JAVA

- A general purpose programing language.
- It is like C++ but with advanced features and simplified features
- Concurrently executes many statements (use multicore)
- Platform independent

Why?

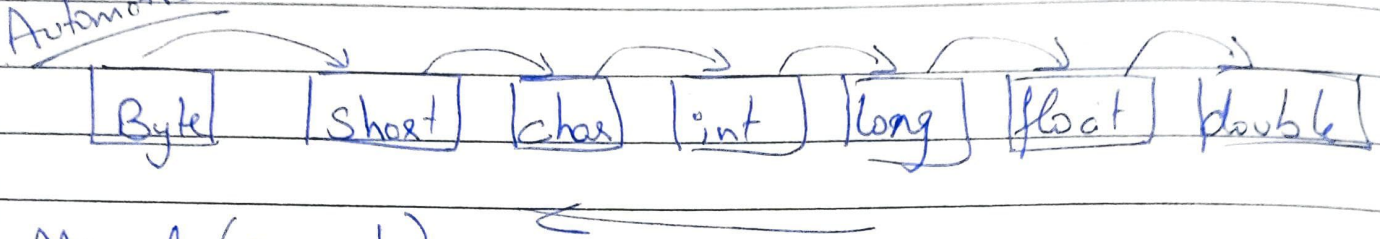
- Easy to learn
- Common language (world wide)
- Large Community
- Java is used at many places like: Servers, IOT, Android, BigData, Continuous Testing, Banking, Communication & many more.
- Java tools like IDE & built in API features make life easier.

How Java Works

Java Source Code (.java) → Java Compiler → Byte Code (.class) → JVM (Mac), JVM (Windows), JVM (Linux)

Type Casting (Widening)

Type Casting is when you assign a value of one primitive data type to another type.



Manual (opposite)

eg `double x = 3.14`
`int y = (int) x;`

Scanner

```
Import java.util.Scanner;
Scanner sc = new Scanner (System.in)
  int x = sc.nextInt();
```

for string : If u next() or nextline()

{ 1 next() is necessary if there u any next before scanning string becz it take enter in string. }

for float

`float x = 12.5f`

The f u necessary else it would be considered as double

1. Arithmatic (+, -, /, *, %)
2. Bitwise (&, |)
3. Assignment (+=)
4. Comparison (==, >, <)
5. Logical (||, &&, !)

Arrays

Array is a data structure used to store multiple values in a single variable, instead of declaring seprate variable for each value.

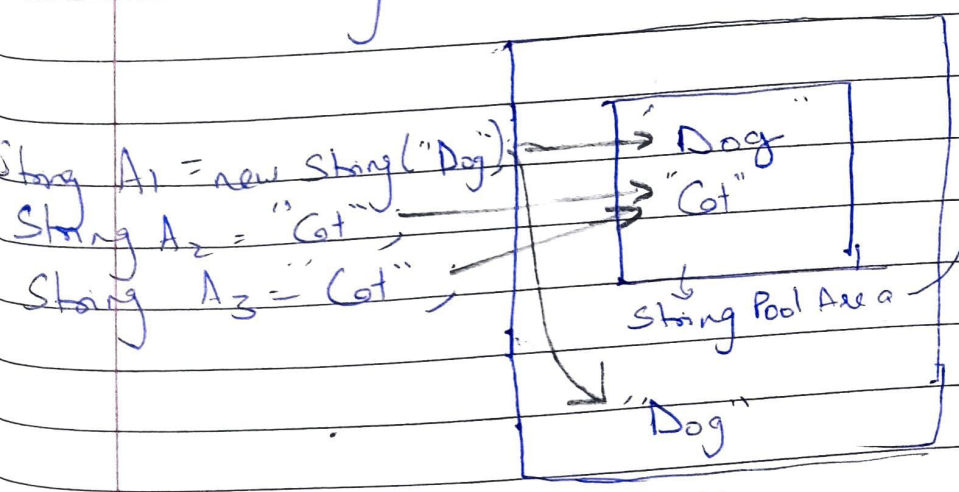
```
datatype [] arrayName
arrayName = new datatype[n]

datatype[] arrayname = new datatype[n]
```

* n cannot be changed once defined
* By default array element has 0 value

String

String is a class

```
String name = "Dev"; (By Literal)
String anotherName = new String("Code") (By object)
```



Strings are immutable, ie strings cannot be deleted

eg String name = "Dev"

name = "Divyansh"

In this Dev is not deleted, & new Divyansh is created in stringpool area.

