Лекция 2

Основы JAVA. Переменные, методы, классы.

Создание объектов

- Чтобы создать какой-нибудь объект, нужно написать имя типа (класс) этого объекта и ключевое слово new перед ним. Допустим, у нас есть класс «кот» – Cat, тогда:

	Код	Описание
1	Cat cat;	Объявляет ссылочную переменную с именем cat и типом Cat. Значение переменной cat – ссылка null.
2	**new** Cat();	Создаёт объект типа Cat.
3	Cat cat = **new** Cat();	Создаёт ссылочную переменную cat типа Cat. Создаёт новый объект типа Cat. Присваивает в переменную cat ссылку на новосозданный объект.
4	Cat murzik = **new** Cat(); Cat vaska = **new** Cat();	Создаются два объекта, ссылки на которые присваиваются двум разным переменным соответственно.
5	Cat murzik = **new** Cat(); Cat vaska = **new** Cat(); vaska = murzik;	Создаются два объекта, ссылки на которые присваиваются двум разным переменным. Затем, переменной vaska присваивается ссылка на объект, содержащийся в переменной murzik. Теперь обе переменных ссылаются на первый созданный объект. (На второй больше никто не ссылается – второй объект считается мусором)

6	Cat murzik = **new** Cat();
	Cat vaska = null;
	vaska = murzik;
	murzik = null;
		Создаётся один объект типа Cat, ссылка на который заносится в первую переменную (murzik), вторая переменная (vaska) содержит пустую (нулевую) ссылку.
		Обе переменных ссылаются на один объект.
		Теперь только vaska ссылается на объект, но не murzik.

- Если просто создать объект и не присвоить ему никакой переменной, то Java-машина создаст его и тут же объявит его мусором (неиспользуемым объектом). А через некоторое время удалит его в процессе «сборки мусора».

- Как только не останется ни одной переменной, хранящей ссылки на данный объект, он будет помечен как мусор и убран: уничтожен Java-машиной при следующей «сборке мусора».

- Пока есть хоть одна ссылка на объект, он считается живым и уничтожаться не будет. Если хотите побыстрее уничтожить объект – обнулите все ссылки на него: присвойте всем переменным, ссылающимся на него, значение **null**.

Видимость переменных

- Под «видимостью переменных» подразумевают места в коде, где к этой переменной можно обратиться. К некоторым переменным можно обращаться отовсюду в программе, к другим только в пределах их класса, к третьим же только внутри одного метода.

```
public class Variables
{
  private static String TEXT = "The end.";

  public static void main (String[] args)
  {
    System.out.println("Hi");
    String s = "Hi!";

      System.out.println(s);
      if (args != NULL)
      {
        String s2 = s;

        System.out.println(s2);

      }
      Variables variables = new Variables();
      System.out.println(variables.classVariables);
      System.out.println(TEXT);

  }

  public String classVariables;

  public Variables()
  {
    classVariables = "Class Variables test.";
  }
}
```

1. Переменная, объявленная в методе, существует/видна с начала объявления до конца метода.
2. Переменная, объявленная в блоке кода, существует до конца этого блока кода.
3. Переменные — аргументы метода — существуют везде внутри метода.
4. Переменные класса/объекта существуют все время жизни содержащего их объекта. Их видимость дополнительно регулируется специальными модификаторами доступа: public, private.
5. Статические переменные классов существуют все время работы программы. Их видимость также определяется модификаторами доступа.

Модификаторы доступа

- Модификатор «**public**».
- К переменной, методу или классу, помеченному модификатором public, можно обращаться из любого места программы. Это самая высокая степень открытости — никаких ограничений нет.
- Модификатор «**private**».
- К переменной или методу, помеченному модификатором private, можно обращаться только из того же класса, где он объявлен. Для всех остальных классов помеченный метод или переменная — невидимы и «как бы не существуют». Это самая высокая степень закрытости – только свой класс.
- Без модификатора.
- Если переменная или метод не помечены никаким модификатором, то считается, что они помечены «модификатором по умолчанию». Переменные или методы с таким модификатором (т.е. вообще без какого-нибудь) видны всем классам пакета, в котором они объявлены. И только им. Этот модификатор еще иногда называют «**package**», намекая, что доступ к переменным и методам открыт для всего пакета, в котором находится их класс.

