileInputStream fis =
23         new FileInputStream("media.properties");
24     try {
25         prop.load(fis);
26     } catch (IOException e) {
27         e.printStackTrace();
28     }
29 }
```

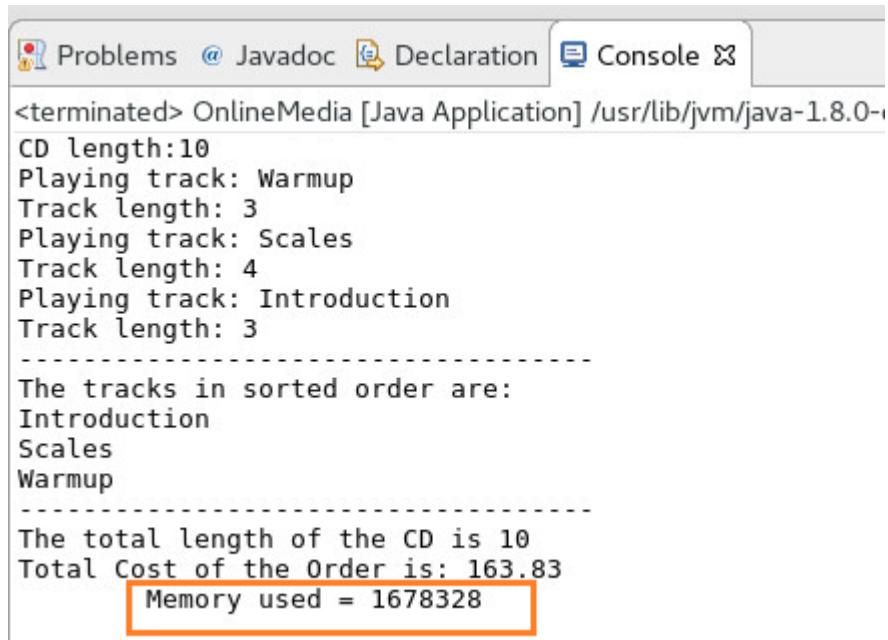
c. **Сохраните** сделанные изменения.

9. Проверьте работу программы.

a. Запустите **OnlineMedia** с помощью кнопки **Run**.

b. Откройте представление **Console**.

с. Обратите внимание в листинге на строчки, описывающие утилизацию памяти:



The screenshot shows a Java application window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the following output:

```
<terminated> OnlineMedia [Java Application] /usr/lib/jvm/java-1.8.0-
CD length:10
Playing track: Warmup
Track length: 3
Playing track: Scales
Track length: 4
Playing track: Introduction
Track length: 3
-----
The tracks in sorted order are:
Introduction
Scales
Warmup
-----
The total length of the CD is 10
Total Cost of the Order is: 163.83
Memory used = 1678328
```

The last line, "Memory used = 1678328", is highlighted with a red rectangle.

## Упражнение 9. Исключения

---

### О чём это упражнение:

В этой лабораторной работе рассматривается работа с исключениями и их обработка.

Создается класс исключений, который называется **PlayerException** и который наследуется от класса **Exception**. Эта исключительная ситуация будет возникать, когда метод `play()` должен проиграть нечто, с длиной 0. По ходу этого упражнения редактируются многие классы и методы, чтобы вызывать ошибки и обрабатывать их.

### Что вы должны будете сделать:

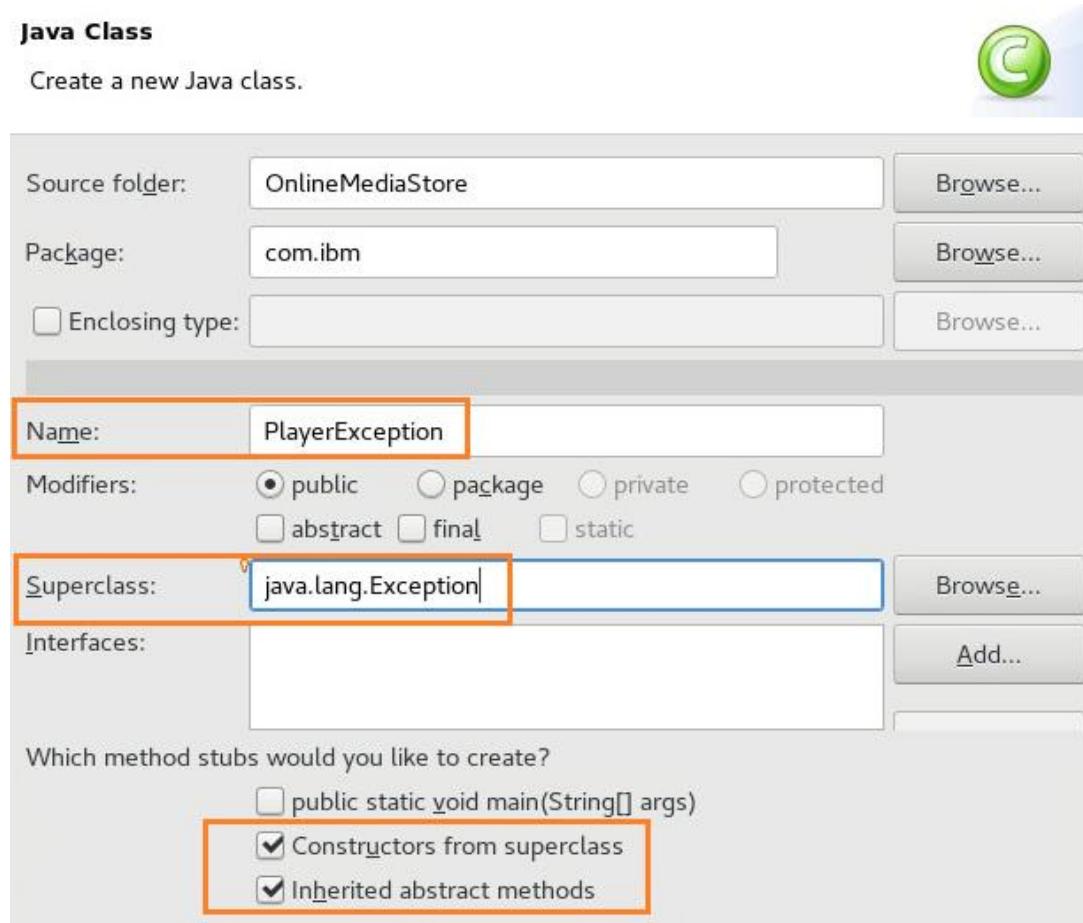
- Создавать классы ошибок
- Выбрасывать ошибки
- Ловить ошибки

## **Раздел 1. Создание класса *PlayerException***

В данном разделе создается класс, описывающий ошибку, возникающую при проигрывании медиа ресурса в соответствующем методе.

1. Откройте новое рабочее пространство
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/worksplaces/Exceptions** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
2. Импортируйте проекты в рабочее пространство из каталога **/root/labfiles/Exceptions/import/OnlineMediaStore**. Если вы забыли последовательность действий, то можно воспользоваться инструкцией из начала прошлого упражнения.
3. Перейдите в перспективу **Java**.
4. Создайте класс **PlayerException**.
  - a. Из главного меню перейдите: **File -> New -> Class**.  
Также можно нажать правой кнопкой мыши по проекту **OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.
  - b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.
  - c. В названии пакета укажите **com.ibm**.
  - d. Укажите **PlayerException** как имя класса.
  - e. **Superclass: java.lang.Exception**.
  - f. **Не устанавливайте опцию public static void main(String[] args)**.

## g. Установите опцию Constructors from superclass.



h. Нажмите **Finish**.

## Раздел 2. Реализация логики генерации и обработки исключений

В данном разделе редактируется логика методов **play()** для выбрасывания исключений при нулевой длине контента. Также меняется логика главного модуля для обработки исключительной ситуации. Далее приложение тестируется с соответствующими данными.

1. Обновите логику метода **play()** для класса **DigitalVideoDisc**.

а. Замените текущую реализацию на следующую:

```
if(this.getLength() <= 0) {  
    System.err.println("ERROR: DVD length is 0");  
    throw (new PlayerException());  
}  
System.out.println("Playing DVD: " +  
    this.getTitle());
```

```
System.out.println("DVD length: " +  
this.getLength());
```

- b. После этого изменения возникает ошибка, связанная с недопустимостью выбрасывания такого исключения в данном классе. Измените объявление метода в классе на следующее:
- ```
public void play() throws PlayerException
```

2. Аналогичным образом поменяйте логику метода **play()** для класса **Track**.

- a. Вы можете использовать следующий код:

```
if (this.getLength() <= 0) {  
    System.err.println("ERROR: Track length is 0");  
    throw (new PlayerException());  
}  
  
System.out.println("Playing Track: " +  
this.getTitle());  
  
System.out.println("Track length: " +  
this.getLength());  
}
```

- b. Аналогично поменяйте объявление метода в классе, обозначив возможность выбрасывания **PlayerException**.

3. Измените интерфейс **Playable** таким образом, чтобы сделать возможность выбрасывать исключения типа **PlayerException**:



```
1 package com.ibm;  
2  
3 public interface Playable {  
4     public void play() throws PlayerException;  
5 }
```

4. Измените метод **play()** для класса **CompactDisc**. Нужно проверять длину альбома, и если она нулевая, то выбрасывать исключение. Но в этом же методе вызывается проигрывание каждого трека с диска. И если при проигрывании трека возникает **PlayerException**, то эта ошибка должна быть обработана в методе **play()** для класса **CompactDisk**. Также следует добавить в объявление метода возможность выбрасывания соответствующего исключения.

- а. По итогам всех манипуляций должен получиться код, похожий на представленный ниже:

```
public void play() throws PlayerException {
    if (this.getLength() <= 0) {
        System.err.println("ERROR: CD length is 0");
        throw (new PlayerException());
    }
    System.out.println("Playing CD: " + this.getTitle());
    System.out.println("CD length:" + this.getLength());
    for (Track track : tracks) {
        try {
            track.play();
        }
        catch (PlayerException e) {
            e.printStackTrace();
        }
    }
}
```

5. **Сохраните** все сделанные изменения.

6. Обновите логику класса **OnlineMedia** так, чтобы там обрабатывались ошибки типа **PlayerException** при вызове методов **play()**.

а. Откройте класс **OnlineMedia** в редакторе.

б. Найдите все вызовы метода **play()**. Поместите их в блоки **try/catch**. Например, для вызова `dvd.play()`:

```
try {
    dvd.play();
} catch(PlayerException e) {
    e.printStackTrace();
}
```

с. **Сохраните** сделанные изменения.

7. Запустите класс **OnlineMedia**. Он завершает свою работу так же, как и в прошлом упражнении, так как на данный момент продолжительность у любого артефакта больше 0.

8. Измените **media.properties** так, чтобы возникло исключение.

а. Найдите в этом файле строчку, начинающуюся с `t2.length` и поставьте значение 0.

б. **Сохраните** изменения.

с. Перезапустите программу.

9. Вы должны увидеть листинг, похожий на показанный ниже, говорящий о возникшей исключительной ситуации.

```
-----  
The tracks currently are in the order:  
t1.title = Warmup  
t2.title = Scales  
t3.title = Introduction  
-----  
Playing CD: IBM Symphony  
CD length:6  
Playing Track: Warmup  
Track length: 3  
-----  
The tracks in sorted order are:  
Introduction  
Scales  
Warmup  
-----  
The total length of the CD is 6  
Total Cost of the Order is: 163.83  
ERROR: Track length is 0  
com.ibm.PlayerException  
    at com.ibm.Track.play(Track.java:25)  
    at com.ibm.CompactDisc.play(CompactDisc.java:35)  
    at com.ibm.OnlineMedia.main(OnlineMedia.java:76)  
Memory used = 2013928|
```

10. Проверьте также случаи с нулевой длиной DVD и CD.
11. После проведения всех экспериментов верните данные в состояние, при котором исключительных ситуаций возникать не должно.

## Упражнение 10. JavaBeans

---

### О чём это упражнение:

В этой лабораторной работе рассматриваются вопрос использования JavaBeans API.

В первой части упражнения создается бин **Media** со связанным свойством **cost**. То есть, любое изменение значения этого свойства приводит к появлению события **PropertyChangeEvent**. Это событие может обрабатываться любым количеством зарегистрированных слушателей. В частности создается слушатель **MediaPriceChangeListener**, который фиксирует изменения этого свойства и выводит новое значение в консоль.

Во второй части упражнения поле **category** класса **Media** становится свойством с ограничением. В этом случае фиксируется список зарегистрированных слушателей **VetoableChangeListeners**. Перед изменением свойства, слушатели решают, возможно ли произвести это изменение. Если нет, то выбрасывается исключение **PropertyVetoException**.

### Что вы должны будете сделать:

- Создавать параметры JavaBeans
- Использовать события и слушателей

## Раздел 1. Создание связанного свойства cost

1. Откройте новое рабочее пространство
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/workspaces/JavaBeans** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
2. Импортируйте проекты в рабочее пространство из каталога **/root/labfiles/JavaBeans/import/OnlineMediaStore**. Если вы забыли последовательность действий, то можно воспользоваться инструкцией из начала прошлого упражнения.
3. Перейдите в перспективу **Java**.
4. Добавьте в класс **Media** свойство **cost**.
  - a. Откройте в редакторе класс **Media.java**.
  - b. Для отслеживания изменений свойства **cost** добавьте в класс объявление новой переменной.

```
private PropertyChangeSupport changes = new  
PropertyChangeSupport(this);
```

**PropertyChangeSupport** – вспомогательный класс для отправки событий и управления слушателями.

- c. Добавьте в класс методы регистрации и удаления слушателей:

```
public void  
addPropertyChangeListener(PropertyChangeListener p)  
{  
    changes.addPropertyChangeListener(p);  
}  
  
public void  
removePropertyChangeListener(PropertyChangeListener p) {  
    changes.removePropertyChangeListener(p);  
}
```

- d. Измените метод **setCost()** так, чтобы создавалось событие **PropertyChangeEvent** для всех зарегистрированных слушателей.

