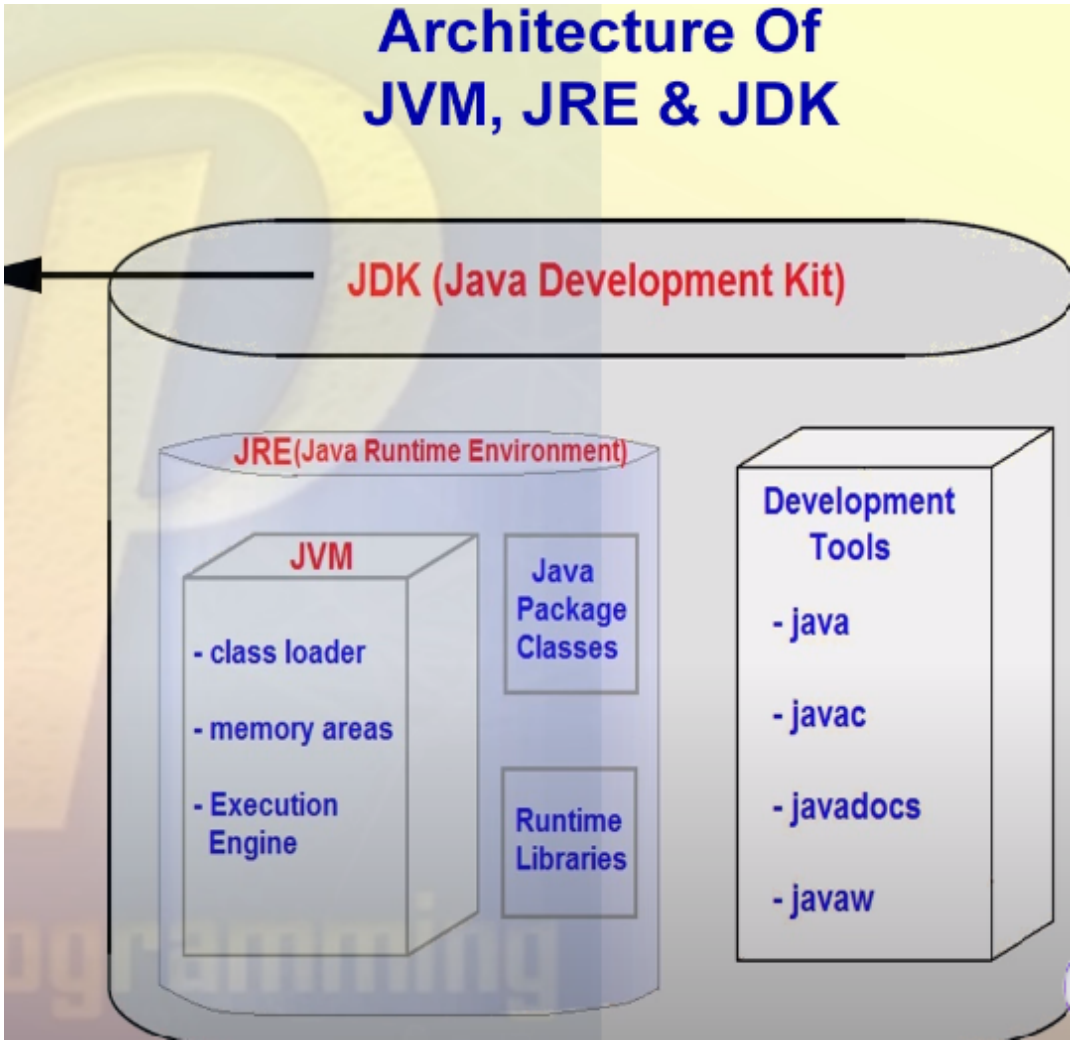
# Core JAVA + AWT+ JDBC+JSP Servlets + Hibernate + Spring + Java Web Services/ Restful API + Spring Boot

## JAVA DEFINITION

JAVA is a general purpose, statically typed, platform independent, object oriented, high level programming language designed for having lesser implementational dependencies, with the main goal of serving as a server side scripting language. It was developed by James Gosling at Sun Microsystems in 1995 and later acquired by Oracle Corporation. It is fast, reliable and secure and thus used in various fields like Web development, Android development,  desktop app development.

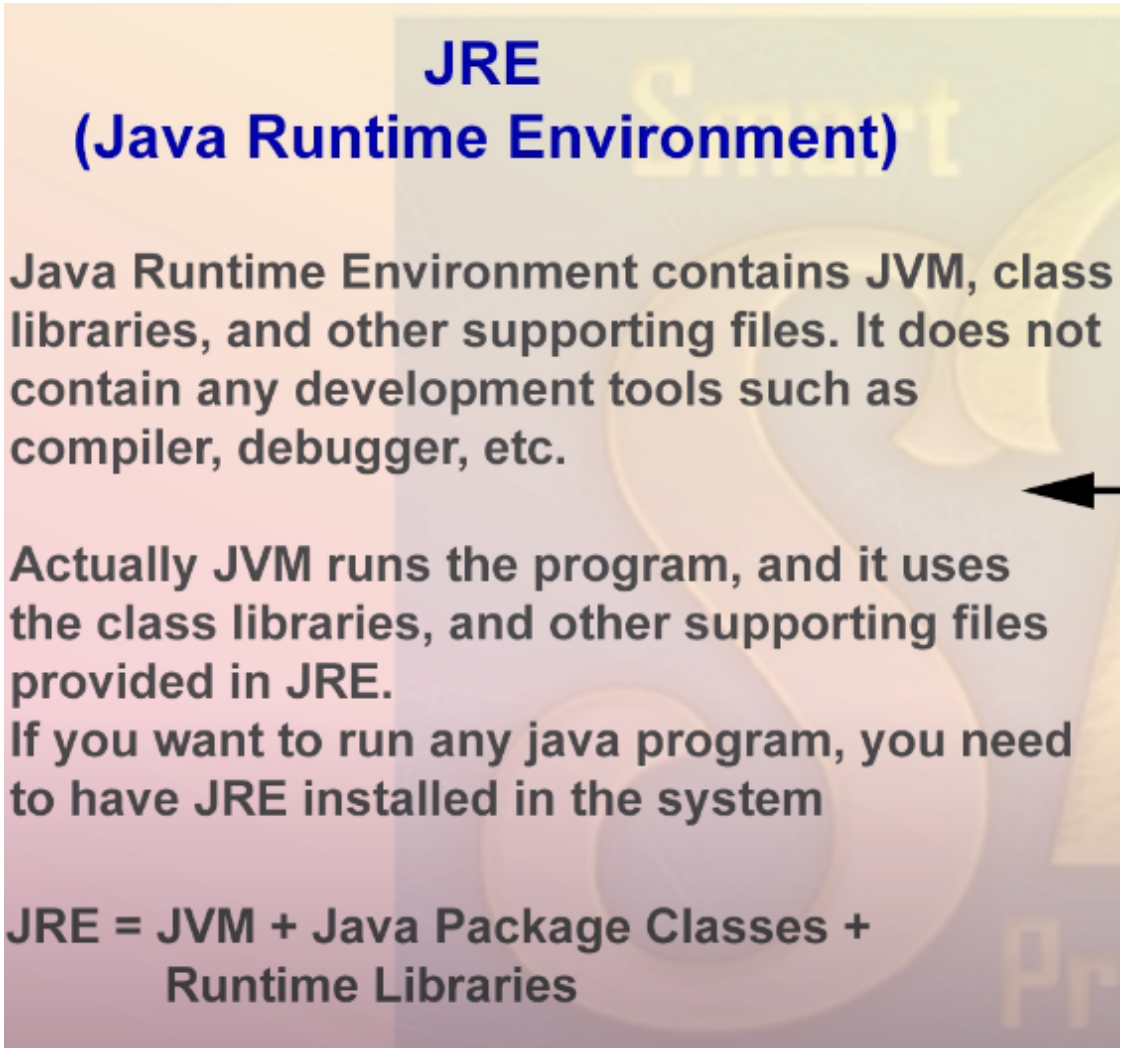
## ARCHITECTURE OF JDK


Architecture Of
JVM, JRE & JDK
JDK (Java Development Kit)
JRE(Java Runtime Environment)
JVM
- class loader
- memory areas
- Execution Engine
Java Package Classes
Runtime Libraries
Development Tools
- java
- javac
- javadocs
- javaw

## WHAT IS JDK ?


JDK
(Java Development Kit)
Java Developer Kit contains tools needed to develop the Java programs, and JRE to run the programs. The tools include compiler (javac.exe), Java application launcher (java.exe), Appletviewer, etc
JDK is mainly targeted for java development. i.e. You can create a Java file (with the help of Java packages), compile a Java file and run a java file.
JDK = JRE + Development Tools

## WHAT IS JRE ?

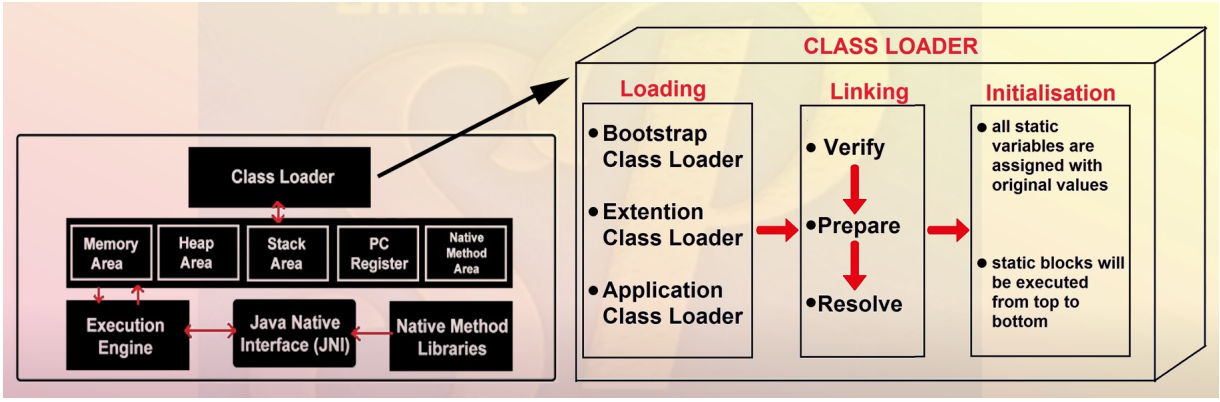

JRE
(Java Runtime Environment)
Java Runtime Environment contains JVM, class libraries, and other supporting files. It does not contain any development tools such as compiler, debugger, etc.
Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in JRE.
If you want to run any java program, you need to have JRE installed in the system
JRE = JVM + Java Package Classes + Runtime Libraries

