# Java Programming

## Course Contents



## Object Oriented Programming System (OOPS)



# Introduction

Java is a true object-oriented programming, which is used to develop number of applications like

- Console application
- Windows application
- Distributed application
- Web application

FAQ in interview by this topic

What is JAVA?

Application/uses of java?



## Features of Java

- WORA Principle– write once, runs anywhere.
- Platform independent language (machine independent).
- **Platform-independent:** Java is platform-independent, which means it can run on any platform that has a Java Virtual Machine (JVM) installed. This makes it easy to write once and run anywhere.
- Memory Management
- Multi-Threading
- Secure
- Object-Oriented

# C++ v/s JAVA

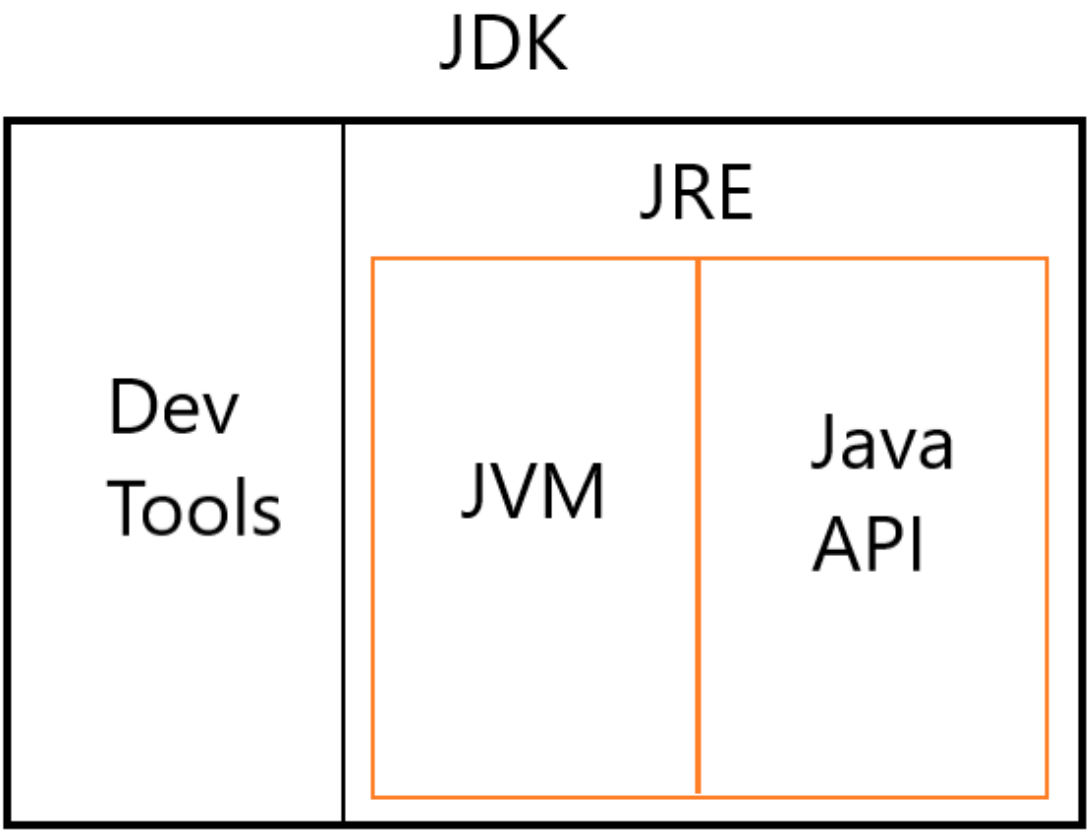
C++ and Java are two popular programming languages that are used for different purposes. While they share some similarities, there are also many differences between them. Here are a few key differences:

| Feature | C++ | Java |
|---|---|---|
| Syntax | More complex syntax, harder to learn | Simpler syntax, easier to learn |
| Memory Management | Manual memory management | Automatic memory management (JVM-managed) |
| Platform Independence | Platform-specific compilation | Platform-independent bytecode (JVM) |
| Performance | Generally considered faster | Optimized for portability, may be slower |
| Object-oriented | Supports both procedural and OOP | Strict adherence to object-oriented rules |
| Standard Library | Standard library is less extensive | More extensive standard library |

# JDK (Java Development Kit)

The Java Development Kit (JDK) is a software development kit that includes a set of tools for developing Java applications. Some of the key development tools in the JDK include:

1. Java Compiler (javac): This tool is used to compile Java source code into bytecode that can be executed by the Java Virtual Machine (JVM).
2. Java Virtual Machine (JVM): The JVM is responsible for executing Java bytecode on different platforms.
3. Java Archive (JAR) tool: The JAR tool is used to create, modify, and extract Java archive files, which are used to package Java classes and resources into a single file.
4. Java Debugger (jdb): The Java Debugger is a command-line tool that allows developers to debug Java programs by setting breakpoints, inspecting variables, and stepping through code.
5. JavaDoc tool: The JavaDoc tool is used to generate documentation for Java source code in HTML format.
6. JavaFX tools: JavaFX is a set of tools and libraries for building rich client applications in Java. The JDK includes tools for developing JavaFX applications, such as the JavaFX Packager tool, which is used to package JavaFX applications for distribution.
7. Java Mission Control (JMC): JMC is a tool for monitoring and managing Java applications, providing insights into application performance and behavior.




**JDK – JDK** is an application software. JDK is a combination of JRE and development tools.

**Development tools -** Development Tools are **software applications that help developers to make easier the entire process of developing, testing, and deployment in Java.**

**JRE** – JRE stands for java runtime environment, JRE is an application software, JRE is a combination of JVM and Java API.

**JVM** (Java Virtual Machine) – JVM is responsible to load and to execute a .class or bytecode file.

**JAVA API -** APIs are **important software components bundled with the JDK**. APIs in Java include classes, interfaces, and user Interfaces. They enable developers to integrate various applications and websites and offer real-time information.

FAQ in interview by this topic

What is JDK?

What is JVM?

What is JRE?

What is JAVA API?

Example of development tools in java?

**There are three main parts of JVM.**

1. **CLS**
2. **Memory**

3. **Execution Engine**

# JVM Architecture

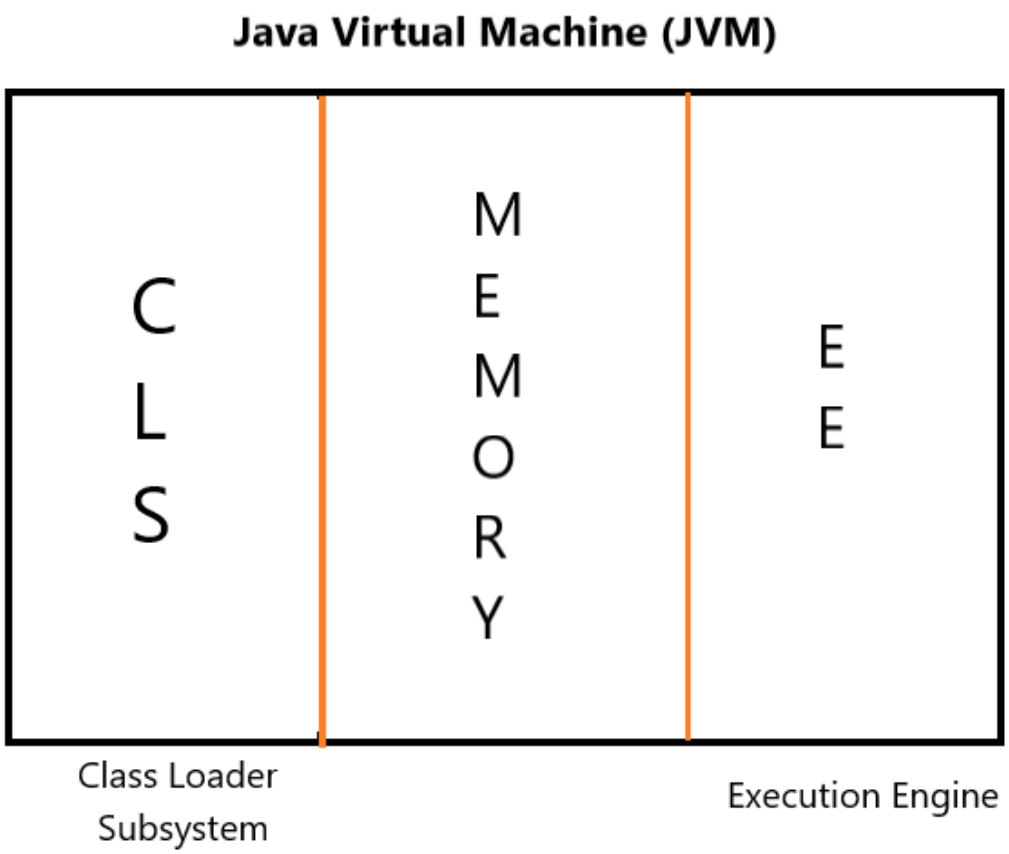
JVM (Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that calls the main method present in a java code. JVM is a part of JRE (Java Runtime Environment).

- JVM is platform dependent.

**JVM has main three parts.**

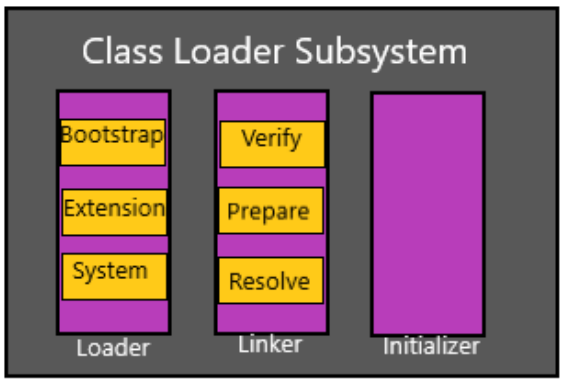
1. CLS
2. MEMORY
3. Execution Engine






**Class loader subsystem (Loader)**

Class loader subsystem is responsible to load a .class file.




Class loader subsystem consist main three parts: -

1. Loader
2. Linker
3. Initializer

**Loader**

Loader is a part class loader subsystem. It loads .class/byte-code file in various memory.

The Class loader reads the ".class" file, generate the corresponding binary data and save it in the method area.
For each ".class" file, JVM stores the following information in the method area.

