Лекция 3

Основы JAVA. Тип boolean, циклы, конкатенация.

- В этом коде объявлены две переменных count. В строке 3 объявлена переменная класса, а в строке 8 – переменная метода.

```
1. class Main
2. {
3.    public int count = 0;    //объявили переменную класса
4.
5.    public void run()
6.    {
7.       count = 15;         //обращение к переменной класса
8.       int count = 10;     //объявили локальную переменную метода
9.       count++;           //обращение к переменной метода
10.   }
11.}
```

- Когда метод run будет исполняться, то произойдет вот что:

- В строке 7 мы обращаемся к переменной класса и присваиваем ей значение 15

- В строке 8 объявляется (создается) новая переменная метода – count. Она закрывает собой переменную класса. Любой дальнейший код в методе будет видеть (обращаться) именно к переменной метода.

- Переменная метода закрывает собой переменную класса. Т.е. обращение будет происходить к переменной метода. Но к переменной класса тоже можно обратиться, только более сложным способом.

Статическая переменная	Обычная переменная
ClassName.variableName Примеры: Cat.catsCount	**this**.variableName Примеры: **this**.catsCount

Тип boolean

- Переменные типа boolean могут принимать всего два значения – **true** (истина/правда) и **false** (ложь).

- Этот тип неявно используется во многих местах. Так же, как и результат любого сложения – число, то и результат любого сравнения – истина или ложь – тип **boolean**

	Код	Пояснение
1	**boolean** m;	Два данных выражения эквивалентны. Значение переменой типа boolean по умолчанию **false** («ложь»).
2	**boolean** m = false;	
3	**if** (a > b) System.out.println(a);	В переменную m будет записан результат сравнения – **true** или **false**. Условие выполняется, если переданное в него выражение истинно – **true**.
4	**boolean** m = (a > b); **if** (m) System.out.println(a);	
5	**boolean** m = (a > b); **if** (m == true) System.out.println(a);	Не нужно сравнивать логическую переменную (типа boolean) с **true** или **false**. Результат сравнения сам будет иметь тип boolean, и в точности будет совпадать со значением сравниваемой переменой: true == true – истина. Результат выражения – true. true == false – ложь. Результат сравнения – false.
6	**boolean** m = (a > b); **if** (!m) System.out.println(b);	

	Код	Пояснение

1	`public boolean isALessThanB (int a, int b)`
	`{`
	`  if (a < b)`
	`  return true;`
	`  else`
	`  return false;`
	`}`
		Данный метод проверяет, что число А меньше числа Б.
		
		Показаны четыре эквивалентных записи сравнения. Внизу самая компактная и корректная. Всегда старайся использовать компактную запись.

2	`public boolean isALessThanB (int a, int b)`
	`{`
	`  boolean m = (a < b);`
	`  if (m)`
	`  return true;`
	`  else`
	`  return false;`
	`}`	

3	`public boolean isALessThanB (int a, int b)`
	`{`
	`  boolean m = (a < b);`
	`  return m;`
	`}`	

4	`public boolean isALessThanB (int a, int b)`
	`{`
	`  return a < b;`
	`}`	

- В Java есть три логических оператора: AND (и), OR (или) и NOT (не). С помощью них можно строить условия различной сложности. Эти операторы можно применять только к выражению, имеющему тип boolean. Т.е. (a+1) AND (3) написать нельзя, а (a>1)AND (a<3) – можно.

- Оператор NOT – унарный – его действие распространяется только на выражение справа от него. Он больше похож на знак «минус» перед отрицательным числом, чем на знак умножить

- С переменными типа boolean (логический тип) можно выполнять различные операции.