## WHAT IS JVM ?

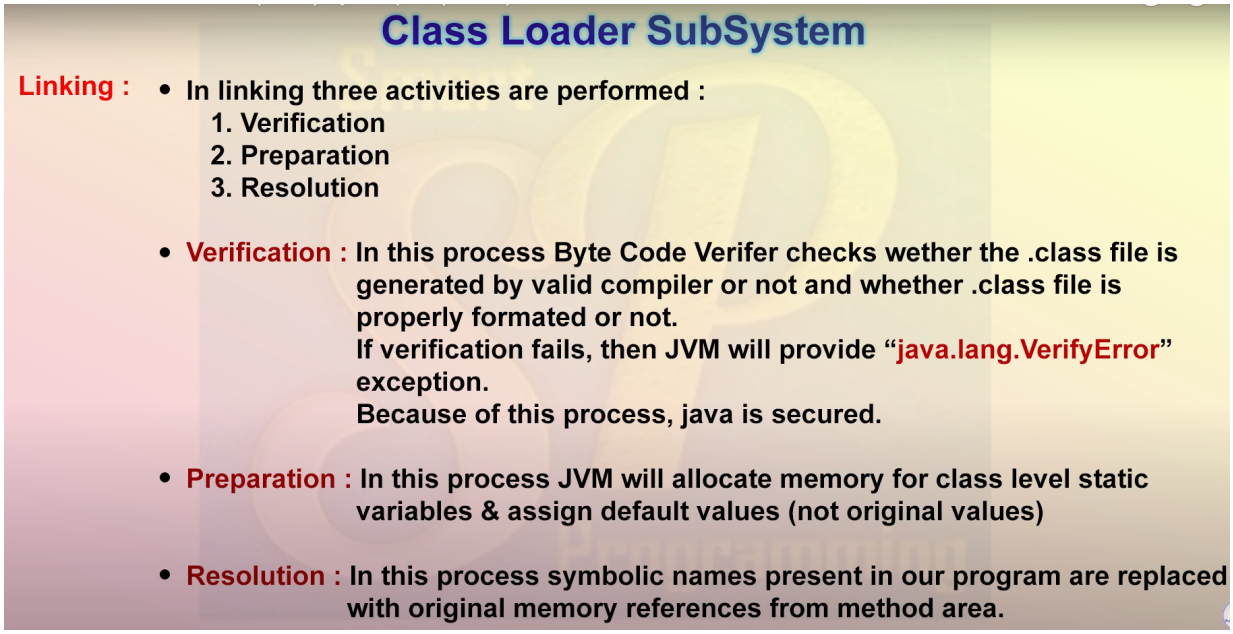

## ARCHITECTURE OF JVM

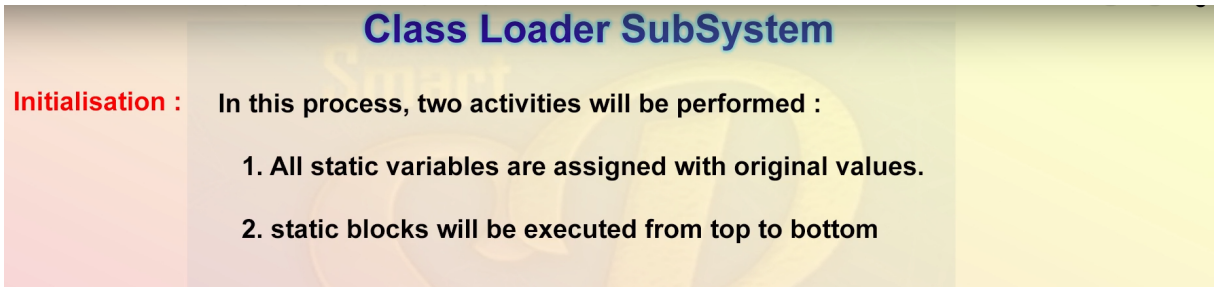
## Virtual Machine

- Virtual Machine is a software simulation of a machine which can perform operations similar to physical machine.
- Virtual Machine is not physically present.
- A virtual machine, usually known as a guest is created within another computing environment referred as a "host." Multiple virtual machines can exist within a single host at one time.




# CLASS LOADER




## MEMORY AREAS :

The different memory areas are:

1. Method Area (stores .class file and static variables)
2. Heap Area (stores objects, object variables, arrays )
3. Stack Area (currently running methods and local variables)
4. PC Register (hold the address of the next executing information)
5. Native Method Stacks (All the methods invoked by the thread are called in the native method stacks)

## Memory Areas

**1. Method Area :**

- Method area is created when JVM is started.
- It stores .class file information and static variables.
- Per JVM one memory area, therefore multiple threads can access this area, so it is not thread safe.

**2. Heap Area :**

- Heap area is created when JVM is started.
- It stores objects, instance variables and arrays (as every arrays is an object in java).
- It can be accessed by multiple threads, so the data stored in heap area is not thread safe.

**3. Stack Area :**

- Whenever a new thread is created, a separate stack area will also be created
- It stores the current running method and local variables.
- When the method is completed, the corresponding entry from the stack will be removed.
- After completing all method calls, the stack will become empty and that empty stack will be destroyed by thee JVM just before terminating the thread.
- The data stored in the stack is available only for the corresponding thread and not available to the remaining thread, so this area is thread safe.

**4. PC Register :**

- It holds the address of next executing instruction.
- For every thread, a separate pc register is created, so it is also thread safe.

**5. Native Method Stacks :**

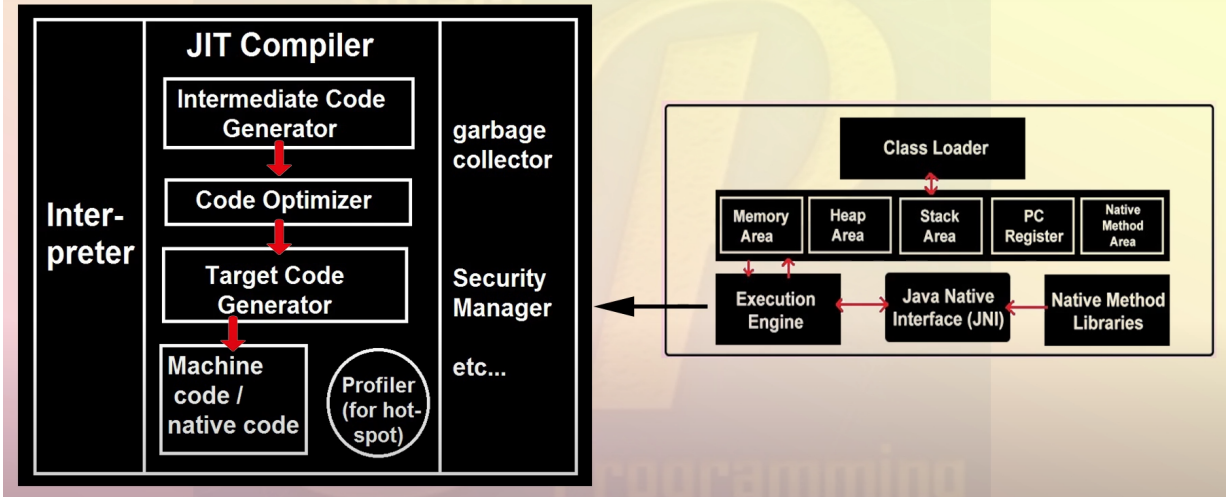
- All native method calls invoked by the thread will be stored in the corresponding native method stack.
- For every thread separate native method stack will be created
- It is also thread safe.

# EXECUTION ENGINE

## Execution Engine

- Execution Engine is responsible to execute java class file.
- It contains mainly two components :
  1. Interpreter
  2. JIT Compiler

**Interpreter :** A module that alternately decodes and executes every statement or line in some body of code. The Java interpreter decodes and executes bytecode for the Java virtual machine.


Execution Engine (central component of JVM)

## JIT Compiler

JIT Complier :

- JIT stands for Just-in-Time which means that code gets compiled when it is needed, not before runtime.
- The main purpose of JIT compiler is to improve performance.
- JVM maintains a count as of how many time a function is executed. If this count exceeds a predefined limit or say threshold value, the JIT compiles the code into machine language which can directly be executed by the processor (unlike the normal case in which javac compile the code into bytecode and then java - the interpreter interprets this bytecode line by line converts it into machine code and executes).
  Also next time this function is calculated same compiled code is executed again unlike normal interpretation in which the code is interpreted again line by line. This makes execution faster.
- JIT compilation is applicable only for repeatedly required methods, not for every method.

## JAVA NATIVE INTERFACE(JNI)

### Java Native Interface (JNI)

- An interface that allows Java to interact with code written in another language.
- It acts as mediator for java method calls & the corresponding native libraries i.e. JNI is responsible to provide information about native libraries to the JVM.
- Native Method Library provides or holds native library information.
- The java command-line utility is an example of one such application, that launches Java code in a Java Virtual Machine.