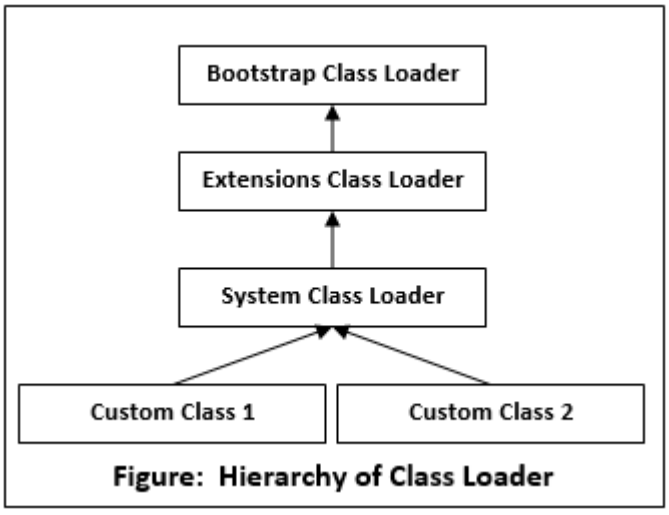


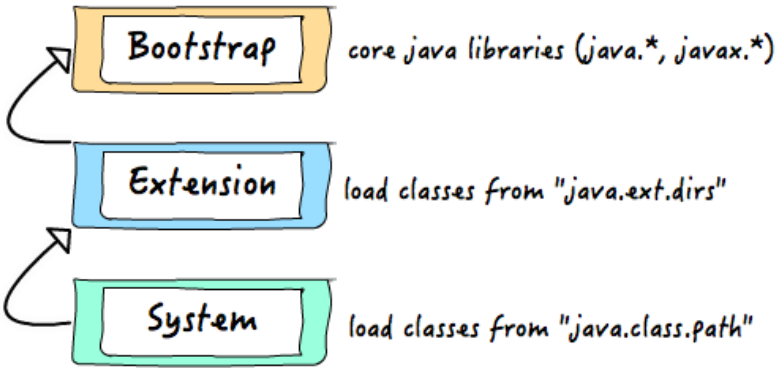
Figure: Hierarchy of Class Loader

- The fully qualified name of the loaded class and its immediate parent class.
- Whether the ".class" file is related to Class or Interface or Enum.

- Modifier, Variables and Method information etc.

**There are three types of class loaders.**

1. Bootstrap Loader
2. Extension Loader
3. Application/System Loader




After loading the ".class" file, JVM creates an object of type Class to represent this file in the heap memory. Please note that this object is of type Class predefined in java.lang package.

These Class object can be used by the programmer for getting class level information like the name of the class, parent name, methods, and variable information etc. To get this object reference we can use getClass() method of Object class.

Note: For every loaded ".class" file, only one object of the class is created.

**Bootstrap class loader:** Every JVM implementation must have a bootstrap class loader, capable of loading trusted classes. It loads core java API classes present in the "JAVA_HOME/jre/lib" directory.

This path is popularly known as the bootstrap path. It is implemented in native languages like C, C++.

**Extension class loader:** It is a child of the bootstrap class loader. It loads the classes present in the extensions directories "JAVA_HOME/jre/lib/ext." (Extension path) or any other directory specified by the java.ext.dirs system property. It is implemented in java by the sun.misc.Launcher$ExtClassLoader class.

**System/Application class loader:** It is a child of the extension class loader. It is responsible to load classes from the application class path. It internally uses Environment Variable which mapped to java.class.path. It is also implemented in Java by the sun.misc.Launcher$AppClassLoader class.

**Linker**

Linker is a computer system program that takes one or more object files (generated by a compiler or an assembler) and combines them into a single executable file, library file, or another "object" file.

**Linker performs operations on byte code: -**

Verify, prepare, and resolve.

**Verify: -** It ensures the correctness of the .class file i.e., it checks whether this file is properly formatted and generated by a valid compiler or not.

Verify is a part of linker, it checks .class file generated by a valid compiler or not.

**Prepare: -** Prepare is a part of linker. Which is situated in class loader subsystem. Allocates memory for member variable, local variable and initialized with default values.

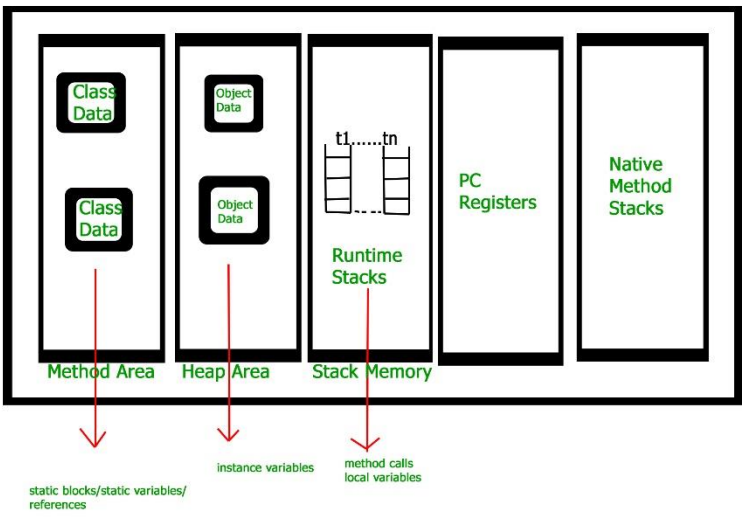
**Resolve: -** It is a process of changing the symbolic references with the original memory references from the method area.

**Initializer**

Initializer is part of class loader subsystem. It is responsible to allocate memory for static variables and initialize its respective values.

**Memory**

- Method area
- Heap area
- Stack area
- PC registers
- Native method stack



**Method area**

In the method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables. There is only one method area per JVM, and it is a shared resource.

Method area is type of various memory area in JVM architecture. Each JVM has its own method area, it contains all information of class.

**Heap area**

Information of all objects is stored in the heap area. There is also one Heap Area per JVM. It is also a shared resource.

**Stack area**

For every thread, JVM creates one run-time stack which is stored here. Every block of this stack is called activation record/stack frame which stores methods calls. All local variables of that method are stored in their corresponding frame. After a thread terminates, its run-time stack will be destroyed by JVM. It is not a shared resource.

**Frame -** In stack area every thread creates its own stack is called frame.

Frame stack divided into three parts

i. Local variable array - array of primitive data types

ii. Operand stack – expression evaluation

iii. Frame data – it is a part of stack frame it holds the symbolic reference of method



**PC Registers**

PC register is part various memory areas in JVM architecture. Each thread has its own pc

register. That contain the information of next performed operation.

**Native method stack**

For every thread, a separate native stack is created. It stores native method information.

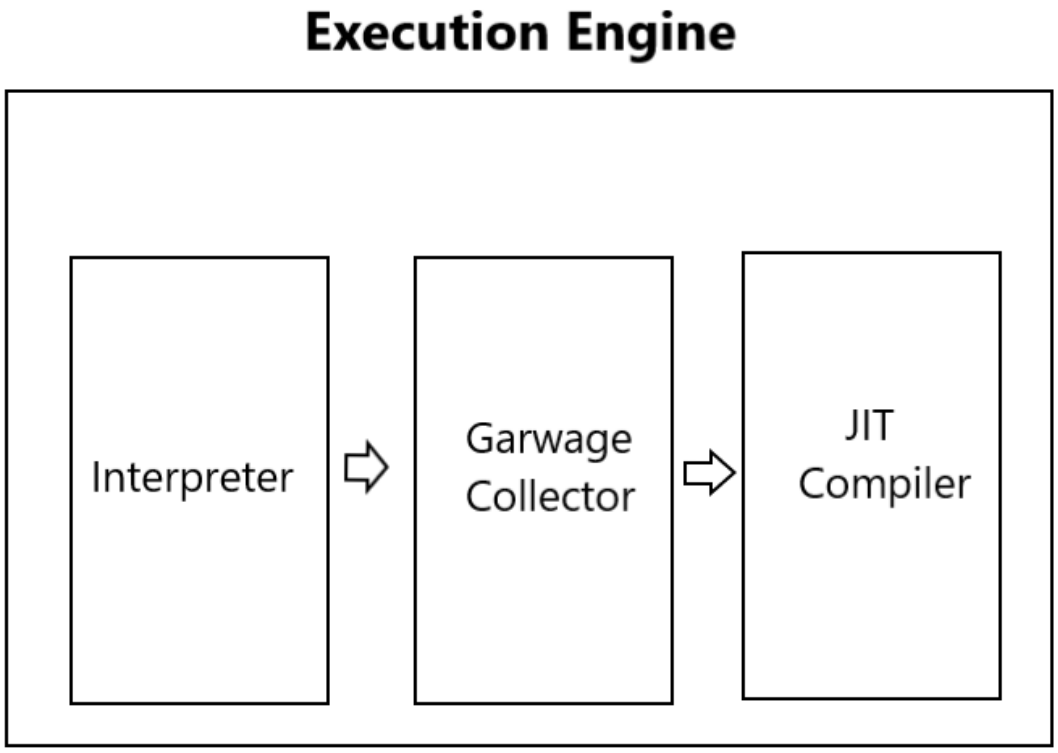
***NOTE***

*Local variables -> Stack area*
*Member variables -> Heap area*
*Static variables -> Method area*

**Execution Engine**

Execution engine executes the ".class" (bytecode). It reads the byte-code line by line, uses data and information present in various memory area and executes instructions. It can be classified into three parts:

- Interpreter
- JIT-Compiler
- Garwage Collector



**Interpreter:** Interpreter is a part of execution engine in java. It directly communicates with various memory area. Interpreter load, interprets, and execute each line of code, line-by-line and converted bytecode +into native .exe is called interpreter.

It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is required. So, we introduce JIT-Compiler.

**Just-In-Time Compiler (JIT):** It is used to increase the efficiency of an interpreter.

It compiles the entire bytecode and changes it to native code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not required, thus efficiency is improved.

FAQ in interview by this topic

What is JIT Compiler?

What is garbage +++collector?

FAQ in interview by this topic

What is interpreter?

- **JIT is used to increase the efficiency of an interpreter, whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not required, thus efficiency is improved.**

**+**

**Garbage Collector:** It destroys un-referenced objects.

**Java Native Interface (JNI):**

It is an interface that interacts with the Native Method Libraries and provides the native libraries (C, C++) required for the execution. It enables JVM to call C/C++ libraries and to be called by C/C++ libraries which may be specific to hardware.

FAQ in interview by this topic

What is JNI (Java Native Interface)?

What is profiler?

**Native Method Libraries:**

It is a collection of the Native Libraries (C, C++) which are required by the Execution Engine.

**Profiler**

Profiler is a part of execution engine in JVM architecture. The Profiler monitor native code, check the performance of native code is good or bad. If performance is bad that inform to garbage collector to remove it.

# Hello world in Java.

**Program**

```
1. public class Main {
2.     // main is method
3.     // public access modifier
4.     public static void main(String[] args) {
5.         System.out.println("Hello World!");
6.     }
7. }
```

// OUTPUT
// Hello World

FAQ in interview by this topic

What is command line arguments?

**args** - args is command line arguments.

-> Command Line Arguments?
Our console is a command prompt each line into command prompt is called command line, and arguments passes into that line is called command line arguments.

