Модуль #4
Понятие Java приложения

Модуль 4

- **Описание класса**
- Понятие Java-приложения
- Работа с экземпляром класса
- Понятие classpath
- Пакеты
- Java-архивы
- Отладка приложений

Описание класса

- Идентификатор – слово используемое для наименования переменной, метода, класса или метки.
  - Ключевые слова и зарезервированные слова не могут быть использованы в качестве идентификаторов.
  - Идентификатор должен начинаться с буквы, символа доллара ($) или символа подчеркивания (_).
  - Последующими символами могут быть: буквы, $, _, цифры.

```
   foobar                    // legal
   BIGinterface              // legal:
embedded keywords are ok
   $incomeAfterTaxes         // legal
   3_node5                   // illegal:
starts with a dig
```

Ключевые слова Java

abstract	class	extends	implements	null	strictfp
assert	const	false	import	package	super
boolean	continue	final	instanceof	private	switch
break	default	finally	int	protected	synchr
byte	do	float	interface	public	this
case	double	for	long	return	throw
catch	else	goto	native	short	throws
char	enum	if	new	static	transie

Описание класса

- Класс – набор данных и методов работы с этими данными.
- Класс необходимо представлять как “штамп”, производящий экземпляры этого класса.
- Экземпляр – создается в runtime (в процессе выполнения программы).
- Представляет из себя пямять, выделенную под данные и инструкции JVM, как вызывать методы класса.

Описание класса

- Декларация класса:

```
<modifier> class <class_name> {
        <data>
        <methods>
}
```

- `<modifier>` определяет видимость класса, т.е. какие классы могут получить доступ к данному классу.
- Модификатор **`public`** говорит, что к такому классу могут обращаться откуда угодно.
- Может быть определен только один **`public`** класс в файле.

Описание класса

- Тело класса – это кодовый блок, содержащий данные. Его тело должно заключаться в { }.

- Данные – декларация переменных и методы.

- Стоит различать переменные класса и переменные экземпляра (об этом в следующих модулях).

Описание класса

- Метод – набор инструкций работы с данными класса, а так же с параметрами метода.

```
<modifier> <return_type> <name>(<params>)
{
    ...
    <return_value>
}
```

- Метод также представляет кодовый блок и заключается в **{ }**.

```
public class Person {
    String name;
    int age;

    int getAge() {
        return age;
    }
}
```

Модуль 4

- Описание класса
- **Понятие Java-приложения**
- Работа с экземпляром класса
- Понятие classpath
- Пакеты
- Java-архивы
- Отладка приложений

Описание класса

- Работа любой java программы (java приложения) начинается с выполнения метода `main`.
- Простейшее приложение состоит из одного метода `main`.
- Для запуска программы в JVM, необходимо задать в командной строке java:

```
<полное_имя_класса_содержащего_main>
```

Внимание! Если указанный класс не содержит `main`, или не верна его сигнатуры, Возбуждается `java.lang.NoSuchMethodError`

Сигнатура метода

- Сигнатура метода `main`.

```
public static void main(String[] args)
```

  - **`void`** – возвращаемый тип.
  - **`args`** – список аргументов коммандной строки.
  - Метод `main` должен быть объявлен как статический.
  - Статический метод может и должен быть вызван так:

```
TheClass.staticMethod(...)
```

Внимание! Время жизни программы равно времени выполнения метода main.

Описание класса

- Метод создает экземпляры классов, вызывает методы этих объктов, которые могут вызывать другие методы, запрашивать ввод с консоли и т.д.

- Данная цепочка вызовов методов называется поток выполнения(execution thread).

- JVM ведет выполнение метода строго последовательно, от выражения к выражению.

Модуль 4

- Описание класса
- Понятие Java-приложения
- **Работа с экземпляром класса**
- Понятие classpath
- Пакеты
- Java-архивы
- Отладка приложений

Описание класса

- Класс – это файл, содержащий байт-код, определяющий данные и методы.
- Экземпляр – память, выделенная под структуры данных, декларированные в классе.
- Для того, чтобы JVM могла работать в классом (создавать экземпляры), она должна загрузить код класса.