	Модификаторы	Доступ из...		
		Своего класса	Своего пакета	Любого класса
1	**private**	Есть	Нет	Нет
2	нет модификатора (**package**)	Есть	Есть	Нет
3	**public**	Есть	Есть	Есть

Создание переменных

	Пример	Пояснение
1	String s1 = **new** String(); String s2 = "";	Создание двух идентичных пустых строк.
2	**int** a;	Создание переменной типа int;
3	**int** a = 5;	Создание переменной a типа int, установка её значения равным 5
4	**int** a = 5, b = 6;	Создание переменной a типа int, установка ей значения 5 Создание переменной b типа int, установка ей значения 6
5	**int** a = 5, b = a + 1;	Создание переменной a типа int, установка ей значения 5 Создание переменной b типа int, установка ей значения 6
6	Date date = **new** Date();	Создание объекта типа «Дата». В каждый объект типа «дата» после создания заносится текущее время и дата.
7	**boolean** isTrue = true;	Переменная логического типа инициализируется значением true (истина)
8	**boolean** isLess = (5 > 6);	В переменную isLess заносится значение false (ложь). Других значений тип boolean не принимает.

Переменные-ссылки

- Переменные-ссылки — это переменные всех типов, кроме примитивных. Такие переменные содержат в себе только адрес объекта (ссылку на объект).

- Переменные примитивных типов хранят в себе значения, а переменные типов-классов хранят ссылку на объекты этого же класса, ну или хранят null.

- Объект и ссылка на него связаны, примерно, как человек и его телефонный номер. Телефонный номер не является человеком, но номер можно использовать, чтобы звонить человеку, спрашивать у него какую-то информацию, руководить им или давать команды. Ссылка тоже используется для взаимодействия с объектом. Все объекты взаимодействуют друг с другом при помощи ссылок.

- При присваивании примитивного объекта, его значение копируется (дублируется). При присваивании же ссылочной переменной, копируется только адрес объекта (телефонный номер), сам же объект при этом не копируется.

- Ссылка даёт ещё одно преимущество: можно передать ссылку на объект в какой-нибудь метод, и этот метод будет в состоянии модифицировать (изменять) наш объект используя ссылку на него, вызывая его методы и обращаясь к данным внутри объекта.

- Переменным a и b присваиваются только значения 5(m) и 6(n) соответственно, a и b ничего не знают про (никак не влияют на) m и n.

Пример 1 Тут значение m и n не меняется.	И вот почему. Данный код аналогичен коду слева

```
public class References
{
  public static void main (String[] args)
  {
    int m = 5;
    int n = 6;

    System.out.println("M=" +m+ " N=" + n);
    swap(m, n);
    System.out.println("M=" +m+ " N=" + n);
  }

  private static void swap(int a, int b)
  {
    int c = a;
    a = b;
    b = c;
  }
}
```

```
public class References
{
  public static void main (String[] args)
  {
    int m = 5;
    int n = 6;

    System.out.println("M=" +m+ " N=" + n);
    int a = m, b = n;

    int c = a;
    a = b;
    b = c;

    System.out.println("M=" +m+ " N=" + n);
  }
}
```

- Переменным a и b присваиваются ссылки на olga и vera соответственно, a и b меняют значения внутри объектов olga и vera.