**out –** out is an object of **PrintStream** class, define into system class.

**System –** System is class define into **java.lang.*** package.

**println – println** is a public static member method of **PrintStream** class.

# Data type

FAQ in interview by this topic

What is data type?

Data type specifies the type of data that can be stored in a variable and the number of bytes required to represent a type.

Java is strongly types of language. This means that every variable must have a declared type.

There are eight primitives' data types in java.

4 -> Integer type

2 -> Floating-point number

1 -> character

1 -> boolean

**Integer types**

The integer types are for numbers without fractional parts. Negative values are allowed. Java provides the four integer types.

| Type | Storage requirement | Range (inclusive) |
|---|---|---|
| int | 4 bytes | -2,147,483,648 to -2,147,483,647 (just over 2 billion) |
| short | 2 bytes | - 32768 to 32767 |
| long | 8 bytes | Range (2^32-1)/2 |
| byte | 1 byte | -128 to 127 |

Under java, the range of integer types do not depend on the machine on which you will be running the code.

Note that java does not have any unsinged versions of the int, long, short, or byte types.

**Floating point type**

The floating-point types denote numbers with fractional parts. The two floating-point types shown.

| Type | Storage requirement | Range (inclusive) |
|---|---|---|
| float | 4 bytes | Approximately -+ 3.40x10^38 (6-7 significant decimal digits. |
| double | 8 bytes | Approximately -+ 1.79x10^308 (15 significant decimal digits. |

**The char Type**

a character constant with value 65. It is different from "A", a string containing a single character. Values of type char can be expressed as hexadecimal values that run from \u0000 to \uFFFF. For example, \u2122 is the trademark symbol (™) and \u03C0 is the Greek letter pi (π).

**Escap sequece**

a character constant with value 65. It is different from `"A"`, a string containing a single character. Values of type `char` can be expressed as hexadecimal values that run from `\u0000` to `\uFFFF`. For example, `\u2122` is the trademark symbol (™) and `\u03C0` is the Greek letter pi (π).

Escape sequence Meaning

\\ - \ character

\' - ' character

\" - " character

\? - ? character

\a - Alert or bell

\b - Backspace

\f - Form feed

\n - Newline

\r - Carriage return

\t - Horizontal tab

\v - Vertical tab

\ooo - Octal number of one to three digits



**The Boolean type**

The Boolean type has two values, false and true. It is used for evaluating logical conditions. You cannot convert between integers and Boolean values.

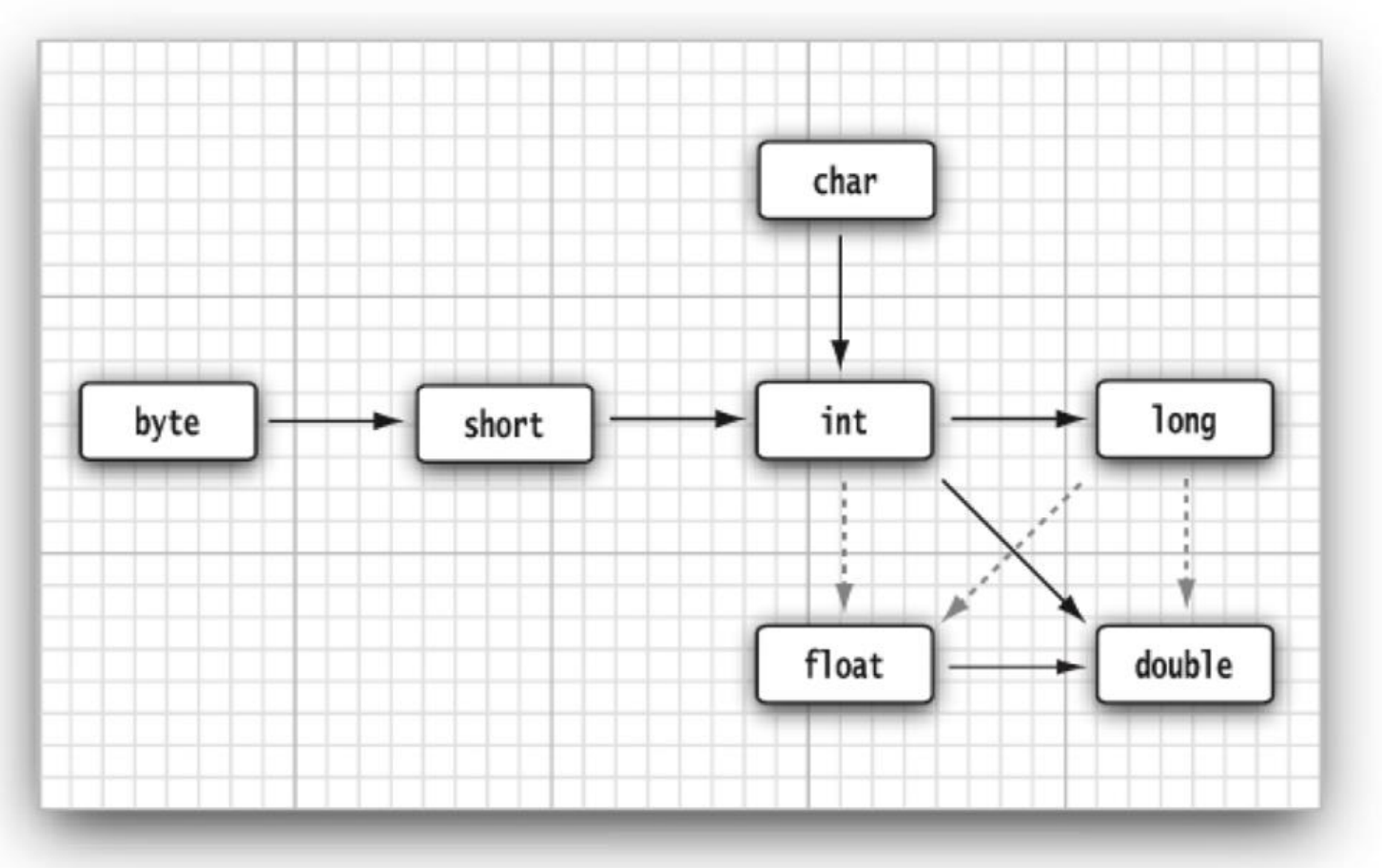
# Enumerated Types

Sometimes, a variable should only hold a restricted set of values, for example, you may sell clothes or pizza in four Size: small, medium, large and extra large.

Enumerated types must be global not local

```
class Main {
   enum Size {SMALL, MEDIUM, LARGE, EXTRA_LARGE};
   public static void main(String args[]) {
      Size s = Size.MEDIUM;
      System.out.println(s);
   }
}
```

# Type conversion




```
class Main {
   public static void main(String args[]) {
```

```
        double a = 9.997;
        int nx = (int) a;
        System.out.println(nx); // 9
    }
}
```

Now, the variable `nx` has the value `9` because casting a floating-point value to an integer discards the fractional part.

```
class Main {
    public static void main(String args[]) {
        double a = 9.997;
        int nx = (int) a;
        System.out.println(nx); // 9
        nx = (int) Math.round(a);
        System.out.println(nx); // 10

    }
}
```

# Variable & Constant

FAQ in interview by this topic

What is variable?

What is constant?

What is final keyword?

How to declare variable and constant?



**Variable**

Variable is a quantity, that can be changed at any time during program execution.

**Constant**

Variable is a quantity, that can not be changed in program and during program execution.
ex. 1,2, %, @, 's' etc.
Final keyword is used to denote a constant. For example

```
final float CM_PER_INCH = 2.54F;
final float PI = 3.14F;
```

By naming convention of java constant must be written in UPPERCASE latter.

Const is a reserved java keyword, but it is not currently used for anything, you must use final for a constant.

**Variable declaration**

In java, every variable has a type. You declare a variable by placing the type first, followed by the name of the variable. Here some example

```
int age;
```

**Initializing of variable/assigning**
Initializing of vaiable mean assign/put the value in the variable.

```
int age = 20;
```

```
class Main {
  public static void main(String args[]) {

    // here is a variable and 10 is constant.
    int a = 10;
    System.out.println(a); // output -> 10

    a = 11;
    // here value of changed
    System.out.println(a); // output -> 11
  }

}
```

A variable of type size can hold only one of the values listed in the type decleration, or the special value null that indicates that the variable if not set to any value at all.
final float

CM_PER_INCH = 2.54F;

final float

PI = 3.14F; By naming convention of java constant must be written in UPPERCASE latter. Const is a reserved java keyword, but it is not currently used for anything , you must use final for a constant.

Variable declaration

In java, every variable has a type. You declare a variable by placing the type first, followed by the name of the variable. Here some example

int age;

Initializing of variable/assigning

Initializing of variable mean assign/put the value in the variable.

int age = 20;

```
class Main {
   public static void main(String args[]) {
      // here is a variable and 10 is constant.
      int a = 10;
      System.out.println(a); // output -> 10

      a = 11;
      // here value of changed
      System.out.println(a); // output -> 11
   }
}
```



Keywords

Comments are lines of code that are ignored by the compiler.

They are used to adding human-readable explanations and notes to the source code, and can help to document and clarify the purpose of the code.

-> There are 67 keywords in Java

→ Comments are used to explain the programming logic, which useful in the future.

There are two types of comments in Java:

Single-line comments and multi-line comments.

Single-line comments start with a double forward slash (//) and extend to the end of the line. For example:

// This is a single-line comment

// comments are never executed

Multi-line comments start with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/). Anything between these markers is considered a comment. For example:

```
/* This is a
   multi-line comment */
```

```
/*
 this is a
 multiline comment
 */
```



Keywords & Identifier

Identifier

The identifier is the name given to entities such as variables, functions, structures etc. Identifiers must always be unique. They are created to give a unique name to an entity to identify it during the execution of the program.

Rules of defining Java variable/identifier: -

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

A variable name must start with a letter or the underscore character

A variable name cannot start with a number.

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _ )

Identifier names are case-sensitive (age, Age and AGE are three different variables)

Variable name can you start it from $, this is valid in c just like PHP

Keywords

Keywords are reserved words, which has special meaning to the Java compiler, there are 67 reserved words in Java, that 67 keywords

Keywords we can't use as name of an identifier.

Example – class, if, else etc.

# Operator

FAQ in interview by this topic

What is an operator?

How many types of operators in java?

Operator is nothing, but it is a symbol, that is used to perform mathematical and logical operations.