	Логический оператор	Обозначение в JAVA	Выражение	Результат
1	AND (и)	&&	true && true	true
			true && false	false
			false && true	false
			false && false	false
2	OR (или)	||	true || true	true
			true || false	true
			false || true	true
			false || false	false
3	NOT(не)	!	! true	false
			! false	true
4	Распространённые комбинации и выражения		m && !m	false
5			m || !m	true
6			! (a && b)	!a || !b
7			! (a || b)	!a && !b

Циклы

- С помощью цикла можно выполнить какую-нибудь команду или блок команд несколько раз. Общий вид цикла такой:

```
while(условие типа boolean)
   команда;

или

while(условие типа boolean)
   блок команд в фигурных скобках
```

- Команда или блок команд выполняются снова и снова, пока условие цикла истинно – равно true. Сначала проверяется условие, затем выполняется тело цикла (блок команд), затем снова проверяется условие и снова выполняется тело цикла. И так до тех пор, пока условие не станет ложным. Если оно всегда истинно, программа никогда не прекратит работать и будет всегда выполнять цикл. А если всегда ложно, то тело цикла не выполнится ни разу.

	Код	Описание

1	**int** i = 3;
	while (i >= 0)
	{
	  System.out.println(i);
	  i--; //уменьшение i на 1
	}
		3
		2
		1
		0

2	**int** i = 0;
	while (i < 3)
	{
	  System.out.println(i);
	  i++; //увеличение i на 1
	}
		0
		1
		2

3	**boolean** isExit = false;
	while (!isExit)
	{
	  String s = buffer.readLine();
	  isExit = s.**equals**("exit");
	}
		Программа будет вводить строки с клавиатуры, пока не будет введена строка «exit».

4	**while** (true) System.out.println("C");	Программа будет вечно печатать на экран букву C.

5	**while** (true)
	{
	  String s = buffer.readLine();
	  if (s.**equals**("exit"))
	  break;
	}
		Программа будет вводить строки с клавиатуры, пока не будет введена строка «exit».

Цикл for

- Цикл называется for. Это ещё одна запись цикла while, просто более компактная и удобная [для программистов]

- Данные циклы эквиваленты. Если **while** содержит в скобках одно условие, то for – три. Но при компиляции **for** превращается в такой же цикл, как и **while**.

- Первое выражение в цикле **for** (выделено зеленым) выполняется один раз до цикла.

- Второе выражение выполняется каждый раз перед исполнением тела цикла – аналогично условию в цикле while.

- Третье – каждый раз после исполнения тела цикла.

	while	for

1	**int** i = 3;
	while (i >= 0)
	{
	  System.out.println(i);
	  i--;
	}
		for (**int** i = 3; i >= 0; i--)
		{
		  System.out.println(i);
		}

2	**int** i = 0;
	while (i < 3)
	{
	  System.out.println(i);
	  i++;
	}
		for (**int** i = 3; i >= 0; i--)
		{
		  System.out.println(i);
		}

3	**boolean** isExit = false;
	while (!isExit)
	{
	  String s = buffer.readLine();
	  isExit = s.equals("exit");
	}
		for (**int** i = 3; i >= 0; i--)
		{
		  System.out.println(i);
		}

4	**while** (true)
	  System.out.println("C");
		for (**int** i = 3; i >= 0; i--)
		{
		  System.out.println(i);
		}

5	**while** (true)
	{
	  String s = buffer.readLine();
	  if (s.equals("exit"))
	  break;
	}
		for (**int** i = 3; i >= 0; i--)
		{
		  System.out.println(i);
		}

Конкатенация

- Склеивание или объединение строк ещё называют таким коротким словом, как конкатенация.

- Принцип склеивания строк простой. Если «складывать» строку и ещё что-то, то это что-то преобразовывается в строку посредством неявного вызова метода toString().

