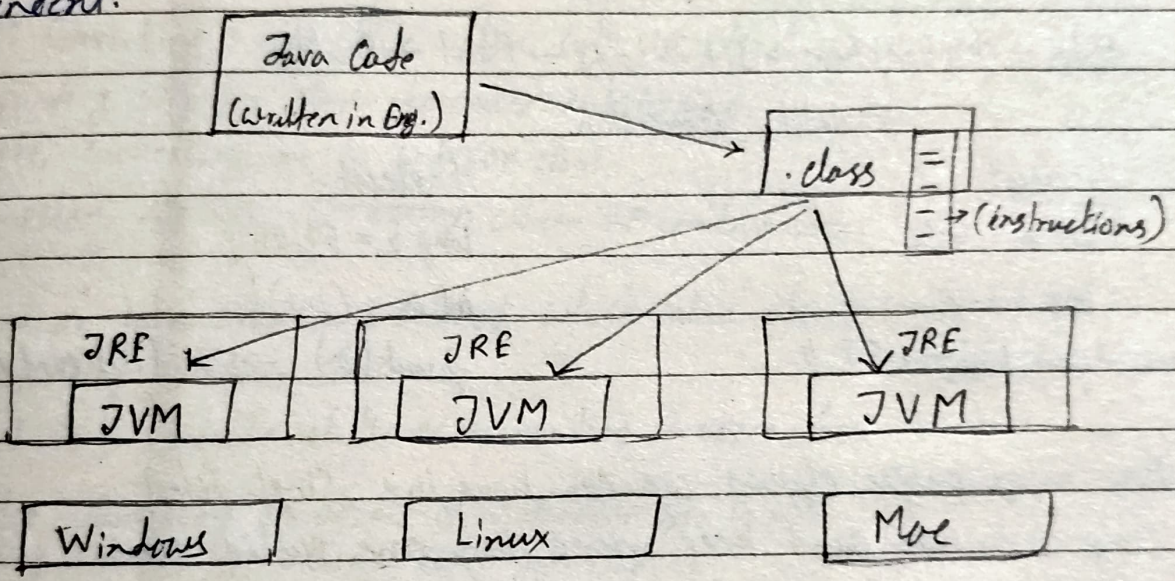
Core Java / Session-1 / 23/10/2021

Why Java is platform independent?

Java is platform independent as it does not depend on any platform i.e it can run on any platform i.e Windows, Mac, Linux etc.
Though Java Code can run platform independently it requires JVM (Java Virtual Machin) to run on platforms which in turn is platform dependent.



What is JDK?

Java Development kit (JDK) is a development enviroment for building applications, applets & components using the Java Programming language
(

What is IDE? → Intregrated Development Enviroment. We are currently using. (eclipse IDE for Java & DSL Developers 2021-09)

In One class we can't have two main method.

Is Java a 100% Object Oriented Programming language?

⇒ As it contains followings.

1. Primitive Data type : which are non-object types.

2. Static keyword: when we create a class as static then it can be used without creating an object.

3. Wrapper class: Wrapper classes provide mechanism to convert primitive to Object & vice-versa.

Major type of conversion:

Implicit Conversion →

Byte → short → char → Int → Long → Float → Double

← Explicit conversion.