# CORE JAVA

## 1. Basic Questions

1. Why JAVA is not  100% object Oriented?
2. What are keywords in JAVA?
3. What are identifiers and literals in JAVA?

4. **WHAT IS OOP ?**
5. **WHAT IS A CLASS**
    1. Class is a set of object which shares common characteristics/ behavior and common properties/ attributes.
    2. Class is not a real world entity. It is just a template or blueprint or prototype from which objects are created.
    3. Class does not occupy memory.
    4. Class is a group of variables of different data types and group of methods.
6. **WHAT IS AN OBJECT ?**
7. Why is the return type of main method always void?
8. Difference between " public static void <func_name> " and " static public void <func_name> " ?
9. What are big Omega, big Theta, big Oh.

| S.No. | Big Oh (O) | Big Omega (Ω) | Big Theta (Θ) |
|---|---|---|---|
| 1. | It is like (<=) rate of growth of an algorithm is less than or equal to a specific value. | It is like (>=) rate of growth is greater than or equal to a specified value. | It is like (==) meaning the rate of growth is equal to a specified value. |

10. What type of sort is used by Java?

Sorting of Primitive Datatypes --> Double Pivot Quick Sort

Sorting of Objects --> Merge Sort

11. What are the default values and java and how they are initialized?

```java
/*****************************************************************************

1. Class Level, Instance Level Variables are initiated with default value even if those are not assigned.
2. Local Variables are not initialized automatically.
*****************************************************************************/
public class Main
{
    static int z;
    public static void main(String[] args) {
        int a;

        System.out.println(Sounak.b+" "+(new Sounak()).c+" "+z );
    }
}

class Sounak{
    static int b; //class level variable
    int c; // object/Instance level variable
}
```

input

```
0 0 0
```

| Data Type | Default Value (for fields) |
|---|---|
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| float | 0.0f |
| double | 0.0d |
| char | '\u0000' |
| String (or any object) | null |
| boolean | false |

## 12. Why is java slow?

There are two main reason why java is slow:

1. Dynamic Linking: Unlike C java is linked at the runtime instead of the compile time,every time a program is run.
2. Run-time Interpreter : The conversion of byte code into native machine code is done at the runtime which makes the program very slow.

# 2. ALL PILLARS OF OOP

- **Abstraction** : Data **abstraction** is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either **abstract classes** or **interfaces**.

  ### IMPORTANT CONCEPTS ABOUT INTERFACES

  - The variable in an interface is public, static, and final by default.
  - If any variable in an interface is defined without public, static, and final keywords then, the compiler automatically adds the same.
  - No access modifier is allowed except the public for interface variables.
  - Every variable of an interface must be initialized in the interface itself.
  - The class that implements an interface can not modify the interface variable, but it may use as it defined in the interface.
- **Encapsulation :** **Encapsulation** is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, that it is a protective shield that prevents the data from being accessed by the code outside this shield.

- **Polymorphism :** The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

  **Types of polymorphism**

In Java polymorphism is mainly divided into two types:

1. Compile-time Polymorphism
2. Runtime Polymorphism

- **Inheritance :**

# 3. ACCESS SPECIFIERS/ MODIFIERS

| Access Modifier | within class | within package | outside package by subclass only | outside package |
|---|---|---|---|---|
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

# 4. NON ACCESS SPECIFIERS/ MODIFIERS

| | |
|---|---|
| **static** | The member belongs to the class, not to objects of that class. |
| **final** | Variable values can't be changed once assigned, methods can't be overriden, classes can't be inherited. |
| **abstract** | If applied to a method - has to be implemented in a subclass, if applied to a class - contains abstract methods |
| **synchronized** | Controls thread access to a block/method. |

| | |
|---|---|
| **volatile** | The variable value is always read from the main memory, not from a specific thread's memory. |
| **transient** | The member is skipped when serializing an object. |
| **native** | |

# 5. Primitive Data types in java

| Data Type | Size | Description |
|---|---|---|
| `byte` | 1 byte | Stores whole numbers from -128 to 127 |
| `short` | 2 bytes | Stores whole numbers from -32,768 to 32,767 |
| `int` | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |

| | | |
|---|---|---|
| `long` | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| `float` | 4 bytes | Stores fractional numbers. Gives up-to 7 decimal digits. |
| `double` | 8 bytes | Stores fractional numbers. Gives up-to 15 decimal digits. |
| `boolean` | 1 bit | Stores true or false values |
| `char` | 2 bytes | Stores a single character/letter or ASCII values |

# 5. Overcoming the drawbacks of Primitive data types in Java

1. BigInteger
2. BigDecimal

# 6. Important ways to control Decimal Arithmetic in Java

It can be done in three ways :

1. By using BigDecimal
2. By using String.format
3. By using Formatter class
4. By using Math.round

## BigDecimal

```
import java.math.BigDecimal;
public class Main{
    public static void main(String args[]){
        BigDecimal obj1 = new BigDecimal("3.22");
        BigDecimal obj2 = new BigDecimal("4.35");
```

```
        System.out.println(obj1.add(obj2));

/*
            function
              ___
            add
            subtract
            multiply
            divide
              ____
        */
    }
}
```

## By using String.format

```
/*
    While printing it sets the precision
*/
```

## By using Formatter Class

```

```

## By using Math.round

```

```

# 6. Operators in Java

# 7. IO in Java

- **Java I/O** (Input and Output) is used *to process the input* and *produce the output*. Java uses the concept of a stream to make I/O operation fast.
- The java.io package contains all the classes required for input and output operations.
- We can perform **file handling in Java** by Java I/O API.

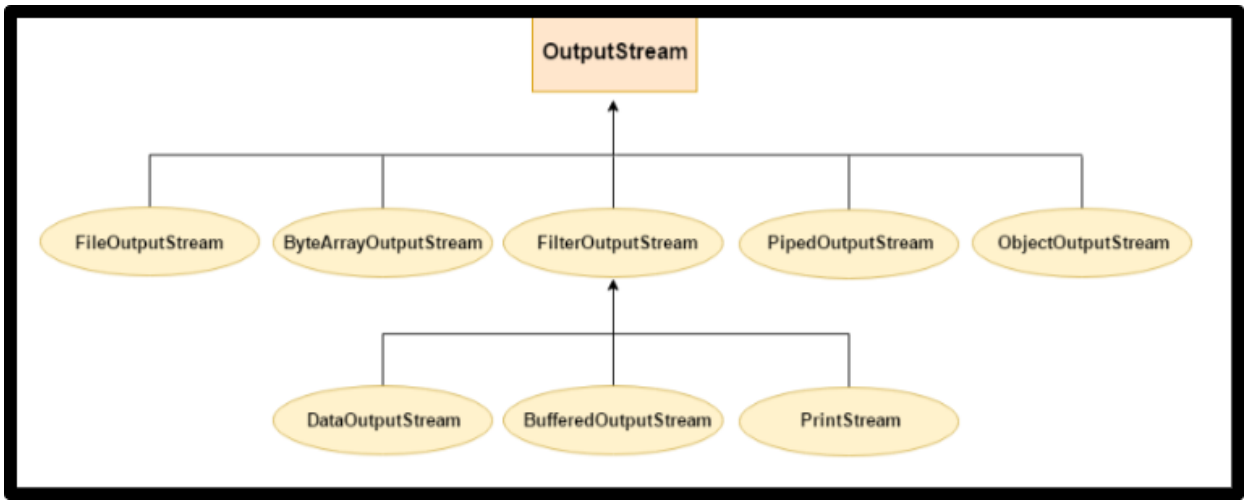
## Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow. In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) **System.out:** standard output stream
2) **System.in:** standard input stream
3) **System.err:** standard error stream

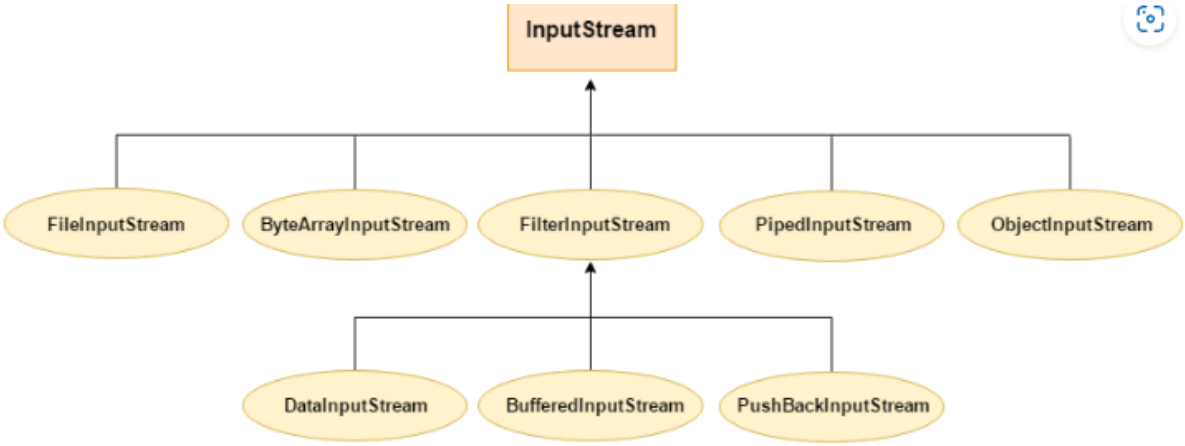
## OutputStream Class

- Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.
- OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.