	Код	Что будет на самом деле
1	Cat cat = **new** Cat(); String text = "Cat is " + cat;	Cat cat = new Cat(); String s = cat.toString(); String text = "Cat is " + s;
2	**int** a = 5; String text = "A is " + a;	int a = 5; String s = Integer.toString(a); String text = "A is " + s;
3	**int** a = 5; String text = a + "A is ";	int a = 5; String s = Integer.toString(a); String text = s + "A is ";
4	Cat cat = **new** Cat(); **int** a = 5; String text = "Cat is " + cat + a;	Cat cat = new Cat(); String s1 = cat.toString(); String s2 = Integer.toString(a); String text = "Cat is " + s1 + s2;
5	Cat cat = **new** Cat(); **int** a = 5; String text = a + "Cat is " + cat + a;	Cat cat = new Cat(); String s1 = cat.toString(); String s2 = Integer.toString(a); String s3 = Integer.toString(a); String text = s3 + "Cat is " + s1 + s2;
6	Cat cat = **new** Cat(); **int** a = 5; String text = cat + a + "Cat is " + cat + a;	Программа не скомпилируется! Порядок выполнения всех операций сложения: слева направо, получаем: String text = (((cat + a) + "Cat is ") + cat) + a; При сложении котов с числами, автоматического преобразования к строке не произойдёт.
7	Cat cat = **new** Cat(); **int** a = 5; String text = cat + (a + "Cat is ") + cat + a; //Это равносильно: Cat cat = **new** Cat(); **int** a = 5; String text = ((cat + (a + "Cat is ")) + cat)+a;	Cat cat = new Cat(); String s1 = cat.toString(); String s2 = cat.toString(); String s3 = Integer.toString(a); String s4 = Integer.toString(a); String text = s1 + s3 + "Cat is " + s2 + s4;

Задачи

1. Вывести на экран числа от **1** до **10** используя цикл while. Каждое значение с новой строки.

2. Вывести на экран числа от **10** до **1** используя цикл while. Каждое значение с новой строки.

3. Используя цикл for вывести на экран чётные числа от **1** до **100** включительно. Каждое значение вывести с новой строки.

4. Реализуйте метод public static void div(int **a**, int **b**). Метод должен делить первое число на второе, и выводить на экран результат деления a на b. На экран должно быть выведено целое число.

```
public class Solution {
   public static void main(String[] args) {
      div(6, 3);
      div(10, 6);
      div(2, 4);
   }

   public static void div(int a, int b) {
      //напишите тут ваш код
   }
}
```

Знакомство с классами: написание своих классов, конструкторы.

Что такое класс?

- Все вещи в физическом мире состоят из атомов. Атомы могут быть разных типов: водород, кислород, железо, уран, ... Комбинируя такие атомы можно создавать различные молекулы, вещества и предметы. Атомы в свою очередь тоже имеют некоторую внутреннюю структуру. Внутри них содержатся электроны и ядро: протоны и нейтроны.

- В Java все очень похоже. Программы состоят из объектов, которые бывают разных типов (классов). Различные классы в свою очередь имеют различные внутренние структуры: переменные и методы. Если рассмотреть программу в целом, то объекты — это блоки, из которых состоит программа. А классы — это типы этих блоков. Блоки разных типов являются объектами разных классов.

- Когда нам нужен новый тип объектов, мы создаем новый класс, и внутри него описываем нужное нам поведение этих объектов.

- С точки зрения внутренней структуры, класс состоит из методов класса, которые что-то делают, и переменных класса, в которых эти методы хранят различные данные.

- Класс – это группа методов, работающих вместе, и переменные, в которых эти методы хранят различные значения, которыми пользуются сообща.

Схема создания класса

1. Программист решает, какие еще объекты ему нужны.
2. Программист разбивает эти объекты на различные типы в зависимости от того, что они должны делать.
3. Для каждого типа программист пишет свой отдельный класс.
4. В классе он объявляет нужные методы и переменные.
5. В каждом методе пишутся команды, чтобы метод делал то, что хочет от него программист.
6. Класс готов, можно создавать его объекты.

- Важной особенностью классов является возможность создавать экземпляры этих классов – объекты.

- Для того, чтобы создать объект класса, в коде надо написать «new имя_класса()». Примеры:

```
Cat cat = new Cat();

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

InputStream is = new FileInputStream(path);
```

- У объекта класса есть две интересные особенности:

- **Первая**. Каждый объект класса хранит свою собственную копию переменных класса. Т.е. если в классе объявлены переменные «x,y» и создано 10 объектов такого класса, то в каждом объекте будут свои переменные. Изменение переменных одного объекта не влияет на переменные другого объекта.