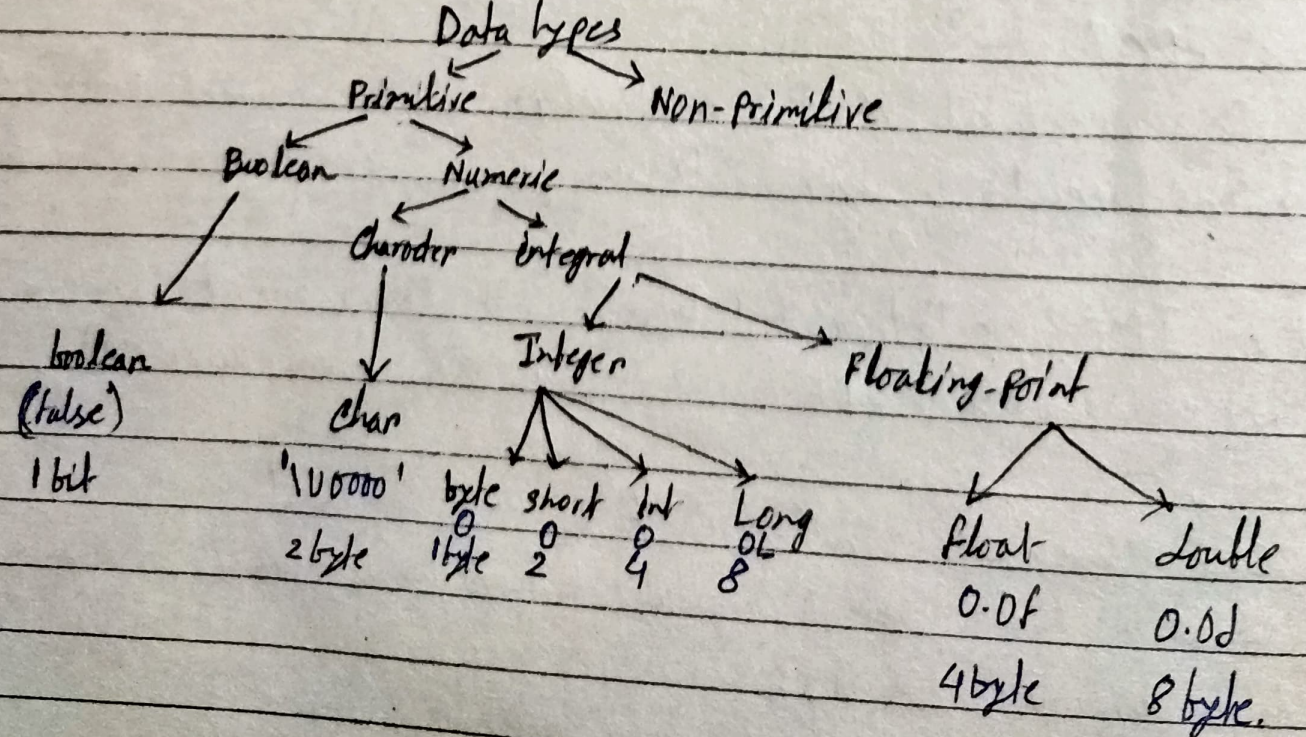
Implicit

```
int a = 57;
long b = a;
Sysout (b) → 57.0
```

Explicit.

```
long b = 57.01 ;
int a = (int) b ;
Sysout (a) → 57 (.01 part lost)
```

How many public classes we can have in a single file?
Only one. We can't keep more than one public class. also filename & class name should be same.



Q) What is Wrapper Class in Java?

→ The Wrapper class in Java provides the mechanism to convert primitive into Object and Object into primitive.

→ chang the value in method
→ Serialization
→ Synchronization
→ java.util package
→ Collection Framwork.

Q) What is Autoboxing & Autounboxing in Java?

→

Autoboxing: Converting a primitive value into an object of the corresponding wrapper class is called autoboxing.
For example, converting int to Integer class.
Java Compiler applies autoboxing when a primitive value is:

- Passed as an parameter to a method which takes object of corresponding wrapper class
- Assigned to a variable of the corresponding wrapper class.

Auto unboxing: Converting an object of a wrapper type to its corresponding primitive type value is called unboxing.
Java Compiler applies auto unboxing when,

- Passed as a parameter to a method that expects a value of Primitive type
- Assigned to a variable of the corresponding

Session-2 / 10/24/2021

Access Modifier: Specifies accessibility or scope of a field, method, constructor, class.