## InputStream class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.




# All class involved in IO from console/terminal or from a file

## Seldomly used in file handling

1. FileOutputStream
2. FileInputStream
3. BufferedOutputStream
4. BufferedInputStream
5. SequenceInputStream
6. ByteArrayInputStream
7. ByteArrayOutputStream
8. DataOutputStream
9. DataInputStream

## Take input from console:

1. Using BufferedReader Class

```
// You have to throw a Exception to make the user aware of a exception that might occur
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
public class Main{
    public static void main(String args[]) throws IOException{
        BufferedReader obj = new BufferedReader(new InputStreamReader(System.in));
        String ans = obj.readLine();
        System.out.println(ans);
    }
}
```

| Method | Description |
| --- | --- |
| int read() | It is used for reading a single character. |
| int read(char[] cbuf, int off, int len) | It is used for reading characters into a portion of an array. |
| boolean markSupported() | It is used to test the input stream support for the mark and reset method. |
| String readLine() | It is used for reading a line of text. |
| boolean ready() | It is used to test whether the input stream is ready to be read. |
| long skip(long n) | It is used for skipping the characters. |
| void reset() | It repositions the stream at a position the mark method was last called on this input stream. |
| void mark(int readAheadLimit) | It is used for marking the present position in a stream. |
| void close() | It closes the input stream and releases any of the system resources associated with the stream. |

2. Using Scanner Class (MOSTLY USED)

```
import java.util.Scanner;
public class Main{
    public static void main(String args[]){
        Scanner obj = new Scanner(System.in);
        int a = obj.nextInt(); //Inputs a integer
    }
}
```

ALL CONSTRUCTORS

| SN | Constructor | Description |
|---|---|---|
| 1) | Scanner(File source) | It constructs a new Scanner that produces values scanned from the specified file. |
| 2) | Scanner(File source, String charsetName) | It constructs a new Scanner that produces values scanned from the specified file. |
| 3) | Scanner(InputStream source) | It constructs a new Scanner that produces values scanned from the specified input stream. |
| 4) | Scanner(InputStream source, String charsetName) | It constructs a new Scanner that produces values scanned from the specified input stream. |
| 5) | Scanner(Readable source) | It constructs a new Scanner that produces values scanned from the specified source. |
| 6) | Scanner(String source) | It constructs a new Scanner that produces values scanned from the specified string. |

METHODS AVAILABLE IN SCANNER CLASS

| SN | Modifier & Type | Method | Description |
|---|---|---|---|
| 1) | void | close() | It is used to close this scanner. |
| 2) | pattern | delimiter() | It is used to get the Pattern which the Scanner class is currently using to match delimiters. |
| 3) | Stream<MatchResult> | findAll() | It is used to find a stream of match results that match the provided pattern string. |
| 4) | String | findInLine() | It is used to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters. |
| 5) | string | findWithinHorizon() | It is used to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters. |

| | | | |
|---|---|---|---|
| 6) | boolean | hasNext() | It returns true if this scanner has another token in its input. |
| 7) | boolean | hasNextBigDecimal() | It is used to check if the next token in this scanner's input can be interpreted as a BigDecimal using the nextBigDecimal() method or not. |
| 8) | boolean | hasNextBigInteger() | It is used to check if the next token in this scanner's input can be interpreted as a BigDecimal using the nextBigDecimal() method or not. |
| 9) | boolean | hasNextBoolean() | It is used to check if the next token in this scanner's input can be interpreted as a Boolean using the nextBoolean() method or not. |
| 10) | boolean | hasNextByte() | It is used to check if the next token in this scanner's input can be interpreted as a Byte using the nextBigDecimal() method or not. |
| 11) | boolean | hasNextDouble() | It is used to check if the next token in this scanner's input can be interpreted as a BigDecimal using the nextByte() method or not. |
| 12) | boolean | hasNextFloat() | It is used to check if the next token in this scanner's input can be interpreted as a Float using the nextFloat() method or not. |
| 13) | boolean | hasNextInt() | It is used to check if the next token in this scanner's input can be interpreted as an int using the nextInt() method or not. |
| 14) | boolean | hasNextLine() | It is used to check if there is another line in the input of this scanner or not. |

| | | | |
|---|---|---|---|
| 15) | boolean | hasNextLong() | It is used to check if the next token in this scanner's input can be interpreted as a Long using the nextLong() method or not. |
| 16) | boolean | hasNextShort() | It is used to check if the next token in this scanner's input can be interpreted as a Short using the nextShort() method or not. |
| 17) | IOExcepti on | ioException() | It is used to get the IOException last thrown by this Scanner's readable. |
| 18) | Locale | locale() | It is used to get a Locale of the Scanner class. |
| 19) | MatchRes ult | match() | It is used to get the match result of the last scanning operation performed by this scanner. |
| 20) | String | next() | It is used to get the next complete token from the scanner which is in use. |
| 21) | BigDecim al | nextBigDecimal() | It scans the next token of the input as a BigDecimal. |
| 22) | BigInteger | nextBigInteger() | It scans the next token of the input as a BigInteger. |
| 23) | boolean | nextBoolean() | It scans the next token of the input into a boolean value and returns that value. |
| 24) | byte | nextByte() | It scans the next token of the input as a byte. |
| 25) | double | nextDouble() | It scans the next token of the input as a double. |
| 26) | float | nextFloat() | It scans the next token of the input as a float. |
| 27) | int | nextInt() | It scans the next token of the input as an Int. |

| | | | |
|---|---|---|---|
| 28) | String | nextLine() | It is used to get the input string that was skipped of the Scanner object. |
| 29) | long | nextLong() | It scans the next token of the input as a long. |
| 30) | short | nextShort() | It scans the next token of the input as a short. |
| 31) | int | radix() | It is used to get the default radix of the Scanner use. |
| 32) | void | remove() | It is used when remove operation is not supported by this implementation of Iterator. |
| 33) | Scanner | reset() | It is used to reset the Scanner which is in use. |
| 34) | Scanner | skip() | It skips input that matches the specified pattern, ignoring delimiters |
| 35) | Stream<String> | tokens() | It is used to get a stream of delimiter-separated tokens from the Scanner object which is in use. |
| 36) | String | toString() | It is used to get the string representation of Scanner using. |
| 37) | Scanner | useDelimiter() | It is used to set the delimiting pattern of the Scanner which is in use to the specified pattern. |
| 38) | Scanner | useLocale() | It is used to sets this scanner's locale object to the specified locale. |
| 39) | Scanner | useRadix() | It is used to set the default radix of the Scanner which is in use to the specified radix. |

3. Using Console Class

```
public class Main{
```

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

        String ans = System.console().readLine();  // Console obj = System.console();
                                                   // String  ans = obj.readLine();
        System.out.println(ans);
        //Enter a password using console class
        char[] pass = System.console().readPassword();
        System.out.println(pass);
    }
}
```

METHODS USED IN CONSOLE CLASS

| | |
|---|---|
| String readLine() | It is used to read a single line of text from the console. |
| String readLine(String fmt, Object... args) | It provides a formatted prompt then reads the single line of text from the console. |
| char[] readPassword() | It is used to read password that is not being displayed on the console. |
| char[] readPassword(String fmt, Object... args) | It provides a formatted prompt then reads the password that is not being displayed on the console. |
| Console format(String fmt, Object... args) | It is used to write a formatted string to the console output stream. |
| Console printf(String format, Object... args) | It is used to write a string to the console output stream. |
| PrintWriter writer() | It is used to retrieve the PrintWriter object associated with the console. |
| void flush() | It is used to flushes the console. |

4. Using Command Line argument (Passing as an argument to the main function)

```
import java.util.Arrays;
public class M1{
        public static void main(String[] arr){
                System.out.println(Arrays.toString(arr));
        }
```

```
}
```

```
D:\Coding>javac M1.java

D:\Coding>java M1 Abhishek Manna
[Abhishek, Manna]
```

Mostly Used in File Handling

# 7. Type Casting in Java

# 8. Sorting in Java

Sorting  in java can be done through various algorithms and  inbuilt functions :

Some inbuilt functions  to use to sort data in java are as follows :

1. Collections.sort --> to sort non primitive data types
2. Arrays.sort()  ->  to sort primitive data type in an array

The classes we use to do comparison in a non orthodox way :

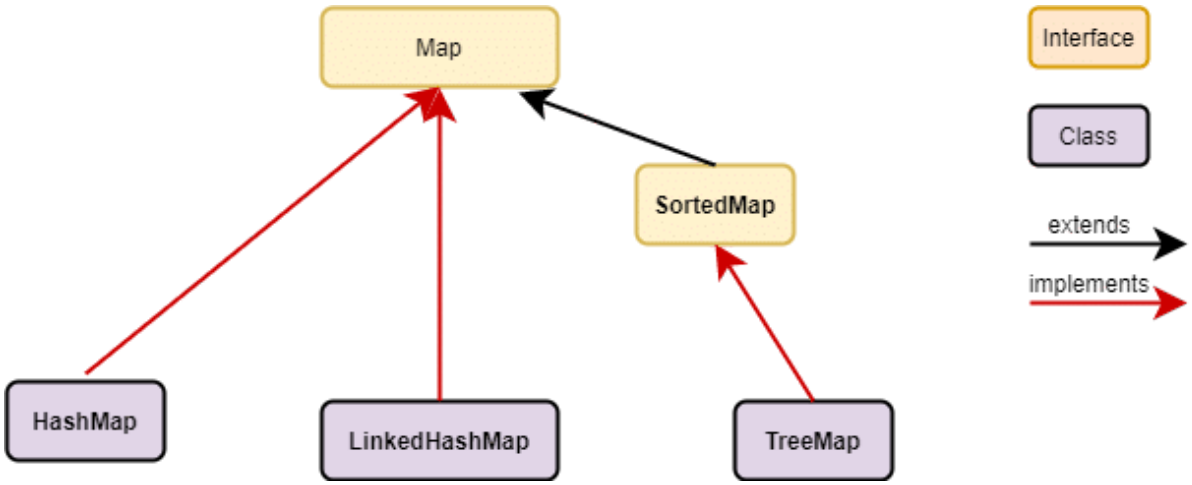
By using :

1. Using Comparator Class
2. Using Comparable Class

# 9. Strings in Java

# 10. Arrays in Java

# 11. Collection Framework in Java



# HashMap

**Important charecteristics about HashMap:**

1. `HashMap` itself doesn't maintain insertion order. Use LinkedHashMap for that.
2. Java HashMap contains values based on the key.
3. Java HashMap contains only unique keys.
4. Java HashMap may have one null key and multiple null values.
5. Java HashMap is non synchronized.
6. The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

**Important Functions**

1. **void obj.put(<key>,<value>);** --> we can insert key value pairs with the help of this in HashMap
2. **<value> obj.get(<key>)**; --> we can get a particular value corresponding to a particular key
3. **arr [] obj.keySet()**; --> Returns an array of all the Keys.
4. **boolean isEmpty()**; --> Checks if the map is empty or not
5. **boolean containsKey(<key>)** --> Checks if the key is present or not
6. **boolean containsValue(<key>)** --> Checks if the value is present or not.
7. **void remove(<key>)** --> removes the key value pair.

**CONSTRUCTOS OF HASHMAP**

| | |
|---|---|
| HashMap() | It is used to construct a default HashMap. |
| HashMap(int capacity) | It is used to initializes the capacity of the hash map to the given integer value, capacity. |
| HashMap(int capacity, float loadFactor) | It is used to initialize both the capacity and load factor of the hash map by using its arguments. |

# Linked HashMap

**CHARECTERTISTICS**

- A LinkedHashMap contains values based on the key. It implements the Map interface and extends the HashMap class.

- It contains only unique elements.
- It may have one null key and multiple null values.
- It is non-synchronized.
- It is the same as HashMap with an additional feature that it maintains insertion order.

## CONSTRUCTORS

**1. LinkedHashMap():** This is used to construct a default LinkedHashMap constructor.

