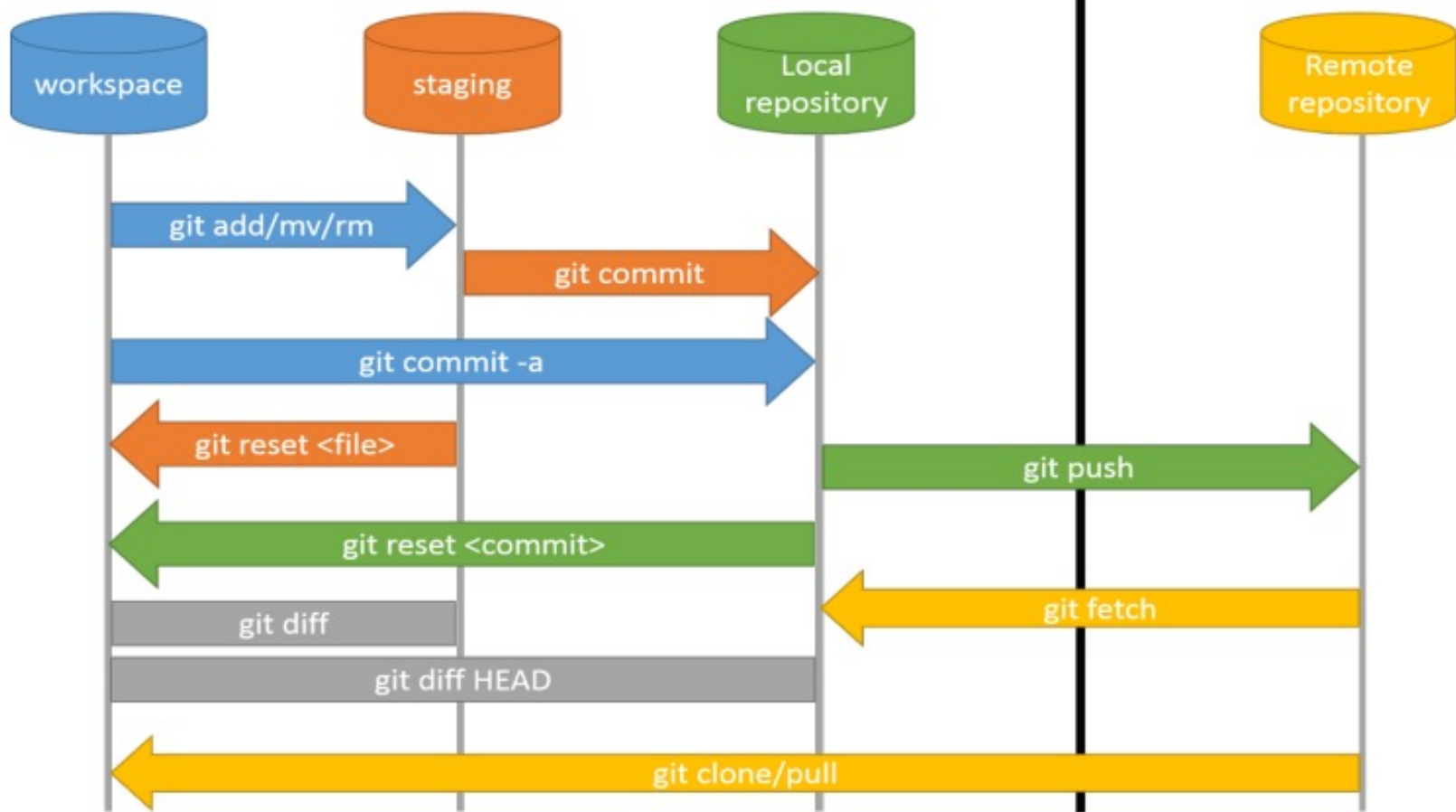
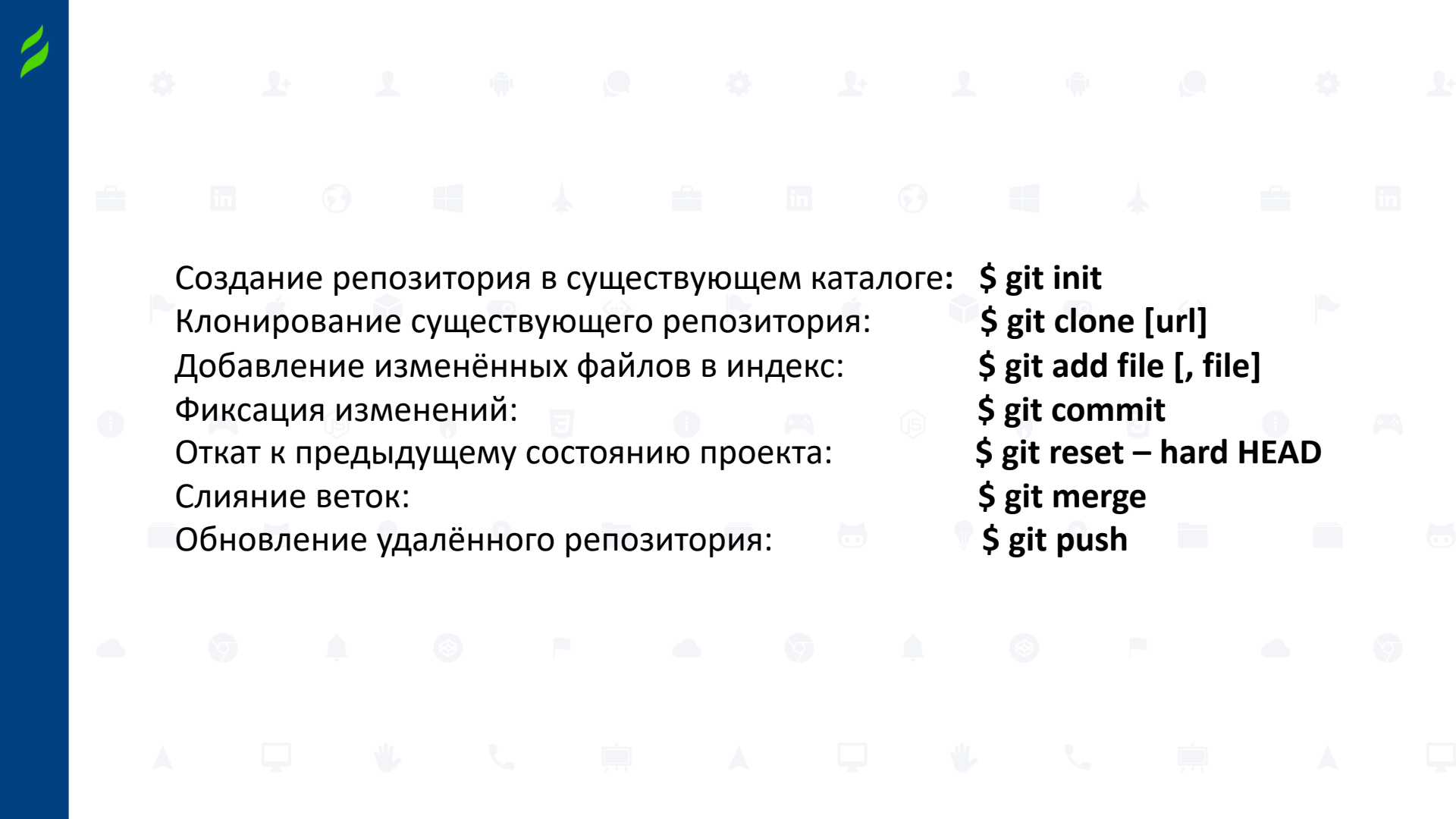