	Within Class	Within Pkg	SP by Subclass	OP by SC	Global
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Default	Yes	Yes	Yes	No	No
Private	Yes	No	No	No	No

Class and Object:

Class: A class is a user defined blueprint or prototype from which objects are created.

Syntax:

~~Public static void Student~~

Public class Employee {

}

- class → Keyword
- Employee → Class name.
- Public → Access Modifier

Object: A Java object is a member (also called an instance) of a Java class. Each object has an identity, a behaviour and a state.

```
Employee Emp-1 = new Employee();
```

"new" Keyword is used to create an instance of the class

What is main() method in Java?

The main() is the starting point for JVM to start execution of a Java Program. Without main() method, JVM will not execute the program.

Why main() method is always public?

The main() is public in Java because it has to be invoked by the JVM. So, if main() is not public JVM would not be able to call it.

What is static method in Java? what's its purpouse?

A static method that belongs to a class rather than an instance of a class (Object). A static method is not part of the objects it creates but is part of a class definition. Unlike instance methods, static methods, a static method is referenced by class name and can be invoked without creating an object of a class.

The static keyword is used to create methods that will exist independly of any instances created for the class. These are generally used to perform an operation that is not dependent upon instance creation.

Why we need to define data types in Java?

Data types are especially important in Java because it is a strogly typed language. This means that all operations are type-checked by the compiler for type compatibility. Illegal operations will not be compiled. Thus, strong type checking helps prevent errors and enhance reliability.

What is garbage collection in Java?

Garbage collection is the process by which Java programs perform automatic memory management. It finds the unused objects (that are no longer used by program) and delete or remove them to free up the memory. (Most popular GC algorithm → Mark and Sweep)

What are non-access modifiers in Java?

Non access modifiers are keywords introduced in Java 7 to notify JVM about a class's behaviour, methods or variables etc. that helps introduce additional functionality.

These are such: **static, final, abstract, synchronized, volatile, & transient.**

What is Object class in Java Programming language?

Object class defined in Java.lang package is the superclass of all other classes defined in Java & every class either directly/indirectly extends from object class.

Non-static methods of Object() → **clone(), toString(), equals(), hashcode(), finalize(), getClass(), wait(), notify(), notifyAll()**

Which access modifier are allowed for a class?

Only public, abstract and final are allowed for a class.

(If we don't specify any access modifier its 'default' and can only be accessed in the same package)

Session-3 / 10-25-2021

OOPs Principles:

Encapsulation: Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them.

Encapsulation in Java is a process of wrapping code and data together into a single unit.

↳ Advantage of Encapsulation in Java:

- By providing getter & setter methods we can make class read only or write only.
- We can set our logic inside setter method to get control over data.
- We can achieve data hiding in Java because other members (classes) will not be able to access data by private attributes.
- The encapsulated class is easy to test so its better for unit testing.

↳ Example of Encapsulation:

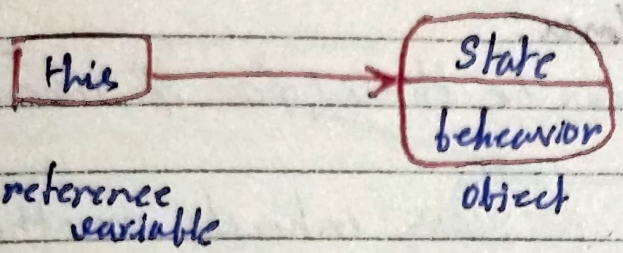
```
Public class Employee{
        int Emp_Id;
        String name;
    }
```

```
Public class Employee {
    private int Emp_Id;
    private String name;

    Public void setEmpId (int EmpId){
        :
    }
    Public int getEmp_Id (){
```

"this" keyword in Java:

There can be a lot of usage of Java this keyword. In Java, this is a reference variable that refers to the current object.



↳ Usage:

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly).
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in method call.
5. this can be passed as an argument in the constructor call
6. this can be used to return the current class instance from the method.