- **Вторая**. При создании объекта, в него можно передавать различные параметры. Это так называемые «стартовые значения». Это почти как дать имя при рождении. Многие объекты не могут быть созданы, если в них не передать такие параметры.

Пакеты (packages)

- Файлы в компьютере группируются по папкам. Классы в Java (а каждый класс лежит в отдельном файле) группируются по пакетам, которые являются папками на диске.

- **Первое**. «Полным уникальным именем класса» является «имя пакета» + «имя класса». Примеры:

	Полное уникальное имя	Имя пакета	Имя класса
1	java.io.FileInputStream	java.io	FileInputStream
2	java.lang.String	java.lang	String
3	java.util.ArrayList	java.util	ArrayList
4	org.apache.tomcat.Servlet	org.apache.tomcat	Servlet
5	Cat	отсутствует	Cat

- Полное имя класса всегда уникально!

- Каждый раз писать длинное имя, например **java.util.ArrayList**, очень неудобно. Поэтому в Java добавили возможность «импортировать классы».

- Делается это конструкцией вида «import java.util.ArrayList;»

- **Второе**. Лучше всегда класть классы в пакеты, а не в корень папки src. Когда классов мало, это ещё не представляет проблему, но когда классов много – очень легко запутаться. Поэтому всегда создавай классы только в пакетах.

- В Java принято давать классам и пакетам осмысленные имена. Многие компании выпускают свои библиотеки (набор классов) и, чтобы не было путаницы, называют пакеты этих классов по имени компании/сайта:

	Имя пакета	Имя компании/проекта
1	org.**apache**.common org.**apache**.tomcat org.**apache**.util	Проект «Апач»
2	com.**oracle**.jdbc	Компания «Oracle»
3	**java**.io **java**x.servlet	Компания Sun, проект Java
4	com.**ibm.websphere**	Компания «IBM», проект WebSphere
5	com.**jboss**	Проект «Jboss»

Создание объектов

- При создании объекта в скобочках можно передавать различные параметры.

Код на Java	Описание

class Cat {
  public String **name**;
 	
public String getName() {
  return name;
  }
 	
public void setName(String name) {
  this.**name** = name;
  }
}
	name – это переменная класса. У нее модификатор доступа public, поэтому она видна везде в проекте.

	Метод getName – это getter, т.е. он возвращает значение переменной-класса name. Имя метода строится по принципу get + «имя переменной с большой буквы».
	Метод setName – это setter, т.е. он используется для присваивания нового значения переменной-класса name. Имя метода строится по принципу set + «имя переменной с большой буквы». В этом методе имя параметра совпадает с именем переменной класса, поэтому мы ставим **this** перед именем переменной.

- Чтобы другие классы могли менять значения приватных переменных, для каждой из них создается пара методов: **get** и **set**. Задача метода **get** вернуть текущее значение переменной тому, кто его вызвал. Задача метода **set** установить новое значение переменной.

- Если мы не хотим, чтобы кто-то менял значения переменных наших объектов, мы можем просто не писать метод **set** для него, или сделать его **private**. Также в этот метод можно добавить дополнительные проверки данных. И если переданное новое значение неверно, то ничего не менять.

- Т.к. переменных в классе может быть много, то методы get и set обычно имеют в своем имени имя той переменной, с которой работают. Если переменная называется name, то методы **setName** и **getName**. И т.д. по аналогии.

Шаг	Код	Описание
1	`new Cat();`	Создаём объект типа Cat
2	`Cat catVaska = new Cat();`	Сохраняем ссылку на объект в переменную catVaska, имеющую тип Cat

3	`catVaska.name = "Vaska";`
	`catVaska.age = 6;`
	`catVaska.weight = 4;`
		Заполняем объект данными: имя, возраст, вес

4	`catVaska.sleep();`	Вызываем метод объекта
5	`catVaska.fight(catBarsik);`	Взаимодействие объектов.