```
LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>();
```

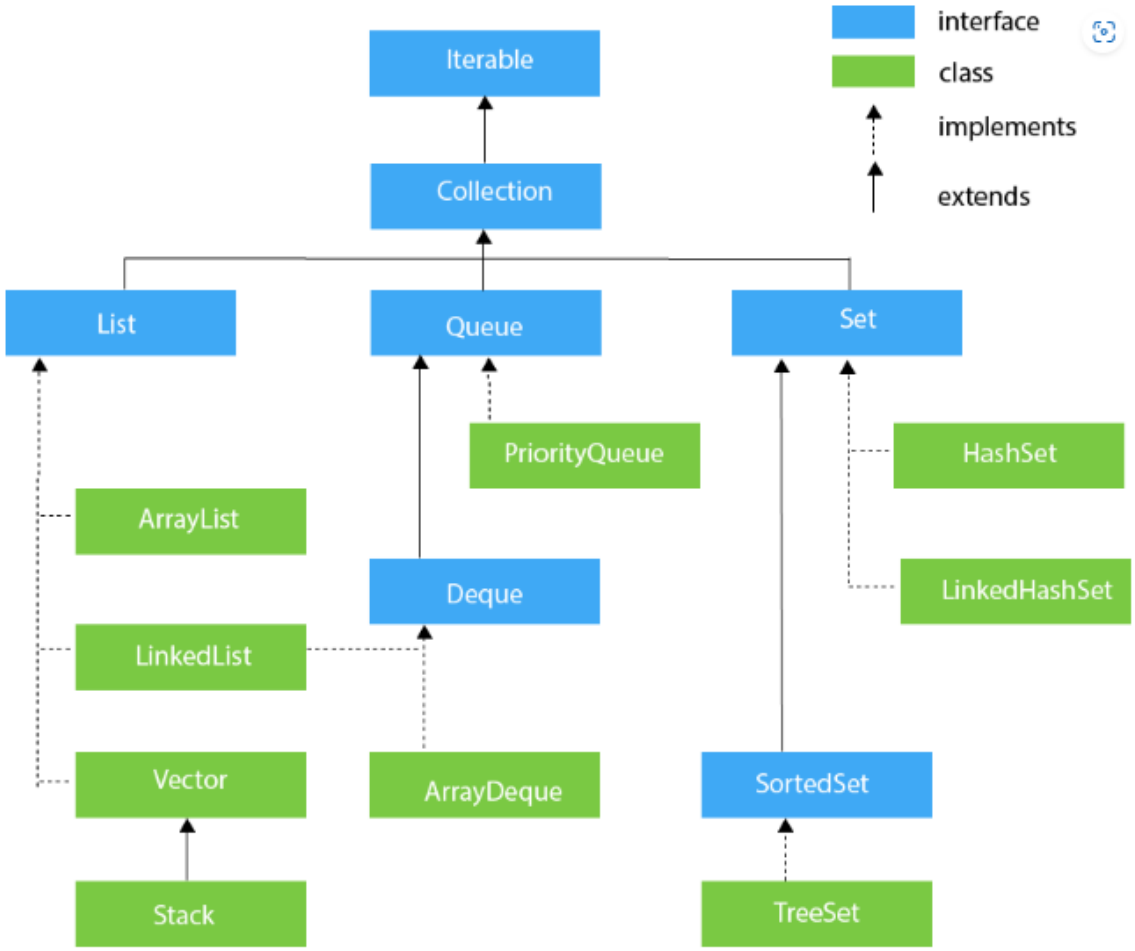
**2. LinkedHashMap(int capacity):** It is used to initialize a particular LinkedHashMap with a specified capacity.

```
LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(int capacity);
```

## FUNCTIONS

1. **void obj.put(<key>,<value>);** --> we can insert key value pairs with the help of this in HashMap
2. **<value> obj.get(<key>)**; --> we can get a particular value  corresponding to a particular key
3. **arr []  obj.keySet()**; --> Returns an array of all the Keys.
4. **boolean isEmpty()**; --> Checks if the map is empty or not
5. **boolean containsKey(<key>)** --> Checks if the key is present or not
6. **boolean containsValue(<key>)** --> Checks if the value is present or not.
7. **void remove(<key>)** --> removes the key value pair.

# TREEMAP




# Priority Queue

PriorityQueue is a data structure used for sequential storage of  data that follows FIFO principal.

**CHARECTERTISTICS**

- PriorityQueue doesn't permit null.
- We can't create a PriorityQueue of Objects that are non-comparable
- PriorityQueue are unbound queues.
- The head of this queue is the least element with respect to the specified ordering. If multiple elements are tied for the least value, the head is one of those elements — ties are broken arbitrarily.
- Since PriorityQueue is not thread-safe, java provides PriorityBlockingQueueclass that implements the BlockingQueueinterface to use in a java multithreading environment.

- The queue retrieval operations poll, remove, peek, and element access the element at the head of the queue.
- It provides O(log(n)) time for add and poll methods.
- It inherits methods from **AbstractQueue**, **AbstractCollection**, **Collection,**and **Object**class.

## USING PRIORITY QUEUE ALONG WITH COMPARATOR INTERFACE AND LAMBDA FUNCTIONS

```
import java.util.*;
public class Main
{
        public static void main(String[] args) {
                //PriorityQueue<XII> obj = new PriorityQueue<>((a,b)->
(a.name).compareTo(b.name));
                PriorityQueue<XII> obj =new PriorityQueue<>(new Aunty());
                obj.add(new XII(69,"Abhishek Manna"));
                obj.add(new XII(75,"Sounak Chakraborty"));
                obj.add(new XII(84,"Ayan Acharyaa"));
                while(!obj.isEmpty()){
                    obj.poll().info();
                }
        }
}
class XII{
    int marks;
    String name;
    public XII(int marks,String name){
        this.marks = marks;
        this.name = name;
    }
    public void info(){
        System.out.println(this.marks+" "+this.name);
    }
}

class Aunty implements Comparator<XII>{
    @Override
    public int compare(XII a,XII b){
        if(a.marks >b.marks) return  1;
        else if(a.marks <  b.marks) return -1;
        return 0;
    }
}
```

## IMPORTANT FUNCTIONS  OF PRIORITY  QUEUE

1. obj.add(<object>);  --> Adds element  to the queue
2. obj.poll(); --> Removes the last added element or the last  element at the queue following the FIFO principal
3. obj.peek() --> Show or returns the last element present in the Queue.
4. obj.isEmpty() --> To check if the queue is empty or not.
5. obj.clear() --> Removes all the element from the list.
6. obj.contains() --> Returns true if the element is present in the list.

CONSTRUCTORS OF PRIORITY QUEUE

```
public PriorityQueue(int initialCapacity,
            Comparator<? super E> comparator)
```

# ARRAYLIST

## CONSTRUCTOR OF ARRAYLIST

**ArrayList**()
Constructs an empty list with an initial capacity of ten.

---

**ArrayList**(**Collection**<? extends **E**> c)
Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

---

**ArrayList**(int initialCapacity)
Constructs an empty list with the specified initial capacity

## FUNCTIONS OF ARRAYLIST

1. add(<element>) --> adds element at the end of the list
2. add(<element>,<index>) --> adds element to the desired index as provided.
3. obj.clear() --> clear all the element from the array.
4. obj.get(index) --> Gives us the element which is present in the current index
5. obj.indexOf(Object) --> Gives the index of the object present.
6. obj.isEmpty() --> Tells us if the arrayList is empty or not.
7. obj.remove(<index>) --> Removes an element present in the particular index.
8. obj.remove(<object>) --> Removes the first occurance of the object found.
9. obj.removeAll(<object>) --> removes all the type of list present in the array.

# LINKEDLIST

## CONSTRUCTOR OF LINKEDLIST

1. *LinkedList ll = new LinkedList();*
2. *LinkedList ll = new LinkedList(<Constructor>);*

## FUNCTIONS OF ARRAYLIST

1. add(<element>) --> adds element at the end of the list
2. add(<element>,<index>) --> adds element to the desired index as provided.
3. addAll(Collection <object>) --> adds all the element of the constructor object in the array.
4. obj.clear() --> clear all the element from the array.
5. obj.get(index) --> Gives us the element which is present in the current index
6. obj.indexOf(Object) --> Gives the index of the object present.
7. obj.isEmpty() --> Tells us if the arrayList is empty or not.
8. obj.remove(<index>) --> Removes an element present in the particular index.
9. obj.remove(<object>) --> Removes the first or last occurance of the object found
10. obj.removeFirstOccurence / obj.removeLastOccurence --> Removes the first or last element and returns it
11. obj.removeAll(<object>) --> removes all the type of list present in the array.
12. obj.peek() --> Retrieves but doesnot remove the last element from the list.
13. obj.poll() --> Retrives and removes the head or the first element from the array.
14. obj.push() --> pushes an element into the stack.
15. obj.toArray() --> Converts the whole linked list into an array.
16. obj.toString() --> Converts the whole linked list into a string.

# VECTORS

Vectors are same as ArrayList but has support of some legacy functions that are not available in the collections framework. They are slower than ArrayList are used in threaded environment as they are synchronized.

## CONSTRUCTOR OF LINKEDLIST

**1. Vector():** Creates a default vector of the initial capacity is 10.