Inheritance: Inheritance in Java is a mechanism in which one object acquires all the properties and behaviour of a parent object. With Inheritance we can create new class that are built upon existing class such that we can reuse methods and fields of the parent class.

Moreover, we can add new methods and fields in our current/child class.

Why to use inheritance:

- For method Overriding (so runtime polymorphism can be used)
- For code reuseability.

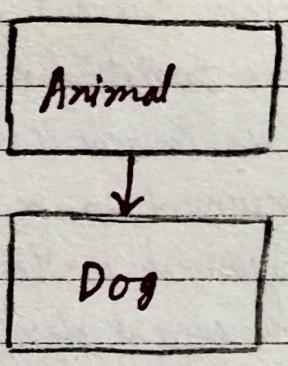
Syntax:

```
class Subclass extends Superclass {
    :
}
```

extends keyword indicates that we are making a new class that derives from an existing class.

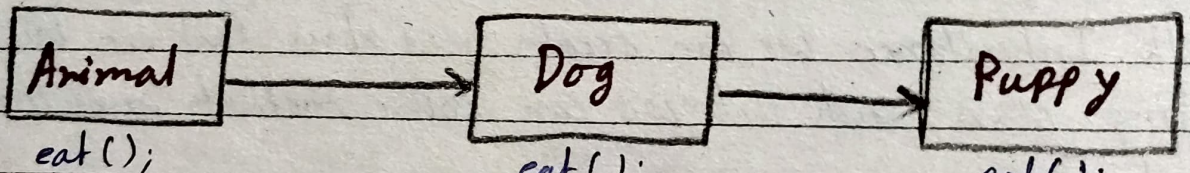
Types of inheritance in Java:
On basis of class, there can be three types of inheritance.

- **Single inheritance:** When a class inherits another class, it is known as single inheritance.

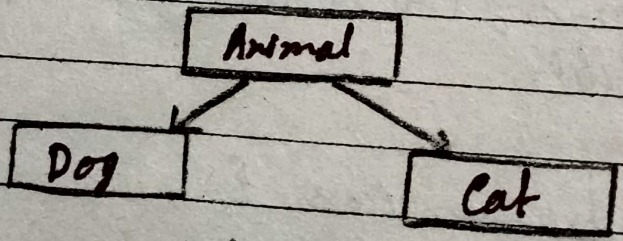


We can use any method (excluding static) in the child class as well as child classes own method.
* We can't use child's method in Parent.

- **Multi-level inheritance:** When we have a chain of inheritance its known as Multilevel inheritance.

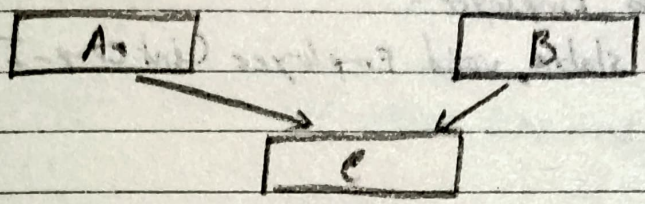


- **Hierarchical inheritance:** When two or more classes inherit same class that is called hierarchical inheritance.



Why multiple inheritance is not supported in Java?

To reduce complexity and simplify the language, multiple inheritance is not supported in Java.



In above scenario if A & B class has same method & child class C called the method it will be ambiguous which method to be called.

Since compile time errors are better than runtime error, Java renders compile-time error if we inherit 2 classes.

Constructor Method: A Constructor method is a special function that creates an instance of the class.

- Typically, Constructor methods accept input arguments to assign the data stored in properties and return an initialized object.
- A class can define a constructor method that overrides the default constructor.
- We can have upto max 65536 methods in a Java class (excluding all constructors)
- We can create a constructor like below.
- We MUST declare a variable if that is to be used in constructor.

```
Public class Employee {
    public static void Employee () {

    }
}
```

The Employee () constructor overrides the default constructor of

Employee class.

