WHAT IS JAVA?

Java is a general-purpose, high-level, object-oriented programming language.

It was developed by James Gosling at Sun Microsystem in 1995

PRINCIPAL OF JAVA:

Five main goals for creation of Java.

1. It must be simple, object-oriented, and familiar.
2. It must be robust and secure.
3. It must be architecture-neutral and portable.
4. It must be executed with high performance.
5. It must be interpreted, threaded, and dynamic.

WHY IS JAVA SO POPULAR?

1. Platform independent.
2. Fundamentally object-oriented.
3. Easy to learn.
4. Versatile.

USE CASES OF JAVA:

1. Building Android Apps
2. Java Web Applications.
3. Software Tools.
4. Scientific Applications.

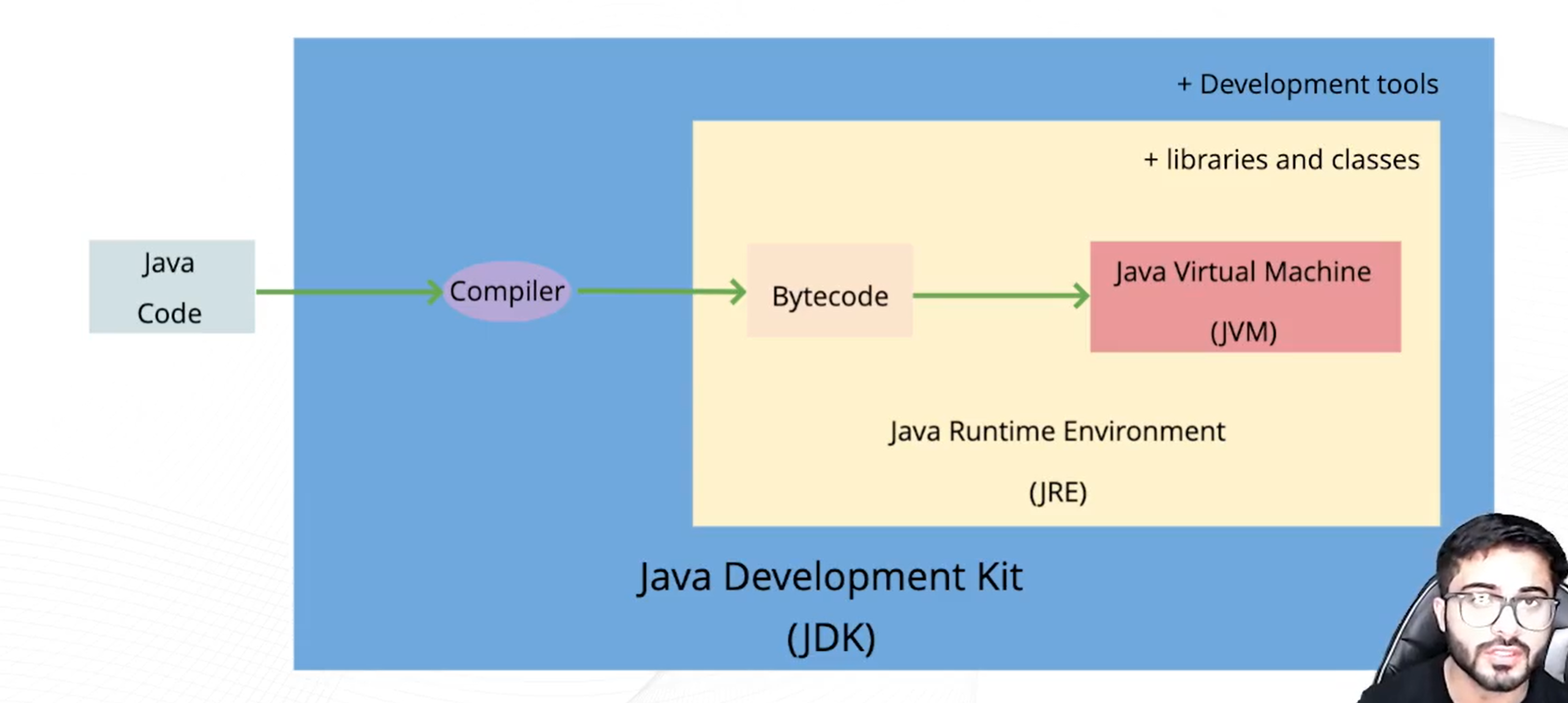
… and many more.