Описание класса

- Для того, чтобы создать экземпляр класса, используется оператор **`new`**.
- При этом в динамической области памяти, называемой кучей (heap), выделяется память под данные экземпляра (объекта).
- Размер кучи можно задать при запуске JVM.
- Оператор **`new`** возвращает ссылку на созданный экземпляр класса.

Загрузка класса

- JVM имеет специальный загрузчик (bootstrap classloader), который умеет загружать байт-код класса из файловой системы, учитывая полное имя класса.

- Байт-код класса можно загрузить также из другого источника. В этом случае необходимо создать специфический загрузчик.

- JVM сама управляет загрузкой классов.

Работа с объектами

- Работа с объектами в Java всегда ведется через ссылку.
- Тип ссылки определяется типом созданного объекта.
- Тип ссылки может совпадать с классом создаваемого объекта, либо отличаться, что является полиморфизмом.

Описание класса

```
public class Person {
    String name;
    int age;

    int getAge() {
        return age;
    }

    public static void main(String[] args) {
        Person personInstance = new Person();
        personInstance.getAge();
    }
}
```

Конструирование и инициализация

- Вызов `Xyz x = new Xyz()` выполняет:
  - Связывание ссылки с ее типом.
  - Выделение памяти под объект.
  - Выполняется явная инициализация атрибутов.
  - Вызывается конструктор.
  - Объектная ссылка возвращается оператором **`new`**.
  - Объектная ссылка присваивается переменной `x`.

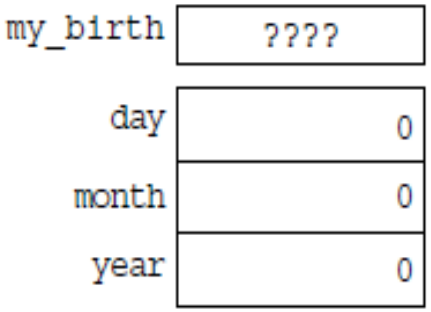
Внимание! Декларация только выделит память для хранения ссылки.

Конструирование и инициализация

- Выделение памяти под ссылку


```
MyDate my_birth;
```

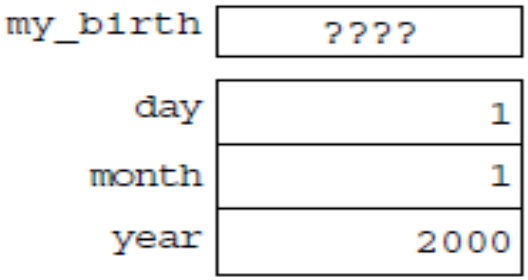
- Выделение памяти под объект.



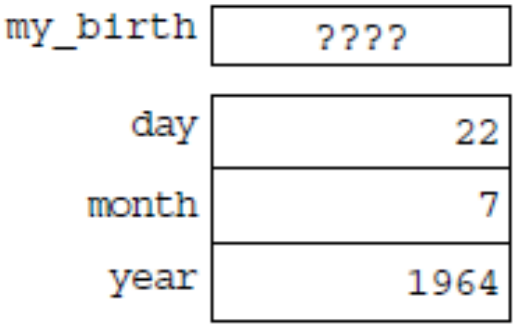

```
new MyDate(31,3,1988)
```

Конструирование и инициализация

- Инициализация атрибутов

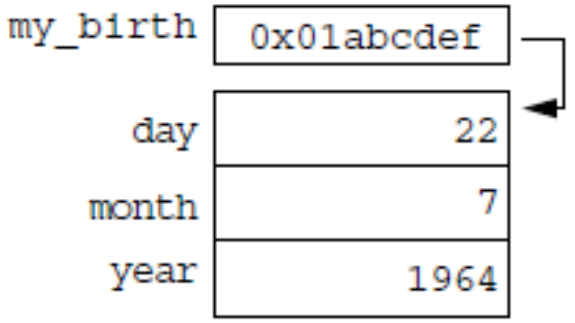


- Вызов конструктора.