- All constructor method names should always be same as class name but can have different sets of arguments.

```
public class Employee {
    public static void Employee (int Emp-ID, String name){

    }

    public static void Employee(int emp-ID, String name, int Emp

    }
```

The no of ~~data~~ arguments / The types of arguments should always be different as if two or more constructors have same no of arg & of same type it causes ambiguity.

X
```
public static void Employee (int emp-ID , String name) {...}
public static void Employee (int emp-ID, String dept) {...}
```

Super Keyword: The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever we create the instance of subclass, an instance of parent class is created implicitly which is referenced by super reference variable.

Usage of Java Super Keyword:

1) Super can be used to refer immediate parent class instance variable.
2) Super can be used to invoke immediate parent class method.
3) Super() can be used to invoke immediate parent class constructor.

Q Can a Java Constructor be static?

→ Java Constructor can not be static. We know static keyword belongs to a class rather than the object of a class. A Constructor is called when an object of a class is created, so no use of the static constructor.

Q Can a Java constructor have void return type method in its signature?

→ No. Java Constructor is a special method which does not return anything. If we put void in its signature explicitly return type is checked and it throws an Compilation error.

Q Can a Java Constructor be Private?

→ If a constructor is declared as private, then its objects are only accessible from within the declared class. It is a special instance constructor used in static member only classes.

Remember only Constructor can have Private access modifier not class.

```
class Animal {
    String color = "Black";
}
class Dog extends Animal {
String color = "white";
void printColor() {
    System.out.println('Dog colour' + color);
    System.out.println('Animal colour' + super.color);
}
}
```

super.color provides color in Animal

- super() should always be the 1st line in child constructor.
- If a parent class constructor has arguments those should be passed in super() else it'll throw error.

Abstraction: Abstraction is a process of hiding the implementation details and showing only functionality to the user.
like when we send a text message we type the text & add the number & send it. We don't know what is going on internally.
- Abstraction lets us focus on what object does instead of how it does it.

There are two ways to achieve abstraction in Java.
1. Abstract class (0 to 100%)
2. Interface (100%)

▷ **Abstract class**: An Abstract class is a generic class declared abstract keyword. It can have abstract & non-abstract methods.

- Its NOT mandatory that an abstract class should have any abstract method but if a class has one abstract method that class should be abstract class.

Points to Remember:

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It CANNOT be instantiated. (No object can be created)
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change body of the method.

Syntax:

Abstract class: abstract class A {}

Abstract method: A method which is declared as abstract and does not have implementation is known as an abstract method.

```
abstract void printStatus();
```

- If we are extending an abstract class that has abstract method, we MUST either provide the implementation of the method or make this class abstract.

2> Interface: An interface in Java is a blue print of a class. It has static and abstract methods.

Interface is used to achieve abstraction and multiple inheritance in Java. We can say that interfaces can have abstract methods and variables. It can not have a method body.

Interface can not be instantiated just like abstract class.

- Since Java 8, we can have default and static methods in an interface.
- Since Java 9, we can have private methods in an interface.

Why use Java interface?

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling. (It means that the classes are independent of each other. The only knowledge one class has about other class is what the other class has exposed through its interface in loose coupling.)

To declare an interface **interface** keyword is used. It provides total abstraction; means all methods inside interface declared with empty body.
A class that implements interface must implement all the methods declared in the interface.