Пример 2

```
public class Primitives
{
 public static void main(String[] args)
 {
  Student olga = new Student();
  olga.name = "Olga";
  olga.age = 21;

  Student vera = new Student();
  vera.name = "Veronika";
  vera.age = 15;

  System.out.println("Olga is " + olga.age);
  System.out.println("Vera is " + vera.age);

  ageSwap(olga, vera);

  System.out.println("Olga is " + olga.age);
  System.out.println("Vera is " + vera.age);
 }

 private static void ageSwap(Student a, Student b)
 {
  int c = a.age;
  a.age = b.age;
  b.age = c;
 }

 static class Student
 {
  String name;
  int age;
 }
}
```

И вот почему.

```
public class Primitives
{
 public static void main(String[] args)
 {
  Student olga = new Student();
  olga.name = "Olga";
  olga.age = 21;

  Student vera = new Student();
  vera.name = "Veronika";
  vera.age = 15;

  System.out.println("Olga is " + olga.age);
  System.out.println("Vera is " + vera.age);

  Student a = olga, b = vera;

  int c = a.age;
  a.age = b.age;
  b.age = c;

  System.out.println("Olga is " + olga.age);
  System.out.println("Vera is " + vera.age);
 }


 static class Student
 {
  String name;
  int age;
 }
}
```

- Команды группируют в функции, чтобы потом можно было исполнять их единым блоком — как одну сложную команду. Для этого надо написать имя функции(метода) и в скобках после него перечислить значения-параметры.

- В примере ниже написана функция, которая выводит на экран переданную строку 4 раза. Затем вызвана функция print4 в строке номер 6.

- Когда программа дойдет до выполнения строчки 6, она перескачет на строчку 9 — переменной s будет присвоено значение “SIRIUS IS”

- Затем будут выполнены строки 11-14, и, наконец, функция завершится и программа продолжит работу со строчки номер 7.

```
1
2   public class MethodCall
3   {
4       public static void main(String[] args)
5       {
6           print4("SIRIUS IS");
7       }
8
9       public static void print4(String s)
10      {
11          System.out.println(s);
12          System.out.println(s);
13          System.out.println(s);
14          System.out.println(s);
15      }
16  }
```

- Функция вычисляет какое-то значение и отдает его тем, кто ее вызвал с помощью команды return.

```
public class MethodCall
{
  public static void main(String[] args)
  {
    int a = 5, b = 7;
    int m = min(a, b);
    System.out.println("Minimum is "+ m);
  }

  public static int min(int c, int d)
  {
    int m2;
    if (c < d)
        m2 = c;
    else
        m2 = d;

    return m2;
  }
}
```

- Некоторые функции просто что-то делают, но никаких значений не вычисляют и не возвращают, как метод **main**(), например. Для них придуман специальный тип результата – **void** – пустой тип. Функции делятся на две категории – возвращающие значение и не возвращающие значение.

- Когда Java-машина выполняет команду return, она вычисляет значение выражения, стоящего справа от слова return, сохраняет это значение в специальной части памяти и **тут же завершает работу функции.** А сохранённое значение использует как результат вызова функции в том месте, где её вызвали.

Полное имя класса

- Полным именем класса считается имя, состоящее из всех пакетов, перечисленных через точку и имени класса.

	Имя класса	Имя пакета	Полное имя
1	String	java.lang	java.lang.**String**
2	FileInputStream	java.io	java.io.**FileInputStream**
3	ArrayList	java.util	java.util.**ArrayList**
4	IOException	java.io	java.io.**IOException**;

- Чтобы использовать класс в своём коде, вам нужно указывать его полное имя. Хотя можно использовать и краткое имя — только лишь имя класса, но для этого нужно «проимпортировать данный класс» — указать его имя перед объявлением вашего класса, со словом **import**. Классы из пакета java.lang импортируются по умолчанию. Их указывать не обязательно.

Полное имя класса:

```
public class FileCopy2
{
   public static void main(String[] args) throws
java.io.IOException
   {
      java.io.FileInputStream fileInputStream =
                  new java.io.FileInputStream("c:\\data.txt");
      java.io.FileOutputStream fileOutputStream =
                  new java.io.FileOutputStream("c:\\result.txt");

      while (fileInputStream.available() > 0)
      {
         int data = fileInputStream.read();
         fileOutputStream.write(data);
      }

      fileInputStream.close();
      fileOutputStream.close();
   }
}
```