```
Vector<E> v = new Vector<E>();
```

**2. Vector(int size):** Creates a vector whose initial capacity is specified by size.

```
Vector<E> v = new Vector<E>(int size);
```

### FUNCTIONS OF ARRAYLIST

-> Same as that of ArrayList

## STACK

It provides a Stack class that models and implements a **Stack data structure**. The class is based on the basic principle of last-in-first-out. It inherits from the vector class.

### CONSTRUCTORS OF STACK

```
public class Stack<E> extends Vector<E>
```

### FUNCTIONS OF STACK

1. **push(<e>) --> Adds a particular element at the end of the list or stack in LIFO concept**
2. **pop() --> Removes and returns the last element from the top of the stack following the LIFO concept**
3. **peek() --> Retrieves the upper most element from the stack.**
4. **empty() --> Returns true if the top of the stack us empty**
5. **search(<object>) --> Searches and gives us a boolean answer if the element is present or not**

**ALL OTHER FUNCTIONS OF STACK ARE SIMILAR TO VECTOR AS IT IS INHERITED FROM THE LATER.**

## HASHSET

### CHARECTERISTICS

- Implements Set Interface.
- The underlying data structure for HashSet is Hashtable.
- As it implements the Set Interface, duplicate values are not allowed.
- Objects that you insert in HashSet are not guaranteed to be inserted in the same order. Objects are inserted based on their hash code.
- NULL elements are allowed in HashSet.
- HashSet also implements **Serializable**and **Cloneable**interfaces.

### FUNCTIONS

| | |
|---|---|
| add(E e) | Used to add the specified element if it is not present, if it is present then return false. |
| clear() | Used to remove all the elements from set. |

| | |
|---|---|
| contains(Object o) | Used to return true if an element is present in set. |
| remove(Object o) | Used to remove the element if it is present in set. |
| iterator() | Used to return an iterator over the element in the set. |
| isEmpty() | Used to check whether the set is empty or not. Returns true for empty and false for a non-empty condition for set. |
| size() | Used to return the size of the set. |
| clone() | Used to create a shallow copy of the set. |

## LINKED HASHSET

### CHARECTERISTICS

- Contains unique elements only like HashSet. It extends the HashSet class and implements the Set interface.
- Maintains insertion order.

### FUNCTIONS

**-->Same as that of HashSet**

## TREESET

# ADVANCED JAVA

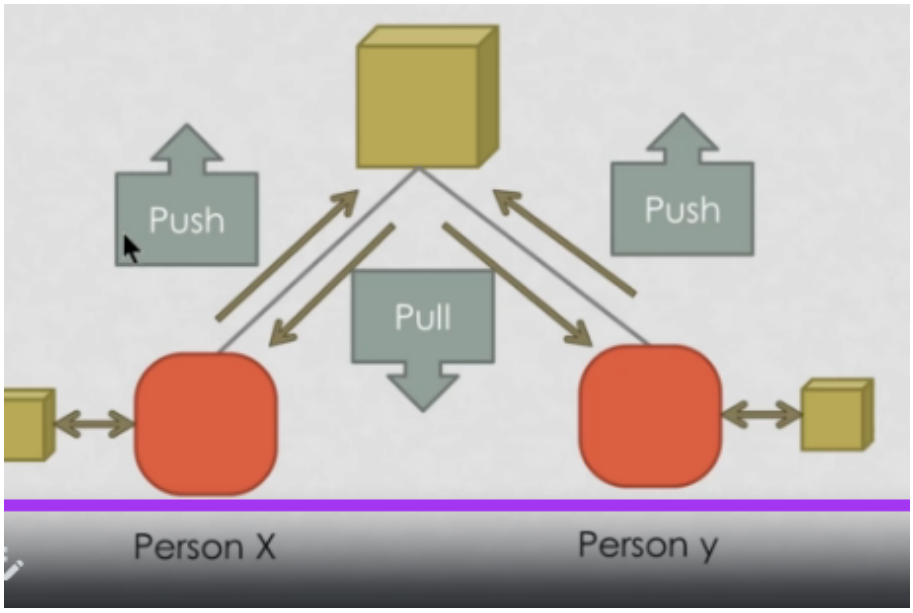
**Contents :**

1. **How to use GIT**

2. **Error Handling**
3. **Multi Threading**

# 1. GIT

1. Git was created by Linus Torvalds in 2005.
2. It helps manage projects with multiple contributor.
3. Distributed version control system.
4. Version Control
5. Pushing and Pulling data
6. Contribute
7. Help in resolving conflicts.



# 2. ERROR HANDLING IN JAVA

## Error Handling is done using 5 keyword:

1. try
2. catch
3. finally
4. throw
5. throws

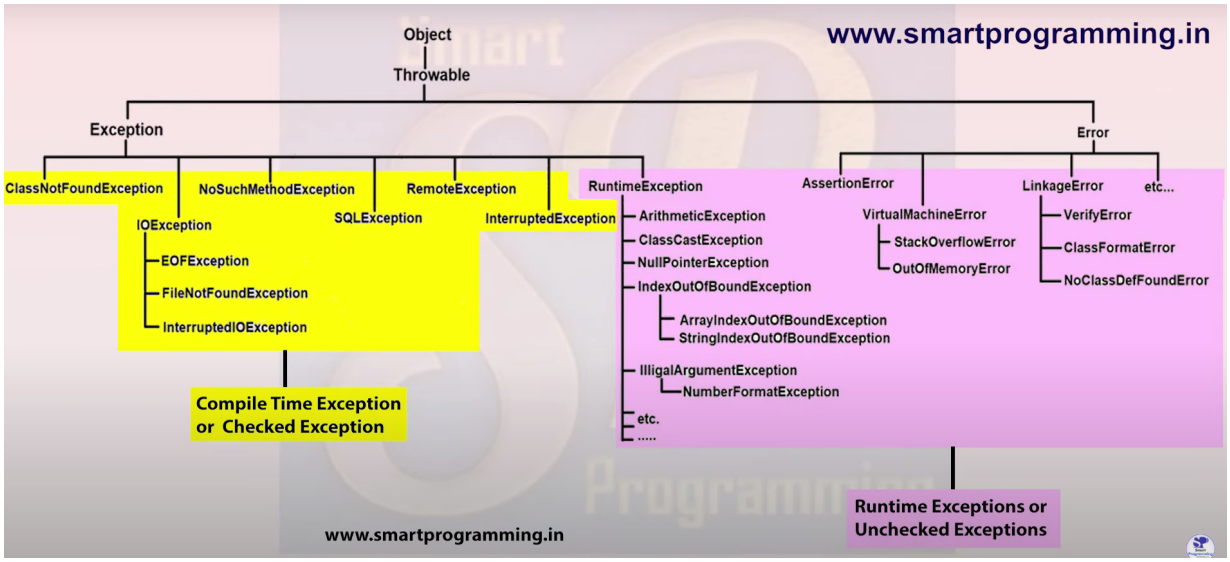
## Error Handling using try and catch block in Java

```
public class Main
{

        public static void main(String[] args) {
            try{
                    System.out.println("Hello World");
                    int a=10/0;
            }
            catch(Exception e){
                System.out.println(e);