Инициализация объектов

- Когда объект создаётся – его переменным нужно присвоить стартовые данные. Чтобы не было ситуаций, когда вы обращаетесь к объекту, а внутри у него нет нужной ему информации для правильной работы.

- Рассмотрим для примера объект типа File (файл). Минимальной необходимой информацией для файла является его имя.

- Для этого добавим в наш класс метод **initialize**().

```
class MyFile
{
  private String filename = null;

  public void initialize(String name)
  {
    this.filename = name;
  }
...
}
```

- Мы добавили метод **initialize**, чтобы с объектом можно было работать – вызывать его методы. Это можно делать сразу после вызова метода **initialize**. Если с объектом работать нельзя, его называют **невалидным** (invalid), если можно – **валидным** (valid). Основная задача метода initialize – передать в объект все необходимые данные, чтобы сделать его валидным.

- Представим, что другому программисту, который будет использовать наш класс, удобнее передавать в него не полное имя файла, а директорию и короткое имя файла. Мы бы смогли реализовать эту функциональность с помощью ещё одного метода **initialize** (Java позволяет создавать несколько методов с одинаковыми именами). Тогда наш класс стал бы выглядеть так:

```
class MyFile
{
   private String filename = null;
   public void initialize(String name)
   {
      this.filename = name;
   }

   public void initialize(String folder, String name)
   {
      this.filename = folder + name;
   }
...
}
```

- Вызывать метод initialize нужно сразу после создания объекта, чтобы перевести его в валидное состояние:

```
MyFile file = new MyFile();
file.initialize("c:\data\a.txt");

String text = file.readText();
MyFile file = new MyFile();
file.initialize("c:\data\", "a.txt");

String text = file.readText();
MyFile file = new MyFile();
file.initialize("c:\data\a.txt");

MyFile file2 = new MyFile();
file2.initialize(file, "a.txt");

String text = file2.readText();
```

- Если нужно создать временную копию файла рядом с текущим, то пишут:

```
class MyFile
{
  private String filename = null;
  public void initialize(String name)
  {
    this.filename = name;
  }

  public void initialize(String folder, String name)
  {
    this.filename = folder + name;
  }

  // Файл filename будет находиться в той же директории, что и file.
  public void initialize(MyFile file, String name)
  {
    this.filename = file.getFolder() + name;
  }

...
}
```

- getFolder() должен возвращать строку с именем папки, в которой находится наш файл.

Конструкторы

- Существует сокращённая запись создания и инициализации объекта:

	Без использования конструктора	С использованием конструктора
1	MyFile file = **new** MyFile(); file.initialize("c:\data\a.txt"); String text = file.readText();	MyFile file = **new** MyFile("c:\data\a.txt"); String text = file.readText();
2	MyFile file = **new** MyFile(); file.initialize("c:\data\", "a.txt"); String text = file.readText();	MyFile file = **new** MyFile("c:\data\", "a.txt"); String text = file.readText();

3	MyFile file = **new** MyFile();
	file.initialize("c:\data\a.txt");
	
	MyFile file2 = **new** MyFile();
	file2.initialize(file, "a.txt");
	String text = file2.readText();
		MyFile file = **new** MyFile("c:\data\a.txt");
		
		MyFile file2 = **new** MyFile(file, "a.txt");
		String text = file2.readText();

- Объявить конструктор в классе очень легко. Конструктор — это тот же метод initialize, но с двумя отличиями:
  а) Имя метода-конструктора совпадает с именем класса (вместо initialize).
  б) У метода-конструктора нет типа (никакой тип не указывается вообще).

- Фактически тот же метод initialize, но с небольшими отличиями.