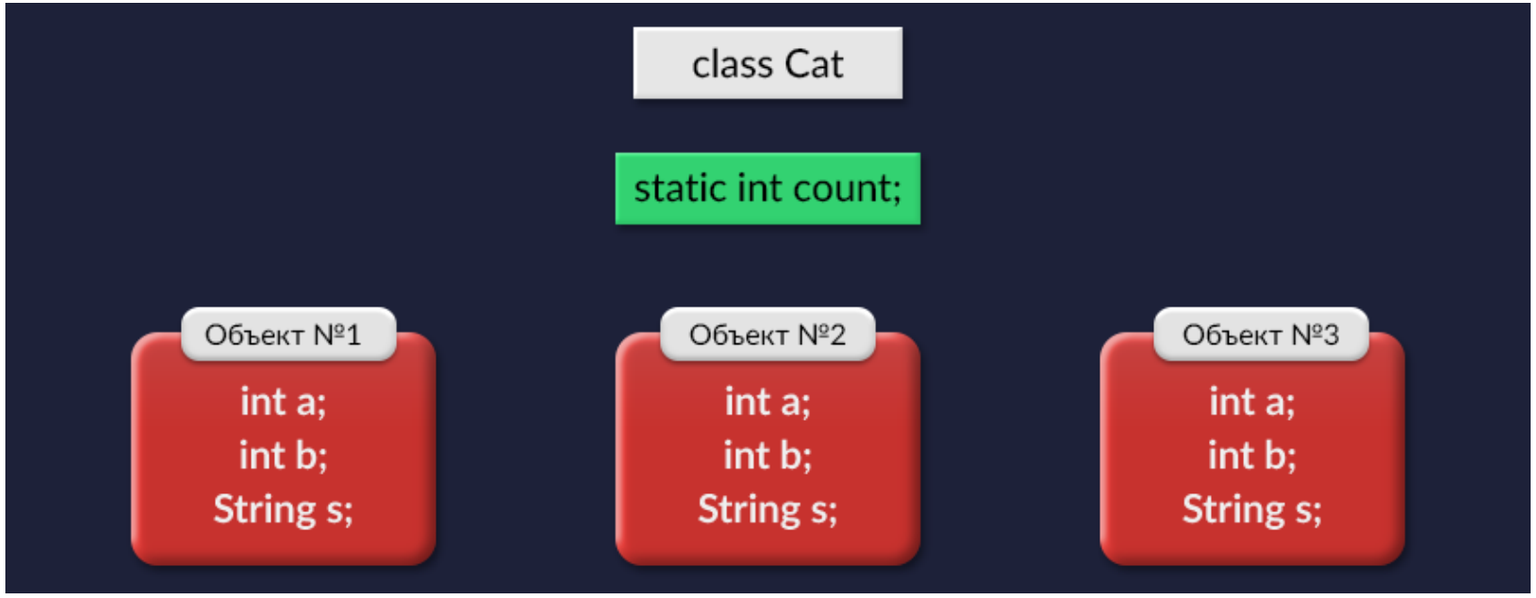
Короткое имя класса:

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileCopy
{
   public static void main(String[] args) throws
IOException
   {
      FileInputStream fileInputStream =
                  new FileInputStream("c:\\data.txt");
      FileOutputStream fileOutputStream =
                  new FileOutputStream("c:\\result.txt");

      while (fileInputStream.available() > 0)
      {
         int data = fileInputStream.read();
         fileOutputStream.write(data);
      }