Конструирование и инициализация

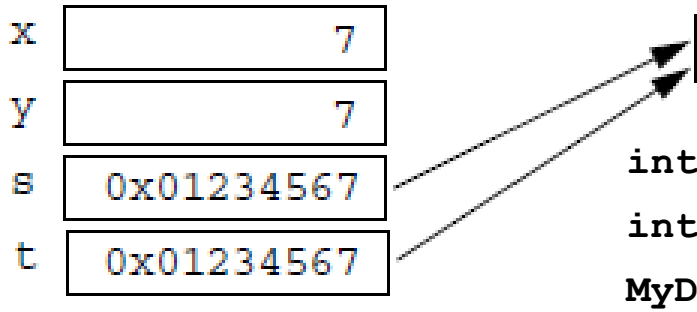
- Присвоение объектной ссылки




```
MyDate my_birth =
new MyDate(22, 7, 1964)
```

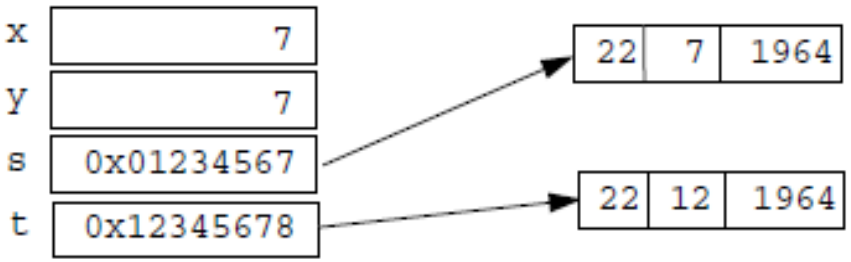
Конструирование и инициализация

- Две переменные ссылаются на один объект




```
int x = 7;
int y = x;
MyDate s = new MyDate(22, 7, 1964);
MyDate t = s;
```

- При переприсваивании t она ссылается на новый объект:




```
t = new MyDate(22, 7, 1964);
```

Конструирование и инициализация

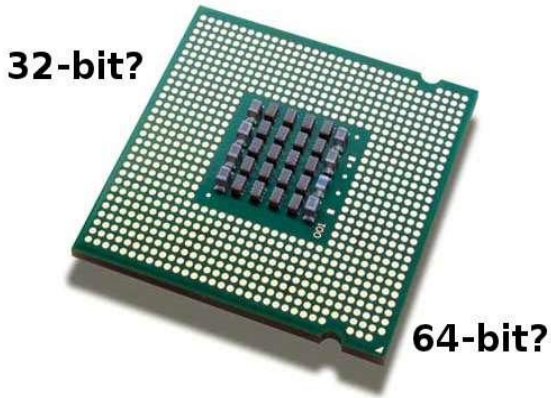
- Java поддерживает следующие виды переменных, отличающихся в области памяти и временем жизни:
  - Свойство объекта (member)

    Создается, когда создается объект. Размещается в куче.
  - Стековая переменная (local variable)

    Определяется в методе.
  - Переменная класса (class member)

    Создается, когда класс загружается загрузчиком классов. Может существовать только одна копия.

Размер объектов (основы)



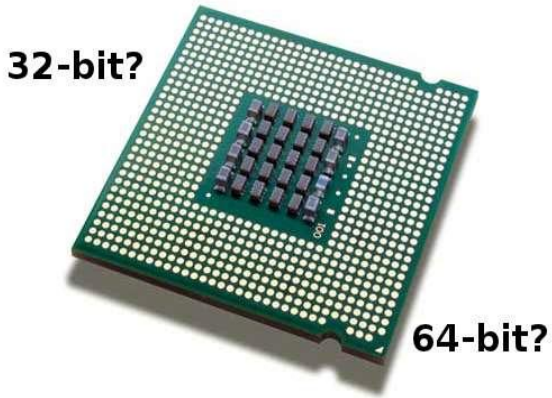
Бит. 0 или 1

Байт. 8 бит

Мегабайт (MB). ~1 миллион или 2^20 байт

Гигабайт (GB). ~1 миллиард или 2^30 байт

Размер объектов (основы)



Бит. 0 или 1

Байт. 8 бит

Мегабайт (MB). ~1 миллион или 2^20 байт

Гигабайт (GB). ~1 миллиард или 2^30 байт

Старые компьютеры: Имеется ввиду 32-разрадный компьютер с указателем в 4 байта.