# Lesson 2

16.12.2021





Создание репозитория в существующем каталоге:	<b>\$ git init</b>
Клонирование существующего репозитория:	<b>\$ git clone [url]</b>
Добавление изменённых файлов в индекс:	<b>\$ git add file [, file]</b>
Фиксация изменений:	<b>\$ git commit</b>
Откат к предыдущему состоянию проекта:	<b>\$ git reset – hard HEAD</b>
Слияние веток:	<b>\$ git merge</b>
Обновление удалённого репозитория:	<b>\$ git push</b>

## Структура класса Java (полей и методов)

```
package org.allyourcode.myfirstproject;
```

```
public class MyFirstJavaClass {  
  
    /**  
     * @param args  
     */  
    public static void main (String[] args) {  
        javax.swing.JOptionPane.showMessageDialog  
            (null, "Hello");  
    }  
}
```

# Комментарии

## 1. Строчные ( // )

```
1 // Строчный комментарий
2 System.out.println("Hello, World!");
```

## 2. Блочные (/\* \*/)

```
1 /*
2  * Блочный комментарий
3  */
4 System.out.println("Hello");
```

## 3. Java docs (/\*\* \*/)

```
1 package test;
2
3 /**
4  * This is a JavaDoc class comment
5  */
6 public class JavaDocTest {
```