            }
    }
}
```

## All throwable Hierarchy  which includes exception and errors

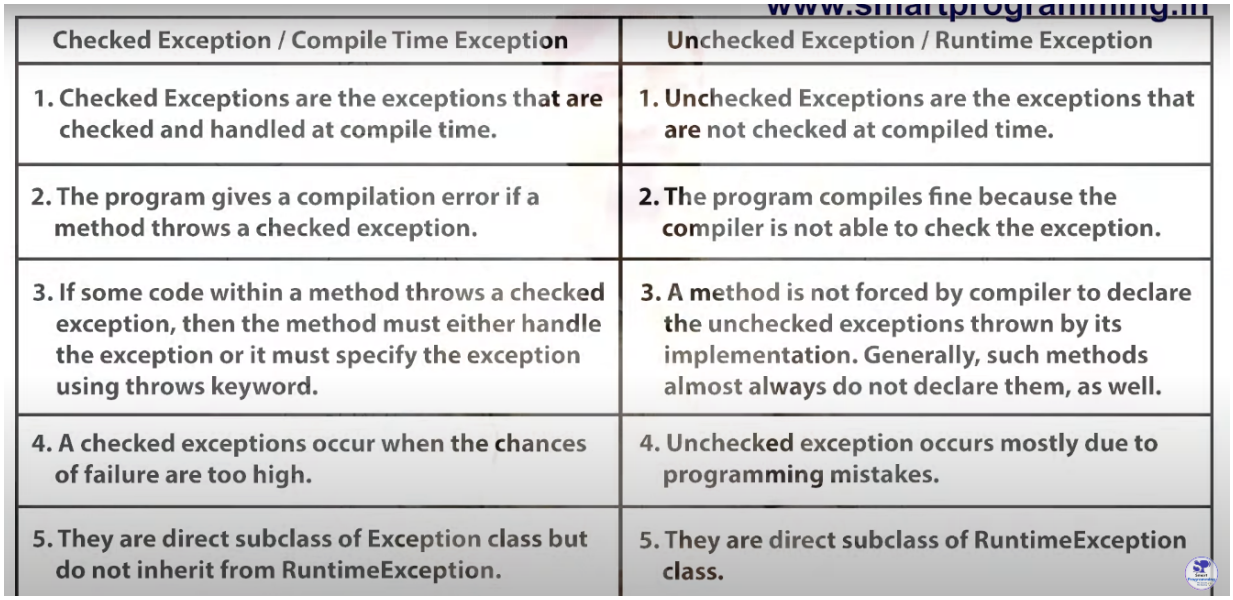



# Difference between Error and Exceptions in Java

| Errors | Exceptions |
|---|---|
| Recovering from Error is not possible. | We can recover from exceptions by either using try-catch block or throwing exceptions back to the caller. |
| All errors in java are unchecked type. | Exceptions include both checked as well as unchecked type. |
| Errors are mostly caused by the environment in which program is running. | Program itself is responsible for causing exceptions. |
| Errors can occur at compile time as well as run time. Compile Time: eg Syntax Error Run Time: Logical Error. | All exceptions occurs at runtime but checked exceptions are known to the compiler while unchecked are not. |

| | |
|---|---|
| They are defined in java.lang.Error package. | They are defined in java.lang.Exception package |
| Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError | Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundException, NullPointerException, ArithmeticException. |

## Difference between Checked/Compile Time and Exceptions/Runtime Exceptions in Java



| Checked Exception / Compile Time Exception | Unchecked Exception / Runtime Exception |
|---|---|
| 1. Checked Exceptions are the exceptions that are checked and handled at compile time. | 1. Unchecked Exceptions are the exceptions that are not checked at compiled time. |
| 2. The program gives a compilation error if a method throws a checked exception. | 2. The program compiles fine because the compiler is not able to check the exception. |
| 3. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword. | 3. A method is not forced by compiler to declare the unchecked exceptions thrown by its implementation. Generally, such methods almost always do not declare them, as well. |
| 4. A checked exceptions occur when the chances of failure are too high. | 4. Unchecked exception occurs mostly due to programming mistakes. |
| 5. They are direct subclass of Exception class but do not inherit from RuntimeException. | 5. They are direct subclass of RuntimeException class. |

## Different ways to handle and print the Exception:

1. System.out.println(e); / System.out.println(e.toString());
2. System.out.println(e.getMessage()); (worst way only shows description)
3. ae.printStackTrace(); (best wat to print a error)

## Difference between Throw and Throws in Java

| | | | |
|---|---|---|---|
| **1.** | **Point of Usage** | The **throw** keyword is used inside a function. It is used when it is required to throw a custom made Exception logically. | The **throws** keyword is used in the function signature. It is used when the function has some statements that can lead to exceptions. |
| **2.** | **Exceptions Thrown** | The **throw** keyword is used to throw an exception explicitly. It can throw only one exception at a time. | The **throws** keyword can be used to declare multiple exceptions, separated by a comma. Whichever exception occurs, if matched with the declared ones, is thrown automatically then. |
| **3.** | **Syntax** | Syntax of **throw** keyword includes the instance of the Exception to be thrown. Syntax wise throw keyword is followed by the instance variable. | Syntax of **throws** keyword includes the class names of the Exceptions to be thrown. Syntax wise throws keyword is followed by exception class names. |

| | | | |
|---|---|---|---|
| 4. | **Propagation of Exceptions** | **throw** keyword cannot propagate checked exceptions. It is only used to propagate the unchecked Exceptions that are not checked using the throws keyword. | **throws** keyword is used to propagate the checked Exceptions only. |

## Example of throws and throw in Code

```
import java.io.*;
public class Main
{

        public static void main(String[] args) throws IOException{
            //try{
            BufferedReader obj = new BufferedReader(new InputStreamReader(System.in));
            long a = Long.parseLong(obj.readLine());

            if(a>28){
                throw new test("Aladin");
            }
            System.out.println("Hello ng");
    }
}
class test extends RuntimeException{
    test(String msg){
        super(msg);
    }
}
```

```
import java.io.*;
public class Main
{

        public static void main(String[] args) throws IOException{
         try{
            BufferedReader obj = new BufferedReader(new InputStreamReader(System.in));
            long a = Long.parseLong(obj.readLine());
```

```
            if(a>28){
                throw new test("Aladin");
            }
            System.out.println("Hello ng");
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
class test extends Exception{
    test(String msg){
        super(msg);
    }
}
```

# 3. MULTI-THREADING IN JAVA

**Multi Tasking :** **Multi-tasking is the process of doing several tasks together at a time.**

**There are two ways in which multi-tasking is done:**

1. **Multi-Threading.**
2. **Multi-Processing**

## MULTI-THREADING

**Multithreading** **in Java** is a process of executing multiple threads simultaneously. Multithreading is a Java feature that allows concurrent execution of two or more parts of a process for maximum utilization of CPU. Each part of such program is called a thread. Threads are light-weight smallest part of a process. Numerous small threads make up  a process.

## MULTI-PROCESSING

When a system is connected to multiple processors to complete a task is known as multi-processing. Each process has an address in memory. In other words, each process allocates a separate memory area. A process is heavyweight. Cost of communication between the process is high.

## ADVANTAGES OF MULTI-THREADING

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

## WHAT IS A THREAD?

A thread is a lightweight sub-process, the smallest unit of processing. It has a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

There  are two ways in which a thread can be created :

1. **Extending the Thread class.**

```
public class Main
{
        public static void main(String[] args) {
                System.out.println("Hello World");
                Dummy obj = new Dummy();
                obj.start();
        }
}
class Dummy extends Thread{
    @Override
    public void run(){
        try{
             for(int i=0;i<5;i++) {
                System.out.println(i);
                this.sleep(2000);
             }

        }
        catch(Exception e){
            e.printStackTrace();
        }

    }
}
```

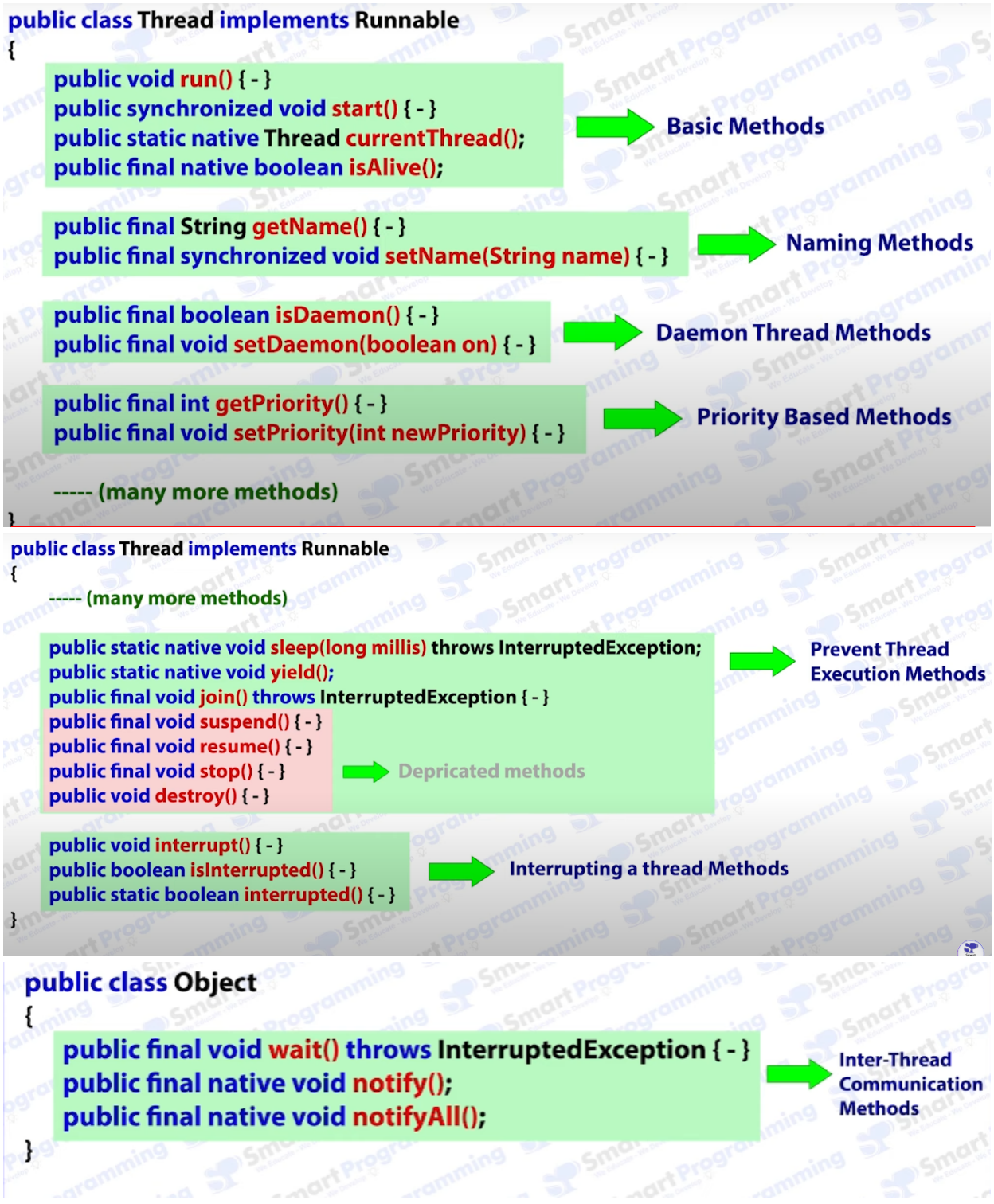
2. ## Implementing  the Runnable interface.