      fileInputStream.close();
      fileOutputStream.close();
   }
}
```

Повторение

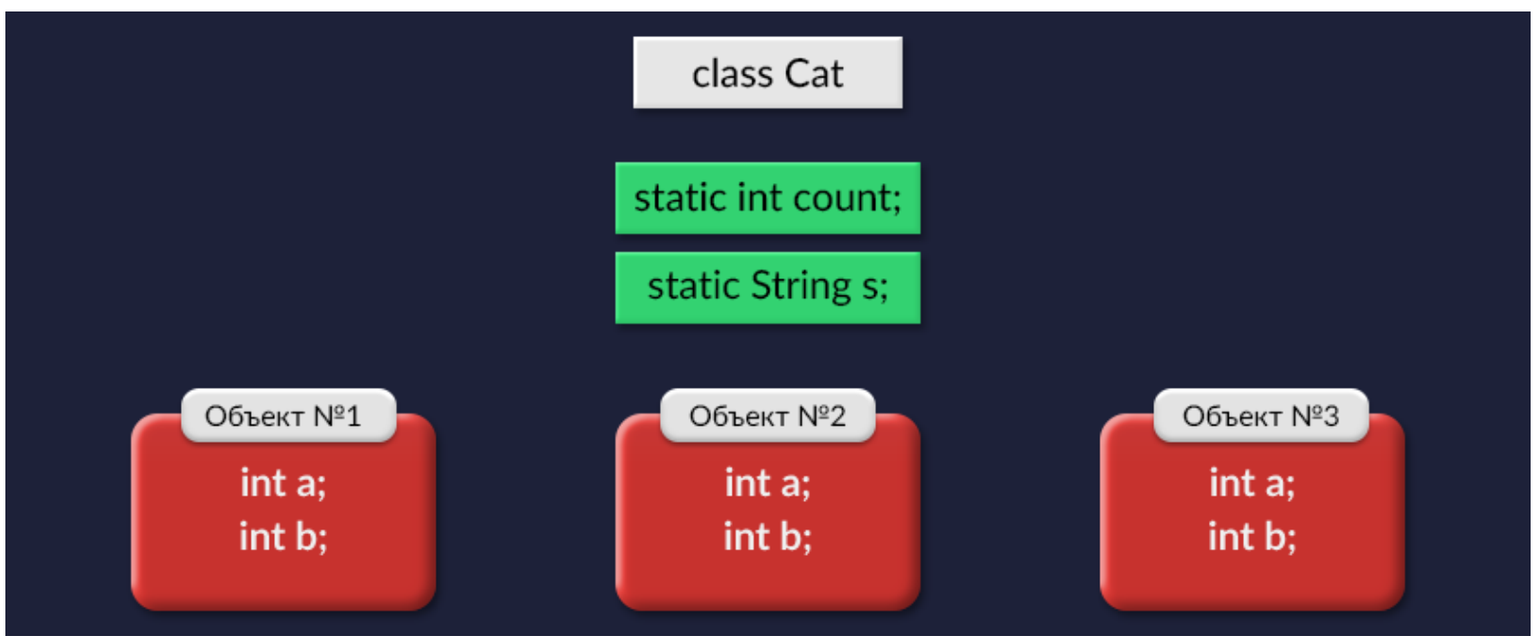
```
public class Variables
{
    private static String TEXT = "The end.";

    public static void main (String[] args)
    {
        System.out.println("Hi");
        String s = "Hi!";
        {
            System.out.println(s);
            if (args != NULL)
            {
                String s2 = s;

                System.out.println(s2);
            }
            Variables variables = new Variables();
            System.out.println(variables.classVariables);
            System.out.println(TEXT);
        }
    }

    public String classVariables;