```
public void setCost(float cost) {  
    Float oldCost = new Float (this.cost);  
    this.cost = cost;  
    changes.firePropertyChange("cost", oldCost,  
        new Float(this.cost));  
}
```

Изменения в методе выделены жирным шрифтом.

## 5. Импортируйте классы **PropertyChangeSupport** и

### **PropertyChangeListener**.

a. Нажмите правой кнопкой мыши в редакторе и выберите **Source -> Organize Imports**.

b. Сохраните сделанные изменения. На вкладке **Problems** не должно быть ошибок компиляции.

## 6. Создайте класс **MediaPriceChangeListener**.

a. Из главного меню перейдите: **File -> New -> Class**.

Также можно нажать правой кнопкой мыши по проекту **OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.

b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.

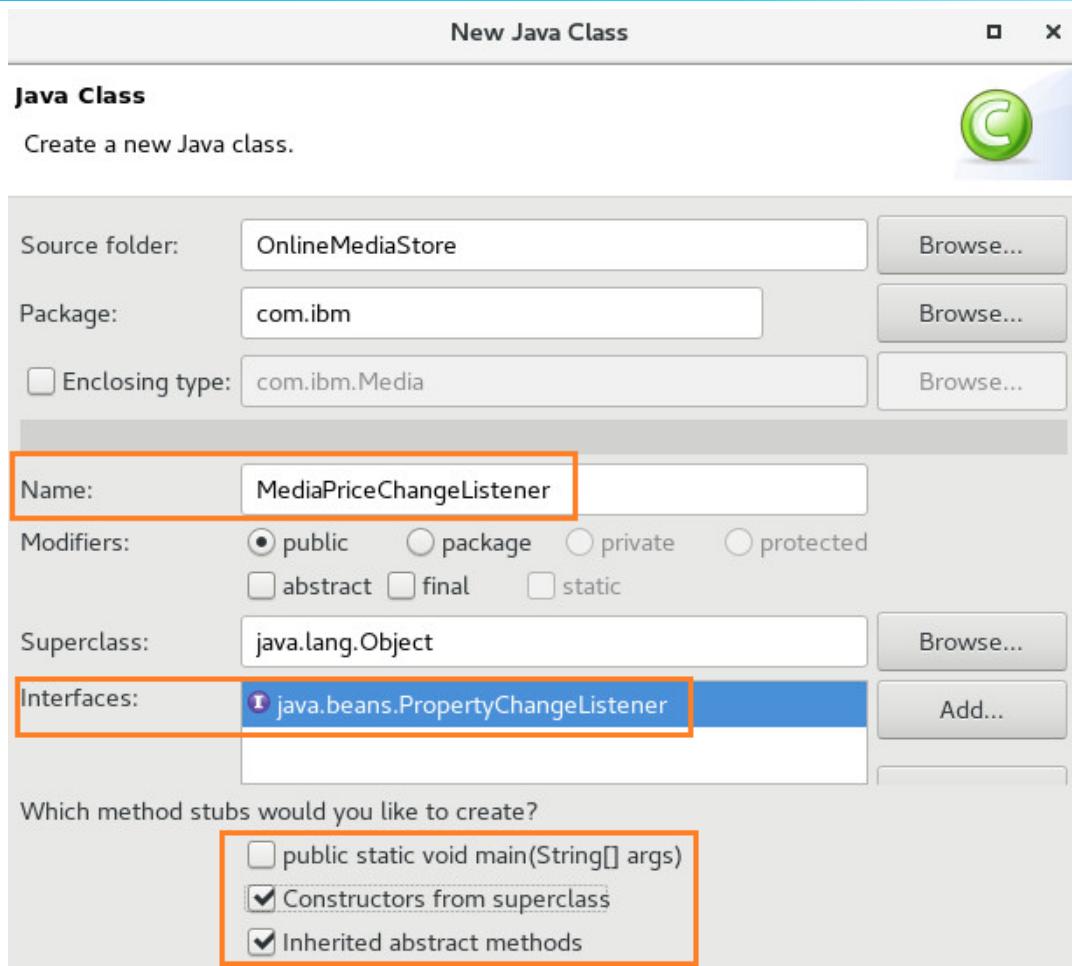
c. В названии пакета укажите **com.ibm**.

d. Укажите **MediaPriceChangeListener** как имя класса.

e. Interfaces: **java.beans.PropertyChangeListener**

f. Не устанавливайте опцию **public static void main(String[] args)**.

g. Выберите опции **Constructors from superclass** и **Inherited abstract methods**.



## h. Нажмите Finish.

7. Реализуйте метод **propertyChange** в созданном классе. Сделайте так, чтобы в консоль выводилась информация о том, уменьшилась или увеличилась стоимость и указывались бы старое и новое значения.

a. Используйте следующий код:

```
public void propertyChange(PropertyChangeEvent evt)
{
    float oldvalue = ((Float)
evt.getOldValue()).floatValue();
    float newvalue = ((Float)
evt.getNewValue()).floatValue();

    if (oldvalue < newvalue) {
        System.out.println("[The " +
evt.getPropertyName() + " of "
+ (Media)
evt.getSource()).getTitle()
```

```
        + " has increased from $" + oldvalue
        + " to $" + newvalue
        + " ]");
    } else {
        System.out.println("[The " +
evt.getPropertyName() + " of "
        + (Media)
evt.getSource()).getTitle()
        + " has decreased from $" + oldvalue
        + " to $" + newvalue
        + " ]");
    }
}
```

- b. Импортируйте нужные классы, если возникнет такая необходимость:

```
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
```

- c. Сохраните сделанные изменения.

8. Отредактируйте код главного модуля, чтобы зарегистрировать слушателя, увеличить и уменьшить значение свойства.

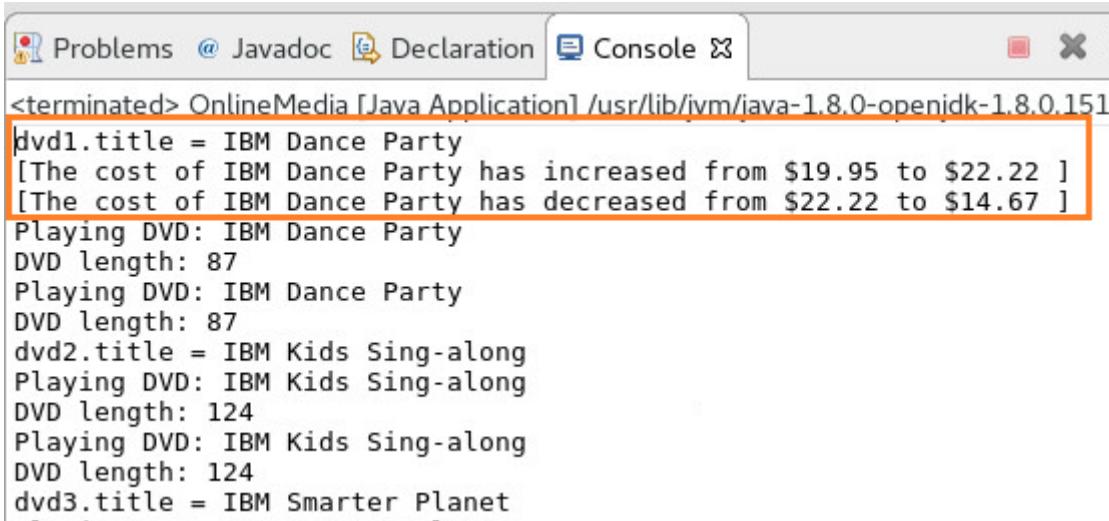
- a. Откройте в редакторе класс **OnlineMedia**.

- b. В методе **main** сразу после начала цикла создания DVD и получения переменной **dvd** в этом цикле (то есть, до блока **try**) добавьте следующий код:

```
if (dvdNumber == 1) {
    MediaPriceChangeListener mpl = new
MediaPriceChangeListener();
    //Register the new listener with dvd1
    dvd.addPropertyChangeListener(mpl);
    //Increase the cost of dvd1
    dvd.setCost(22.22f);
    //Decrease the cost of dvd1
    dvd.setCost(14.67f);
}
```

- c. Сохраните сделанные изменения.

9. Запустите класс **OnlineMedia**. В консоли должен открыться следующий листинг:



The screenshot shows an IDE interface with tabs for Problems, Javadoc, Declaration, and Console. The Console tab displays the application's output. A red box highlights the following text:  
<terminated> OnlineMedia [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151  
dvd1.title = IBM Dance Party  
[The cost of IBM Dance Party has increased from \$19.95 to \$22.22 ]  
[The cost of IBM Dance Party has decreased from \$22.22 to \$14.67 ]  
Playing DVD: IBM Dance Party  
DVD length: 87  
Playing DVD: IBM Dance Party  
DVD length: 87  
dvd2.title = IBM Kids Sing-along  
Playing DVD: IBM Kids Sing-along  
DVD length: 124  
Playing DVD: IBM Kids Sing-along  
DVD length: 124  
dvd3.title = IBM Smarter Planet

## Раздел 2. Создание свойства с ограничением category

1. Добавьте в класс **Media** поддержку для свойства с ограничением.
  - a. Откройте в редакторе класс **Media.java**.
  - b. Для поддержки ограниченных изменений свойства **cost** добавьте в класс объявление новой переменной.

```
private VetoableChangeSupport vetaes = new  
VetoableChangeSupport(this);
```

VetoableChangeSupport – вспомогательный класс для отправки событий и управления слушателями.

- c. Добавьте в класс методы регистрации и удаления слушателей:

```
public void  
addVetoableChangeListener(VetoableChangeListener v)  
{  
    vetaes.addVetoableChangeListener(v);  
}  
  
public void  
removeVetoableChangeListener(VetoableChangeListener v) {  
    vetaes.removeVetoableChangeListener(v);  
}
```

- d. Измените метод **setCategory()** так, чтобы создавалось событие **PropertyChangeEvent** для всех зарегистрированных слушателей.

```
public void setCategory(String category) {  
    String oldCategory = new String(this.category);  
    this.category = category;  
    vetoes.fireVetoableChange("category",  
    oldCategory, this.category);  
}
```

Изменения в методе выделены жирным шрифтом.

2. Импортируйте классы **VetoableChangeSupport** и

**VetoableChangeListener**.

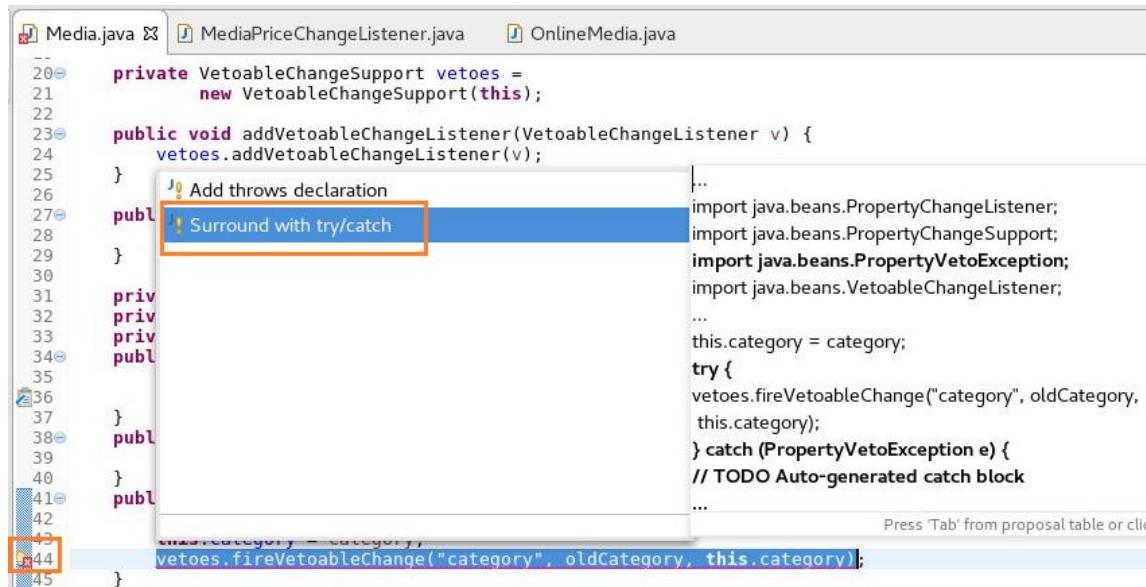
- a. Нажмите правой кнопкой мыши в редакторе и выберите **Source -> Organize Imports**.

3. Добавьте логику обработки ошибки **PropertyVetoException**.

- a. Нажмите по лампочке, указывающей на ошибку, слева от кода:

```
vetoes.fireVetoableChange("category", oldCategory,  
this.category);
```

- b. Выберите вариант **Surround with try/catch**.



- c. Замените строку:

```
e.printStackTrace();
```

на строку:

```
System.out.println(e.getMessage());
```

- d. Модифицированный метод должен быть похож на показанный ниже:

```
    }
    public void setCategory(String category) {
        String oldCategory = new String(this.category);
        this.category = category;
        try {
            vetoes.fireVetoableChange("category", oldCategory, this.category);
        } catch (PropertyVetoException e) {
            // TODO Auto-generated catch block
            System.out.println(e.getMessage());
        }
    }
```

- e. Сохраните сделанные изменения.

## 4. Создайте класс **MediaCategoryChangeListener**.

- a. Из главного меню перейдите: **File -> New -> Class**.

Также можно нажать правой кнопкой мыши по проекту

**OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.

- b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.

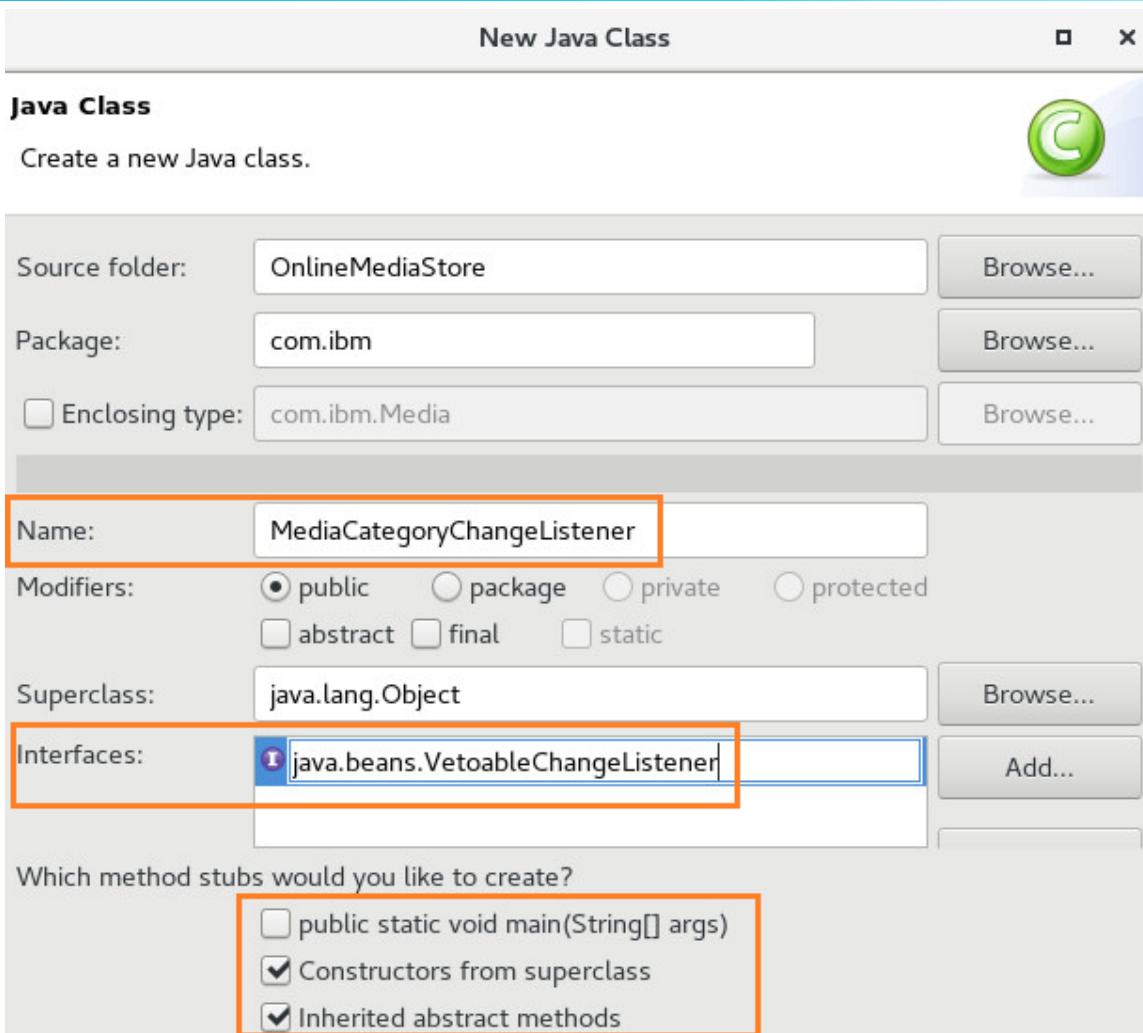
- c. В названии пакета укажите **com.ibm**.

- d. Укажите **MediaCategoryChangeListener** как имя класса.

- e. Interfaces: **java.beans.VetoableChangeListener**

- f. Не устанавливайте опцию **public static void main(String[] args)**.

- g. Выберите опции **Constructors from superclass** и **Inherited abstract methods**.



## h. Нажмите **Finish**.

5. Реализуйте метод **vetoableChange** в созданном классе. Этот метод будет использоваться для принятия решения – разрешено ли изменение. Если оно запрещается, метод будет выбрасывать исключение **PropertyVetoException**. Предполагаемая логика заключается в том, что категорию для **CompactDisc** менять можно, а для других артефактов нельзя.

### a. Используйте следующий код:

```
public void vetoableChange(PropertyChangeEvent evt)
    throws PropertyVetoException {
    String oldvalue = (String) evt.getOldValue();
    String newvalue = (String) evt.getNewValue();
    if (evt.getSource() instanceof CompactDisc) {
        System.out.println("[The " +
            evt.getPropertyName()
            + " of this item has been changed
            from " + oldvalue
```

```
        + " to " + newValue + " ]");  
    } else {  
        throw new PropertyVetoException(  
            "[Sorry - the category for this  
media type can not be changed.]",  
            evt);  
    }  
}
```

- b. Импортируйте при необходимости недостающие классы.  
c. **Сохраните** сделанные изменения.
6. Отредактируйте код главного модуля, чтобы зарегистрировать слушателя, увеличить и уменьшить значение свойства.
- a. Откройте в редакторе класс **OnlineMedia**.
- b. В методе **main** сразу после создания **cd** добавьте следующий код:
- ```
System.out.println("-----");  
MediaCategoryChangeListener mcl = new  
MediaCategoryChangeListener();  
cd.addVetoableChangeListener(mcl);  
book.addVetoableChangeListener(mcl);  
cd.setCategory("Heavy metal");  
book.setCategory("Mystery");
```
- В этом коде для объектов **cd** и **book** назначается слушатель. После чего, для них устанавливаются категории.
- c. Перед запуском приложения, нужно изменить начальное значение категорий для экземпляров **Media**. Откройте этот класс и поменяйте объявление поля **category** на следующее:
- ```
private String category = "No category";
```
- d. **Сохраните** сделанные изменения.
- a. Запустите класс **OnlineMedia**. В консоли должен открыться следующий листинг:

The screenshot shows a Java application window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the following text:

```
<terminated> OnlineMedia [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-5.b12.el7_4.
Playing DVD: IBM Smarter Planet
DVD length: 90
Book title = Java Programming
cd1.title = IBM Symphony
Cost of CD = 29.95
-----
[The category of this item has been changed from Instrumental to Heavy metal ]
[Sorry - the category for this media type can not be changed.]
-----
The tracks currently are in the order:
t1.title = Warmup
t2.title = Scales
t3.title = Introduction
-----
Playing CD: IBM Symphony
```

A red box highlights the error message in the console.

**Конец упражнения**

## Упражнение 11. Сериализация

---

### О чём это упражнение:

В этом упражнении вы научитесь сохранять и восстанавливать сериализованные данные.

Для классов **Media** и **Order** организуется реализация интерфейса **java.io.Serializable**. Также создается новый класс **OrderSaver**, который будет использоваться для чтения и записи заказов в файловой системе.

### Что вы должны будете сделать:

- Реализовать интерфейс `Serializable`
- Сохранять и восстанавливать сериализованные данные

## Раздел 1. Объявление классов сериализуемыми

В данном разделе все классы, для которых требуется соответствующая возможность объявляются сериализуемыми.

6. Откройте новое рабочее пространство
    - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
    - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
    - c. Выберите каталог **/root/labfiles/workspaces/Serialization** и нажмите **Launch**.
    - d. Закройте страницу **Welcome**.
  7. Импортируйте проекты в рабочее пространство из каталога **/root/labfiles/Serialization/import/OnlineMediaStore**. Если вы забыли последовательность действий, то можно воспользоваться инструкцией из начала прошлого упражнения.
  8. Перейдите в перспективу **Java**.
  9. Объявите класс **Order** как сериализуемый.
    - a. Откройте файл **Order.java**.
    - b. Добавьте в объявление класса **Order** замечание о реализации интерфейса **Serializable**:

```
implements java.io.Serializable
```
- import java.util.ArrayList;
- public class Order implements java.io.Serializable{
- private ArrayList<Media> itemsOrdered = new ArrayList<Media>();
- public Order() {
- super();
- c. Нажмите по лампочке с предупреждением в редакторе. Выберите опцию **Add default serial version ID** из списка.

```
1 package com.ibm;
2
3 import java.util.ArrayList;
4
5 public class Order implements java.io.Serializable{
6     private A
7     public Or + Add default serial version ID
8         super + Add generated serial version ID
9         // TO
10    } ...
```

d. Сохраните сделанные изменения.

10. Сделайте все дочерние для Media классы реализующими интерфейс **java.io.Serializable**.

- Так как класс **Order** был объявлен как сериализуемый, а в его атрибутах есть список элементов **Media** (а все поля тоже должны быть сериализуемы), то соответствующие классы тоже должны реализовать данный интерфейс.
- Если какие-то из атрибутов не будут сериализуемыми, то во время их записи будет возникать исключительная ситуация **java.io.NotSerializableException**.
- Самый легкий способ обеспечить реализацию интерфейса всеми подклассами **Media** – это реализовать его на уровне самого родительского класса **Media**, чтобы все дочерние классы это унаследовали.
- Добавьте в объявление класса **Media** соответствующее замечание:

```
implements java.io.Serializable
```

```
1 package com.ibm;
2
3+ import java.beans.PropertyChangeListener; []
4
5 public class Media implements java.io.Serializable{
6     private String title;
7     private String category = "No category";
8     private float cost;
```

- Нажмите по лампочке с предупреждением в редакторе. Выберите опцию **Add default serial version ID** из списка.
- Сохраните сделанные изменения.

11. Сделайте класс **Track** сериализуемым. Это актуально, так как в заказе могут быть экземпляры класса **CompactDisc**, а в нем в свою очередь хранится информация о треках.

- Откройте **Track.java**.
- Добавьте в объявление класса **Track** замечание о реализации интерфейса **Serializable**:

```
implements java.io.Serializable
```

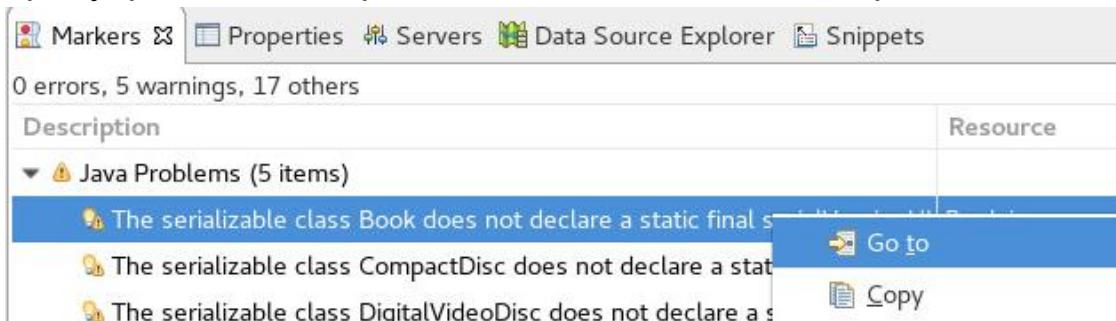


```
1 package com.ibm;
2
3 public class Track implements Playable,
4     Comparable<Track>, java.io.Serializable {
5     private String title;
6     private int length;
7     public Track() {
8         // TODO Auto-generated constructor stub
9     }
}
```

- Нажмите по лампочке с предупреждением в редакторе. Выберите опцию **Add default serial version ID** из списка.
- Сохраните** сделанные изменения.

12. Среда разработки выдает несколько предупреждений относительно добавления полей serial version ID в ряд классов: **Book**, **CompactDisc** и **DigitalVideoDisc**, **PlayerException**.

- Нажмите правой кнопкой мыши по одному из таких предупреждений в представлении **Markers**. Выберите **Go To**.



- Нажмите по лампочке с предупреждением в редакторе. Выберите опцию **Add default serial version ID** из списка.
- Сохраните** сделанные изменения.
- Повторите процедуру для остальных классов.

## Раздел 2. Создание класса для сериализации и его тестирование

В данном разделе создается класс **OrderSaver** с двумя статическими методами, которые будут использоваться для записи/чтения заказов в/из файловой системы.

1. Создайте новый класс **OrderSaver**.

a. Из главного меню перейдите: **File -> New -> Class**.

Также можно нажать правой кнопкой мыши по проекту

**OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.

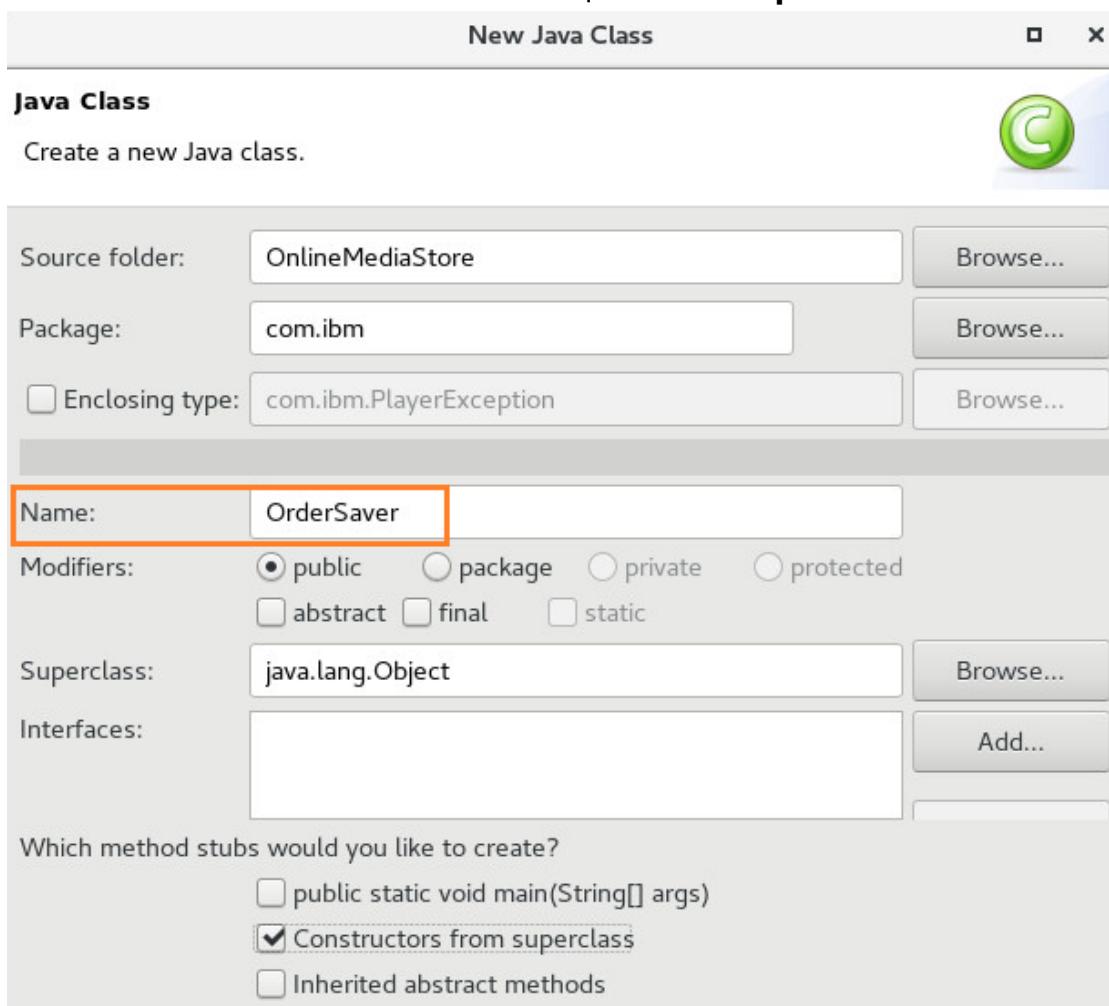
b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.

c. В названии пакета укажите **com.ibm**.

d. Укажите **OrderSaver** как имя класса.

e. **Constructors from superclass** – опция **выбрана**.

f. **Inherited abstract methods** – опция **не выбрана**.



g. Нажмите **Finish**.

2. Реализуйте методы **saveOrder** и **restoreOrder**. Это будут статические методы. Первый должен получать в качестве входных параметров: сохраняемый заказ и имя файла для сохранения. Второй – имя файла, откуда нужно считать данные. **restoreOrder** будет возвращать объект типа **Order**, считанный из файла.

a. В методе **saveOrder** будет использоваться метод **writeObject()** класса  **ObjectOutputStream**. Экземпляр соответствующего класса создается с использованием объекта типа  **FileOutputStream**, при создании которого, в свою очередь, требуется объект типа  **File**, который создается в начале и связывается с именем файла, куда нужно производить запись.

b. Вы можете использовать следующий код:

```
public static void saveOrder(String fileName, Order order1) {
    try {
        // Create a file to store the Order object in
        java.io.File objectFile = new
        java.io.File(fileName);
        if (objectFile.exists()) {
            objectFile.delete();
        }
        // Create the output stream to write out the
        file
        java.io.FileOutputStream fos = new
        java.io.FileOutputStream(objectFile);
        java.io.ObjectOutputStream oos = new
        java.io.ObjectOutputStream(fos);

        // Write the Order object out to the file
        oos.writeObject(order1);
        oos.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

c. В методе **restoreOrder()** будут использоваться похожие классы, но не Output, а Input:  **FileInputStream** и  **ObjectInputStream**.

d. Вы можете использовать следующий код:

```
public static Order restoreOrder(String fileName) {  
    try {  
        // Create a file to store the Order object in  
        java.io.File objectFile = new  
        java.io.File(fileName);  
  
        // Create the input stream to write out the  
        file  
        java.io.FileInputStream fis = new  
        java.io.FileInputStream(objectFile);  
        java.io.ObjectInputStream ois = new  
        java.io.ObjectInputStream(fis);  
  
        Order retrieved = (Order) ois.readObject();  
        ois.close();  
  
        return retrieved;  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    return null;  
}
```

e. Сохраните сделанные изменения.