```
public class Main
{
        public static void main(String[] args) {
                System.out.println("Hello World");
                Thread obj = new Thread(new Dummy());
                obj.start();
        }
}
class Dummy implements Runnable{
    @Override
    public void run(){
        try{
             for(int i=0;i<5;i++) {
                System.out.println(i);
                Thread.sleep(2000);
             }

        }
        catch(Exception e){
            e.printStackTrace();
        }

    }
}
```

## FUNCTIONS IN THREAD CLASS




# What is Synchronization?

It is the process we control the accessibility of multiple threads to a particular shared resource.

## Problems which can occur without synchronization?

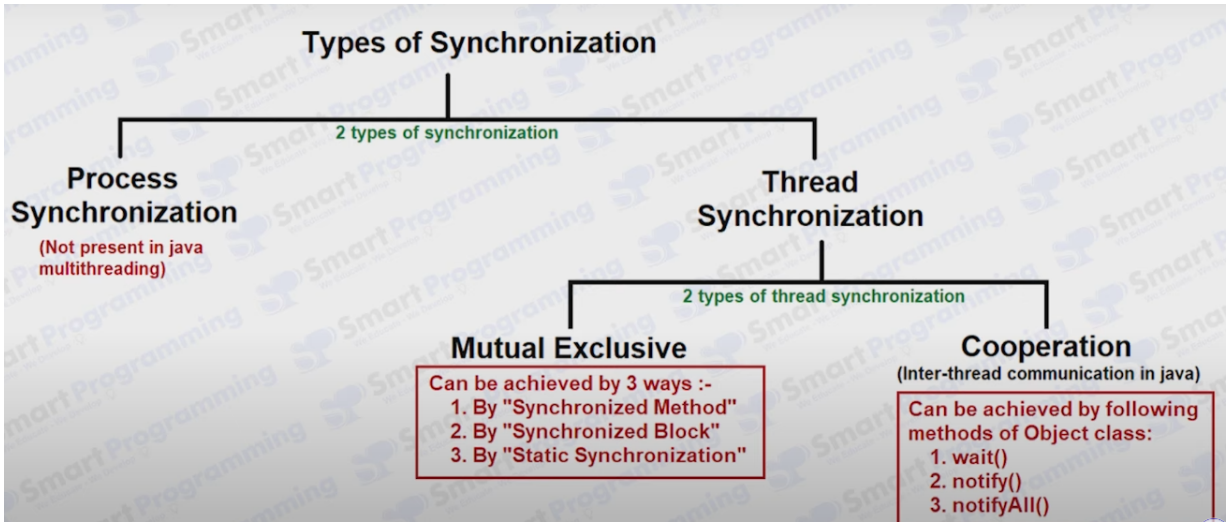
1. Data inconsistantcy
2. Thread Intereference

## Advantages of Synchronization

1. Removes data inconsistantcy
2. Removes thread interference

## Dis-advantages of Synchronization

1. Increasing waiting time for threads
2. Decreases performance




# 4. Servlets IN JAVA

Servlets are simple java programs that run on server and are capable of handling request and generating dynamic response.
Package to be used :

1. javax.servlet

Interface to be used:

1. Servlet
    1. public abstract void init(javax.servlet.ServletConfig())
    2. public  ServletCongif getServletConfig();
    3. public void
        1. service.javax.servlet.ServletRequest,
    4. public abstract java.lang.String getServletInfo();
    5. public abstract void destroy();

1,3,5 are Life Cycle methods.

These methods have to be overridden

We have to map : using web.xml (Deployment Descriptor)

JDBC Connectivity

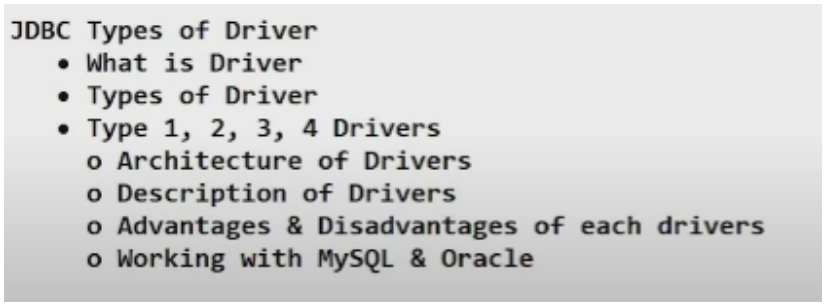
1. Register the driver class. It is present within the sql-connector library so we have to import that jar file into our library.
2. Creating the connection

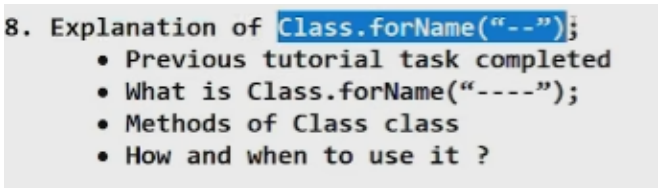
# JDBC IN JAVA

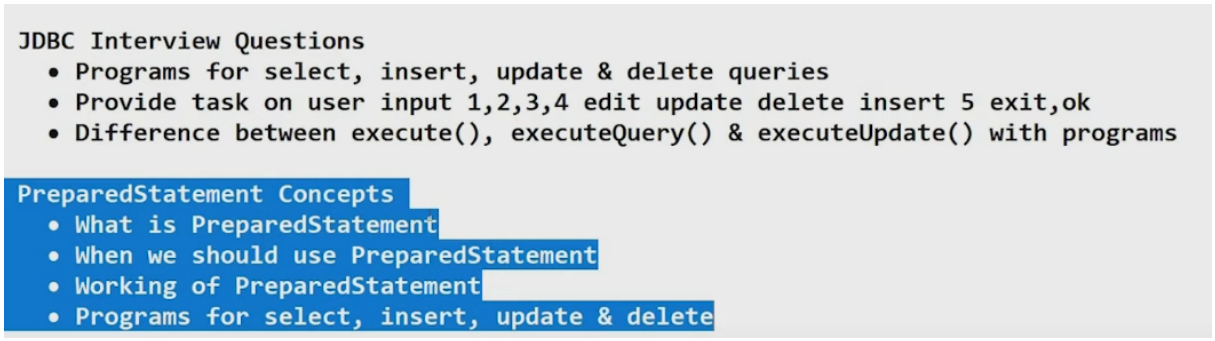
What is JDBC ?

1. JDBC stands for Java Database Connectivity.
2. It is a technology used to connect a Java application to a database.
3. JDBC is an API containing two packages with some classes and interfaces used to provide connectivity.
4. The two packages are : java.sql and javax.sql
5. JDBC is an abstraction which is provided by Sun Microsystems and is implemented by Database Vendors.

## THINGS TO LEARN IN JDBC




DRIVERS IN JDBC

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

Difference between execute(), executeUpdate(), executeQuery()
execute() --> It is used for DDL commands
executeUpdate() --> It i used for DML commands
executeQuery() --> It is used for selection statements.
Code:

# APPLETS IN JAVA

An applet is a small program created to be implemented and embedded in a webpage to generate dynamic content. The salient feature of the applet is that it can run inside a webpage on the client side (User interface side).

Features/ keywords of Applets:

- **Generating dynamic content.**
- **Works at the client side**
- **Response is fast**
- **Object--->Container---> Component-->Panel-->Applet--->JApplet**
- **import java.awt.applet**

**The main applet is present within --> import java.applet.Applet (This class has been discontinued)**

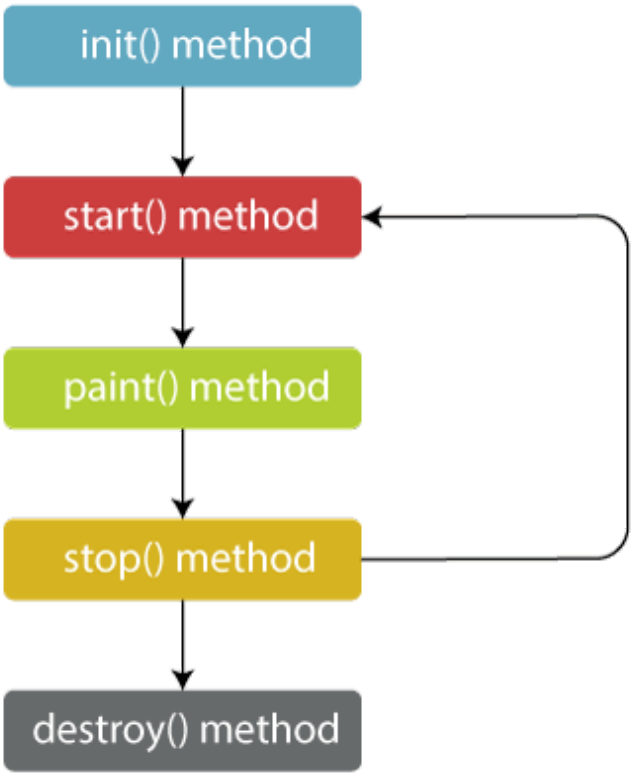
There are two type of ways to run an applet.

1. **Inside a HTML file.**
2. **By Applet Viewer Tool**

**There are two types of applets present in java:**

1. **The applets which are used in the AWT (Abstract Window Toolkit) packages by extending its  Applet Class.**
2. **Applets used inside swing package by extending the JApplet class inside it.**

**The life cycle of Applet is as follows:**


init() method
start() method
paint() method
stop() method
destroy() method