    public Variables()
    {
      classVariables = "Class Variables test.";
    }
}
```

1. Переменная, объявленная в методе, существует/видна с начала объявления до конца метода.
2. Переменная, объявленная в блоке кода, существует до конца этого блока кода.
3. Переменные — аргументы метода — существуют везде внутри метода.
4. Переменные класса/объекта существуют все время жизни содержащего их объекта. Их видимость дополнительно регулируется специальными модификаторами доступа: public, private. **Если нет ни одного объекта, то нет и ни одной копии переменной.** К переменной можно обращаться (переменная видна) из всех методов класса, независимо от того, объявлены они до неё или после неё. Для каждого объекта создаётся своя, независимая от других объектов, переменная. Доступ к переменной невозможен из статических методов.
5. Статические переменные классов существуют все время работы программы. Их видимость также определяется модификаторами доступа. Если переменная объявлена статической – помечена ключевым словом **static**, то она существует все время, когда существует её класс. Обычно JVM загружает класс в память при первом его использовании, тогда же и инициализируются статические переменные.



- В примере выше у нас объявлен класс Cat, у которого есть 4 переменные: a,b,s – обычные, а count – статическая. Если создать несколько объектов такого класса (например, 3 шт.), то каждый из них будет содержать свою собственную копию обычных переменных класса. Статическая же переменная – общая на всех. Формально она даже не находится внутри этих объектов, т.к. существует даже тогда, когда ни одного объекта класса Cat создано не было.

- Вот что произойдет, если мы объявим переменную s статической:



Область действия переменных

- Все переменные, объявленные внутри метода, должны иметь уникальные имена. Аргументы метода также считаются его переменными.

- Переменные класса тоже должны быть уникальными в рамках каждого конкретного класса.

- Есть исключение – имена переменных метода и переменных класса могут совпадать.

- Если в методе видны (доступны) несколько переменных, например, переменная класса и переменная метода, то тогда обращение произойдет к переменной метода.

Статические и нестатические методы

Вот так работает обычный **нестатический** метод

Как выглядит код	Что происходит на самом деле

Cat cat = **new** Cat();
String name = cat.getName();
cat.setAge(17);
cat.setChildren(cat1, cat2, cat3);
	Cat cat = **new** Cat();
	String name = Cat.getName(cat);
	Cat.setAge(cat,17);
	Cat.setChildren(cat, cat1, cat2, cat3);

При вызове метода в виде **«объект»** точка «имя метода», на самом деле вызывается метод класса, в который первым аргументом передаётся **тот самый объект**. Внутри метода он получает имя **this**. Именно с ним и его данными происходят все действия.

Вот так работает **статический** метод

Как выглядит код	Что происходит на самом деле
Cat cat1 = **new** Cat(); Cat cat2 = **new** Cat(); **int** catCount = Cat.**getAllCatsCount**();	Cat cat1 = **new** Cat(); Cat cat2 = **new** Cat(); **int** catCount = Cat.**getAllCatsCount**(**null**);

При вызове статического метода, никакого объекта внутрь не передаётся. Т.е. this равен null, поэтому статический метод не имеет доступа к нестатическим переменным и методам (ему нечего передать в такие методы в качестве this).

Задачи

1. Создать объект типа Cat **2 раза**. Сохраните каждый экземпляр в свою переменную. Имена переменных должны быть разные.

```
public class Solution {
   public static void main(String[] args) {
      //напишите тут ваш код
   }

   public static class Cat {

   }
}
```

2. Закомментируйте максимальное количество строк, чтобы на экран вывелось число **19.**

```
public class Solution {
   public static void main(String[] args) {
      int x = 1;
      int y = 0;
      y = y + 3 * x;
      x = x * 2;
      x = x * 16;
      y = y + 2 * x;
      y = y + x;
      System.out.println(y);
   }
}
```

3. Реализуйте метод print3. Метод должен вывести на экран переданную строку **3 раза**. Каждый раз с новой строки.

```
public class Solution {
   public static void print3(String s) {
      //напишите тут ваш код

   }

   public static void main(String[] args) {
      print3(«JAVAjavaJAVA!");
   }
}
```

4. * Помогите коту обрести имя с помощью метода setName.

```
public class Cat {
   private String name = "безымянный кот";

   public void setName(String name) {
      //напишите тут ваш код

   }

   public static void main(String[] args) {
      Cat cat = new Cat();
      cat.setName("Жужик");
      System.out.println(cat.name);
   }
}
```

Ввод с клавиатуры

- Для вывода данных на экран мы использовали **System.out**. Для ввода данных будем использовать **System.in**. Но у **System.in** есть минус — он позволяет считать с клавиатуры только коды символов. Чтобы обойти эту проблему и считывать большие порции данных за один раз, мы будем использовать более сложную конструкцию:

Ввод строки и числа с клавиатуры

```
InputStream inputStream = System.in;
Reader inputStreamReader = new InputStreamReader(inputStream);
BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

String name = bufferedReader.readLine(); //читаем строку с клавиатуры
String sAge = bufferedReader.readLine(); //читаем строку с клавиатуры
int nAge = Integer.parseInt(sAge); //преобразовываем строку в число.
```

Более компактная запись первой части:

```
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String name = reader.readLine();
String sAge = reader.readLine();
int nAge = Integer.parseInt(sAge);
```

Еще более компактная