## Класс vs. Файл

```
// Compiling this Java program would  
// result in multiple class files.
```

```
class Sample  
{  
  
}
```

```
// Class Declaration  
class Student  
{  
  
}
```

```
// Class Declaration  
class Test  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Class File Structure");  
    }  
}
```

# main()

```
1 | public static void main(String[] args) { }
```

***public*** – модификатор доступа

***static*** – ключевое слово которое показывает что не надо создавать экземпляр класса для вызова метода

***void*** – возвращаемое значение (ничего не возвращать)

***main*** – имя метода которое позволяет JVM идентифицировать начальную точку запуска приложения

***String[] args*** – массив параметров с которыми может запускаться приложение



## Имена переменных и методов

1. Имя должно состоять из латинских букв. Хотя джава и поддерживает кириллицу — все же лучше называть переменные латиницей. От себя добавлю — давайте им английские названия.
2. Название должно нести некий смысл. Это означает что переменные с именем *a* или *newVariable* не будут нести никакого смысла. Постарайтесь чтобы имя было «говорящим». Это нелегкий труд — придумывать имена для переменных, классов и методов. Со временем и с опытом делать это будет легче.
3. Нельзя использовать цифры в начале имени. Это не только негласное правило, но и правило языка java. Если Вы попытаетесь использовать цифры в начале имени — программа не скомпилируется. Если, же цифры в середине или в конце имени — то их использование допустимо если же цифра будет нести некий смысл переменной, а не просто *a1*, *a2*.
4. Хотя использование символа \$ и \_ допустимо в начале названия, я бы посоветовал Вам по возможности, избегать эти символы в именах.



5. Программисты, которые пишут программы на языке java используют camelCase (верблюжийСтиль). Если имя переменной состоит больше, чем с одного слова следующее слово пишется с большой буквы: *userName*, *jackpotFunctionalView*.

6. Нельзя использовать зарезервированные слова в качестве имени. На рисунке ниже представлены все служебные 54 слова в языке java. Учить и запоминать их не нужно. Со временем они сами «засядут» в голове.

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

# Типы данных Java

## Примитивные типы (primitive types)

Логический тип  
`boolean`

## Числовые типы (numeric types)

### Целые типы (integral types)

`byte`

`short`

`int`

`long`

`char`

### Вещественные типы (floating point types)

`float`

`double`

## Ссылочные типы (reference types)

Классы

Массивы

Интерфейсы

Переменные  
типов


## Примитивные типы данных

Тип данных	Представляет	Объём памяти	Диапазон допустимых значений	Значение по умолчанию
byte	Целые числа	1 байт	от -128 до 127	0
short	Целые числа	2 байт	-32768 до 32767	0
int	Целые числа	4 байт	-2147483648 до 2147483647	0
long	Целые числа	8 байт	от -9223372036854775808 до 9223372036854775807	0L
float	Вещественные числа	4 байт	от $\sim 1,4 \cdot 10^{-45}$ до $\sim 3,4 \cdot 10^{+38}$	0.0f
double	Вещественные числа	8 байт	$\sim 4,9 \cdot 10^{-324}$ до $\sim 1,8 \cdot 10^{+308}$	0.0d
char	Символ Unicode	2 байт	от '\u0000' (или 0), до '\uffff' (или 65535)	\u0000
boolean	Логические значения	1 байт	false / true	false

## Создание и инициализация переменной

1. С помощью операции присваивания знака = мы присваиваем значение переменной. Причём она всегда обрабатывает справа-налево:

k = 10




Значение 10 присваивается переменной k

2. Присваивать значение переменной мы можем 2-мя способами:  
в 2 строчки или в 1 строчку

```
1 class Test {  
2     public static void main(String args[]){  
3         int k;  
4         k = 10;  
5         System.out.println (k);  
6     }  
7 }
```

```
1 class Test {  
2     public static void main(String args[]){  
3         int k = 10;  
4         System.out.println (k);  
5     }  
6 }
```



3. Также необходимо запомнить:

Если присваиваем значение переменной типа String, заключаем его в двойные кавычки

**String title = "Обожаю Java";**

Если присваиваем значение переменной типа char, заключаем его в одинарные кавычки

**char letter = 'M';**

Если присваиваем значение переменной типа float, обязательно после него добавляем букву f

**float pi = 3.14f;**


4. Слово "инициализация" - это тоже самое, что и "присвоить начальное значение переменной"

## String

**Строка** представляет собой последовательность символов. Для работы со строками в Java определен класс String, который предоставляет ряд методов для манипуляции строками. Физически объект String представляет собой ссылку на область в памяти, в которой размещены символы.