Example: - +, -, ^, %

```
class Operator {
   public static void main (String args []) {
      // sum of two number
      // here a and b both are operands
      int a = 5;
      int b = 10;

      // here + is operator used to add two number
      System.out.println(a + b);

   }
}
```

**Types of operators**

There are following types of operators in Java:

- Arithmetic operators
- Relational operators/comparison
- Logical operators
- Bitwise operators
- Assignment operators
- Ternary operators

## Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

| Operator | Name | Description | Example |
|---|---|---|---|
| + | Addition | Adds together two values | x + y |
| - | Subtraction | Subtracts one value from another | x - y |
| * | Multiplication | Multiplies two values | x * y |
| / | Division | Divides one value by another | x / y |
| % | Modulus | Returns the division remainder | x % y |
| ++ | Increment | Increases the value of a variable by 1 | ++x |
| -- | Decrement | Decreases the value of a variable by 1 | --x |

## Relational Operators

Comparison operators are used to compare two values:

| Operator | Name | Example |
|---|---|---|
| == | Equal to | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |

| | | |
|---|---|---|
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

## Logical operators

Logical operators are used to determine the logic between variables or values:

| Operator | Name | Description |
|---|---|---|
| && | Logical and | Returns true if both statements are true |
| \|\| | Logical or | Returns true if one of the statements is true |
| ! | Logical not | Reverse the result, returns false if the result is true |

## Bitwise operators

Bitwise operators are used to performing the manipulation of individual bits of a number.

Example

5 -> 1001

7 -> 0111

| Operator | Name | Description | Example |
|---|---|---|---|
| & | Bitwise and | Perform bitwise and | 5&7<br>1 000<br>0 1 1 1<br>-----<br>0000 -> 0 |
| \| | Bitwise or | Perform bitwise or | 5\|7<br>1 000<br>0 1 1 1 |

| | | | |
|---|---|---|---|
| | | | ----- |
| | | | 1 1 1 1 -> 15 |
| ^ | Bitwise xor | Perform xor operation | 5^7 |
| | | | 1 000 |
| | | | 0 1 1 1 |
| | | | ----- |
| | | | 1 1 1 1 -> 15 |
| ~ | Once complement | Bitwise complement | ~5 (1001) |
| | | | 1 0 0 1 -> 0 1 1 0 + 1 |
| | | | 0 1 1 1 -> 7 |

# Bit-Shift Operators (Shift Operators)

Shift operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number by two, respectively. They can be used when we have to multiply or divide a number by two.

**Types of Shift Operators:**

Shift Operators are further divided into 4 types. These are:

Signed Right shift operator (>>)

Unsigned Right shift operator (>>>)

Left shift operator

Unsigned Left shift operator (<<<)

# Ternary operator

Java ternary operator is the only conditional operator that takes three operands. It's a one-liner replacement for the if-then-else statement and is used a lot in Java programming.

Syntax:

variable = Expression1 ? Expression2: Expression3

# Assignment operators

Assignment operators are used to assign values to variables.

In the example below, we use the assignment operator (=) to assign the value 10 to a variable called x:

| Operator | Example | Same As |
|---|---|---|
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| *= | x *= 3 | x = x * 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| &= | x &= 3 | x = x & 3 |
| \|= | x \|= 3 | x = x \| 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

# Priority And Associativity of operators

| Operators | Precedence | Associativity |
|---|---|---|
| postfix increment and decrement | ++ -- | left to right |
| prefix increment and decrement, and unary | ++ -- + - ~ ! | right to left |
| multiplicative | * / % | left to right |
| additive | + - | left to right |

| | | |
|---|---|---|
| **shift** | << >> >>> | left to right |
| **relational** | < > <= >= instanceof | left to right |
| **equality** | == != | left to right |
| **bitwise AND** | & | left to right |
| **bitwise exclusive OR** | ^ | left to right |
| **bitwise inclusive OR** | \| | left to right |
| **logical AND** | && | left to right |
| **logical OR** | \|\| | left to right |
| **ternary** | ? : | right to left |
| **assignment** | = += -= *= /= %= &= ^= \|= <<= >>= >>>= | right to left |

# Big numbers

These are classes for manipulating numbers with an arbitrarily long sequence of digits. The BigInteger class implements arbitrary-precision integer arithmetic, and BigDecimal does the same for floating-point numbers.

```
import java.math.BigInteger;
// BigInteger class has inbuilt add, multiply, divide methods

class Main {
    public static void main(String args[]) {
```

```
        BigInteger bInt = new BigInte-
ger("22223224462942044552973989346190996720666693909649976499097960 0");
        System.out.println(bInt);
    }
}
```

# Input/Output

Reading from the "standard input stream" System.in isn't quite as simple. To read console input, you first construct a Scanner class object that is attached to System.in.

FAQ in interview by this topic

How to take input from user and display?

How to read sentence?

How to read an integer?

```
Scanner sc = new Scanner(System.in);
```

Theire are many methods are available in Scanner class for example: -

next() -> read only one word input

nextLine() -> reads a line of input

nextInt() -> to read an integer input

nextDouble() -> to read a double input



***NOTE***

*The scanner class is not suitable for reading a password from a console since the input is painly visible to anyone. Use the Console class for this purpose. To read a password from the user. Use the following code:*

```
Console cons = System.console();
String username = cons.readLine("User name: ");
char[] password = cons.readPassword("Password: ");
```

```
Console cons = System.console();
String username = cons.readLine("User name: ");
char[] password = cons.readPassword("Password: ");
```

OUTPUT

To displaying output to the standard output stream (that is, the console window) just by calling System.out.println() -> with new line

System.out.print() -> without new line

System.out.println() -> with new line

System.out.printf() -> formatted output

```
double f = 1000.0/3.0;
System.out.printf("%.3f", f);

Output-> 333.333
```



# BufferedReader

BufferedReader is a class that reads text from a character input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.

The key benefit of using BufferedReader is that it reads the input data in chunks, which makes it more efficient than reading one character at a time. It also provides methods for reading lines and arrays of characters, as well as for skipping over characters in the input stream.

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

```
public class Main{
   public static void main(String[] args) {
      try {
         BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
         String str = reader.readLine();
         System.out.println(str);
            reader.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

read() method to read individual characters.

readLine() method to read entire lines of text.

## File input and Output

To read from a file, construct a Scanner class object like this:

```
import java.nio.charset.StandardCharsets;

Scanner fileInput = new Scanner(Path.of("file.txt"), StandardCharsets.UTF_8);
```

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
```

```
public class Main{
   public static void main(String[] args) {
      try {
         BufferedReader reader = new BufferedReader(new FileReader("input.txt"));
         String line = reader.readLine();
         while (line != null) {
            System.out.println(line);
            line = reader.readLine();
         }
         reader.close();

      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```



# InputStreamReader

InputStreamReader is a class that is used to convert bytes from an InputStream into characters using a specified character encoding. This is useful when you are reading text data from an input stream, such as a file or a network socket, and you need to convert the raw bytes into characters that can be processed by your application.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
```

```
public class Main{
   public static void main(String[] args) {
      try {
         InputStream input = new FileInputStream("myfile.txt");
         InputStreamReader reader = new InputStreamReader(input, "UTF-8");
         int c = reader.read();
         while (c != -1) {
            System.out.print((char) c);
            c = reader.read();
         }
         reader.close();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

# Conditional Statements



Conditional statements in C allow you to execute a block of code only if a certain condition is met. There are three types of conditional statements in C: if, if-else, and switch.

The if statement is used to execute a block of code if a certain condition is true. The syntax for an if statement is:

Copy code

```
if (condition) {
   // code to be executed if condition is true
}
```

For example:

Copy code

```
int x = 10;
if (x > 5) {
```

```
    printf("x is greater than 5\\n");
}
```

The if-else statement is similar to the if statement, but it allows you to specify a block of code to be executed if the condition is false. The syntax for an if-else statement is:

```
Copy code
if (condition) {
   // code to be executed if condition is true
} else {
   // code to be executed if condition is false
}
```

For example:

```
Copy code
int x = 10;
if (x > 5) {
  System.out.println("x is greater than 5");
} else {
   System.out.println("x is not greater than 5");
}
```

The switch statement is another way to perform different actions based on different conditions. It is used when you have multiple conditions that depend on the value of a single expression. The syntax for a switch statement is:

```
Copy code
switch (expression) {
   case value1:
      // code to be executed if expression equals value1
      break;
   case value2:
      // code to be executed if expression equals value2
      break;
   ...
   default:
      // code to be executed if expression does not match any case
}
```

For example:

```
Copy code
int x = 2;
switch (x) {
   case 1:
      System.out.println("x is 1\\n");
```

```
      break;
   case 2:
      System.out.println("x is 2\\n");
      break;
   case 3:
      System.out.println("x is 3\\n");
      break;
   default:
      System.out.println("x is not 1, 2, or 3\\n");
}
```

Overall, conditional statements in Java allow you to control the flow of your program and execute different blocks of code based on different conditions.



# Function

FAQ in interview by this topic

What is Function?

Type of function?

Function return type?

Example of real world (Function)?

Function is a bloack of code that perform a specific task.

Or

Function is a piece of code that perform a specific task, function enables code reusability.

Main two parts of function

Function definition

Function return type

**Example**

`static` means that the method belongs to the Main class and not an object of the Main class.

```java
class Main {
    // here n1 and n2 is parameter
    static int sub(int n1, int n2) {
        return (n1-n2);
    };
    // return type is int
    // function name sum -> perform addition of two number
    static int sum(int a, int b) {
        return (a + b);
    }
    public static void main(String[] args) {
        // 3, 4 arguments
        int addition = sum(3, 4);
        int subtration = sub(4, 2);
        System.out.println("Sum of two number is : " + addition);
        System.out.println("Sub of two number is : " + subtration);
    }
}
```

**Method overloading**

Same with different signature is called method overloading, can be declared at multiple times with different arguments

> FAQ in interview by this topic
>
> What is Method overloading?
>
> Why do we need?

mean one method and definition.



```java
class Main {
    static int sum(int n1, int n2) {
        return (n1 + n2);
    }

    static double sum(double n1, double n2) {
        return (n1 + n2);
    }
```

```
    static float sum(float n1, float n2) {
        return n1 + n2;
    }

    public static void main(String[] args) {
        int addInt = sum(3, 4);
        double addDoub = sum(3.4, 5.6);
        float addFloat = sum(4.5f, 3.2f);
        System.out.println("Add1 : " + addInt);
        System.out.println("Add2 : " + addDoub);
        System.out.println("Add3 : " + addFloat);
    }
}
```

```
/*
Add1 : 7
Add2 : 9.0
Add3 : 7.7
*/
```



# Array

FAQ in interview by this topic

What is an Array?

How to declare and initiaize an array?

Array is a variable, which stores multiple values of similar data types of data on **contiguous fashion (manner).**

**Decleration of array**

```
data_type[] ArrayName = new data_type[size];

data_type ArrayName[size] = new data_type[size];

int[] marks = new int[3];
```

**Initialization of Array**

```
ArrayName[0] = value;
ArrayName[1] = value;
```

```
ArrayName[2] = value;
ArrayName[3] = value;
.
.
.
ArrayName[N] = value;


// declaration
int[] arr = new int[3];

// initialization
arr[0] = 34;

arr[1] = 45;
arr[2] = 90;
```

## 2D-Array

2d arrays is like a matrix, 2d arrays have row and columns, indexing starts with zero. Horizontal line is called row and vertical line called colunm

```
Declaration
Syntax:
data_type arr[][] = new data_type[row][column];
```

```
int arr[][] = new int[2][2];
```

## String

String is a non-primitive data type used to store characters. **Strings are immutable (unchangable) in** sequence of **java.**

> FAQ in interview by this topic
>
> What is String?
>
> Difference between 'A' and "A".

**String declaration**

```
Syntax:
String varName = “values”;

// String declaration
String name = "Heera Singh Lodhi";
// display string
System.out.println(name);
```

```
// input a string
Scanner sc = new Scanner(System.in);
String name = sc.next(); // for single word
// take a sentence input
// sc.nextLine()
System.out.println("Your name is: "+name);

String firstName = "Heera";
String lastName = "Singh";
String fullName = firstName +" "+ lastName;
System.out.println(fullName); -> Heera Singh
```

String length – string.length() -> return the length of string

For example

```
/*
charAt()
Returns the char value in this sequence at the specified index. The first char
value is at index 0, the next at index 1, and so on, as in array indexing.