Новые компьютеры: Имеется ввиду 64-разрадный компьютер с указателем в 8 байт.

- Можно использовать больше памяти
- Указатели используют больше места

Некоторые JVM “сжимают” указатели до 4х байт.

Размер объекта

type	bytes
byte	1
short	2
int	4
long	8
float	4
double	8
char	2

Примитивные типы

type	bytes
char[]	2N + 24
int[]	4N + 24
double[]	8N + 24

Одномерные массивы

type	bytes
char[][]	~2MN
int[][]	~4MN
double[][]	~8MN

Двумерные массивы

Внимание! Примитивные типы более подробно рассмотрены в модуле 5.

Размер класса

Заголовок класса.	16 байт.
Ссылка.	8 байт.
Выравнивание снизу.	Каждый объект выравнивается до кратности 8 байт.

Размер класса

Заголовок класса. 16 байт

Ссылка. 8 байт

Выравнивание снизу. Каждый объект выравнивается до кратности 8 байт

Пример 1. Объект Data использует *32 байт* памяти

```
class Date
{
    int day;
    int month;
    int year;
    ...
}
```

Размер класса

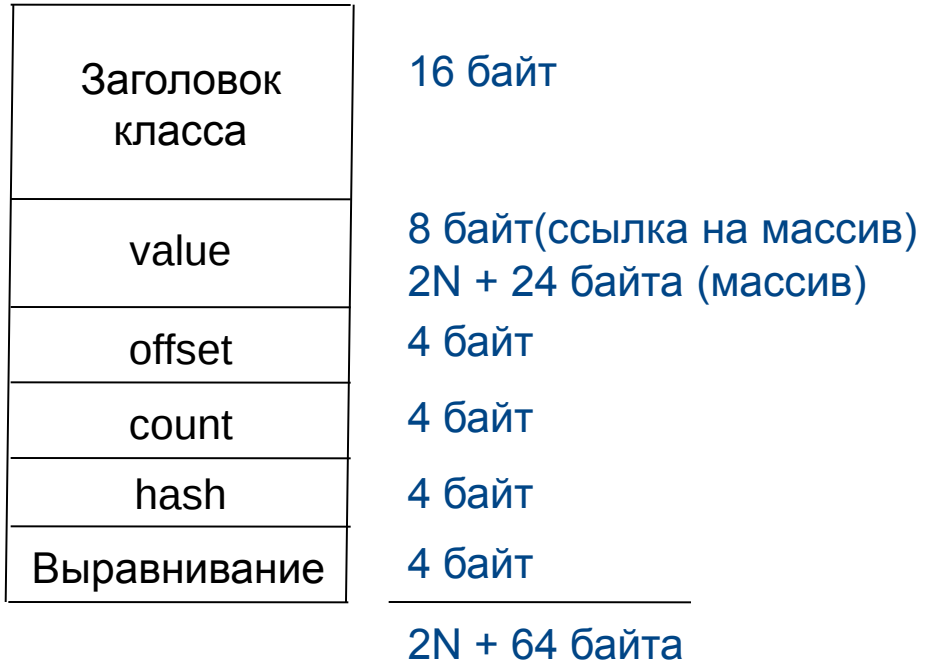
Заголовок класса. 16 байт.

Ссылка. 8 байт.

Выравнивание снизу. Каждый объект выравнивается до кратности 8 байт.

Пример 2. Пустая строка длинны *N* использует *~2N байт* памяти.

```
class String
{
    char[] value;
    int offset;
    int count;
    int hash;
    ...
}
```



Размер класса

- Сколько памяти использует объект QuickUnion как функция от N?

```
class QuickUnion
{
    int[] id;
    int[] size;
    int count;
    …
}
```

Размер класса (итог)

Использование памяти для различных типов.