3. Измените логику класса **OnlineMedia**. Вам необходимо проверить логику сериализации и десериализации. Можно сделать все за один тестовый прогон, воспользовавшись кодом ниже. Но лучше сделать два тестовых прогона с изменением программы. В первом случае сохранить заказ. Во втором случае считать заказ из файловой системы и убедиться, что это именно тот заказ, который был сохранен в предыдущем teste.

a. Вы можете воспользоваться такой шпаргалкой с примером использования созданных методов:

```
OrderSaver.saveOrder("order.ser", anOrder);  
Order newOrder =  
OrderSaver.restoreOrder("order.ser");  
  
System.out.print ("New Order's total cost is: ");
```

```
System.out.println (newOrder.totalCost());
```

- b. Сохраните сделанные изменения.
- c. Запустите **OnlineMedia** для проверки работы указанных методов.

```
<terminated> OnlineMedia [Java Application] /usr/lib/jvm/java-1.8.0-
[Sorry - the category for this media type can not be determined]
-----
The tracks currently are in the order:
t1.title = Warmup
t2.title = Scales
t3.title = Introduction
-----
Playing CD: IBM Symphony
CD length:10
Playing Track: Warmup
Track length: 3
Playing Track: Scales
Track length: 4
Playing Track: Introduction
Track length: 3
-----
The tracks in sorted order are:
Introduction
Scales
Warmup
-----
The total length of the CD is 10
Total Cost of the Order is: 158.55
    Memory used = 1678048
    Memory used = 2013648
    Memory used = 2349248
    Memory used = 2685832
    Memory used = 3021648
    Memory used = 3357328
New Order's total cost is: 158.55
    Memory used = 3693248
```

## Раздел 3. Использование нового API для ввода/вывода

В данном разделе от вас требуется адаптировать класс **MemoryDaemon** таким образом, чтобы информация выводилась не только в консоль, но и в файл **/tmp/mdaemon.log**. При этом необходимо использовать новый API с классами **Path** и **Files**.

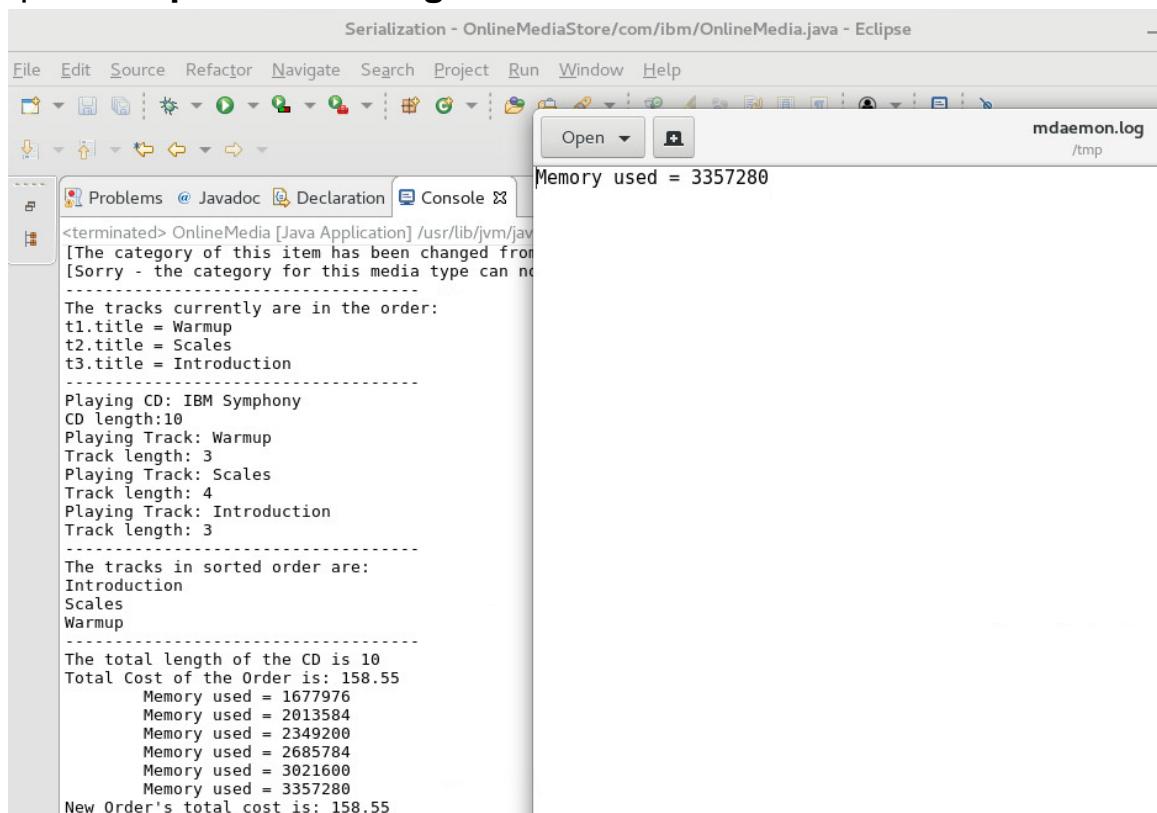
1. Внесите изменения в класс **MemoryDaemon**.
  - a. Откройте **MemoryDaemon** в редакторе.
  - b. Рядом со строками для вывода уровня утилизации памяти в консоль добавьте строчки для записи информации в файл. Вы

можете воспользоваться следующим примером:

```
Path path = Paths.get("/tmp/mdaemon.log");
try(BufferedWriter writer =
Files.newBufferedWriter(path, Charset.forName("UTF-
8"))){
    writer.write("Memory used = " + used + "\n");
} catch(IOException ex) {
    ex.printStackTrace();
}
```

c. Импортируйте необходимые классы.

d. Запустите **OnlineMedia** и убедитесь, что информация попадает в файл **/tmp/mdaemon.log**.



## Конец упражнения

## Упражнение 12. JUnit

---

### О чём это упражнение:

В этой лабораторной работе рассматривается вопрос создания тест кейсов в фреймворке JUnit и их запуск с помощью Eclipse.

Тесты создаются для методов класса Converter, который импортируется в рабочее пространство в начале упражнения.

Класс необходим для перевода единиц измерения. Запуск тестов и просмотр результатов осуществляется с помощью представления JUnit.

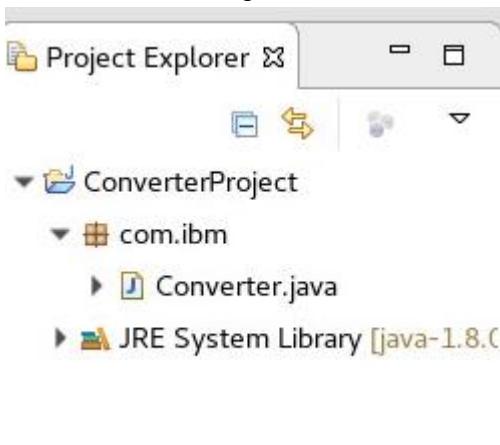
### Что вы должны будете сделать:

- Использовать JUnit для тестирования
- Создавать тест кейсы
- Запускать тесты

## Раздел 1. Генерация документации

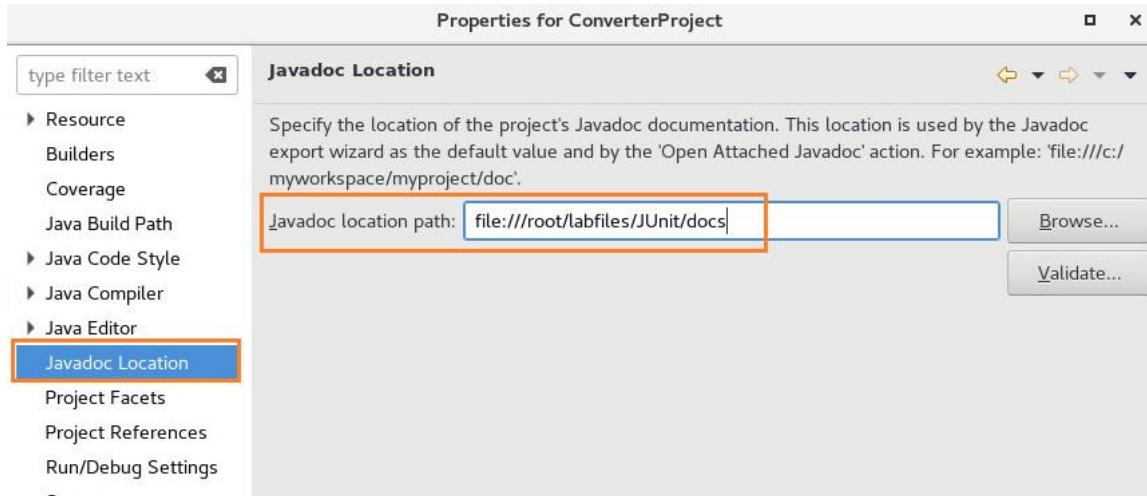
В данном разделе перед написанием тестов вы сгенерируете и изучите документацию по тестируемому классу **Converter**.

6. Откройте новое рабочее пространство
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/workspaces/JUnit** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
7. Импортируйте проекты в рабочее пространство из каталога **/root/labfiles/JUnit/import/ConverterProject**. Если вы забыли последовательность действий, то можно воспользоваться инструкцией из начала прошлого упражнения.
8. После импорта в рабочем пространстве появляется проект **ConverterProject** и класс **Converter**.



9. Перейдите в перспективу **Java**.
10. Укажите расположение каталога для генерации документации проекта.
  - a. В представлении **Project Explorer** нажмите правой кнопкой по проекту **ConverterProject** и выберите **Properties**.
  - b. Перейдите в раздел **Javadoc Location**.

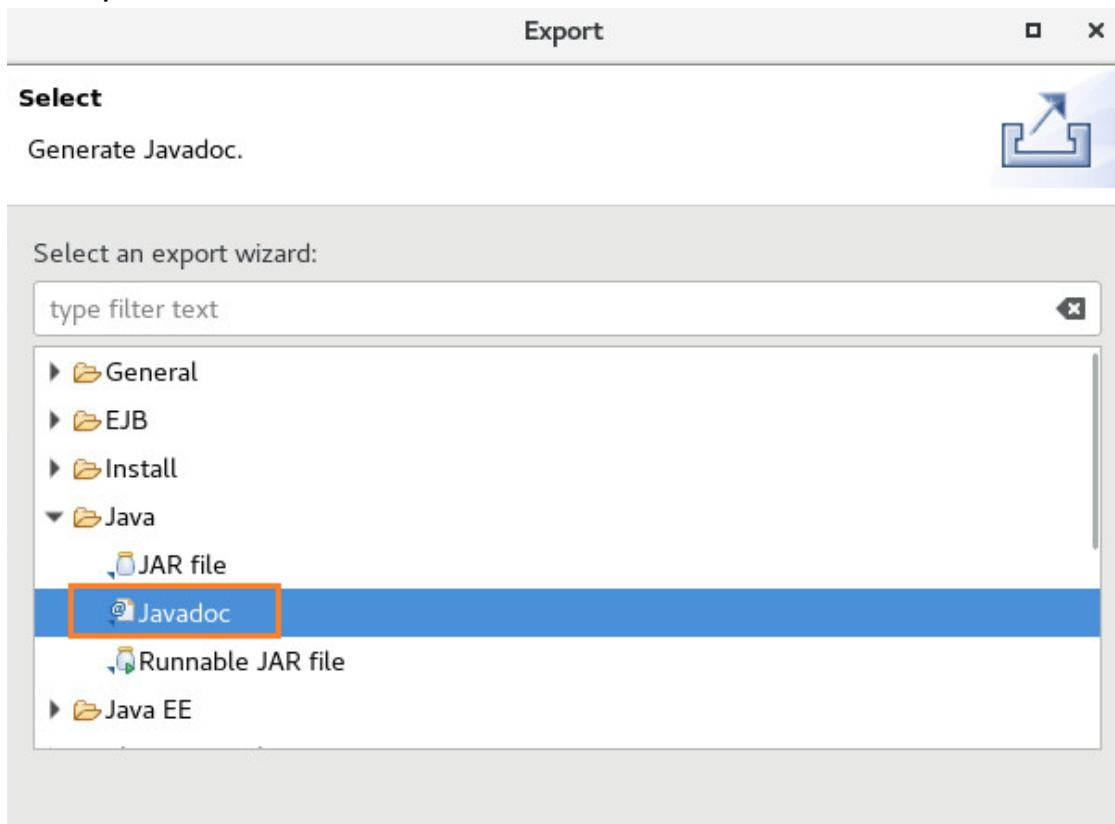
c. Введите `file:///root/labfiles/JUnit/docs.`



d. Нажмите **Apply and Close**.

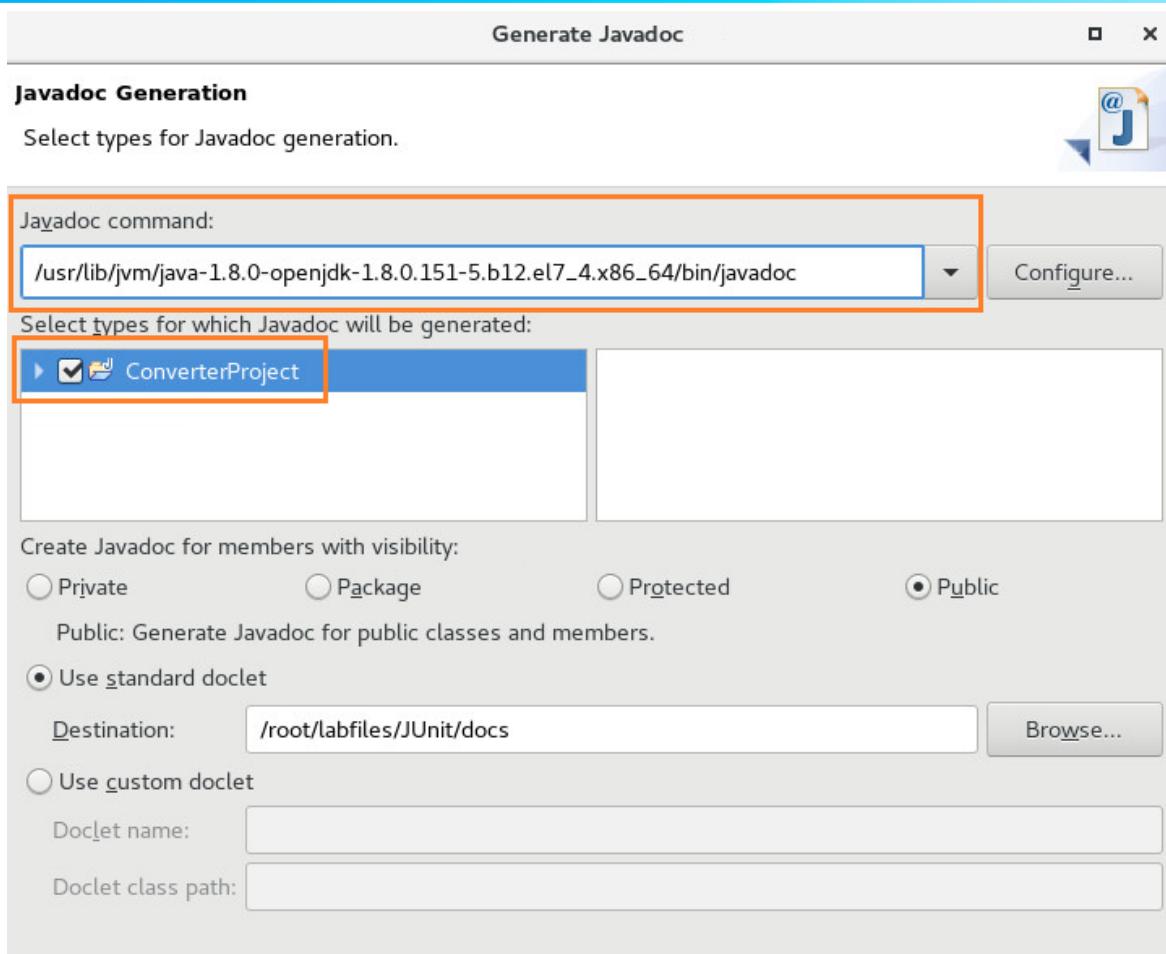
11. Сгенерируйте документацию.