Syntax:

```
public interface <interface-Name>{
    :
}
```

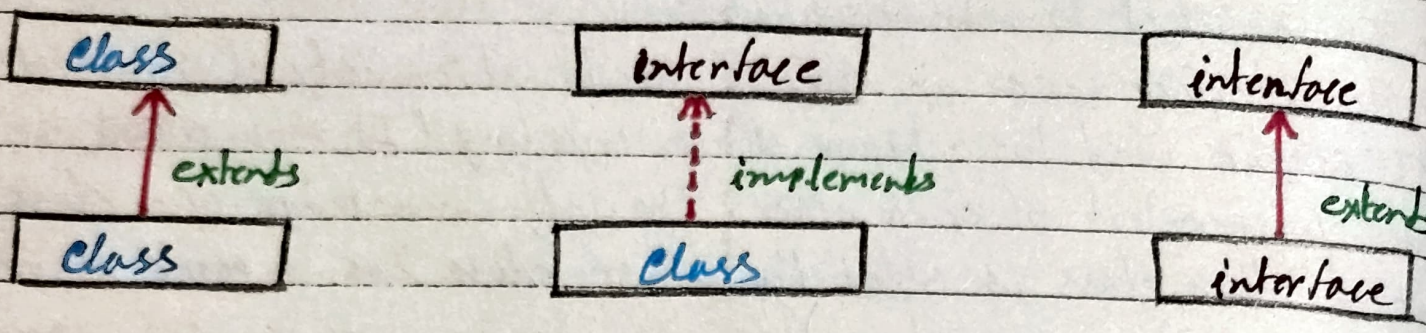
Java compiler adds public and abstract keywords ~~by default~~ before interface methods. Moreover, it adds public, static and final keyword before its data members.

```
public interface Printable{
    int MIN = 5;
    void print();
}
```

→ Compiler →

```
public interface Printable{
    public static final MIN=5
    public abstract void print
}
```

The relationship b/w classes and interface:



Representation with code:

```
Public interface Printable {
    void print();
}

Public class Printer implements Printable {
    Public void print() {
        System.out.println("Hello");
}
```

Abstract Class VS Interface:

Abstract class	Interface.
1) Abstract class can have abstract and non-abstract methods	1) Interface can have only abstract methods. Since Java 8 it can have default and static methods also.
2) Abstract class does not support multiple inheritance.	2) Interface supports multiple inheritance

Abstract class	Interface
3) Abstract class can have final, non-final, static and non-static variables	3) Interface has only static and final variables.
4) Abstract class can provide the implementation of interface	4) Interface can't provide the implementation of Abstract class.
5) The abstract keyword is used to declare abstract class	5) The interface keyword is used to declare interface.
6) An abstract class can extend another Java class and implement multiple Java interfaces.	6) An interface can extend another Java interface only.
7) An abstract class can be extended using keyword "extends"	7) An interface can be implented using keyword "implements"
8) A Java abstract class can have members like private, protected etc.	8) Members of Java interface are public by default.

Example:
public abstract class Shape {
public abstract void draw();
}
	Example:
	public interface Drawable {
	void draw();
	}

Polymorphism: The word polymorphism means having many forms. Simply, we can define polymorphism as the ability of a message to be displayed in more than one form.
Real life example of such would be a man. He can be an Engineer, Husband, Father, Friend anything.

- Polymorphism allows us to perform a single action in different ways.

- **Static/Compile time Polymorphism:** This type of polymorphism is achieved by function overloading or Operator overloading. But Java does not support the Operator Overloading.

Method Overloading: When there are multiple functions with same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by changing no. of arguments or/and changing type of arguments.

```
class Employee {
    public Employee (int emp_ID, String name) {
        :
    }
    public Employee (int emp_ID, String name, String dept) {
        :
    }
}
```

No. of arg chng!

- **Runtime Polymorphism/Dynamic polymorphism:** This is also known as Dynamic Method dispatch. It is a process in which a function call to overriden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Method Overriding : When a derived class has a definition for one of the member functions of the base class. That base function is said to be overriden.

```
class Parent {
    void Print() {
        System.out.println("Parent");
    }
}

class subclass extends Parent {
    @Override
    void Print() {
        System.out.println("Sub class");
    }
}
```

Print() is being overriden by subclass.

What is the significance of writing @Override in Java?

⇒ The @Override annotation indicates that the child class method is over-writing its base class method. It extracts a warning from the compiler if the annotation method doesn't actually Override anything. It can improve the readability of the source code

Conditional Statements in Java:

We have two different types of conditional statements in Java. If-else if-else or Switch case statements.