- Примитивные типы: 4 байта для int, 8 байт для double....
- Объектная ссылка: 8 байт
- Массивы: 24 байта + память для каждого элемента.
- Объект: 16 байт + память для каждого члена класса
- Вложенный класс: 8 байт
- Выравнивание: кратно 8 байт.

Работа с экземпляром класса

- К свойству объекта можно обратиться с помощью оператора « . »
- Аналогично можно вызвать метод объекта. Если метод не принимает аргументы, нужно указать `()`, чтоб отличить метод от свойства.
- Аргументы заключаются в скобки `()`.

Работа с экземпляром класса

```
public class Person {
    String name;
    int age;

    int getAge() {
        return age;
    }

    void setAge(int ageToSet) {
        age = ageToSet;
    }

    public static void main(String[] args) {
        Person personInstance = new Person();
        personInstance.getAge();
        String theName = personInstance.name;
        personInstance.setAge(29);
    }
}
```

Упражнение 5

Разработка приложения из 2-х классов.

Модуль 4

- Описание класса
- Понятие Java-приложения
- Работа с экземпляром класса
- **Понятие classpath**
- Пакеты
- Java-архивы
- Отладка приложений

Понятие classpath

- Программа Java – это цепочка вызовов методов объектов некоторых классов, т.о. В процессе компиляции и работы программы требуется информация, где расположен байт-код необходимых классов.

- При запуске программы `javac` можно указать список путей файловой системы, где необходимо искать зависимые классы.

- Это спецификация путей поиска и называется **`classpath`**.

Понятие classpath

- В Windows список путей разделяется “;”, в Unix “:”

- Если компилируемый java файл **Person.java** использует некоторые скомпилированные классы, находящиеся в **c:\lib\classes**, то необходимо указать компилятору данный путь через флаг **-classpath**

```
javac -d bin
      -sourcepath src
      -classpath C:\lib\classes
      Person.java
```