- В представлении **Project Explorer** нажмите правой кнопкой по проекту **ConverterProject** и выберите **Export**.
- Выберите **Java -> Javadoc**.



c. Нажмите **Next**.

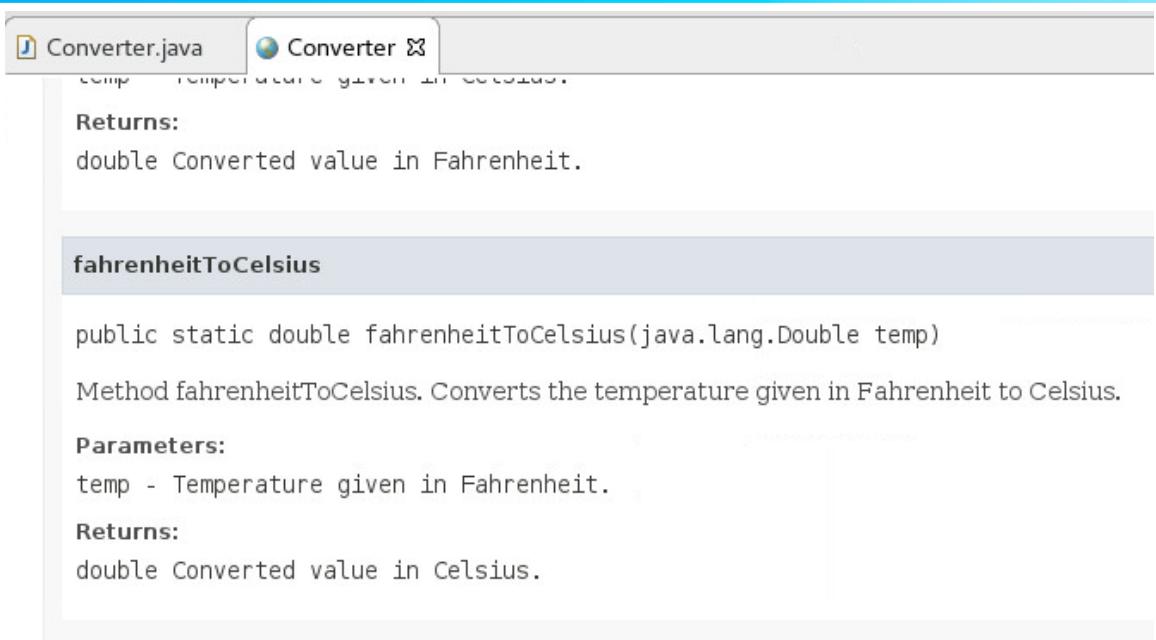
- Среда разработки по умолчанию предлагает правильный путь до исполняемого файла **javadoc** в поле **Javadoc command**. Убедитесь, что выбран проект **ConverterProject**.



е. Нажмите **Finish**. Ответьте **Но** на вопрос среди относительно изменения каталога. Документации должна сгенерироваться.

## 12. Просмотрите Javadoc.

- a. Откройте файл **Converter.java**.
- b. Нажмите **Shift + F2**. Документация откроется во встроенным браузере **Eclipse**.
- c. Выберите любой из методов файла и посмотрите Javadoc для него.



## Раздел 2. Создание тест кейсов

В данном разделе создается несколько тест кейсов, проверяющих работу методов импортированного ранее класса **Converter**. Каждый тест кейс возвращает результат **pass** или **fail**.

1. Создайте новый пакет для классов с тестами. Это хорошая практика хранить тесты в отдельном пакете. При этом такой пакет называется, как правило, так же, как и пакет, классы которого будут тестируться с дополнительным суффиксом **.test**. В нашем случае создается пакет для тестов **com.ibm.test**, в котором будут лежать тесты для класса/классов из пакета **com.ibm**.

- В главном меню выберите **File -> New -> Package**.
- Укажите имя для пакета **com.ibm.test**.
- Нажмите **Finish**.

2. Создайте JUnit тест кейс.

- В главном меню выберите **File -> New -> JUnit Test Case**
- Введите следующую информацию:

**New JUnit 4 test:**

выбрано

**Source folder:**

ConverterProject

**Package:**

com.ibm.test

**Name:**

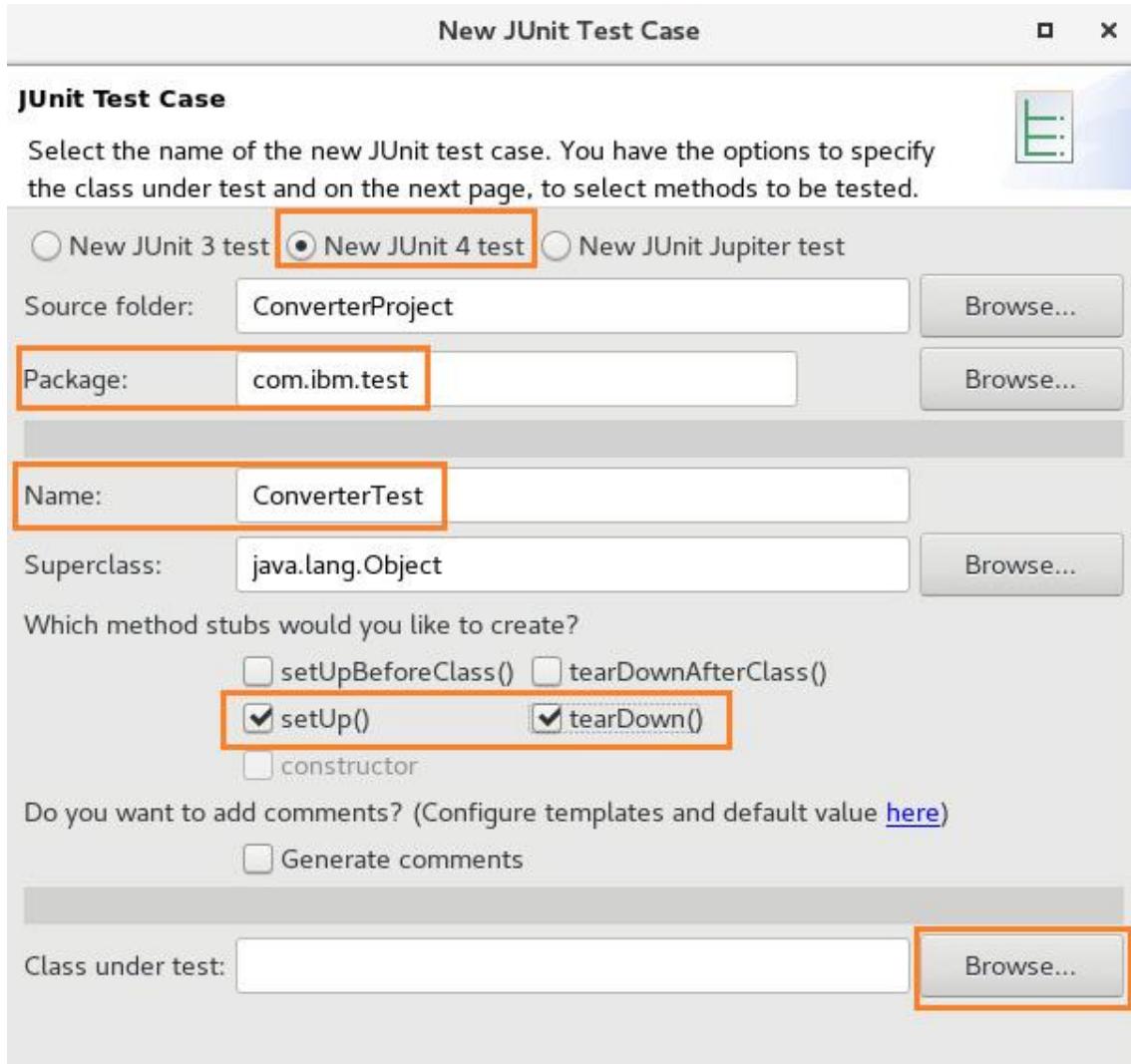
ConverterTest

**setUp():**

выбрано

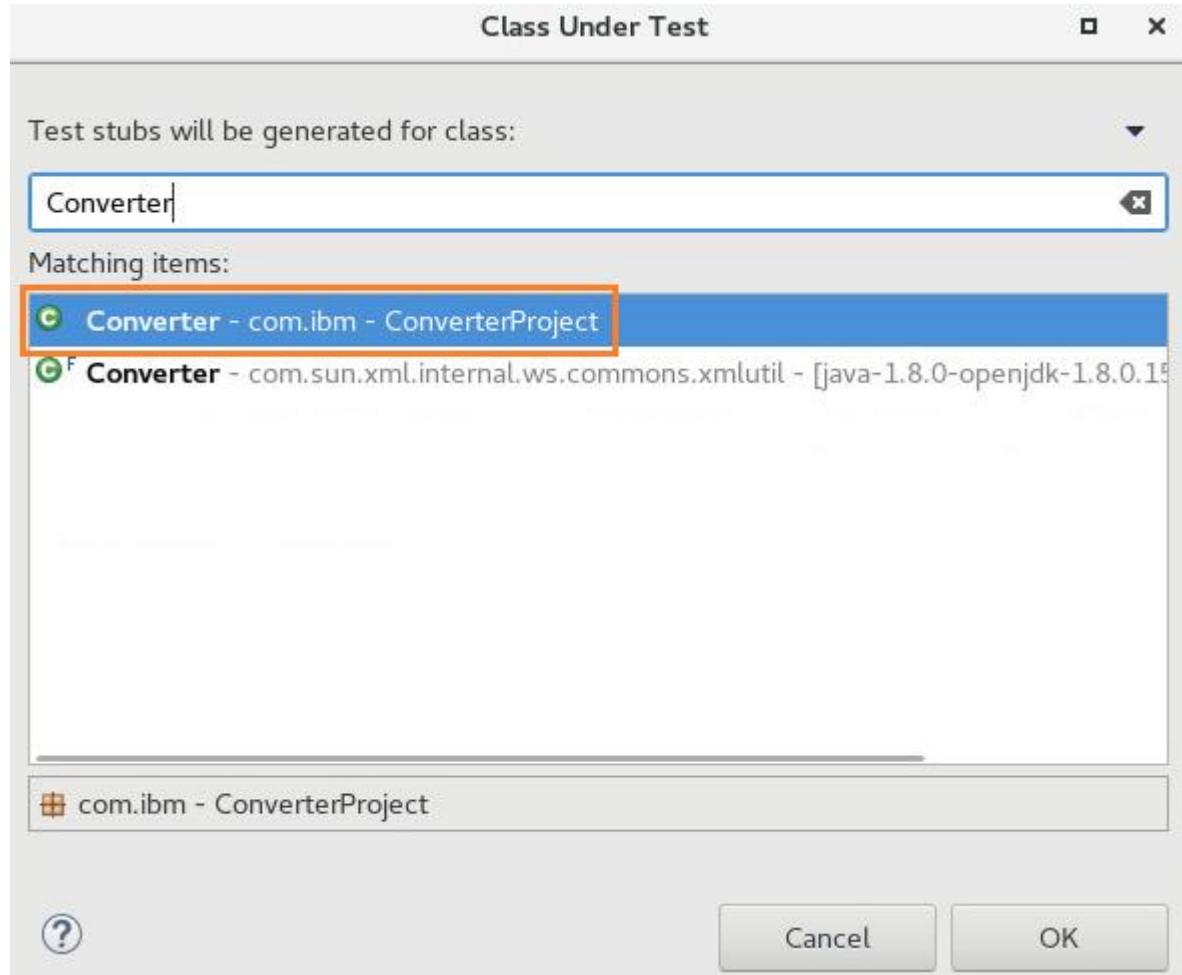
**tearDown():**

выбрано



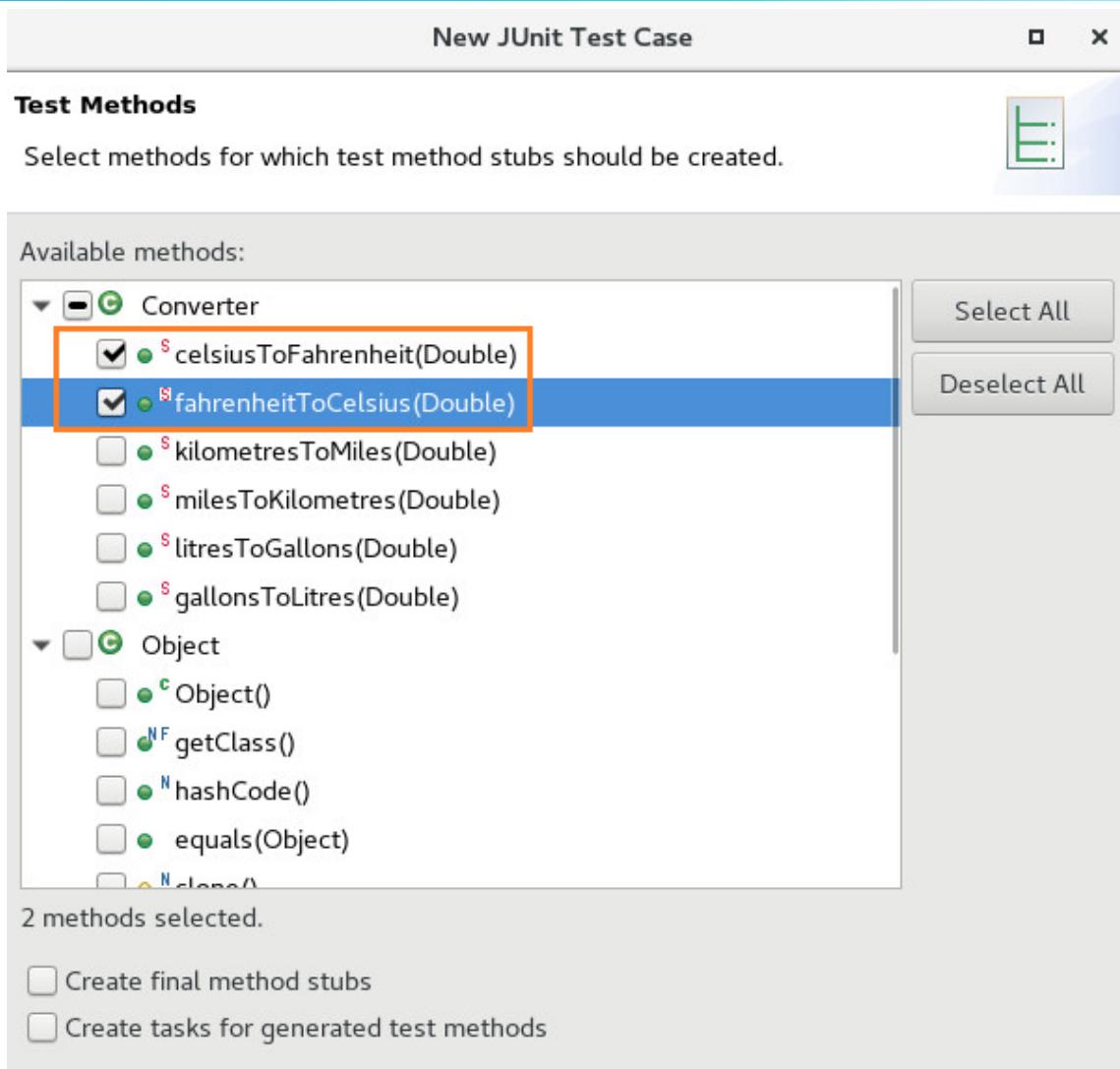
с. Нажмите **Browse** рядом с полем **Class under test**.

d. Введите **Converter – com.ibm – ConverterProject**. Нажмите **OK**.