Для создания новой строки мы можем использовать один из конструкторов класса String, либо напрямую присвоить строку в двойных кавычках:

```
String str1 = "Java";  
String str2 = new String(); // пустая строка  
String str3 = new String(new char[] {'h', 'e', 'l', 'l', 'o'});  
String str4 = new String(new char[] {'w', 'e', 'l', 'c', 'o', 'm', 'e'}, 3, 4);
```



При работе со строками важно понимать, что объект `String` является неизменяемым (**immutable**). То есть при любых операциях над строкой, которые изменяют эту строку, фактически будет создаваться новая строка.

Поскольку строка рассматривается как набор символов, то мы можем применить метод **`length()`** для нахождения длины строки или длины набора символов:

```
1 String str1 = "Java";  
2 System.out.println(str1.length()); // 4
```

А с помощью метода **`toCharArray()`** можно обратно преобразовать строку в массив символов:

```
1 String str1 = new String(new char[] {'h', 'e', 'l', 'l', 'o'});  
2 char[] helloArray = str1.toCharArray();
```

Строка может быть пустой. Для этого ей можно присвоить пустые кавычки или удалить из строки все символы:

```
1 String s = ""; // строка не указывает на объект
2 if(s.length() == 0) System.out.println("String is empty");
```

Класс String имеет специальный метод, который позволяет проверить строку на пустоту - **isEmpty()**. Если строка пуста, он возвращает true:

```
1 String s = ""; // строка не указывает на объект
2 if(s.length() == 0) System.out.println("String is empty");
```

Переменная String может не указывать на какой-либо объект и иметь значение **null**:

```
1 String s = null; // строка не указывает на объект
2 if(s == null) System.out.println("String is null");
```



## Основные методы класса String

Основные операции со строками раскрываются через методы класса String, среди которых можно выделить следующие:

**concat():** объединяет строки

**valueOf():** преобразует объект в строковый вид

**join():** соединяет строки с учетом разделителя

**compareTo():** сравнивает две строки

**charAt():** возвращает символ строки по индексу

**getChars():** возвращает группу символов

**equals():** сравнивает строки с учетом регистра

**equalsIgnoreCase():** сравнивает строки без учета регистра

**regionMatches():** сравнивает подстроки в строках

**indexOf():** находит индекс первого вхождения подстроки в строку

**lastIndexOf():** находит индекс последнего вхождения подстроки в строку

**startsWith():** определяет, начинается ли строка с подстроки

**endsWith():** определяет, заканчивается ли строка на определенную подстроку

**replace():** заменяет в строке одну подстроку на другую



**trim():** удаляет начальные и конечные пробелы

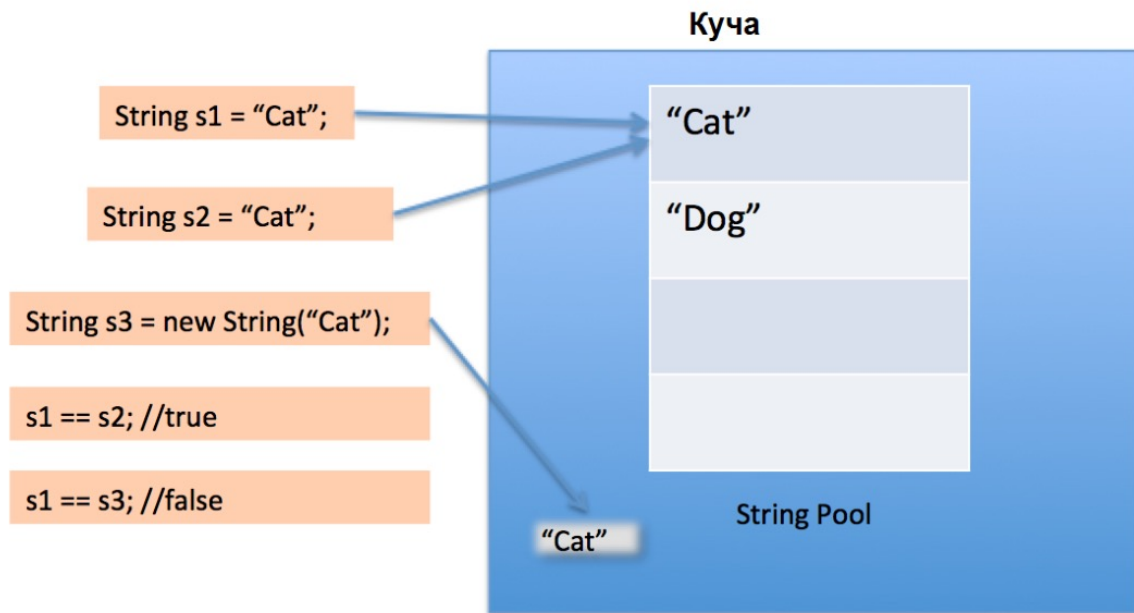
**substring():** возвращает подстроку, начиная с определенного индекса до конца или до определенного индекса