The index argument must be greater than or equal to 0, and less than the length
of this sequence.
```

```
If the char value specified by the index is a surrogate, the surrogate value is
returned.

Parameters:

index the index of the desired char value.
Returns:

the char value at the specified index.
Throws:

IndexOutOfBoundsException - if index is negative or greater than or equal to
length(). */

*/

System.out.println(fullName.length()); //  -> 11


// accessing string character by character

for (int i=0; i<fullName.length(); i++) {
    System.out.println(fullName.charAt(i));
}

// OUTPUT
H
e
e
r
a

S
i
n
g
h
```

**String comparison / String equality**

**Syntax:**

str1.compareTo(str2)

```
String s1 = "Heera";
String s2 = "Heera";

// condition of compareTo funtion
// if s1 > s2 -> any +ve value
// if s1 < s2 -> any -ve value
// if s1 == s2 -> zero/0

if (s1.compareTo(s2) == 0) {
    System.out.println("String are equal");
} else {
    System.out.println("String are not equal");
}
// OUTPUT
// Strings are equal

// ------------------------------------------------------------------------

class Main {
    public static void main(String[] args) {
        String s1 = new String("AB");
        String s2 = new String("AB");
        String s3 = s1;
        /*
        * to test weather two string are equal, use the equals() method.
        * The expression str1.equals(str2)
        * return true if strings str1 and str2 are equal, false otherwise.
        * Note that str1 and str2 can be string variables or string literals
        *   "AB".equals(s1);
        * is perfectly legal.To test whether two strings are identical except for
the
        * upper/lowercase letter distinction, use the equalsIgnoreCase method.
        * "AB".equalsIgnorCase(s1);
        *
```

```
        * Do not use the == operator to test whether two strings are equal! It
only
        * determines whether or not the strings are stored in the same location.
Sure, if
        * strings are in the same location, they must be equal. But it is en-
tirely possible
        * to store multiple copies of identical strings in different places.
        *
        * */

        if (s1 == s2) {
            System.out.println("Same");
        } else {
            System.out.println("Not same");
        }

        if (s1.equals(s2)) {
            System.out.println("Same");
        } else {
            System.out.println("Not same");
        }

        if (s1 == s3) {
            System.out.println("Same");
        } else {
            System.out.println("Not same");
        }

        if (s1.equals("AB")) {
            System.out.println("Same");
        } else {
            System.out.println("Not same");
        }

        if ("AB" == s1) {
            System.out.println("Same");
        } else {
            System.out.println("Not same");
        }
```

```
    }
}


// OUTPUT
// Not same
// same
// same
// same
// Not same
```

C programmers never use == to compare strings but use strcmp instead. The Java

```
class Main {
    public static void main(String args[]) {
        String s1 = "Heera";
        if (s1.compareTo("Heera") == 0) {
            System.out.println("Strings are equal");
        } else {
            System.out.println("Strings are not equal");
        }
    }
}

// OUTPUT
// Strings are equal
```

but it seems clearer to use equals instead

***NOTE - what is difference between == and equals()***

*== is relational operator, it check reference of an obj*
*check the content of an object.*

FAQ in interview by this topic

What is difference between == and .equals comparison in string?

**Empty and null strings**

The empty string “” is a string of length 0. You can test whether a string is empty by calling

```
if (str.length() == 0)
```

or

```
if (str.equals(“”))
```

An empty string is a Java object which holds the string length (namely, 0) and an empty contents. However, a String variable can also hold a special value, called null, that indicates that no object is currently associated with the variableTo test whether a string is null, use

```
if (str == null)
```

Sometimes, you need to test that a string is neither null nor empty. Then use

```
if (str != null && str.length() != 0)
```

You need to test that str is not null first. it is an error to invoke a method on a null value.

**Substring or string slicing (in python)**

```
import java.util.*;

class Main {
    public static void main(String[] args) {
        String s1 = "Heera";
        String s2 = "Heera";
        // functioning of string.substring(bignning index, ending index)
       // ending index is exclude
        System.out.println(s1.substring(2, 5)); // same as s1[2:5] in python
        // OUTPUT -> era

        System.out.println(s1.substring(3));
        // OUTPUT -> ra
    }
```

```
}
```

**parseInt method of Integer class**

```
import java.util.*;

class Main {
    public static void main(String[] args) {
        String str = "123";
        int number = Integer.parseInt(str);
        System.out.println(number);
    }
}
```

**toString method of Integer class**

```
import java.util.*;

class Main {
    public static void main(String[] args) {
        int number = 123;
        String str = Integer.toString(number);
        System.out.println(str);
    }
}
```

# Task

- ➔ Given an email like this -> Heera222@gmail.com your task is to remove @gmail.com domain.
- ➔ Given a string for example 'Heera', your task is to replace 'e' with 'I'.

```
import java.util.*;

class Main {
    public static void main(String[] args) {
```

```
        String email = "Heera222@gmail.com";
        System.out.println(email.substring(0, email.indexOf('@')));

        String name = "Heera";
        System.out.println(name.replace('e', 'i'));
    }
}

// OUTPUT
// Heera222
// Hiira
```

**Q5 – Subsequence of a string.**

```
class Main {
    static void sequenceDis(String str, String newStr, int start, int count) {

        if (start == count) {
            System.out.println(newStr);
            return;
        }
        char currChar = str.charAt(start);
        sequenceDis(str, newStr+currChar, start+1, count);
        sequenceDis(str, newStr, start+1, count);

    }
    public static void main(String[] args) {
        sequenceDis("abc", "", 0, 3);
        return;
    }
}
```

# Strings builder

StringBuilder is a class that is used to manipulate and create strings. It is similar to the String class, but with one key difference: String objects are immutable, meaning that once you create a String object, you cannot change its value. StringBuilder objects, on the other hand, are mutable, meaning that you can modify their contents after they are created.

StringBuilder provides a set of methods for adding, inserting, and deleting characters in a string. These methods include append(), insert(), delete(), and replace(), among others.

```
import java.util.*;

class Main {
    public static void main(String[] args) {
        // it's time consuming
        String firstName = "Heera";
        String lastName = "Singh";
        String fullName = firstName + lastName;
        System.out.println(fullName);

        // it's better idea
        StringBuilder fname = new StringBuilder("Heera");
        StringBuilder lname = new StringBuilder("Singh");
        StringBuilder fuName = fname.append(lname);
        System.out.println(fname);
        System.out.println(fuName.charAt(0));
        System.out.println(fuName.length());
    }
}

// OUTPUT
// HeeraSingh
// HeeraSingh
// H
// 10
```

**String reverse**

```
import java.util.*;

class main {
```

```
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("hello");

        for(int i=0; i<sb.length()/2; i++) {
            int front = i;
            int back = sb.length() - 1 - i;

            char frontChar = sb.charAt(front);
            char backChar = sb.charAt(back);

            sb.setCharAt(front, backChar);
            sb.setCharAt(back, frontChar);
        }

        System.out.println(sb); // -> olleh
        StringBuilder str = new StringBuilder("Heera");
        // System.out.println(str.reverse());
        // same as
        str.reverse();
        System.out.println(str);// -> areeH
    }
}
```



## String v/s String builder/Buffer

| String | StringBuffer/StringBuilder |
|---|---|
| String is less efficient than StringBuffer. | StringBuffer/StringBuilder is efficient than string. |
| String is Immutable sequence of characters. | StringBuffer/StringBuilder is mutable sequence characters. |
| In string take more time on operations than String Builder. | StringBuffer/StringBuilder take less time than String. |
| String class overridden equals () method. | StringBuffer/StringBuilder class use Object class equals () method. |

| | |
|---|---|
| Thread is safe. | Thread is not safe. |
| Nature of string object is read only. | Readable and writable. |
| | |

## String builder v/s StringBuffer

| StringBuilder | StringBuffer |
|---|---|
| String is efficient than StringBuffer. | StringBuffer is less efficient than StringBuilder. |
| | |



## Object Oriented Programming

## Introduction

## Class

Class is a logical entity contain the information of their members like member method, static method, member variable, static variable, access modifier, constructor, constant.

Every file creates separate .class file. If single file has multiple class, so multiple .class file created separately.

- Every class directly or indirectly inherited from Object class.