STEPS:

1. Introduction To JAVA
2. Setup & Installation
3. Getting Started With JAVA
4. Operators And Control Statements
5. Loops
6. Functions
7. Arrays
8. Strings
9. 2D Arrays
10. Autoboxing
11. Collections And Important Libraries
12. Introduction To OOPs

HOW JAVA WORKS UNDER THE HOOD:

1. Programmer writes the source code.
2. Compiler converts source code into bytecode/machine code.
3. Bytecode is executed over the Java virtual machine.
4. The program works over the machine.



HOW INSTALL JAVA AND INTELLIJ:

* Search intellij on google
* Open jetbrains website from google search result
* Click on download button
* Click on windows
* Download community edition for free use
* Run .exe file
* Check Java installed or not: open command prompt and run “java -version”, if it showing Java version means Java is installed otherwise it’s not installed.
* Go to google and search windows JDK.
* Open oracle website from google search
* Go to downloads
* Select latest Java version
* Select OS
* Download any Installer
* Run .exe file
* Next > Next > Close
* Set environment variable JAVA\_HOME and JRE\_HOME till Java’s bin folder

GETTING STARTED WITH JAVA: AGENDA:

* Hello world program
* Variable and datatype
* Ranges of datatype
* Typecasting
* Comments
* Constants in Java – final
* Reading Input – Scanner

FIRST PROGRAM – HELLO WORLD:

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

Each Java program has to be written inside a class as it is one of the main principals of object-oriented programming that java strictly follows.

The main method is the entry point of every Java program.

All the code which executed is written either inside a main method or it is called from main method.

Inside main method we have written print statement for string hello world.

Each and every statement is ended with semicolon (;). without this java code throw error during compilation.

VARIABLE:

A variable is a container which stores a value in a java program. Each variable has a type associated with it which is defined at its declaration.

Java is a **strongly types language**. So, every variable must be declared before using it.

public class Main {  
 public static void main(String[] args){  
 int num = 100;  
 System.*out*.println(num);}  
}

Here **num** is a variable of integer type storing value 100.

**Variable name:** A label for a memory location.

**Value:** Entity that stored in a variable.

**Storage:** A place where data can be stored. Storage tells us where in memory variable stored.

**Declaration:** Announcing a variable (usually) at the beginning of a program.

**Naming convention:** A set of rules about the names of variables.

**Assignment:** Giving(setting) a variable a value.

**Data Type:** type of data variable can store.

E.g., int mark = 100;

**Variable name:** mark.

**Value:** 100.

**Storage:** by default, storage is in RAM, however working in larger project we can change to secondary memory.

**Declaration:**  int mark; is declaration of variable, initially if no value is assigned to variable the default value is considered. If value is assigned then that value is considered.

| **Data Type** | **Default Value** | **Description** |
| --- | --- | --- |
| byte | 0 | 8-bit signed integer |
| short | 0 | 16-bit signed integer |
| int | 0 | 32-bit signed integer |
| long | 0L | 64-bit signed integer (L suffix is standard) |
| float | 0.0f | 32-bit floating point (f suffix required) |
| double | 0.0d | 64-bit floating point (d is optional) |
| char | '\u0000' | Null character (Unicode 0) |
| boolean | false | Logical type |

**Naming convention:**

We use Uppercase and Lowecase letters, digit from 0 to 9, Underscore(\_) and doller($)

Variable name must start with either lowercase alphabets or “\_” or “$” but not start with number. (like “myValues”, “$name”, “\_name” and not like “1count”)

Camel casing. (“userName”, “maxCount”, “totalAmount”)

No special characters except “\_” & “$”. (not like “my-value” or “max@speed”)

No reserved words. (not like ”int”, “class”, “public” etc.,)

Uses combination of alphabets numbers “$” and “\_”.