**toLowerCase():** переводит все символы строки в нижний регистр

**toUpperCase():** переводит все символы строки в верхний регистр

Пул строк (**String Pool**) — это множество строк в кучи ([Java Heap Memory](#)). Мы знаем, что String — особый класс в java, с помощью которого мы можем создавать строковые объекты.

На диаграмме ниже мы видим как именно строковый пул расположен в памяти Java Heap. И как разные способы создания строк влияют на расположение их в памяти.





Когда мы используем двойные кавычки, чтобы создать новую строку, то первым делом идет поиск строки с таким же значением в пуле строк. Если java такую строку нашла, то возвращает ссылку, в противном случае создается новая строка в пуле, а затем возвращается ссылка.

Однако использование оператора `new` заставляет класс `String` создать новый объект `String`. После этого можем использовать метод `intern()`, чтобы поместить этот объект в пул строк или обратиться к другому объекту из пула строк, который имеет такое же значение.



В Java все операторы можно разбить на 3 группы:

**арифметические**

**логические**

**побитовые**

Арифметические операторы

Сложение ( + )

Вычитание ( - )

Умножение ( \* )

Деление ( / )

Сокращённые арифметические операторы

`m += 7; // это всё равно, что m = m+7;`

`m -= 7; // это всё равно, что m = m-7;`

`m *= 7; // это всё равно, что m = m*7;`

`m /= 7; // это всё равно, что m = m/7;`

## Инкремент и декремент

В программировании очень часто приходится выполнять операции, когда:

переменная должна увеличиться на единицу  $\rightarrow +1$

переменная должна уменьшиться на единицу  $\rightarrow -1$

Поэтому придумали отдельные операции с переменными, которые называются инкремент и декремент.

**Инкремент** - отвечает за увеличение переменной на единицу. Обозначается как `++`. Например, если у нас есть переменная `i` и мы к ней применим инкремент, тогда это будет записано как `i++`. А это значит, что значение переменной `i` должно быть увеличено на 1.

**Декремент** - отвечает за уменьшение переменной на единицу. Обозначается как `--`. Например, если у нас есть переменная `n` и мы к ней применим декремент, тогда это будет записано как `n--`. А это значит, что значение переменной `n` должно быть уменьшено на 1.

## Две формы инкремента и декремента

Вы должны знать, что существует 2 формы инкремента:

**постфиксная ( $n++$ )**

**префиксная ( $++n$ )**

А также 2 формы декремента:

**постфиксная ( $n--$ )**

**префиксная ( $--n$ )**

Так а в чём же отличие между постфиксной и префиксной формами?

**В постфиксной форме:**

**сначала используется старое значение** в вычислениях

далее в последующих вычислениях используется уже новое значение

**В префиксной форме:**

**сразу используется новое значение** в вычислениях

## Логические операторы

Ниже в таблице приведены все логические операторы, которые есть в Java. Как использовать данные операторы мы с Вами рассмотрим на практике, когда дойдём до темы "Условный оператор if. Оператор switch"

ОПЕРАТОР	ПРИМЕР ИСПОЛЬЗОВАНИЯ	ВОЗВРАЩАЕТ ЗНАЧЕНИЕ "ИСТИННО", ЕСЛИ...
>	<code>a &gt; b</code>	a больше b
>=	<code>a &gt;= b</code>	a больше или равно b
<	<code>a &lt; b</code>	a меньше b
<=	<code>a &lt;= b</code>	a меньше или равно b
==	<code>a == b</code>	a равно b
!=	<code>a != b</code>	a не равно b
&&	<code>a &amp;&amp; b</code>	a и b истинны, b оценивается условно (если a ложно, b не вычисляется)
	<code>a    b</code>	a или b истинно, b оценивается условно (если a истинно, b не вычисляется)
!	<code>!a</code>	a ложно
&	<code>a &amp; b</code>	a и b истинны, b оценивается в любом случае
	<code>a   b</code>	a или b истинно, b оценивается в любом случае