```
Syntax:
class ClassName {
    // data member
}
```

Implementation of Student class

```java
/*
 * Author: Heera Singh Lodhi
 * Date: 19/07/2022
 * Program: Demonstration of student class
 *
 */
class Student {
    public int rollNum;
    public String name;
    public String course;
    public int m1,m2,m3;

    public int total() {
        return m1+m2+m3;
    }

    public double average() {
        return (m1+m2+m3)/3;
    }

    public void showInfo() {
        System.out.println("Student name is: "+name);
        System.out.println("Student roll number is: "+rollNum);
        System.out.println("Student course: "+course);
        System.out.println("Student total marks: "+total());
        System.out.println("Student average marks: "+average());
    }
}

class Main {
    public static void main(String args[]) {
        Student s1 = new Student();
        s1.rollNum = 01;
        s1.name = "Heera Singh Lodhi";
        s1.course = "Btech";
        s1.m1 = 88;
        s1.m2 = 85;
        s1.m3 = 90;
```

```
        s1.showInfo();
    }
}

// OUTPUT
// Student name is: Heera Singh Lodhi
// Student roll number is: 1
// Student course: Btech
// Student total marks: 263
// Student average marks: 87.0
```



```
import java.util.*;

class Pen {
    String color;
    String type;

    public void write() {
        System.out.println("Write someting");
    }

    public void printColor() {
        System.out.println(this.color);
    }
}

class Student {
    String name;
    int age;

    public void printInfo() {
        System.out.println(this.name);
        System.out.println(this.age);
```

```
    }

}

class Main {
    public static void main(String[] args) {
        Pen p1 = new Pen();
        p1.color = "blue";
        p1.type = "gel";
        p1.write();

        Student s1 = new Student();
        s1.name = "Heera Singh Lodhi";
        s1.age = 20;
        s1.printInfo();
    }
}
```

## Abstract Class

Abstract mean – Partial (adhoora), not completed.

Abstraction can be achieved with either abstract classes or interfaces.

The abstract keyword is a non-access modifier, used for classes and methods:

- Abstract class
- Abstract method

**Abstract class:** is a restricted class that cannot to be create objects (to access it, it must be inherited from another class, inherited class must be abstract class).

**Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

If we declare method of class as abstract, it is mandatory to declare class as abstract.

- Can't declare object of abstract class.
- Abstract class contain regular methods and normal methods.
- Abstract class contain normal methods with definition.

- Normal class/concreate can't override abstract class methods. It must be declared as abstract.
- In abstract class normal methods can be declare as static, it is possible.
- In abstract class can't declare methods as static abstract.

**An abstract class can have both abstract and regular methods:**

```
// abstract class
abstract class AbstractClass {
    // abstract method
    abstract public void s();

    // regular method
    public void t() {
        System.out.println("HI");
    }
}

class InheritedClass extends AbstractClass {
    public void s() {
        System.out.println("Hello");
    }
}


class Main {
    public static void main(String[] args) {
        InheritedClass obj = new InheritedClass();
        obj.t();
        obj.s();
    }
}
```

```
// yadi inherited class me kewal ek abstact method (abstract base class) ko de-
fine hai to ko abstract nahi banana padega, yadi ek se adhik method (abstract
base class) ko define karna hai to inherited class ko bhi abstract class banana
```

```
hoga or in method ko use karne inherited class se ki ek or class inherited karna
hoga or eska object bana ke in sabhi methods ko use kar payenge.

// abstract class
abstract class AbstractClass {
    // abstract method
    abstract public void s();
    abstract public void h();

    // regular method
    public void t() {
        System.out.println("HI");
    }
}

// this class inherited from Super class (Abstract class)
abstract class InherAbstractClass extends AbstractClass {
    public void s() {
        System.out.println("Hello");
    }
    public void h() {
        System.out.println("Heera");
    }
}

class InheritedClass extends InherAbstractClass { }

class Main {
    public static void main(String[] args) {
        InheritedClass c1 = new InheritedClass();
        c1.s();
        c1.t();
        c1.h();
    }
}
```

Why And When To Use Abstract Classes and Methods?

To achieve security - hide certain details and only show the important details of an object.

```
abstract class Super {
    public Super() {
        System.out.println("Super constructor");
    }

    public void meth1() {
        System.out.println("Meth1 of super");
    }
    abstract public void meth2();
}

class Sub extends Super {
    public void meth2() {
        System.out.println("Sub meth2");
    }
}

class Main {
    public static void main(String[] args) {
        Super su = new Sub();
        su.meth1();
        su.meth2();
    }
}
/*
Super constructor
Meth1 of super
Sub meth2
 */
```

**Rules of abstract class (do's and don't of abstract class)**

- Reference of super class but can't create an object of super class.
- If one method has abstract of a class. If must be declare as abstract class.

- Abstract can't make as final.
- Abstract method can't as final.
- Abstract class don't declare as static same as abstract method.

## Constructor

Constructor is a special type of method which never return a value, construct is used to initialize member of a class.

- Default constructor
- Parameterized constructor
- Copy constructor – copy object to another object.

```
import java.util.*;

class Student {
    String name;
    int age;


    // default constructor
    Student() {
        System.out.println("Default constructor called");
    }

    // parameterized constructor
    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

class Main {
    public static void main(String args[]) {
```

```
        Student s1 = new Student("Heera Singh", 20);

    }
}
```

Exmple of copy constructor

```
import java.util.*;

class Student {
    String name;
    int age;
    Student() {

    }

    public void printInfo() {
        System.out.println("Student name is : "+ this.name);
        System.out.println("Student age is : " + this.age);
    }
    // Copy constructor
    Student(Student s2) {
        this.name = s2.name;
        this.age = s2.age;
    }
}

class Main {
    public static void main(String args[]) {
        Student s1 = new Student();

        s1.name = "Heera Singh";
        s1.age = 20;

        Student s2 = new Student(s1);
        s2.printInfo();
    }
```

```
}
```

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

    public void printInfo(String name) {
        System.out.println(name);
    }

    public void printInfo(int age) {
        System.out.println(age);
    }

    public void printInfo(String name, int age) {
        System.out.println(name + " " + age);
    }
}

class Main {
    public static void main(String args[]) {
        Student s1 = new Student();

        s1.name = "Heera Singh";
        s1.age = 20;

        s1.printInfo(s1.age);
        s1.printInfo(s1.name);
        s1.printInfo(s1.name, s1.age);
    }
}
```

## Constructor Chaining

Constructor chaining is the process of calling one constructor from another constructor with respect to current object.

One of the main uses of constructor chaining is to avoid duplicate codes while having multiple constructor (by means of constructor overloading) and make code more readable.

- **Within same class:** It can be done using this () keyword for constructors in the same class.
- **From base class:** by using super() keyword to call the constructor from the base class.

```
class Super {
    Super() {
        System.out.println("HI X");
    }

    Super(int x) {
        System.out.println("HI parameter of X");
    }
}

class Sub extends Super {
    Sub() {
        System.out.println("HI Y");
    }

    Sub(int y) {
        this(); // this is a keyword reference to the current object
        System.out.println("HI parameter of Y");
    }
}

class Main {
    public static void main(String[] args) {
        Sub y1 = new Sub(1);
    }
```

```
}
// HI X
// HI Y
// HI parameter of Y


class Sub extends Super {
    Sub() {
        System.out.println("HI Y");
    }

    Sub(int y) {
        super(y); // super is keyword reference to super class
        System.out.println("HI parameter of Y");
    }
}

class Main {
    public static void main(String[] args) {
        Sub y1 = new Sub(1);
    }
}

// HI parameter of X
// HI parameter of Y
```

# Polymorphism

One name different action. Poly -> many, Morphism -> forms => many forms.

In java polymorphism achieve through method overloading as well as method orderriding.

**Method overloading**

A class have two methods same with different signature is called method overloading, compiler has to be decided at compile time which method should we called, also know as compile time polymorphism. Method overloading achieve in single class but method overriding achieve in must have two classes super class and sub class.

```
/*
 * Author: Heera Singh Lodhi
 * Date: 19/07/2022
 * Program: Method overloading
 *
 */

class Test {
    public int max(int a, int b) {
        return a>b?a:b;
    }

    public int max(int a, int b, int c) {
        if(a>b && a > c)
            return a;
        else if(b > c)
            return b;
        return c;
    }
}

class Main {
    public static void main(String[] args) {
        Test t1 = new Test();
        System.out.println(t1.max(2, 5));
        System.out.println(t1.max(4,5,6));
    }
}
```

**Method overriding**

Redefining a method of super class inside class is called method overriding. Also called as run-time polymorphism.

```
/*
 * Author: Heera Singh Lodhi
 * Date: 19/07/2022
 * Program: Method Overriding
 */

class Super {
    public void display() {
        System.out.println("Super display");
    }
}

class Sub extends Super {
    public void display() {
        System.out.println("Sub display");
    }
}

class Main {
    public static void main(String[] args) {
        Super su = new Super();
        su.display(); // -> super class display called

        Super su1 = new Sub();
        su1.display(); // -> sub class display called
    }
}
```

Here two class say Super and Sub, both have method name with same signature, compiler decided which method should we call at run-time is called run-time polymorphism.

## Method overloading v/s Method overriding.

| S. No. | Overloading | Overriding |
|---|---|---|

| | | |
|---|---|---|
| 1. | Method overloading is a compile-time polymorphism. | Method overriding is a run-time polymorphism. |
| 2. | We achieve method overloading in same class. | We achieve overriding in two different classes with inheritance relationships. |
| 3. | Method overloading may or may not require inheritance. | Method overriding always need inheritance. |
| 4. | Method must have same name with different signature. | Method must have same with same signature. |
| 5. | In method overloading, the return type can or can not be the same, but we just have to change the parameter. | In method overriding, the return type must be the same or co-variant. |
| 6. | Private and final methods can be overloaded. | Private and final methods can't be overridden. |
| 7. | Argument list should be different while doing method overloading. | Argument list should be same in method overriding. |
| 8. | | |

# Inheritance



# Package

Packages contain encapsulation mechanism that grouped classes and interfaces into a single unit. Or Package is a collection of classes, methods, and interfaces.

Types of packages

- In-built packages - java.lang.*, java.util.*
- User-defined packages

Difference between user defined package and built-in package?

We declare user defined packages in program's top the lines.

Why we use package in java?

Name conflict and Security of class, so we use package in java.

# Interface

Interface called as abstract class with all abstract methods. Interfaces are useful for achieving run-time polymorphism.

Interface contain all methods definition partially, but abstract class may or may not.

Abstract class has extended and interfaces are implements.

- Interfaces can be extending.
- By default interface contain all methods abstract and public.
- Interface can't declare methods as static and final, but static block and final variable is also used.
- We can't declare interface methods are protected and private.
- We can't create an object of interface.