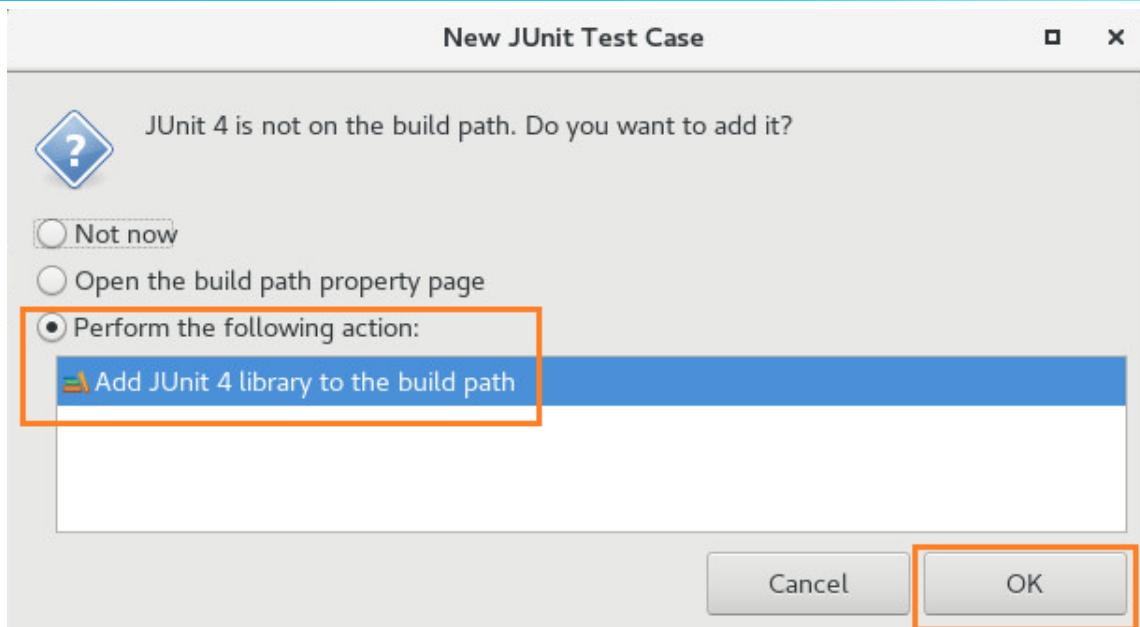
e. Нажмите **Next**.

f. Выберите методы **celsiusToFahrenheit(Double)** и **fahrenheitToCelsius(Double)** для тестирования.

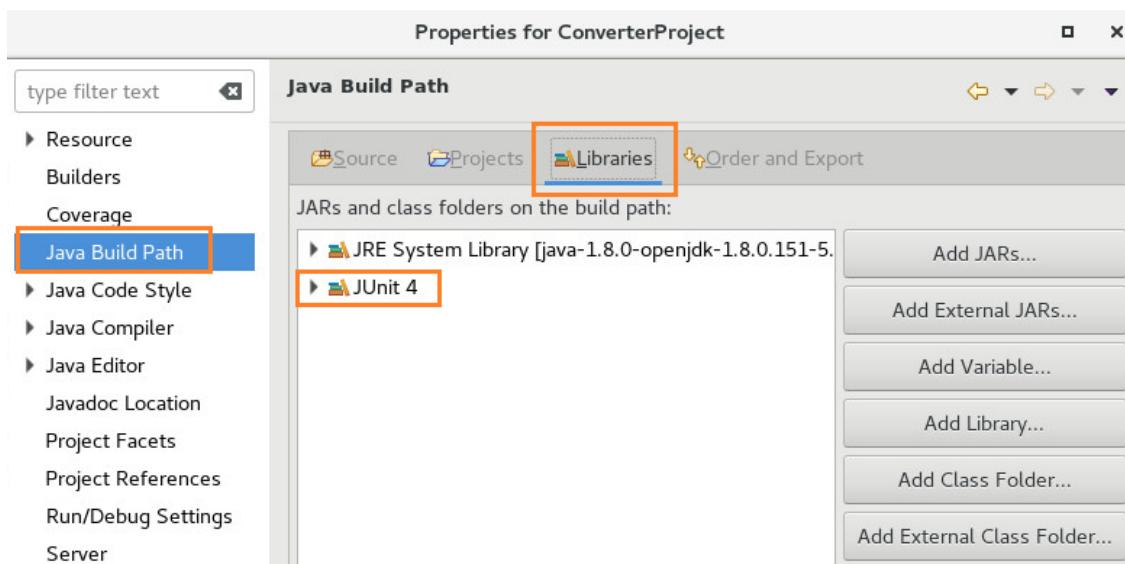


g. Нажмите **Finish**.

3. Открывается окно с сообщением о том, что JUnit 4 отсутствует в **Build Path**. Выберите вариант **Add JUnit 4 library to the build path** и нажмите **OK**.



Убедитесь в том, что нужная библиотека попала в проект можно, перейдя в **Properties** проекта и выбрав раздел **Java Build Path -> Libraries**.



4. Реализуйте логику тест кейса. Перед написанием теста, нужно подготовить среду для работы с теми объектами, которые будут использоваться в этих тестах. Для установки начальных значений, которые могут использоваться в нескольких тестах, можно использовать метод **setUp()**, а для очистки **tearDown()**.
  - a. Добавьте в класс с тестами импорт класса, который будет тестируться:

```
import com.ibm.Converter;
```

- b. Объявите набор переменных, которые будут использоваться в тестах: температура замерзания воды по Кельвину, по Фаренгейту и температура -40:

```
private Double freezingPtC;  
private Double freezingPtF;  
private Double neg40
```

- c. Реализуйте метод **setUp()** для установки значений этих переменных:

```
public void setUp() throws Exception {  
    freezingPtC = new Double ( 0 );  
    freezingPtF = new Double ( 32 );  
    neg40       = new Double ( -40 );  
}
```

- d. Напишите тест, основанный на том, что при переводе температуры замерзания воды по Цельсию должна получиться температура замерзания воды по Фаренгейту:

```
public void testCelsiusToFahrenheit() {  
    assertEquals (  
        freezingPtF.doubleValue () ,  
        Converter.celsiusToFahrenheit (  
            freezingPtC ) ,  
        0  
    );  
}
```

Используемый метод проверяет равенство первого параметра второму с учетом погрешности, указанной в третьем.

- e. Напишите тест для перевода температуры по Фаренгейту в температуру по Цельсию, с учетом того факта, что в двух этих системах значения температур -40 – идентичны:

```
public void testFahrenheitToCelsius () {  
    assertEquals (  
        neg40.doubleValue () ,  
        Converter.fahrenheitToCelsius ( neg40 ) ,  
        0  
    );
```

}

f. Сохраните сделанные изменения.

## Раздел 3. Создание тест кейсов для обработки исключительных ситуаций

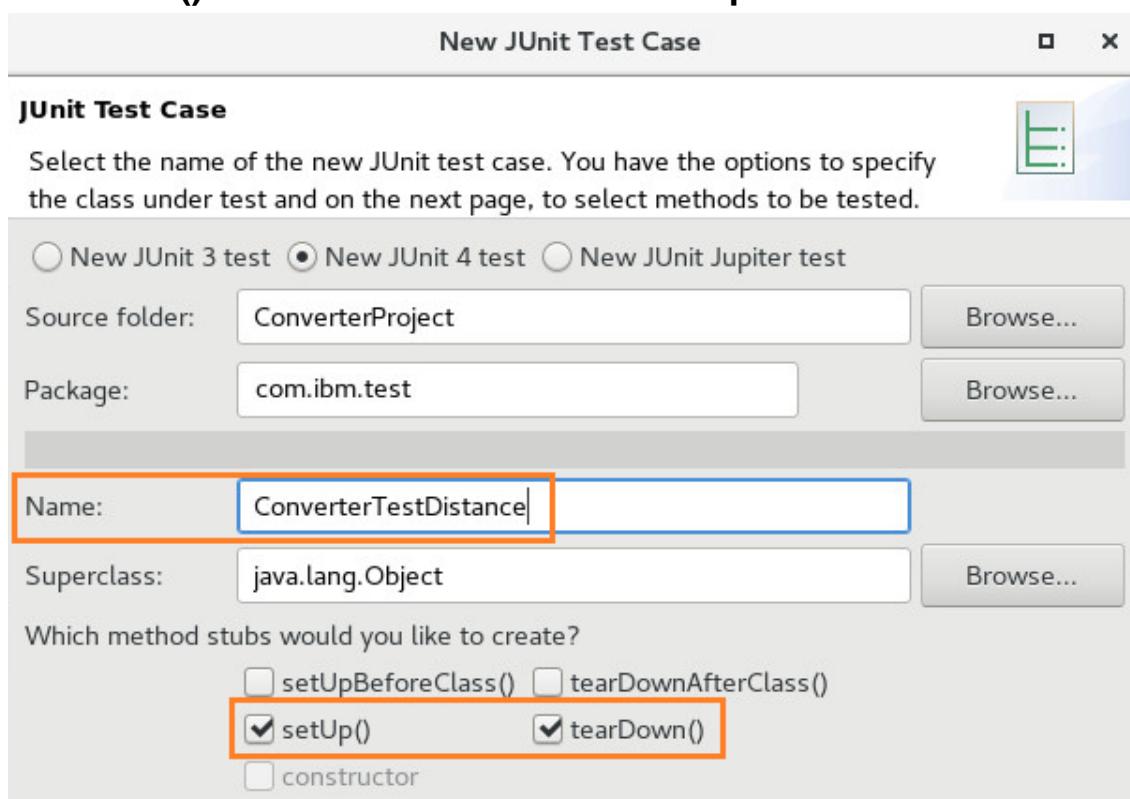
В данном разделе создается тест кейс, который будет анализировать результат работы программы с некорректно введенными данными. Например, с отрицательным расстоянием: если при переводе из миль в километр, указать отрицательное значение, то программа должна указать на соответствующую ошибку.

1. Создайте JUnit тест кейс.

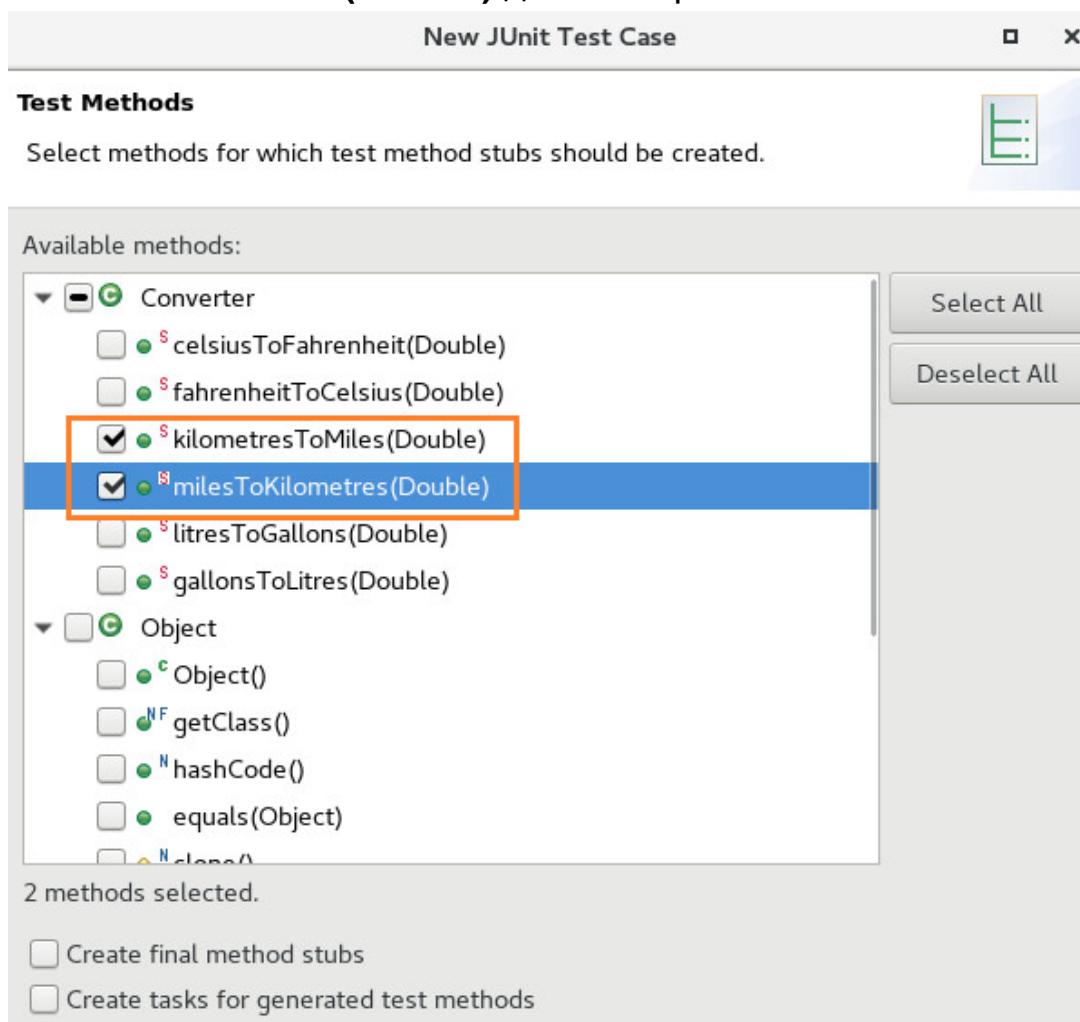
a. В главном меню выберите **File -> New -> JUnit Test Case**

b. Введите следующую информацию:

<b>New JUnit 4 test:</b>	выбрано
<b>Source folder:</b>	<b>ConverterProject</b>
<b>Package:</b>	<b>com.ibm.test</b>
<b>Name:</b>	<b>ConverterTestDistance</b>
<b>setUp():</b>	выбрано
<b>tearDown():</b>	выбрано



- c. Нажмите **Browse** рядом с полем **Class under test**.
- d. Выберите **Converter – com.ibm – ConverterProject**. Нажмите **OK**.
- e. Нажмите **Next**.
- f. Выберите методы **kilometresToMiles(Double)** и **milesToKilometres(Double)** для тестирования.



