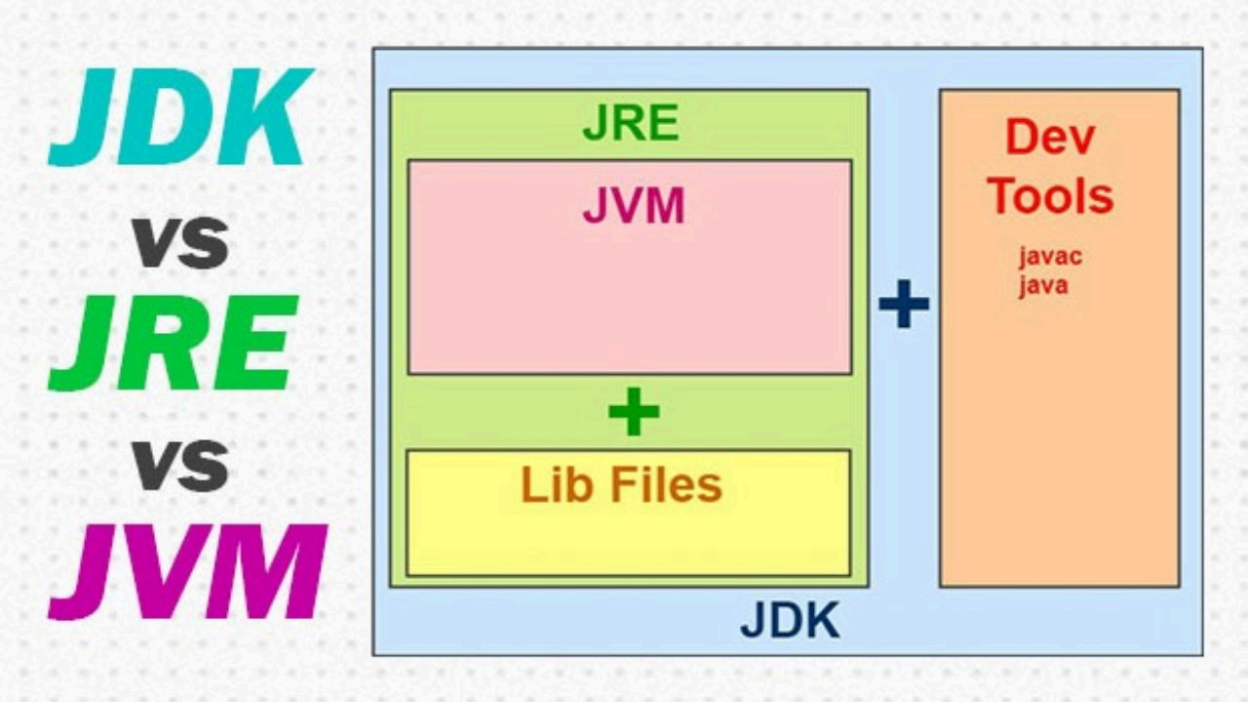
Basic Java I

- Write once, Run anywhere
- Compiled and converted to bytecode, which can be run on any machine with a JVM
- 

- JDK: JDK stands for Java Development Kit. JDK provides an environment to develop and execute Java programs.
- JRE: JRE stands for Java Runtime Environment. JRE provides an environment to run (not develop) the Java programs on your machine. JRE is only used by the end-users of the system.
- JVM: JVM stands for Java Virtual Machine, which is a very important part of both JDK and JRE because it is built into both. Whatever Java program you run using JDK and JRE goes into the JVM, and the JVM is responsible for executing the Java program line by line.
- java main() method is public because the JVM calls the main() outside the class.
- The initialization of the local variable is mandatory. If you don't initialize the variable before use, the compiler will give a compile-time error.
- Instance variables can be accessed only by creating an object of the class. These variables are created when an instance (object) of the class is created and are destroyed when the object is destroyed. Initialization of the instance variable is not mandatory. Even if you don't initialize the instance variable, it has a default value in it.

- Static or Class: These variables are created at the beginning of the program execution and destroyed automatically when the program execution ends. We can create only a single copy of a static variable. To access the static variables, we don't need to create an object of the class. We can simply access the static variable as "class_Name.variable_Name; "

- There are two types of TypeCasting in Java.

    - Widening or Automatic Type Conversion: When we assign a value of a smaller data type to a larger data type, this process is known as Widening Type Casting. It is also known as Automatic Type Conversion because the Java compiler will perform the conversion automatically. This can happen only when the two data types are compatible.

        byte -> short -> int -> long -> float -> double

    - Narrowing or Explicit Type Conversion: When we assign a value of a large data type to a small data type, the process is known as Narrowing Type Casting. This can't be done automatically. We need to convert the type explicitly. If we don't perform casting, the java compiler will give a compile-time error.

        double -> float -> long -> int -> short -> byte

- Overflow in java happens when we assign a value to a variable that is more than its range, and Underflow is the opposite of overflow. In case of overflow and underflow, the Java compiler doesn't throw any error. It simply changes the value. For example, in the case of an int variable, its size is 4 bytes or 32 bits. The maximum value of the int data type is 2,147,483,647 (Integer.MAX_VALUE), and after incrementing 1 on this value, it will return -2,147,483,648 (Integer.MIN_VALUE). This is known as overflow. The minimum value of the int data type is -2,147,483,648 (Integer.MIN_VALUE,) and after decrementing 1 from this value, it will return 2,147,483,647 (Integer.MAX_VALUE). This is known as underflow in Java.

- In Java, there are mainly two ways to get input from the user.

    - Using Scanner class: This class is used to read the input of primitive types such as int, double, long, etc., and String. You need to import the java.util package before using the Scanner class. Java Scanner class provides various methods to

read different primitive data types from the user.

Method	Description
nextInt()	reads an int value from the user.
nextFloat()	reads a float value from the user.
nextDouble()	reads a double value from the user.
nextLong()	reads a long value from the user.
nextShort()	reads a short value from the user.
nextByte()	reads a byte value from the user.
nextBoolean()	reads a boolean value from the user.
nextLine()	reads a line of text from the user.
next()	reads a word from the user.

    - Using the BufferReader class: This class is used to read the stream of characters

- NOTE: The left shift and right shift operators should not be used for negative numbers. If any of the operands is a negative number, it results in undefined behaviour. Also, if the number is shifted more than the integer's size, the behaviour is undefined.

- The *instanceof* operator in Java is used to compare an object to a specified type. We can use it to check if an object is an instance of a class, an instance of a subclass, or an instance of a class that implements a particular interface. The instanceof operator either returns true or false. **Note**: In the case of a null value, it returns false because null is not an instance of anything. It is used in the same way as (=, +, etc.).