Function in Strings (* All operation return new String)

1) char charat (int index)
2) int length ();
3) String substring (int begIndex) // return string till end
4) String substring (int begInd, int endInd)
5) boolean contains (CharSequence s)
6) boolean equals (object anotha) // check value
7) boolean isempty () // Return True if string is empty
8) String concat (String str);
9) String replace (char old, char new)
10) int indexOf (char ch) Return pos^n of first character / else -1
11) String toLowerCase ();
12) String toUpperCase ();
13) String trim ();
    ↓ remove trailing spaces from starting & end.
14) String [] split (String regex)
    → Returns array of string
    → If we want to split about particular character

eg String cars = "Swift, Wagonr, ilo";
String allcars = cars.split (",");

Note Character is a primitive data type & primitive data type cannot be dereferenced by dot (.) operator. We use substring to access character in string & modify it.

To convert int to String

```
int i = 10
String s = String.valueOf (i);
```

OR `String s1 = Integer.toString (i);`

OOPS

Objects

→ Objects have some state & behavior
→ An object can be defined as an instance of a class
→ An object contains an address and takes up some space in memory.
→ eg → a white board, blue pen etc

Class

→ A class is a blueprint for the object
→ It is logical entity
→ Class doesn't consume any space
→ We can create many objects from a class.
eg - pen class, laptop class

Methods

→ Method Overloading
When a class has two or more methods by the same name but different parameters

eg

```
int maxOf(int a, int b) { };
int maxOf(int a, int b, int c) { };
int maxOf(int a, float b) { };
```

* JAVA is pass by value always
→ for both primitive & non-primitive datatypes

Constructor

→ It is of method type (structure)
→ Doesn't return anything nor have any return type
→ Name is equal to that of class

A constructor is similar to method (but not actually a method) that is evoked automatically when an object is instantiated.

eg

```
Class Test {
    Test() {                    // Does not have return type
        // Constructor body
    }
}
```

→ If constructor is not defined then automatically compiler defines a default Constructor in which all default values are set like for int it 0, bool = false, object points to NULL

→ If we define constructor then default constructor cannot be called, we must define default constructor

2 types → Non argument Constructor
→ Argument Constructor

★ this Keyword provides references or points to current object

eg

```
Class Vehicle {
    int wheels
    Vehicle (int wheels) {
        this.wheels = wheels;
    }
}
```

Constructor Overloading

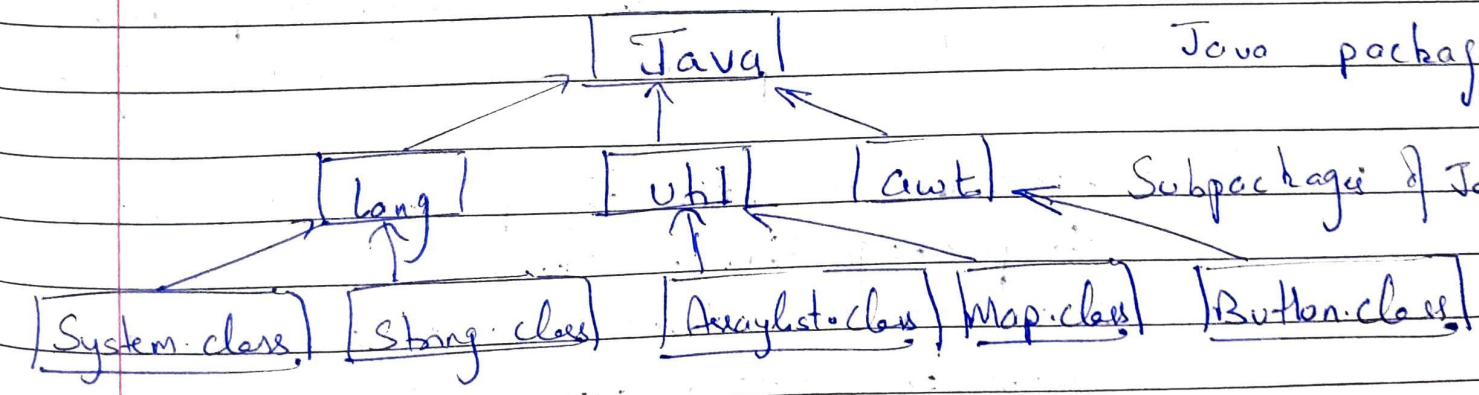
Similar like method overloading, you can also overload constructor if two or more constructor are different in parameter

eg

```
class Test {
    int a;
    Test() {
        a = 5;
    }
    Test(int b) {
        a=b;
    }
}
```

Packages

A Java package is a group of similar type of classes, interfaces and subpackages. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.



Java package

Subpackage of Java

Java pa

* We cannot have 2 classes of same name in 1 package

Advantages

1) Java package is used to categorize the classes & interfaces so that they can be easily maintained
2) Java package provides access protection
3) Java package removes naming collision

Access Modifiers

The access modifiers in Java specifies the accessibility or scope of fields, methods, constructor or class. We can change the access level of them by applying access modifiers on them.