- g. Нажмите **Finish**.
2. Реализуйте логику тест кейса.
  - a. Добавьте в класс с тестами импорт класса, который будет тестируаться:

```
import com.ibm.Converter;
```
  - b. Объявите набор переменных класса, которые будут использоваться в тестах:

```
private Double hundredKM; // 100 kilometres
```

```
private Double negDist; // Negative distance
```

- с. Реализуйте метод **setUp()** для указания верных значений этим переменным:

```
public void setUp() throws Exception {
    hundredKM = new Double ( 100 );
    negDist   = new Double ( -1 );
}
```

- д. 100 км – это приблизительно 62 мили. Используйте этот факт для написания теста метода **kilometresToMiles(Double)**. Так как эти значения равны лишь приблизительно, в методе **assertEquals** становится актуальным использование третьего параметра, где указывается погрешность оценки. Для данного случая можно поставить его значение – 1.

```
public void testKilometresToMiles() throws Exception {
    assertEquals (
        62.0,
        Converter.kilometresToMiles( hundredKM ),
        1
    );
}
```

- е. Для тестирования ситуации с отрицательным расстоянием используйте следующий код:

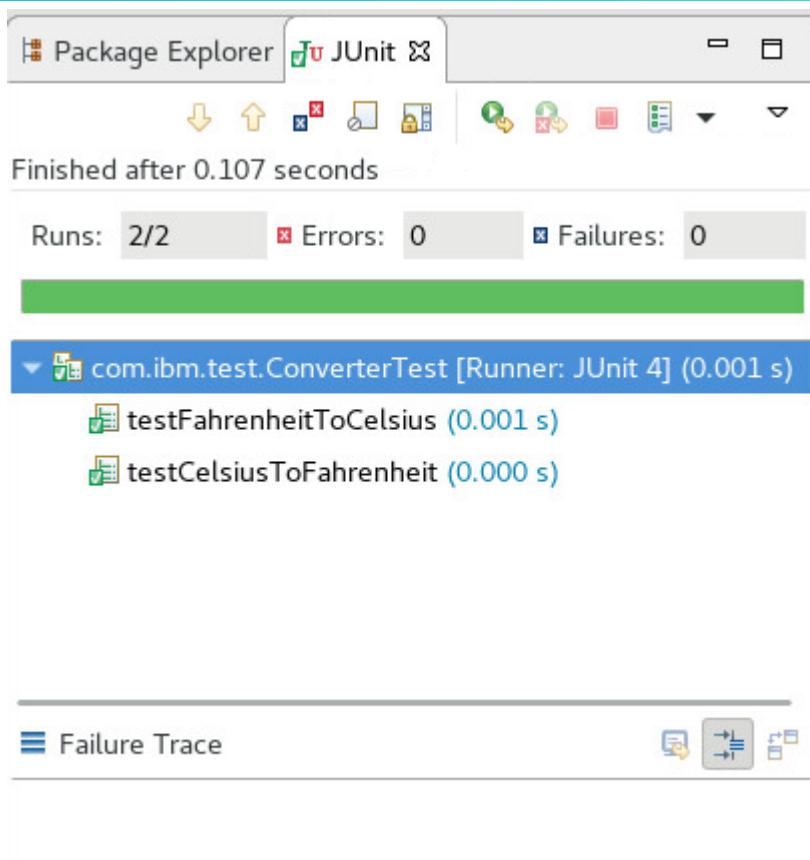
```
public void testMilesToKilometres() {
    try {
        // we expect this statement to throw an
        exception
        Converter.milesToKilometres( negDist );
        // this section should not be reached
        fail("Exception not thrown for negative
values");
    }
    catch ( Exception e ) {}
}
```

- f. В данном тесте мы рассчитываем на то, что код нашего класса сгенерирует исключительную ситуацию при попытке перевести отрицательное расстояние. Если исключение возникает, оно обрабатывается и это считается штатной ситуацией – тест завершается успешно. Если исключения не возникает, то явным образом вызывается метод fail, который переключает статус данного теста в соответствующее значение.
- g. Сохраните сделанные изменения.

## Раздел 4. Запуск тестов

В данном разделе запускаются все созданные кейсы, и проверяется их результат. Результат может быть трех типов: **passed** (успешное завершение), **error** (возникла необработанная исключительная ситуация), **failure** (фактический результат не совпал с ожидаемым).

1. Запустите JUnit Test Case **ConverterTest**.
  - a. В представлении **Project Explorer** выберите **ConverterTest.java**.
  - b. Нажмите на стрелку вниз, рядом с кнопкой **Run**.
  - c. Выберите **Run As -> JUnit Test**.
  - d. Открывается представление **JUnit**.
2. Убедитесь, что все тесты завершились успешно. В поле **Runs** указывается, как много тестовых методов было вызвано.



3. Смоделируйте ошибку в тестах.

а. Измените код метода `testFahrenheitToCelsius()` на следующий:

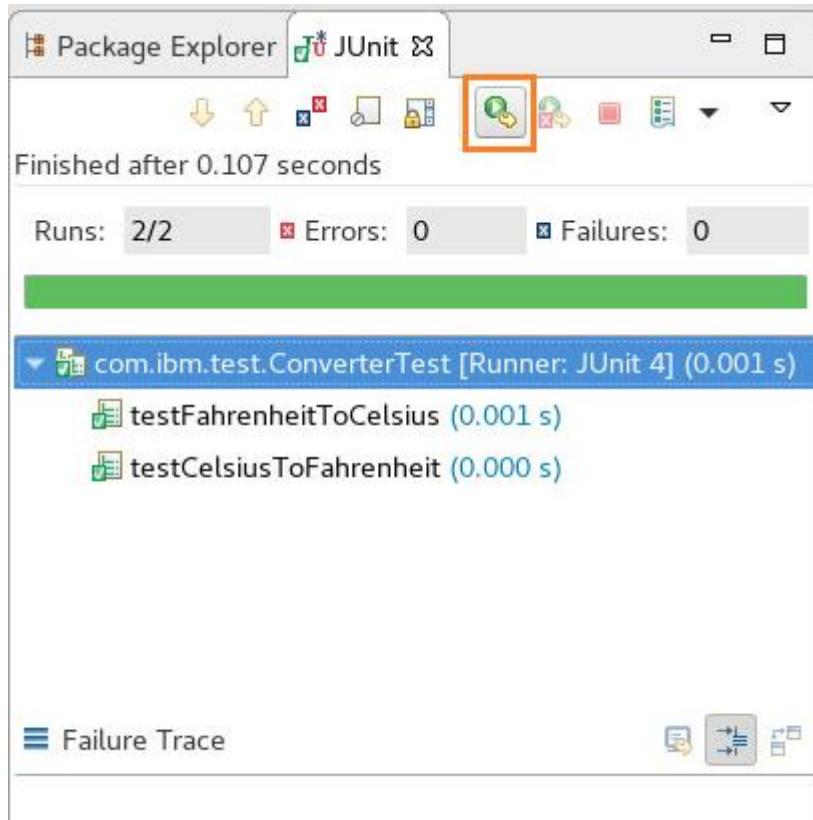
```
public void testFahrenheitToCelsius() {  
    assertEquals (  
        0,  
        Converter.fahrenheitToCelsius( neg40 ),  
        0  
    );  
}
```

б. Сохраните сделанные изменения.

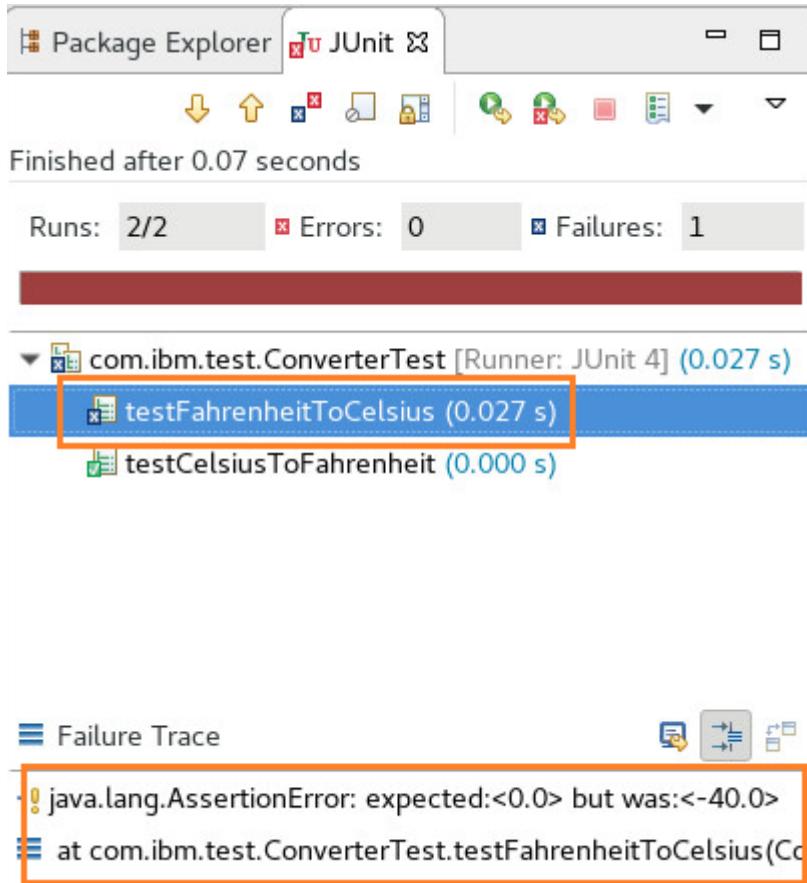
с. Ожидаемый результат заведомо не верен, так что можно ожидать завершения теста со статусом **Failure**.

4. Повторно выполните тесты.

а. Нажмите кнопку **Rerun Last Test** в представлении JUnit.



b. Обратите внимание на красную индикацию (полосу).



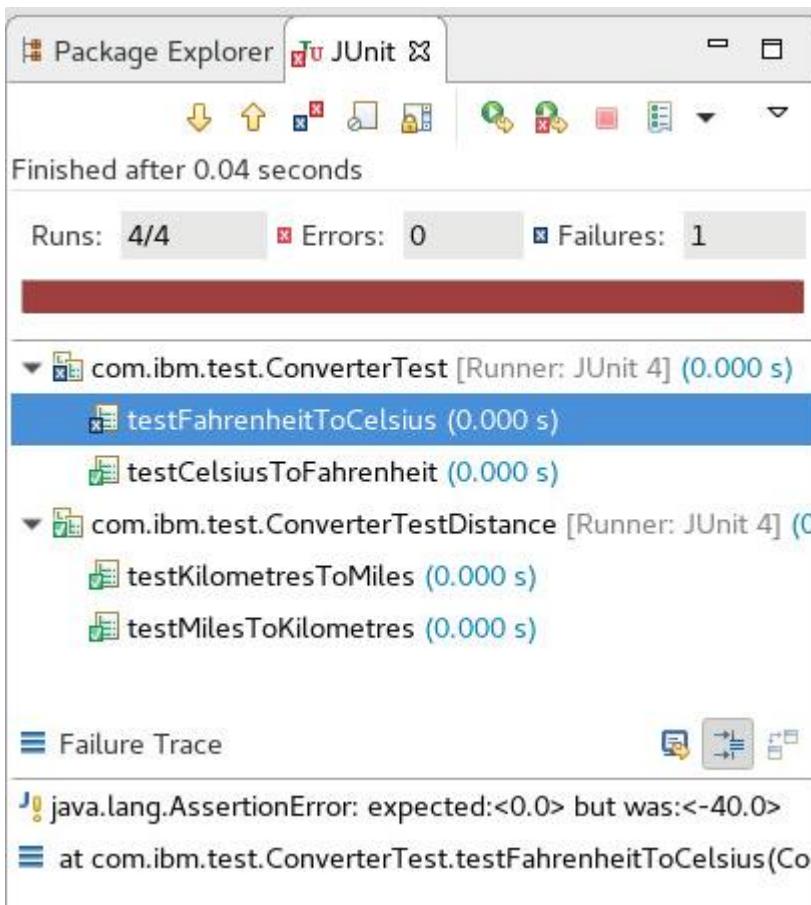
c. Рядом с тестом **testFahrenheitToCelsius** появился характерный знак X, который говорит о неожиданном результате. В панели **Failure Trace** можно посмотреть детали этого теста и результата.

d. Закройте представление **JUnit**.

5. Запустите все JUnit тесты в пакете **com.ibm.test**.

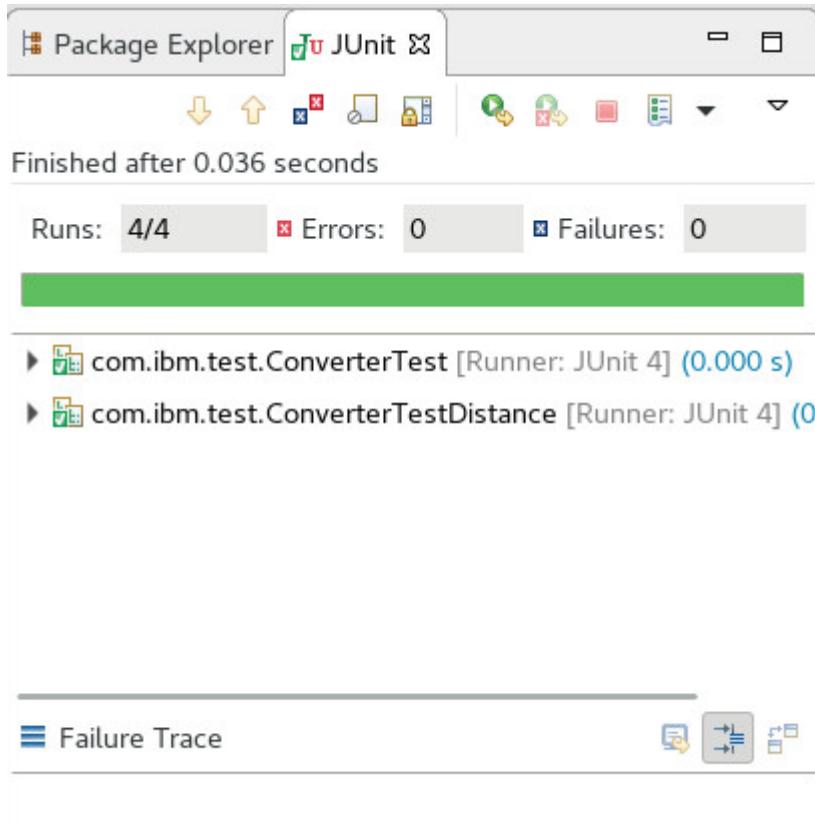
a. В представлении **Project Explorer** нажмите правой кнопкой мыши по пакету **com.ibm.test** и выберите **Run As -> JUnit Test**.

b. В представлении **JUnit** открываются результаты 4 тестов из 2 классов. Один из тестов завершился неудачно из-за смоделированной ранее ошибки.