Must be meaningful and descriptive. (“customerName”, “temp”)

Short but not too short. (not like “x1”)

Avoid underscore unless constant. (“MAX\_SPEED”, “DEFAULT\_PORT”)

**Assignment:** “=”, here we are assigning 100 value to variable mark using assignment operator (A.K.A equality operator) i.e., “=”

**Datatype:** int (integer)

DATATYPES IN JAVA:

Data Types mean to identify the type of the data and associated operations that can be done on the data values. Data types define the value that a variable can take.

Data types also tell us information about:

The Size of the memory location.

The maximum and minimum value of range that can store in the memory location.

Different types of operation can be done on the memory location (+, -, \*, /, %).

TYPES OF DATATYPES:

1. Primitive Data Type (Java provided datatype):
2. Numeric:
3. Floating point:

float

double

1. Non-Floating point/Integers:

byte

short

int

long

1. Non-Numeric

char

boolean

1. Non-Primitive Data Type (Programmer/User provided datatype):

strings

arrays

user defined classes

public class Main {  
 public static void main(String[] args) {  
 float f=30.0f/7.0f;  
 double d=30.0/7.0;  
 byte b=8;  
 short s=1000;  
 int i=100000000;  
 long l=1000000000000000000l;  
 boolean bool=true;  
 char c='=';  
 String str="Shubham";  
 System.*out*.println(f);  
 System.*out*.println(d);  
 System.*out*.println(b);  
 System.*out*.println(s);  
 System.*out*.println(i);  
 System.*out*.println(l);  
 System.*out*.println(bool);  
 System.*out*.println(c);  
 System.*out*.println(str);  
 }  
}

output:

4.285714

4.285714285714286

8

1000

100000000

1000000000000000000

true

=

Shubham

**✅ Java Primitive Data Types – Size & Range**

| **Data Type** | **Size** | **Minimum Value** | **Maximum Value** |
| --- | --- | --- | --- |
| byte | 8 bits | -128 | 127 |
| short | 16 bits | -32,768 | 32,767 |
| int | 32 bits | -2,147,483,648 | 2,147,483,647 |
| long | 64 bits | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |
| float | 32 bits | ~1.4e-45 | ~3.4e+38 |
| double | 64 bits | ~4.9e-324 | ~1.8e+308 |
| char | 16 bits (unsigned) | 0 (\u0000) | 65,535 (\uffff) |
| boolean | 1 bit (JVM dependent) | false | true |

**💡 Notes:**

* **Integer types (byte, short, int, long)** are signed and follow **2’s complement** representation.
* **char** is **unsigned**, which is why its range is from 0 to 65,535 (can hold any Unicode character).
* **float and double** are based on IEEE 754 standard (floating-point).
* **boolean** technically occupies **1 bit** conceptually but **actual size is JVM-dependent**. It just holds true or false.

Java by default interpret floating point number as double and non-floating-point number as int. we have to explicitly mention the number is float or long. For float we can use “f” or “F” at the end of floating point number and for long we can use “l” or “L” at the end of long integer number.

e.g.,

long l=1000000000000000000l;

float f=28.4567f;

Below will throw error:

long l=1000000000000000000;

As we have not added “l” at the end it will consider this number as int and its overflowing size of int so it will throw error though we mention datatype variable as long.

public class Main2 {  
 public static void main(String[] args) {  
 long l=1\_000\_000\_000l;  
 System.*out*.println(l);  
 }  
}

output:

1000000000

In case of long java ignores “\_” from number.

STRINGS IN JAVA:

* A string is a sequence of characters in java.
* Single line strings are defined with double quotes.
* Multiline strings are defined with triple double quotes (“””).
* Java strings have several methods available to them.

public class Main {  
 public static void main(String[] args) {  
 String str1= """  
 Awesome Shubham  
 """;  
 String str2="Pikachu";  
 System.*out*.print(str1);  
 System.*out*.println(str2);  
 System.*out*.println(str1.charAt(0));  
 System.*out*.println(str1.charAt(1));  
 }  
}

output:

Awesome Shubham

Pikachu

A

W

COMMENTS:

Comments are program text used to explain the program logic. They are ignored by the compiler. Comments help to make our code more readable and maintainable. The compiler and interpreter ignore comments, so they do not affect the program’s behaviour or performance.