If-else if-else Statements: Whenever any condition inside is evaluated as true the statement inside that block is run. If else statement can be written in various ways.

- We can write many if statements only. In this way all if blocks are checked.

```
int a=5;
if (a==5){
   System.out.println("a is 5");
}
if (a==7){
   System.out.println("a is 7");
}
```

Why not to write multiple if statements for a value checking?

As we are creating multiple if statements compiler will continue keep checking for a match even if it had a match. So instead if we are checking an value we should be using if-else or if-else if-else for such.

- We can use if-else or if else if else to validate.

```
if (a==5){ Sysout(....)};
else if (a==7) { Sysout(....)}
else { ____ }
```

• **Nested if-elseif-else Conditionals:** We can also create many if-elseif-else blocks in a nested way.

```
int a = 5;
int b = 7;
if (a == 5) {
    if (b == 8) {      }
    else if (b == 0) { ... }
    else { --- }
}
else { .... };
```

Here we could have used && (and) & || (or) to add multiple true & false checks for conditions.

Switch-case Statements: Switch case statements are value checks rather than conditionals as we pass a key and check a value match with cases.

Syntax

```
switch (key) {
    case value:
            // code
            break;
    default:
            // code to be executed if no cases match
}
```

Why we need break statement in case?

If we don't use break statement all the cases in switch statement will get executed. Once control reaches break it skips rest of the cases in switch are get out of switch block.

Iterative loops in Java:

FOR LOOP: For loops are used in java to iterate over finite number of elements present in a collection.

Syntax: for (int i=0; i<10; i++) {

}

```
for (initialization; condition; increment/decrement) {
    // statements
}
```

WHILE LOOP: While loop is used when we are not sure about number of iterations & continue to ~~need~~ run looping until cond is false.

initialization — int i = 0

looping — while (i<5) {

System.out.println(i);

increment/decrement — i++;