***Why we design interface?***

- *We design interface when all requirements are partially defined.*



```
interface Test {
    public void meth1();
    public void meth2();
}

class My implements Test {
    public void meth1() {
        System.out.println("Meth1 of class My");
```

```
    }
    public void meth2() {
        System.out.println("Meth2 of class My");
    }
}


class Main {
    public static void main(String[] args) {
        Test t = new My();
        t.meth1();
        t.meth2();
    }
}
```

Example

```

class Phone {
    public void call() {
        System.out.println("Phone call");
    }

    public void sms() {
        System.out.println("Phone sending sms");
    }
}

interface Camera {
    public void click();
    public void record();
}

interface MusicPlayer {
    public void play();
    public void pause();
    public void resume();
    public void stop();
}
```

```
abstract class SmartPhone extends Phone implements Camera, MusicPlayer {
    public void videoCall() {
        System.out.println("Smart phone video calling");
    }

    public void click() {
        System.out.println("Smart phone clicking photo");
    }

    public void record() {
        System.out.println("Smart phone recording video");
    }

    public void play() {
        System.out.println("Smart phone playing music");
    }

    public void pause() {
        System.out.println("Smart phone pause music");
    }

    public void resume() {
        System.out.println("Smart phone resume music");
    }

    public void stop() {
        System.out.println("Smart phone resume music");
    }
}

class Xiaomi extends SmartPhone {
    public void info() {
        System.out.println("Xiaomi Smart phone");
    }
}

class Main {
    public static void main(String[] args) {
```

```
        Xiaomi s = new Xiaomi();
        s.call();
        s.play();
        s.pause();
        s.stop();
        s.info();
    }
}

/*
 *
 * Phone call
Smart phone playing music
Smart phone pause music
Smart phone resume music
Xiaomi Smart phone
 */
```

#Challange

```
interface Member {
    void callBack();
}

class Customer implements Member {
    String name;
    @Override
    public void callBack() {
        System.out.println("Ok, I will visit "+name);
    }
    Customer(String n) {
        name = n;
    }
}

class Store {
    Member mem[] = new Member[20];
```

```
    int count=0;

    void register(Member m) {
        mem[count++] = m;
    }

    void inviteSale() {
        for(int i=0; i<count; i++) {
            mem[i].callBack();
        }
    }
}

class Main {
    public static void main(String args[]) {
        Store s = new Store();
        Customer c1 = new Customer("John");
        Customer c2 = new Customer("Heera");

        s.register(c1);
        s.register(c2);
        s.inviteSale();
    }

}
```

## Interface v/s Abstract class

If we declare class as abstract is not compulsory to declare all methods as abstract (abstract class contain both regular method and normal method). But interface is a collection of abstract methods.

| SNo. | Interface | Abstract class |
|---|---|---|
| 1. | All methods are defined partially. | In abs class methods are partially may defined or may not. |
| 2. | Interface implement using 'implements' keyword. | Abs class extend using 'extends' keyword. |
| 3. | Interface has only static and final variables. | Abs class can have final, non-final, static, and non-static variables. |

| | | |
|---|---|---|
| 4. | Members of a Java interface are public by default. | A java abstract class can have class members like private, protected, etc. |
| 5. | In interface can't declare constructors and destructors. | An abs class can declare constructors and destructors. |

# Interface v/s multiple inheritance

# Inner class

**Types of inner class**

- **Nested inner class**
- **Local inner class**
- **Anonymous inner class**
- **Static inner class**



**Nested inner class**

Inner class can access member of outer class directly without creating an object of outer class, but outer can access member of inner class creating an object of inner class. So, we can use inner class member.

Every file creates separate .class file. If single file has multiple classes, so multiple .class file created separetly.

Outer class -> Outer.class

Inner class -> Outer$Inner.class

**Implementation**

```
class Outer {
    int x = 10;

    class Inner {
        int y = 20;
```

```
        void innerDisplay() {
            System.out.println(x);
            System.out.println(y);
        }
    }

    void outerDisplay() {
        Inner i = new Inner();
        i.innerDisplay();
        System.out.println(i.y);
    }
}

class Test {
    public static void main(String[] args) {
        Outer o = new Outer();
        o.outerDisplay();

        // creating object of inner class
        Outer.Inner i = new Outer().new Inner();
        i.innerDisplay();
    }
}
```

**Local inner class**

When you want to write a class that is inheriting some existing class or you want to writing a class implementing an interface.

```
// local inner class implementation

class Outer {
    void display() {
        class Inner {
            void innerDisplay() {
                System.out.println("Hello");
            }
        }
```

```
        Inner i = new Inner();
        i.innerDisplay();

       // same as
       // new Inner().innerDisplay();
    }
}

class Main {
    public static void main(String[] args) {
        Outer o = new Outer();
        o.display();
    }
}
```

**Anonymous class**

An anonymous class can define at time of creation of object itself. You define the class as well as create an object. Usually these are usefull for abstract class and interfaces.

```
abstract class My {
    abstract void display();
}

// same as
// interface My {
//     void display();
// }

class Outer {
    public void meth() {
        My m = new My() {
            // overridden method
            public void display() {
                System.out.println("Hello");
            }
        };
```

```
        m.display();
    }
}
```

Why we need of anonymous class?

If you have to implementing an interface and its uses limited you don't have to write separate class and their same place you can implement the interface as an anonymous class.

**Static inner class**

Static inner classes are the member of outer classes, the object of outer classes can be created outside the outer class (using outer class name), The object of static inner class can be created anywhere.

- Static inner class can access the member of outer class directly, only static member of that class not all member.

```
class Outer {
    static int x = 20;
    int y = 20;

    static class Inner {
        void display() {
            System.out.println(x);

            // this will show error
            // System.out.println(y);
        }
    }
}

class Main {
    public static void  main(String[] args) {
        Outer.Inner i = new Outer.Inner();
        i.display();
    }
```

```
}
```

# Static

# Final

Final is a keyword like const in c/c++. In java final used in many ways.

- Final variable
- Final method
- Final class

**Final variable**

Once value assign than after can't change their value. All constant variable name in UPPERCASE this is a convention not a rule.

```
final int MAX = 1;
```

**Final Method**

Final method can't be overridden.

```
class Final {
    final void meth1(){
        System.out.println("HI");
    }
}

class SubFinal extends Final {
    // this will throw an error
    void meth1() {
        System.out.println("Hello");
    }
}
```

**Final Class**

Final class does not extendded or inherited.

```
final class Super {
    .
    .
    .
}

// CTE
class Sub extends Super {
    .
    .
    .
}
```

## Singleton class

A class which creates just a single object more than one object of class are not allowed.

```
class CoffeeMachine {
    private float coffeeQty;
    private float milkQty;
    private float waterQty;
    private float sugarQty;

    static private CoffeeMachine my = null;

    private CoffeeMachine() {
        coffeeQty=1;
        milkQty=1;
        waterQty=1;
        sugarQty=1;
    }

    public void fillWater(float qty) {
        waterQty=qty;
```

```
    }

    public void fillSugar(float qty) {
        sugarQty = qty;
    }

    public float getCoffee() {
        return 0.15f;
    }

    static CoffeeMachine getInstance() {
        if(my == null) {
            my = new CoffeeMachine();
        }
        return my;
    }
}

class Main {
    public static void main(String[] args) {
        CoffeeMachine c1 = CoffeeMachine.getInstance();
        CoffeeMachine c2 = CoffeeMachine.getInstance();
        CoffeeMachine c3 = CoffeeMachine.getInstance();

        System.out.println(c1+" "+c2+" "+c3);
        if(c1 == c2 && c1 == c3) {
            System.out.println("Same");
        }
    }
}
```

#Challange

```
import java.util.Date;

class Student {
    private String rollNo;
```

```
    private static int count = 1;
    private String assignRollNo() {
        Date d = new Date();
        String rno = "Univ-"+(d.getYear()+1900) + "-"+count;
        count++;
        return rno;
    }

    Student() {
        rollNo = assignRollNo();
    }

    public String getRollNo() {
        return rollNo;
    }
}

class Main {
    public static void main(String[] args) {
        Student s1 = new Student();
        Student s2 = new Student();
        Student s3 = new Student();

        System.out.println(s1.getRollNo());
        System.out.println(s2.getRollNo());
        System.out.println(s3.getRollNo());
    }
}
```

# Exception handling

Exceptions are run-time errors. Errors are three types:

a. Syntax errors

b. Logical errors

c. Run-time errors

**Syntax errors –**

While writing a program if programmer doing grammatical mistakes that is syntactical mistake some mistake, spelling mistake, that errors causes is called syntax error.

```
Int x, y;
x = 10
z = x+y;
```

**Logical errors –**

Program runs but doesn't give expected result.

To remove these types of error using tracing/debugger.

r = -b / 2a  =>  r = -b/(2*a)

Above two types of errors face by programmer. And run-time errors face by users

**Run-time errors –**

Cause error – invalid/bad input, unavailability of resources. Run-time errors handle through exception handling.

```
Syntax:

try {
    .
    .
    .
} catch (...){
    .
    .
    .
} catch(...) {
    .
    .
    .
} finally {
```

```
    .
    .
    .
}
```

**Example**

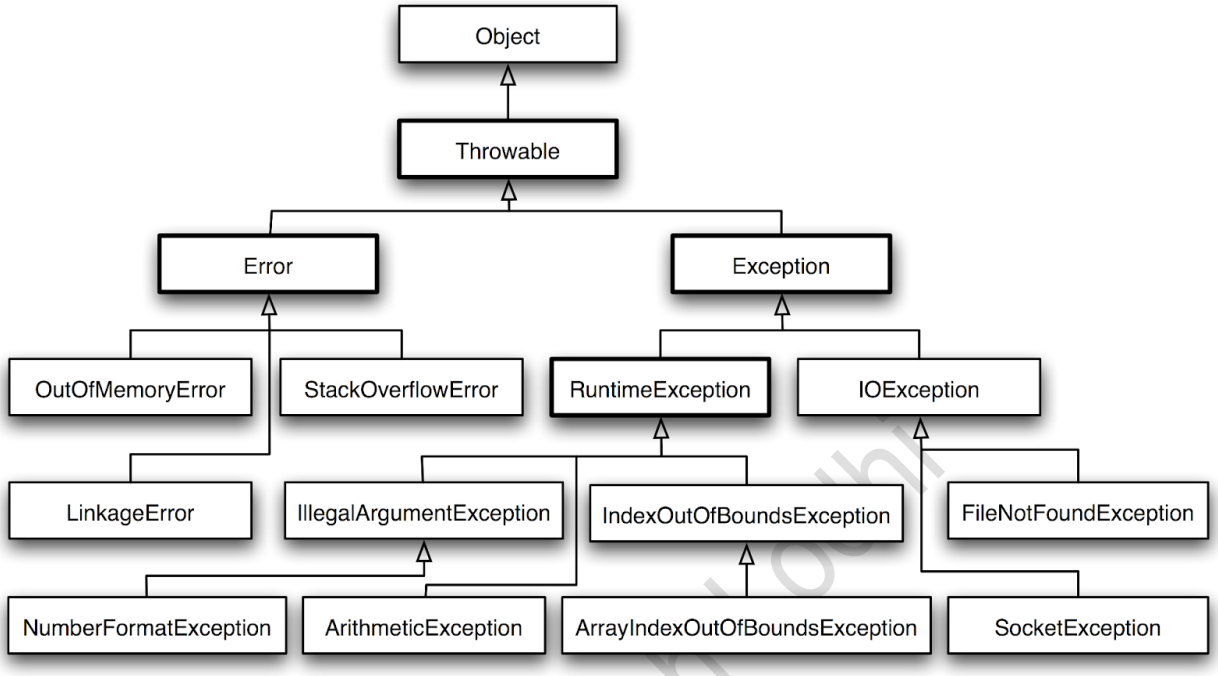
```
class Main {
    public static void main(String[] args) {
        int a, b, c;
        try {
            a = 10;
            b = 0;
            c = a/b;
            System.out.println("Result is : "+c);
        } catch (ArithmeticException e){
            System.out.println("Division error "+e);
        }
    }
}
```

Nested try and catch

**Exception class**

Every user define exception it must be extends from Exception class. We will be able to override Exception class methods like toString(), getMessage(), void printStackTrace().

```
class OwnException extends Exception {
    public String toString() {
        return "Exception";
    }
}
```




# Collection framework

Collection framework is a collection of class and interfaces. Collection framework provided by java developers. Example of collection frameworks are ArrrayList, Stack, Queue, etc.