```
Scanner scanner = new Scanner(System.in);
String name = scanner.nextLine();
int age = scanner.nextInt();
```

- Чтобы считать строку с клавиатуры, удобнее всего воспользоваться объектом **BufferedReader**. Но для этого в него нужно передать объект, из которого он будет вычитывать данные – system.in.

- Но **System.in** и **BufferedReader** не совместимы друг с другом, поэтому мы используем переходник – еще один объект **InputStreamReader**.

- Использовать Scanner довольно удобно, но от этого не очень много пользы. Дело в том, что в будущем в учебе вы будете часто использовать объекты **BufferedReader** и InputStreamReader и очень-очень редко объект типа Scanner. В данной ситуации он удобен, но в будущем толку от него мало. Так что мы пользоваться им не будем.

Ветвления.

Команда и блок команд

- Тело метода состоит из команд. Каждая команда оканчивается точкой с запятой.

	Примеры команд:
1	String s = "Name";
2	System.out.println(1234);
3	**return** a + b * c;
4	**throw new** RuntimeException();
5	;

- Блок команд состоит из нескольких команд, объединённых вместе фигурными скобками. Тело метода является блоком команд.

- Почти для всех ситуаций справедливо правило: там, где можно написать одну команду, можно написать и блок команд. Мы увидим это в примерах для задач в следующих лекциях.

	Примеры:
1	{}

2	{
	throw new RuntimeException();
	}

3	{
	return null;
	}

4	{
	System.out.println(23);
	System.out.println(1);
	System.out.println(14);
	}

Условные операторы

- От программ было бы мало толку, если бы они делали абсолютно одно и то же независимо от того, как меняются внешние обстоятельства. Программе нужно уметь подстраиваться под обстоятельства, и делать одни действия в одних случаях и другие – в других. В Java это реализовано с помощью «условного оператора» — специального ключевого слова, которое позволяет выполнять разные блоки команд в зависимости от правдивости условия.
- Он состоит из трех частей: «условия», «команды 1» и «команды 2». Если условие верно (истинно), тогда выполняется «команда 1», иначе выполняется «команда 2». Команды никогда не выполняются одновременно. Общий вид этого оператора таков:

```
if (условие)
   команда_1;
else
   команда_2;
```

	Код	Пояснение

1	`if (a < b)`
	`  System.out.println("А меньше Б");`
	`else`
	`  System.out.println("Б меньше А");`
		Если a меньше b, то выполнится первая команда, в противном случае – вторая. Команды никогда не выполнятся одновременно.

2	`if (a < b)`
	`{`
	`  System.out.println("А меньше Б");`
	`  System.out.println("Б больше А");`
	`}`
	`else`
	`{`
	`  System.out.println("Б меньше А");`
	`  System.out.println("А больше Б");`
	`}`
		Вместо одной команды можно подставить блок команд. В остальном — то же самое.

3	`if (a < b)`
	`{`
	`  a = 0;`
	`}`
	`else`
	`{`
	`}`
		Блок **else** можно не писать, если он пустой.
		Данные три примера абсолютно эквиваленты.
		Можно не писать скобочки, если нужно выполнить только одну команду (но желательно все-таки их писать). Если у нас несколько команд, то скобочки писать обязательно.

4	`if (a < b)`
	`{`
	`  a = 0;`
	`}`	

5	`if (a < b)` `  a = 0;`	

Условия и Сравнения

- Самые простые операторы сравнения – это меньше (<) и больше (>).
- Так же есть «равно»(==) и «не равно»(!=). А еще «меньше либо равно»(<=) и «больше либо равно»(>=).
- Обратите внимание, что операторов «=<» и «=>» в Java нет!
- Знак «=» используется для операции присваивания, поэтому для равенства используют «==» — два знака равно. Для того, чтобы проверить, что переменные не равны, используют «!=»
- При сравнении двух переменных в Java с использованием оператора «==», происходит сравнение того, что эти переменные в себе содержат.
- Т.е. для переменных примитивных типов происходит сравнение значений.
- Для переменных ссылочных типов происходит сравнение ссылок. Т.е. если объекты идентичны внутри, но ссылки на них разные, сравнение покажет, что они неравны: результатом сравнения будет **false(ложь)**. Сравнение ссылок будет **true(истина)**, только если обе ссылки указывают на один и тот же объект.
- Для сравнения объектов можно использовать специальный метод **equals()**. Этот метод (и все методы класса Object) добавляется компилятором в ваш класс, даже если вы их не объявляли.

	Код	Пояснение
1	**int** a = 5; **int** b = 5; System.out.println(a == b);	Происходит сравнение примитивных типов. На экран будет выведено **true**.
2	Cat cat1 = **new** Cat("Vaska"); Cat cat2 = cat1; System.out.println(cat1 == cat2);	Происходит сравнение ссылок. На экран будет выведено **true**. Обе переменных хранят ссылки на один и тот же объект.
3	String s = **new** String("Mama"); String s2 = s; System.out.println(s == s2)	Происходит сравнение ссылок. На экран будет выведено **true**. Обе переменных хранят ссылки на один и тот же объект.
4	Cat cat1 = **new** Cat("Vaska"); Cat cat2 = **new** Cat("Vaska"); System.out.println(cat1 == cat2);	Происходит сравнение ссылок. На экран будет выведено **false**. Переменные хранят ссылки на два идентичных объекта Cat, но не на один и тот же.
5	String s = **new** String("Mama"); String s2 = **new** String("Mama"); System.out.println(s == s2);	Происходит сравнение ссылок. На экран будет выведено **false**. Переменные хранят ссылки на два идентичных объекта String, но не на один и тот же.
6	Cat cat1 = **new** Cat("Vaska"); Cat cat2 = **new** Cat("Vaska"); System.out.println(cat1.equals(cat2));	Происходит сравнение объектов. Если в классе Cat не переопределен метод equals(), на экран будет выведено **false**. Переменные хранят ссылки на два идентичных объекта Cat, но не на один и тот же.
7	String s = **new** String("Mama"); String s2 = **new** String("Mama"); System.out.println(s.equals(s2));	Происходит сравнение объектов. На экран будет выведено **true**. Переменные хранят ссылки на два идентичных объекта String.

Задачи

5. Напишите метод checkSeason. По **номеру месяца**, метод должен определить время года (зима, весна, лето, осень) и вывести на экран. Пример для номера месяца 2: зима. Пример для номера месяца 5: весна

```
public class Solution {
   public static void main(String[] args) {
      checkSeason(12);
      checkSeason(4);
      checkSeason(7);
      checkSeason(10);
   }

   public static void checkSeason(int month) {
      //напишите тут ваш код

   }
}
```

6. Написать метод compare(int **a**), чтобы он:
   - выводил на экран строку "**Число меньше 5**", если параметр метода меньше **5**,
   - выводил строку "**Число больше 5**", если параметр метода больше **5**,
   - выводил строку "**Число равно 5**", если параметр метода равен **5**.

```
public class Solution {
   public static void main(String[] args) {
      compare(3);
      compare(6);
      compare(5);
   }

   public static void compare(int a) {
      //напишите тут ваш код

   }
}
```

7. Ввести с клавиатуры **номер дня недели**, в зависимости от номера вывести название "понедельник", "вторник", "среда", "четверг", "пятница", "суббота", «воскресенье", если введен номер больше 7 или меньше 1 - вывести "такого дня недели не существует". Пример для номера 5: пятница. Пример для номера 10: такого дня недели не существует.

8. * Подумайте, что делает программа. Исправьте ошибку в программе чтобы переменная age объекта person изменила свое значение. Подсказка: тщательно просмотрите метод adjustAge

```
public class Solution {
   public static void main(String[] args) {
      Person person = new Person();
      System.out.println("Age is: " + person.age);
      person.adjustAge(person.age);
      System.out.println("Adjusted age is: " + person.age);
   }

   public static class Person {
      public int age = 20;

      public void adjustAge(int age) {
         age = age + 20;
         System.out.println("Age in adjustAge() is: " + age);
      }
   }
}
```