// This is single line comment  
  
/\*  
This  
is   
multiline  
comments  
 \*/  
public class Main {  
 // Comment can be written anywhere in the program  
 public static void main(String[] args) { // Comments ignored by compiler  
 System.*out*.println("Comments ignored by java compiler"); // comment should explain the block of code  
 }  
}

output:

Comments ignored by java compiler

CTRL+/ is shortcut for comment in any windows IDE.

TYPECASTING:

Typecasting refers to changing the type of data from one type to other.

There are two types:

1. Widening Type Casting:

Automatically done by Java

double 🡪 float 🡪 long 🡪 int 🡪 short 🡪 byte

🡨---------------------------------------------------------

Increasing order of size

1. Explicit Type Casting:

User Defined

double 🡪 float 🡪 long 🡪 int 🡪 short 🡪 byte

----------------------------------------------------------🡪

Explicit Type casting order

public class Main {  
 public static void main(String[] args) {  
 // Auto typecasting or implicit typecasting or widening  
 int x = 10;  
 long y = x;  
 //explicit typecasting or narrowing  
 long p = 1000l;  
 int q = (int)p;  
 }  
}

In explicit typecasting if value is in the range, then there is no problem however in other case some garbage value will be print

public class GarbageValue {  
 public static void main(String[] args) {  
 int x= 2000;  
 byte y = (byte) x;  
 System.*out*.println(y);  
 }  
}

output:

-48 // Some Garbage Value

CONSTANT:

Constant value creates by giving final keyword at the start of datatype.

Value to constant variable can be assign at compile time or at runtime but once constant value initialised it get locked and it cannot be modified or reinitialised at runtime.

public class Main {  
 public static void main(String[] args) {  
 final double PI = 3.14159265359;  
 }  
}

READING INPUT:

Scanner Class In Java

Java offers a variety of solution for reading inputs.

The simplest and the easiest way of reading inputs is through Scanner class.

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner ip = new Scanner(System.*in*);  
 int x = ip.nextInt();  
 }  
}

To read string we use next() or nextLine() method. next() methodonly read word but nextLine() reads whole line.

import java.util.Scanner;  
  
public class StringRead {  
 public static void main(String[] args) {  
 Scanner ip = new Scanner(System.*in*);  
 String str1 = ip.next();  
 System.*out*.println("single word : "+str1);  
 String str2=ip.nextLine();  
 System.*out*.println("whole line after reading 1st word: "+str2);  
 }  
}

output:

Hi My name is Awesome Shubham

single word : Hi

whole line after reading 1st word: My name is Awesome Shubham

There no function for reading single char. We need to read whole line and use charAt() function to read all single characters.

import java.util.Scanner;  
  
public class SingleCharRead {  
 public static void main(String[] args) {  
 System.*out*.println(new Scanner(System.*in*).nextLine().charAt(0));  
 }  
}

output:

Shubham

S

EXAMPLE: CALCULATOR

Simple interest calculator

Given the values of principal, rate and interest, compute the simple interest.

Sample Input:

P=100

R= 5

T= 2

Sample Output:

10

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner ip=new Scanner(System.*in*);  
 int p=ip.nextInt();  
 int r=ip.nextInt();  
 int t=ip.nextInt();  
  
 int si = (p \* r \* t)/100;  
 System.*out*.println("Simple Interest :"+si);  
 }  
}

output:

100

5

2

Simple Interest :10

CHALLENGE:

1) **Getting Started - 1**

**Problem Description**  
Print the following text in the output:

Hello

World !

Note: There is a empty line you need print between Hello and World !

**Output Format**

Print the required text in the output.

**Example Output**

Hello

World !