Без использования конструктора

```
class MyFile
{
  private String filename = null;

  public void initialize(String name)
  {
    this.filename = name;
  }

  public void initialize(String folder, String
name)
  {
    this.filename = folder + name;
  }

  public void initialize(MyFile file, String
name)
  {
    this.filename = file.getFolder() + name;
  }


…
}
```

С использованием конструктора

```
class MyFile
{
  private String filename = null;

  public MyFile(String name)
  {
    this.filename = name;
  }

  public MyFile(String folder, String
name)
  {
    this.filename = folder + name;
  }

  public MyFile(MyFile file, String name)
  {
    this.filename = file.getFolder() +
name;
  }


…
}
```

Правила именования переменных, классов, методов, констант и т.п. в Java

- Согласно принятым в сообществе Java соглашениям, есть ряд правил, которых желательно придерживаться при написании программы, если вы нарушите эти правила, компилятор не отметит имя как ошибочное и программа будет работать, но вашим коллегам будет трудно читать ваш код, да и вам самим, возможно, в будущем будет сложно понимать свой код, который вы раньше написали, ну и вам соответственно будет сложно читать чужой код, не зная правил.

Соглашения об именовании переменных, классов, методов, интерфейсов, пакетов, констант в Java.

	Тип	Правила именования	Примеры
1	Классы	**Имя класса начинается с большой буквы**, если в имени несколько слов, каждое слово пишется с заглавной буквы слитно. Имена классов должны быть существительными. Старайтесь, чтобы ваши имена классов выглядели просто и наглядно. Используйте целые слова, избегайте сокращений и аббревиатур.	class Raster; class ImageSprite;
2	Интерфейсы	Интерфейсы именуются точно так же как и классы.	interface RasterDelegate; interface Storing;
3	Переменные	**Переменные начинаются со строчной первой буквы**, если в имени несколько слов, каждое следующее слово пишется с заглавной буквы слитно. Имена переменных не должны начинаться с подчеркивания "_" или знака доллара "$". Имена переменных должны быть короткими, но со смыслом. Переменных состоящих из одного символа следует избегать, за исключением временных(одноразовых) переменных. Общие имена для временных переменных: i, j, k, m Общие имена для числовых переменных: n Общие имена для символьных переменных: c, d, e	int i; char c; float myWidth;

4	Методы	**Методы начинаются со строчной первой буквы**, если в имени несколько слов, каждое следующее слово пишется с заглавной буквы слитно. Методы должны быть глаголами.	run();
			runFast();
			getBackground();

5	Константы	**Константы должны состоять из заглавных символов**, если в имени несколько слов, каждое следующее слово отделяется от предыдущего символом подчеркивания "_".	static final int MIN_WIDTH = 4;
			static final int MAX_WIDTH = 999;
			static final int GET_THE_CPU = 1;

6	Пакеты	**Имя пакета пишется только строчными буквами**. Префикс уникального имени пакета должен быть одним из имен верхнего доменного уровня: ru, su, com, org, net, edu, gov и т.п. как указано в стандарте ISO 3166, 1981. Последующие компоненты имени пакета варьируются в зависимости от собственной внутренней организации домена.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese

Задачи

5. Вводить с клавиатуры числа и вычислить среднее арифметическое. Если пользователь ввел **-1**, вывести на экран среднее арифметическое всех чисел и завершить программу. **-1** не должно учитываться. Примеры: при вводе чисел 1 2 2 4 5 -1 получите вывод 2.8

6. Создать класс **Friend** (друг) с тремя инициализаторами (тремя методами initialize):
   - Имя
   - Имя, возраст
   - Имя, возраст, пол

   Примечание: Имя(String), возраст(int), пол(char).

```
public class Solution {

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

  public class Friend {
    //напишите тут ваш код

    public static void main(String[] args) {

    }
  }
```

7. Создать класс **Dog** (собака) с тремя **инициализаторами**:
   - Имя
   - Имя, рост
   - Имя, рост, цвет

8. Создать класс **Dog** (собака) с тремя **конструкторами**:
   - Имя
   - Имя, рост
   - Имя, рост, цвет

9. Программа должна считывать **два числа** с клавиатуры и **выводить** их **сумму** на экран.

```
public class Solution {
   public void main(String[] args) throws Exception {
      BufferReader reader = new BufferedReader(new InptStreamReader(System.in);
      int a = reader.readLine();
      int b = reader.read();

      char sum == a % b;
      System.out.prinln("Sum = " + sum);
}
```