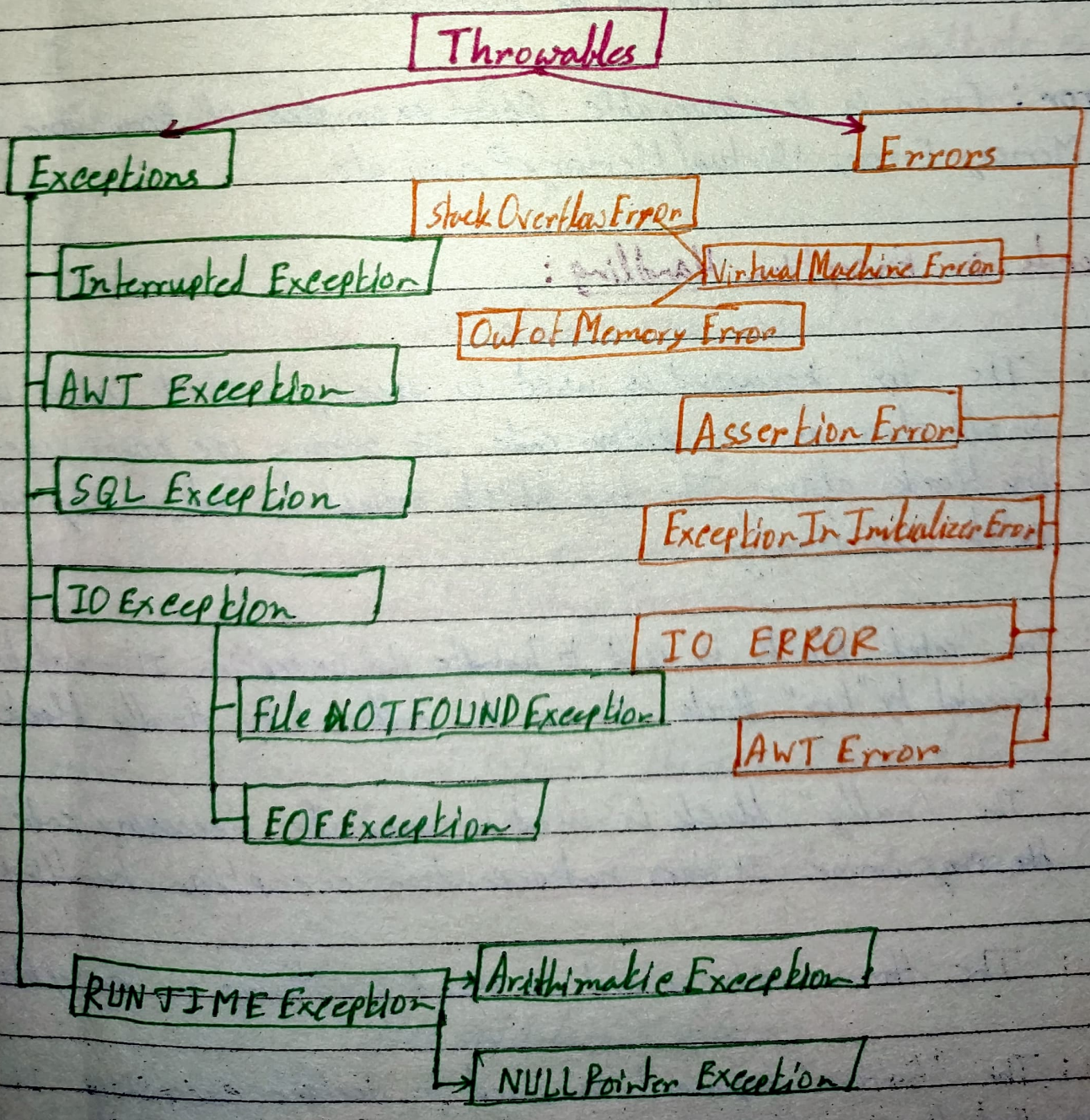
}

DO-WHILE: DO-WHILE loop is similar to while loop with one key difference i.e. when the condition is false one the loop gets executed as condition is checked after execution.

```
do {
    System.out.println(j);
    j++;
} while (j<5)
```

Exceptions in Java: In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.



1> **Checked Exception**: The class directly inherit Throwable class except RuntimeException and Error are known as checked exceptions. These are checked at compile time. Eg: IO Exception, SQL Exception

2> **Unchecked Exception**: The classes inherit the RuntimeException are known as unchecked exceptions. Eg: Anithimatic Exceptions, Nullpointer Exceptions. Unchecked Exception does not come up at compile time rather are thrown at runtime.

3> **Error**: Error is irrecoverable. Some examples of Error are Out of Memory Error, Virtual Memory Error etc.

Keywords in Exception Handling:

try: The 'try' keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.

catch: The "catch" block is used to handle the exception. It must be preceded by "try" block. It can be followed by finally block.

finally: The "finally" block is used to execute necessary code of the programme. It does not depend on exception handling.

throw: The throw keyword is used to throw an exception.

throws: The "throws" keyword is used to declare exceptions. It specifies that there may occur an exceptions in the method. It does not throw an exception. It is always used with method signature.

Structure of Try-Catch-Finally block.

```
try {
    // Code that is to be tried & might throw Exception.
} Catch (IOException e){
.
} Catch (RunTime Exception R){
}
 Catch(Exception I){
}
finally{
    // Code that will always run wheater exception handled or
}   not.
```

More specific exceptions always comes first while using multiple "catch" block.

Q Create a Main class & Employee class. Employee class should have setSalary(s). Throw custom exception from this method if Salary input is negative (InvalidSalaryException). Catch it in Main class.

```
Public setSalary (int Salary) Throws InvalidSalaryException{
        if (Salary <1){
            Throw new InvalidSalaryException();
        }
        else{
            this.setSalary = Salary;
        }
}
```