Answer:

import java.lang.\*;

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        System.out.println("Hello");

        System.out.println();

        System.out.println("World !");

    }

}

2) **Calculate Currency Change**

**Problem Description**

Your friend Rahul plans to visit exotic countries all around the world. Sadly, Rahul's math skills aren't good enough. Take the amount of money Rahul has before the currency exchange and the amount of money that is spent from his savings as **input**, print the amount of money that remains in his savings.

**Input Format**

The first line contains an integer N denoting the total savings, the amount of money before exchange.

The second line contains an integer M denoting the exchanging amount, denoting the amount of money that is spent from the savings.

**Output Format**

Print a single line denoting the amount of money that is left in his savings.

**Problem Constraints**

1 <= N <= 1000

1 <= M <= N

**Example Input**

Input:-

116

12

**Example Output**

Output:-

104

**Note:** The **problem constraints** mean that when we test your code, the test cases used in the backend can have input values only within those constraints. **You need not implement them in your code.** You must ensure your code will work for all such input values!

Answer:

import java.lang.\*;

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        int amt=sc.nextInt();

        int exchangeAmt=sc.nextInt();

        System.out.print(amt-exchangeAmt);

    }

}

3) **Getting Started - 3**

What would be the output of the following snippet?

int a =10;

int b = 20;

int c = 30;

System.out.print(a);

System.out.println(b);

System.out.print(c);

**Output Options:-**

**A**   
1020  
30

**B**  
10 2030

**C**  
102030

**D**  
10  
20  
30

A: Correct

B

C

D

4) **Getting Started - 4**

The given code gives a compilation error. Which of the following changes will remove the compilation error.

int a = 10, b = 5;          //1

int c = (a \* 1.0) / b;      //2

System.out. println(c);  //3

Change line 2 as : float c = (a \* 1.0) / b;

Change line 2 as : double c = (a \* 1.0) / b;

Change line 2 as : int c = a / b;

Both second and third options are correct : Correct

5) **Getting Started - 5**

What should be the output of the following code ?

int a = 10, b = 5;  //1

int c = (a \* 1.0) / b; //2

System.out. println(c); //3

2

2.0

Compilation Error : Correct

None of the above

OPERATORS AND CONTROL STATEMENT IN JAVA:

Intro To Operators and Control Statements:

* Arithmetic Operators
* Relational Operators
* Assignment VS Equality
* Compound Assignment
* Postfix and Prefix Operators
* Branching – if else
* Branching – switch case
* Logical Operators

ARITHMETIC OPERATORS:

public class Main {  
 public static void main(String[] args) {  
 int a = 10;  
 int b = 5;  
 // Addition  
 System.*out*.println(a+b);  
 // Subtraction  
 System.*out*.println(a-b);  
 // Multiplication  
 System.*out*.println(a\*b);  
 // Division  
 System.*out*.println(a/b);  
 // Modulo  
 System.*out*.println(a%b);  
  
 System.*out*.println("Typecasting with arithmetic operator");  
 int x = 10;  
 int y = 3;  
 System.*out*.println(x/y);  
 System.*out*.println((float)(x/y));  
 System.*out*.println((float) x/y);  
 System.*out*.println(x/(float)y);  
 System.*out*.println((float) x/(float) y);  
 }  
}

output:

15

5

50

2

0

Typecasting with arithmetic operator

3

3.0

3.3333333

3.3333333

3.3333333

RELATIONAL OPERATORS:

Relational operators are used to check some relationship between two operands.

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **DESCRIPTION** | **EXAMPLE** |
| == | Is equal to | 10 == 5 return false |
| > | Greater than | 10 > 5 return true |
| < | Less than | 10 < 5 return true |
| >= | Greater than or equal to | 10 >= 5 return true |
| <= | Less than or equal to | 10 <= 5 return false |
| != | Not equal to | 10 != 5 return true |

Result always be in Boolean.

ASSIGNMENT VS EQUALITY OPERATOR:

The ‘=’ is an assignment operator is used to assign the value on the right to the variable on the left.

The ‘==’ is the equity operator which is used to check whether two items are equal in value.

BRANCHING - IF ELSE:

The if statement is used to test a condition. It checks Boolean condition: true or false

It can optionally have an else if and an else statement attached with it as well.