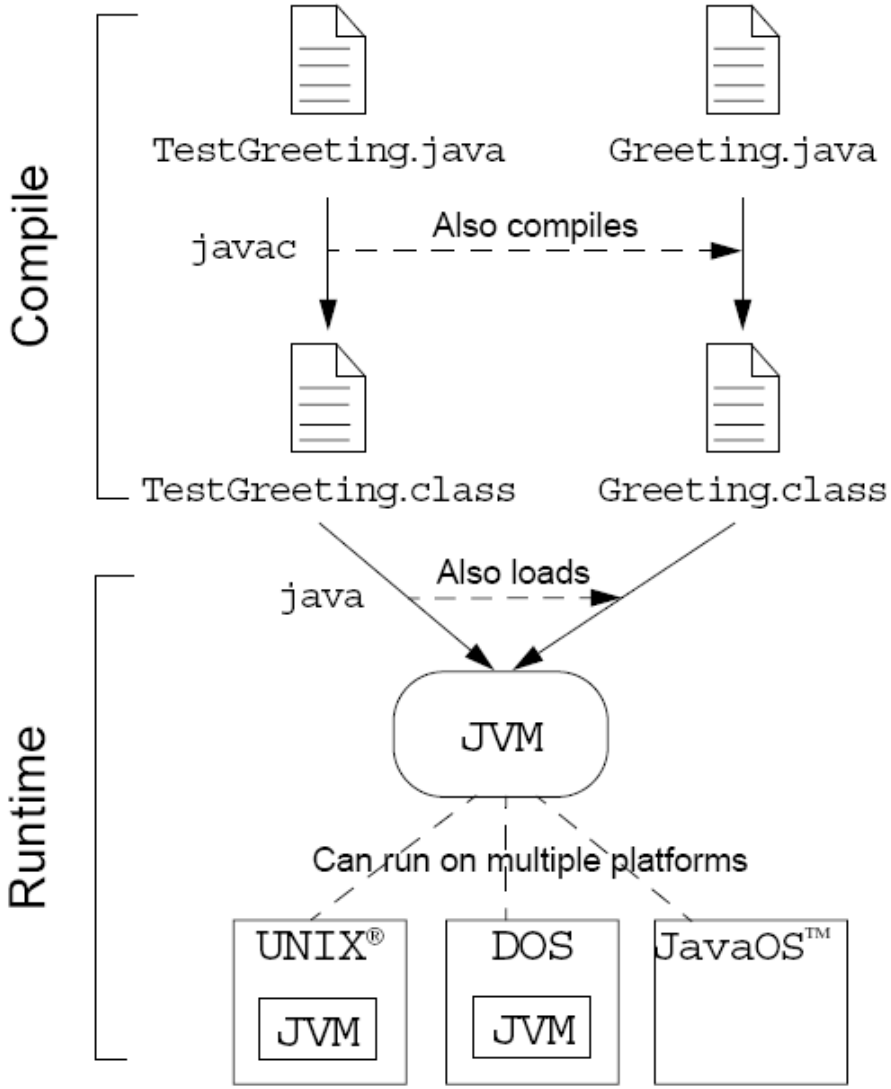
Понятие classpath

- Аналогично можно задать classpath при запуске java приложения. Необходимые классы JVM будет искать в указаных путях.

```
java -classpath C:\lib\classes
     Test
```

Внимание! Для избегания компиляции каждого класса в отдельности, можно указать класс содержащий метод **`main`**.

Компиляция и выполнение



Модуль 4

- Описание класса
- Понятие Java-приложения
- Работа с экземпляром класса
- Понятие classpath
- **Пакеты**
- Java-архивы
- Отладка приложений

Пакеты

- Пакет – способ организации классов. Подобно тому, как два различных файла *readme.txt* могут располагаться в разных директориях, java-классы тоже могут размещаться в разных директориях.

- Пакеты позволяют избежать коллизий классов с одинаковым именем.

Пакеты

- Для того, чтобы разместить файл HelloWorld.java в пакет world, необходимо указать имя пакета в файле с помощью ключевого слова **package**.

- Декларация пакета должна идти в самом начале файла.

```
// only comment can be here
package world;

public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Пакеты

- Для того чтобы использовать класс, необходимо указать его полное имя:

```
package another;

public class Test {
  public static void main(String[] args) {
     world.HelloWorld instance = new world.HelloWorld();
  }
}
```

Пакеты

- Существует способ упростить способ спецификации полного имени класса, импортировав пространство имен.

```
package another;
import world.HelloWorld;

public class Test {
  public static void main(String[] args) {
     HelloWorld instance = new HelloWorld();
  }
}
```

Импорт

- Импорт класса не имеет ничего общего с загрузкой класса и не влияет на скорость компиляции и выполнения

- Можно внести все классы данного пакета в пространство имен.

```
import  package.subpackage.*;
```

Модуль 4

- Описание класса
- Понятие Java-приложения
- Работа с экземпляром класса
- Понятие classpath
- Пакеты
- **Java-архивы**
- Отладка приложений

Jar-архивы

- Существует способ организации классов в zip-архивы для более удобного распространения группы `.class` файлов.
- Архив имеет расширения `.jar`.
- Если классы в jar-файле, то необходимо указывать имя этого файла в classpath.

```
java -classpath c:\libs\myjar.jar
     thepackage.MainClass
```

Jar-архивы

- Для того чтобы создать Jar-файл.

```
jar.exe cf myjar.jar MainClass.class
```

c – создать Jar-файл

f – задает имя файла

Jar-архивы

- В Jar файле можно также вручную определить специальный файл манифеста, описывающий данный JAR файл.
- Манифест располагается в каталоге META-INF архива.
- Он создается по умолчанию инструментом jar.

Jar-архивы

- Например, манифест может указывать имя класса, содержащего `main`():

```
Manifest-Version: 1.0

Class-Path: ojdbc14.jar

Created-By: Eclipse

Main-Class: MyPackage.MyClass
```

- В таком случае JVM можно запустить так:

```
java -jar MyJar.jar
```

Упражнение

Работа с пакетами и jar-файлами

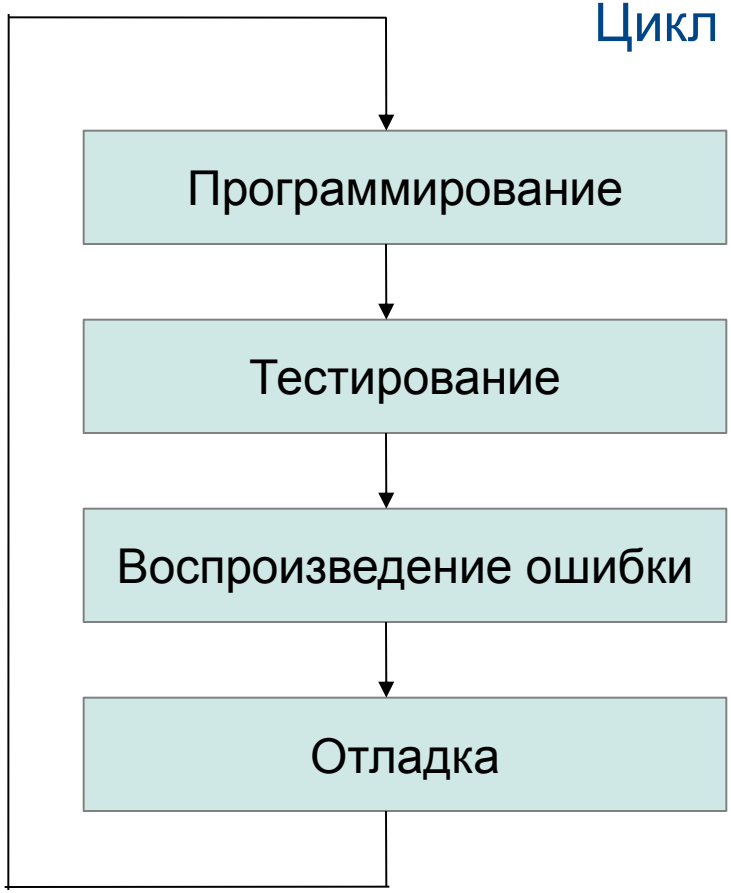
Модуль 4

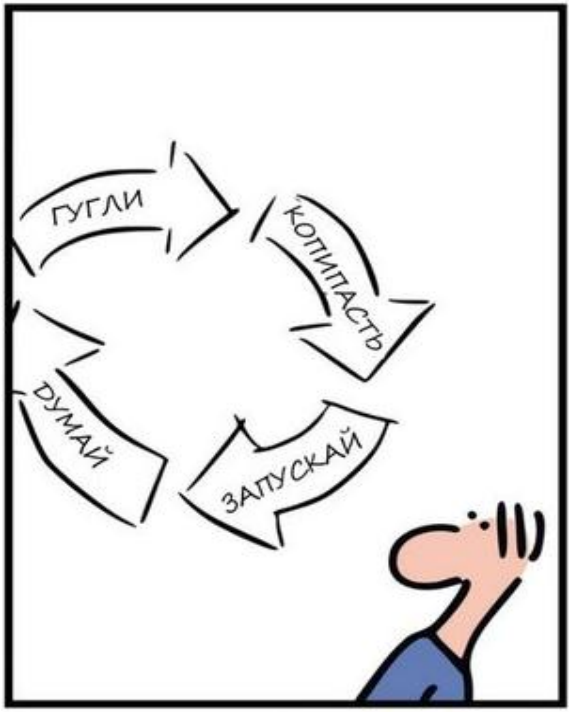
- Описание класса
- Понятие Java-приложения
- Работа с экземпляром класса
- Понятие classpath
- Пакеты
- Java-архивы
- **Отладка приложений**

Отладка Java приложений

- Отладка – этап разработки, на котором обнаруживают, локализуют и устраняют ошибки.

Цикл разработки кода





Отладка Java приложений

Классификация отладок:

- Отладчики и вывод текущего состояния программы.

  System.out.println();

  Logger.log();

  jdb, yourkit, jprofiler, jconsole

- Локальная и удаленная

Упражнение

Научиться пользоваться дебагером