- c. Исправьте допущенную ранее ошибку и **сохраните** изменения.
- d. Нажмите кнопку **Rerun Last Test** в представлении **JUnit**.

е. Убедитесь, что все тесты прошли успешно.



## Конец упражнения

## Упражнение 13. Лямбда-выражения

---

### О чём это упражнение:

В этом упражнении вы научитесь использовать лямбда-выражения.

Вы создаете новый класс **LTester**, который будет запускаться самостоятельно. В нем описываются несколько треков, для которых реализуется лямбда-аппарат. Он состоит из интерфейса **TrackOperation**, метода **operate** для вызова одной из двух операций: **sumLengths** (для суммирования продолжительностей двух треков) и **compareLength** (сравнение продолжительности двух треков).

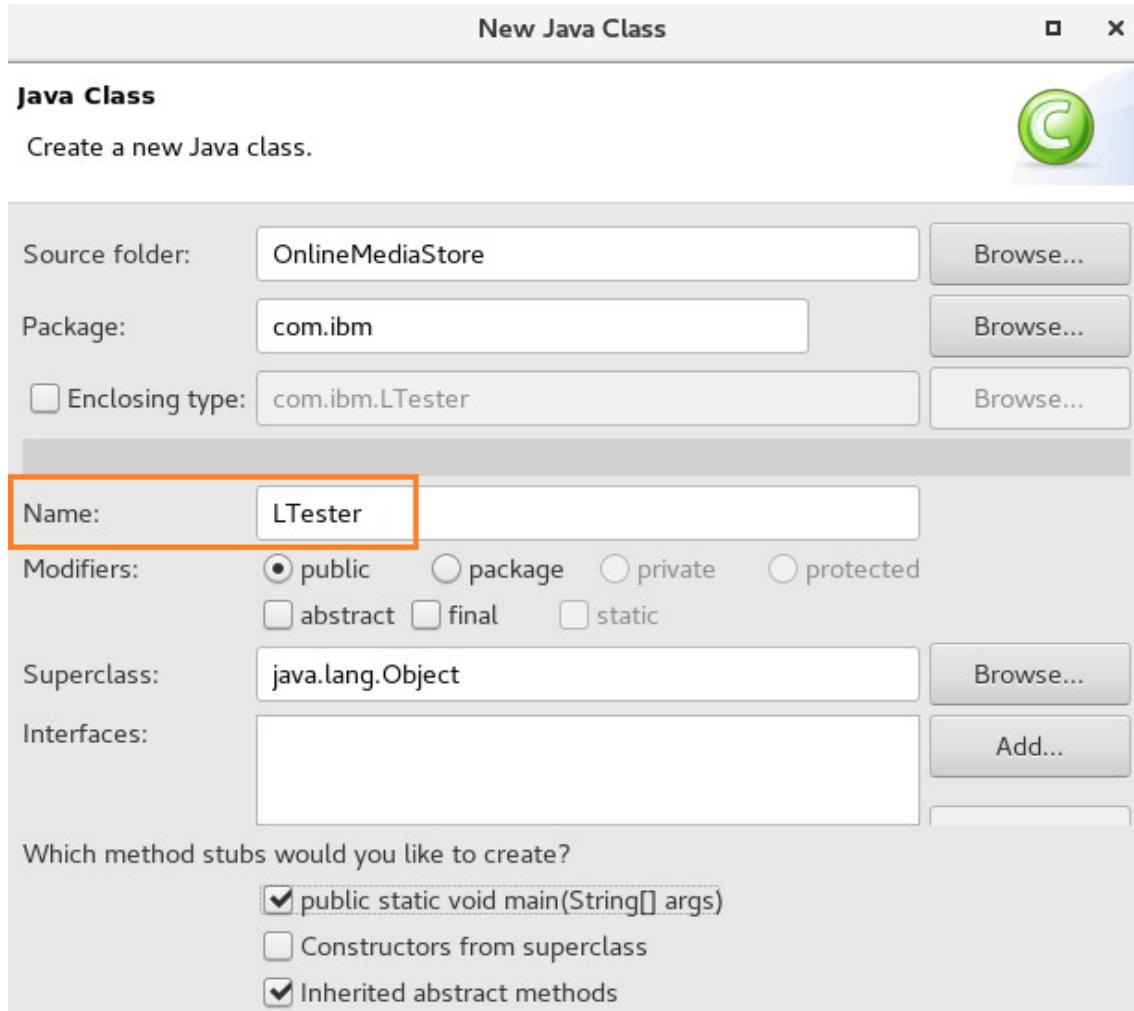
### Что вы должны будете сделать:

- Реализовать лямбда-выражения для работы с треками

## Раздел 1. Создание класса с лямбда-выражениями

1. Откройте новое рабочее пространство
  - a. Если Eclipse уже запущен, выберите в меню **File -> Switch Workspace -> Other**.
  - b. Если Eclipse не запущен, запустите его с помощью ссылки на рабочем столе.
  - c. Выберите каталог **/root/labfiles/workspaces/L** и нажмите **Launch**.
  - d. Закройте страницу **Welcome**.
2. Импортируйте проекты в рабочее пространство из каталога **/root/labfiles/Serialization/import/OnlineMediaStore**. Если вы забыли последовательность действий, то можно воспользоваться инструкцией из начала прошлого упражнения.
3. Перейдите в перспективу **Java**.
4. Создайте класс **LTester**.
  - a. Из главного меню перейдите: **File -> New -> Class**.  
Также можно нажать правой кнопкой мыши по проекту **OnlineMediaStore**, в контекстном меню выбрать **New -> Class**. В любом случае откроется мастер создания нового класса.
  - b. В поле **Source Folder** по умолчанию выбрано **OnlineMediaStore**.
  - c. В названии пакета введите **com.ibm**.
  - d. Укажите **LTester** как имя класса.
  - e. Установите опцию **public static void main(String[] args)**. Это добавит в ваш класс метод **main** для того, чтобы запускать этот класс.

f. Остальные значения не меняйте:



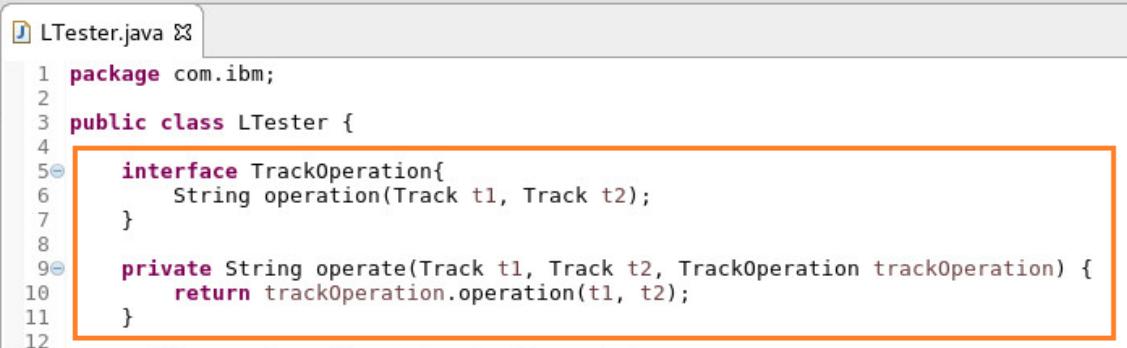
g. Нажмите **Finish**

5. Добавьте в класс описания интерфейса **TrackOperation** и метода **operate** для вызова операции этого интерфейса.

a. Добавьте в класс **LTester** следующий код:

```
interface TrackOperation{
    String operation(Track t1, Track t2);
}

private String operate(Track t1, Track t2,
TrackOperation trackOperation) {
    return trackOperation.operation(t1, t2);
}
```



```
LTester.java
1 package com.ibm;
2
3 public class LTester {
4
5     interface TrackOperation{
6         String operation(Track t1, Track t2);
7     }
8
9     private String operate(Track t1, Track t2, TrackOperation trackOperation) {
10        return trackOperation.operation(t1, t2);
11    }
12
13    public static void main(String[] args) {
14        // TODO Auto-generated method stub
15    }
}
```

6. Внутрь метода **main** добавьте описание двух операций **sumLengths** и **compareLength**

а. Используйте следующий код:

```
TrackOperation sumLengths = (t1, t2) -> {
    int length = t1.getLength() + t2.getLength();
    return "The total length of 2 tracks (" +
t1.getTitle() +
        " and " + t2.getTitle() + " is: " +
length;
};
```

```
TrackOperation compareLength = (t1, t2) -> {
    if (t1.getLength() > t2.getLength()) {
        return "First track is longer";
    }
    else if (t1.getLength() == t2.getLength()) {
        return "Tracks have equal length!";
    }
    else {
        return "Second track is longer";
    }
};
```

```
5  interface TrackOperation{
6      String operation(Track t1, Track t2);
7  }
8
9  private String operate(Track t1, Track t2, TrackOperation trackOperation) {
10     return trackOperation.operation(t1, t2);
11 }
12
13 public static void main(String[] args) {
14     // TODO Auto-generated method stub
15
16     TrackOperation sumLengths = (t1, t2) -> {
17         int length = t1.getLength() + t2.getLength();
18         return "The total length of 2 tracks (" + t1.getTitle() +
19             " and " + t2.getTitle() + " is: " + length;
20     };
21
22     TrackOperation compareLength = (t1, t2) -> {
23         if (t1.getLength() > t2.getLength()) {
24             return "First track is longer";
25         }
26         else if (t1.getLength() == t2.getLength()) {
27             return "Tracks have equal length!";
28         }
29         else {
30             return "Second track is longer";
31         }
32     };
33 }
```

7. Ниже реализуйте код для создания нескольких объектов типа **Track**.

И вызовите для них созданные операции с выводом в консоль.

а. Используйте следующий код:

```
LTester tester = new LTester();

//define tracks
Track track1 = new Track();
track1.setLength(60);
track1.setTitle("Short Track 1 (length 60)");
System.out.println("Track1: " + track1.getTitle());

Track track2 = new Track();
track2.setLength(60);
track2.setTitle("Short Track 2 (length 60)");
System.out.println("Track2: " + track2.getTitle());

Track track3 = new Track();
track3.setLength(300);
track3.setTitle("Long Track (length 300)");
System.out.println("Track3: " + track3.getTitle());

System.out.println("Compare operations:");
```

```
System.out.println("Compare track1 and track2: " +
tester.operate(track1, track2, compareLength));
System.out.println("Compare track2 and track3: " +
tester.operate(track2, track3, compareLength));

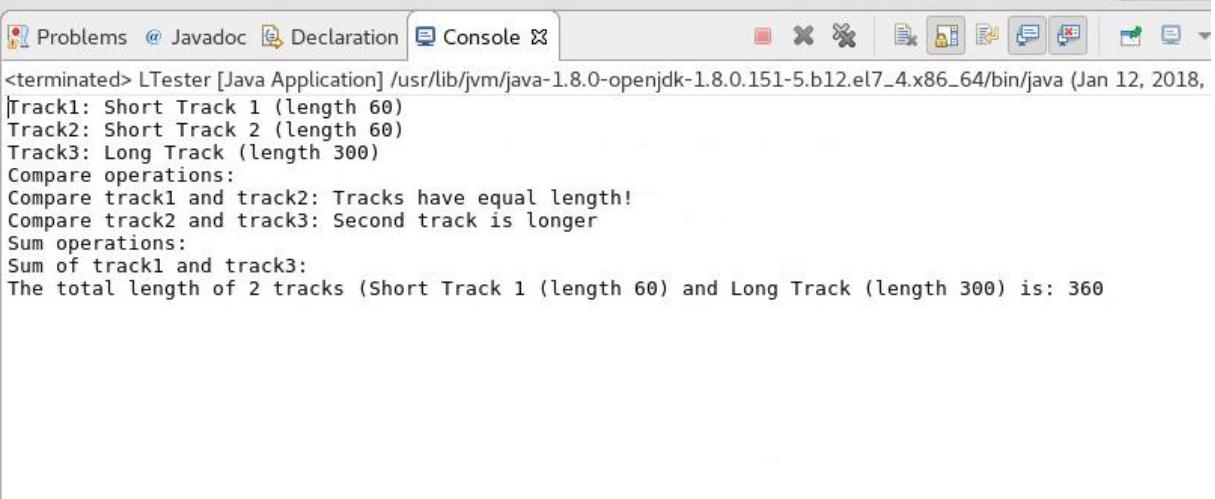
System.out.println("Sum operations:");
System.out.println("Sum of track1 and track3: ");
System.out.println(tester.operate(track1, track3,
sumLengths));
```

The screenshot shows a Java code editor window with the title bar 'LTester.java'. The code is a Java program that creates three Track objects (track1, track2, track3) and prints their titles and lengths. It then compares the lengths of track1 and track2, and track2 and track3. Finally, it calculates and prints the sum of the lengths of track1 and track3. The code uses System.out.println for output and tester.operate for length comparison.

```
31             return "Second track is longer";
32     }
33 }
34
35 LTester tester = new LTester();
36
37 Track track1 = new Track();
38 track1.setLength(60);
39 track1.setTitle("Short Track 1 (length 60)");
40 System.out.println("Track1: " + track1.getTitle());
41
42 Track track2 = new Track();
43 track2.setLength(60);
44 track2.setTitle("Short Track 2 (length 60)");
45 System.out.println("Track2: " + track2.getTitle());
46
47 Track track3 = new Track();
48 track3.setLength(300);
49 track3.setTitle("Long Track 3 (length 300)");
50 System.out.println("Track3: " + track3.getTitle());
51
52 System.out.println("Compare operations:");
53 System.out.println("Compare track1 and track2: " +
54     tester.operate(track1, track2, compareLength));
55 System.out.println("Compare track2 and track3: " +
56     tester.operate(track2, track3, compareLength));
57
58 System.out.println("Sum operations:");
59 System.out.println("Sum of track1 and track3:");
60 System.out.println(tester.operate(track1, track3, sumLengths));
61
62
```

## 8. Сохраните сделанные изменения.

b. Запустите класс **LTester** с помощью кнопки **Run**.



The screenshot shows a software interface with a toolbar at the top and a central text area. The text area displays the output of a Java application named LTester. The output is as follows:

```
<terminated> LTester [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-5.b12.el7_4.x86_64/bin/java (Jan 12, 2018,
Track1: Short Track 1 (length 60)
Track2: Short Track 2 (length 60)
Track3: Long Track (length 300)
Compare operations:
Compare track1 and track2: Tracks have equal length!
Compare track2 and track3: Second track is longer
Sum operations:
Sum of track1 and track3:
The total length of 2 tracks (Short Track 1 (length 60) and Long Track (length 300) is: 360
```

**Конец упражнения**