Interable

^

|

Collection

^

|

List-Queue-Set

Common methods on Collections

- Add
- Size

- Remove
- Iterate
- AddAll
- Removeall
- Clear

**List interface**

- ArrayList
- Linked list
- Vector -> Stack

**Queue Interface**

- Priority queue
- Linked list
- Deque (double ended queue) -> Array Deque

**Set interface.**

Set is a group of objects. all objects are unique.

- HashSet (un-ordered collection of objects)
- LinkedHashSet (ordered collection of objects)
- SortedSet -> TreeSet (sorted collection of objects)

**Map interface**

- HashMap (un-ordered collection of key-value pair)
- LinkedHashMap (ordered collection of key-value pair)
- SortedMap -> TreeMap (sorted collection of key-value pair)
- HashTable (Same as hashmap, but hashtable does not allow null value)

# ArrayList

Array list is variable size list (dynamic array) stores number of objects.

Variable size means it can be expended and shrink in addition or removing to element in Arraylist.

- Variable size
- Non-contiguous
- Stores objects it is not stores primitive data types like int, float, char.

**Methods in ArrayList class**

- ⇨ add
- ⇨ get
- ⇨ set (modify)
- ⇨ remove/delete
- ⇨ iterate

import java.util.ArrayList;

```
import java.util.ArrayList;
import java.util.Collections;
class Main {
    public static void main(String[] args) {
        ArrayList <Integer> list1 = new ArrayList<>();

        // add element
        list1.add(1);
        list1.add(2);
        list1.add(3);
        list1.add(4);

        System.out.println(list1); // -> [1,2,3,4]

        // get element
        int element = list1.get(0);
        System.out.println(element); // -> 1
```

```
        // insert element at specified position
        list1.add(1, 5);
        System.out.println(list1); // [1, 5, 2, 3, 4]

        // set element/replace element with new one
        list1.set(0, 6);
        System.out.println(list1); // -> [6, 5, 4, 3, 4]

        // remove element
        list1.remove(0);
        System.out.println(list1); // -> [5, 2, 3, 4]

        // size of ArrayList
        int size = list1.size();
        System.out.println(size); // -> 4

        // iterate array list
        for(int i=0; i<size; i++) {
            System.out.print(list1.get(i)+" "); // -> 5 2 3 4
        }
        // new line
        System.out.println();

        // sort ArrayList
        Collections.sort(list1);
        for(int i=0; i<size; i++) {
            System.out.print(list1.get(i)+" "); // -> 5 2 3 4
        }
    }
}
```

**Collection framework**

```
import java.util.Stack;

public class Main {
```

```
    public static void main(String[] args) {
        Stack <Integer> s1 = new Stack<>();
        s1.push(4);
        s1.push(3);
        s1.push(2);
        s1.push(1);

        while ( ! s1.isEmpty()) {
            System.out.println(s1.peek());
            s1.pop();
        }
    }
}
```

## HashSet

A HashSet is a collection of items where every item is unique, and it is found in the java.util package.

- Set is unordered collection of items.

**HashSet is same as unordered set in c++.**

```
// It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order
// will remain constant over time. This class permits the null element.

import java.util.HashSet;
```

```
        // SET OF INTEGER VALUES
        HashSet<Integer> set = new HashSet<>();
        set.add(4);
        set.add(3);
        set.add(2);
```

```
        // DISPLAY
        System.out.println(set);
```

**HashSet has many useful built-in methods like-**

add() – add new item beginning of the set.

contains() – check existance of item in set.

remove(item) – remove specified item in the set.

clear() – remove all items.

size() – return size of set.

```
// ITERATOR
        // iterate all item of set
        Iterator it = set.iterator();

        // hashNext(), next()
        // next() -> Returns the next element in the iteration.

        it.next(); // --> return first item of set
        it.next(); // --> return second item of set


        it.hasNext(); // --> return true/false value, if iterator pointing
        // a particular, hasNext() check next element is it or not.

        // iterate set
        while(it.hasNext()) {
            System.out.println(it.next());
        }
```

# HashMap

# Thread And Concurrency

# Thread

A lightweight process is called a thread. Thread is a flow of execution to complete a job. Every program has its own thread, is called main thread. Main thread is controlled by JVM itself.

Ways to implement thread

Implementing runnable interface

Extending thread class

**Implementing Runnable Interface**

Runnable interface contains a run method which is necessary to define in derived class.



```
/**
 * author: Heera Singh Lodhi
 * demonstrating thread example implementing runnable interface
 */


class MyClass implements Runnable {


   /*
    * public abstract method of runnable interface
    */
   public void run() {
      for(int i=0; i<100; i++) {
```

```
            System.out.println("thread class");
        }
    }
}
public class Main {
    /**
     * @param args
     * written program doesn't give same result in same and another computer
     * becasuse of thread, main thread and user created thread running simutaneusly.
     *
     * newly created thread have its onw flow control and main have also separate flow
     * control
     */
    public static void main(String[] args) {
        MyClass myClass = new MyClass();



        Thread newThread = new Thread(myClass);



        newThread.start();
        for(int i=0; i<100; i++) {
            System.out.println("main class");
        }
    }
}
```

**Extending Thread Class**

```
/**
 * author: Heera Singh Lodhi
 * date: 16/06/2023
 * desc: thread demonstration
 */
```

```
class MyClass extends Thread {
    public void run() {
        for(int i=1; i<=100; i++) {
            System.out.println("my class");
        }
    }
}
public class Main {
    public static void main(String[] args) {
        MyClass myClass = new MyClass();
        myClass.start();;
```

```
    for(int i=1; i<=100; i++) {
       System.out.println("main class");
    }
  }
}
```

# Access modifier

Acccess modifiers are used for specifying the access level or visibility of members of a class.

**Types of modifiers**

There are four modifiers in java.

- Default
- Private
- Protected
- Public

```
class Demo {
    int x;

    void show() {...}

    class Inner {...}
}
```

- Outer class can't be private or protected.
- We can use any built-in or user defined class in two ways –

  1. Create an object, 2. Exntends

| | Default | Private | Protected | Public |
|---|---|---|---|---|
| **Same class** | Yes | Yes | Yes | Yes |
| **Same package sub-class** | Yes | No | Yes | Yes |
| **Same package non-sub-class** | Yes | No | Yes | Yes |
| **Different package sub-class** | No | No | Yes | Yes |
| **Different package non-sub-class** | No | No | No | Yes |

# File Handling

We use the Scanner class to read the contents of the text file we created.

File class provide many methods to manipulate files.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        try {
            File file = new File("input.txt");
            // create new file
            // file.createNewFile()

            // return absolute/full path of input.txt file
            // file.getAbsolutePath();
            Scanner sc = new Scanner(file);
```

```
            while (sc.hasNextLine()) {
                System.out.println(sc.nextLine());
            }
        } catch (FileNotFoundException e) {
            System.out.println("An error occured");
        }
    }
}
```

# Networking

**Working with URLs:**

Java provides the java.net.URL class to work with Uniform Resource Locators (URLs). Here's a simple example of how to use it:

**Working with sockets:**

Java provides the java.net.Socket and java.net.ServerSocket classes for working with sockets, allowing communication between clients and servers. Below is a simple example of a basic client and server

```
/**
 * @heera9331
 * @desc - networking demonstration using socket server
 * @date - 10-12-2023
 */

package Java.networking;

import java.net.ServerSocket;
import java.net.Socket;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Server {
```

```
    public static void main(String[] args) {
        try {
            // Create a server socket
            ServerSocket serverSocket = new ServerSocket(8080);

            System.out.println("Server is listening on port 8080...");

            // Wait for client connection
            Socket clientSocket = serverSocket.accept();

            // Read data from the client
            BufferedReader reader = new BufferedReader(new InputStreamReader(clien-
tSocket.getInputStream()));
            String clientMessage = reader.readLine();
            System.out.println("Received from client: " + clientMessage);

            // Close sockets
            reader.close();
            clientSocket.close();
            serverSocket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
/**
 * @heera9331
 * @desc - networking demonstration using socket server, creating client
 * @date - 10-12-2023
 */


package Java.networking;

import java.net.Socket;
import java.io.PrintWriter;

public class Client {
    public static void main(String[] args) {
        try {
            // Create a client socket
            Socket clientSocket = new Socket("localhost", 8080);
```

```
            // Send data to the server
            PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);
            writer.println("Hello from the client!");

            // Close the socket
            writer.close();
            clientSocket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

## Interview Questions and Answers

Q1. What is difference between final, finally and finalize?

**Final** is a keyword and access modifier in java, used to declare constant. Final class can't be inherited.

**Finally,** is the exception block, to execute the important code, whether the exception occur or not?

**Finalize** is the method which is used to perform clean up processing just before object is garwage collected.

Q2. What is ClassLoader in java?

ClassLoader to load .class file, classloader also knows class loader subsystem, situated in JVM.

Q3. What is the difference between stack and heap?

Stack is generally used to store the order of method execution and local variables. In contrast, Heap memory is used to store the objects. After storing, they use dynamic memory allocation and deallocation i.e differences between Heap and Stack Memory in Java?

Q4. What is an Association?

Q5. What is Aggregation?

### Q6. Define Copy Constructor in Java?

A Copy Constructor in Java is a constructor that initializes an object through another object of the same class.

### Q7. What is an object cloning?

An ability to recreate an object entirely like an existing object is known as Object Cloning in Java. Java provides a clone () method to clone a current object offering the same functionality as the original object.

### Q8. What is marker interface?

An empty interface in Java is referred to as a Marker interface. Serializable and Cloneable are some famous examples of Marker Interface.

### Q9. Why is Java not completely object-oriented?

Because java has primitive data so is not considered as pure/fully/complete object-oriented.

### Q10. Define Wrapper Classes in Java?

In Java, when you declare primitive datatypes, then Wrapper classes are responsible for converting them into objects (Reference types).

### Q11. Define Singleton Classes in Java?

In Java, when you make the constructor of a class private, that class can generate only one object. This type of class is popularly known as a Singleton Class.

### Q12. What are generics in Java?

Generics in Java provide a way to create classes, interfaces, and methods with type parameters.

- Generics were introduced in Java 5 to enhance the type system and enable the development of more robust and flexible code.

```
public class Box<T> {
    private T value;
```

```
    public void setValue(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
}
```

## Database concept

Database is a program which deals with storing and organizing the data in a permanent storage like hard disk.

Database is a collection of inter-related data permanently stored in the disk.

Data is organized in the form of tables. Tables may have relationship with other tables.

**Q1. What is difference between files and database?**

Row is also called tuple, record.

Structure of tables is called schema.

**Primary: -**

Primary is a column which contains unique values, such that you search using a primary key.

Primary key is unique and not null.



Back to Top