4 Types of access specifiers

Private : The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

Default :→ Access level is within the package
→ Cannot be accessed from outside the package
→ If we do not specify any access level, it will be default.

Protected :→ Access level is within the package & outside the package through child class.
→ If we do not make any child class, it cannot be accessed from outside the package.

Public :→ Access level is everywhere.

Date / /

Encapsulation

It is a process of wrapping code and data together into a single unit → Java packages
→ Access Modifiers

Why Encapsulation

- In Java, encapsulation helps us to keep related field & methods together which makes our code cleaner & easy to store
- It helps to control the modification of our data fields
- It helps to decouple components of a system. These decoupled can be developed, tested & debugged independently & concurrently. And ~~they~~ any changes in particular component do not effect on other component

How to achieve encapsulation

1. Declare the variables of a class as private
2. Provide public getter & setter methods that will be used to write & read the variable values.

Data hiding v/s Encapsulation

→ Encap. refers to bunding of related feilds & methods together.
→ This allows us to acheive data hiding
→ Encapsulation in itself is not data hiding

Example Bank System
Acc No → variable
ROI → Static variable

DOMS | Page No.
Date / /

→ always belongs to a class, not to an object

Static Keyword (used for memory management)

The static keyword indicates that the particular member belongs to a type itself, rather than to an instance of that type. It is mainly used to help memory management.

* The keyword can be applied to variable, methods, blocks, and nested class.

* We cannot declare static keyword in non-static method
* Static keyword can't be declared in non-static inner type

```
Static int count;  → (accessed by className) eg → ClassName.count
Static string CompName = "Google";
Math.PI
Static int square (int n) {
        return n * n;
}
```

Static Class

We cannot declare a top level class with static modifier, but we can declare a nested class as static. (Static Inner Class)

In Java you can define a class, within another class. Such a class is called nested class. Nested class are of two types → static & other is → non-static

⚠ We cannot create static variable in a function

DOMS | Page No. | Date / /

```
class outer class {
    static class staticNested Class {
    }
    class Inner Class {
    }
}
```

eg
```
public class A {
    class B { }
    static class C { }
}
```

Benefit of static nested class is that we can access that class outside the class as it is outside parent & not nested

Here, diff. b/w B & C is that if we want to make object of class B, then we have to first create obj of A & then B. but we can directly make object of C.

eg
```
A objA = new A();
A.B objB = objA.new B();
A.C objC = new A.C();
```

Static Block
```
Static {
    SysOut ("Block 1");

public static void main (string [] args) {
    SysOut ("Main");

Stat {
    Sys out ("Block 2");
}
```

OUTPUT
Block 1
Block 2
Main

Inheritance

It is an is-a relationship We use inheritance only if is-a relationship is present b/w two classes

eg

A car is a vehicle
Orange is a fruit

1 parent can have multiple children but a single children can only have 1 parent

eg

```
public class Teacher extends Person {
                ↑                ↑
              child            parent
}
```

Method Overriding

If the same method is defined in both the superclass (parent) and subclass (child) then the method in subclass overrides the method in the superclass

Upcasting & Down Casting

Suppose Animal is parent & Cat is child

```
Cat c = new Cat();
Animal a = c;
```

} UPCASTING / Implicit Casting

```
Cat c = new Cat();
Animal a = c;
Cat c1 = (cat) a;
```

} DOWNCASTING / Explicit

boolean y = t instanceof teacher
return boolean
eg SysOut (" c instance of Cat");

Super Keyword

It is used to refer to immediate parent class of a child class. In other words, super keyword is used by a subclass whenever it needs to refer its immediate superclass. eg - super.eat()

If we make any new constructor in parent class then in every child class we need to create constructor which calls constructor of parent class with the help of super keyword
eg super(name)

Singleton Pattern

It is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.

So in this we need create only 1 object of a class which stores data & holds it in RAM.

```
public class AppConfig {
    private AppConfig() { };
    private static AppConfig obj = Null;
    public static AppConfig getInstance() {
        if (obj == null) {
            obj = new AppConfig();
        }
        return obj;
    }
}
```

```
public class mainClass {
    public static void main (String [] arg) {
        AppConfig obj1 AppConfig.getInstance();
        AppConfig obj2 AppConfig.getInstance();
    }
}
// In this both objects are same
```

* Java does not support multiple inheritance because two classes may define Different ways of doing the same thing, and the subclass choose which one to pick. It induces diamond problem.

Polymorphism

It allows us to perform one action in diff ways or
It allows you to define one interface & have multiple implementations

	Run-time polymorphism	Compile Time polymorphism
1.	It is also known as dynamic binding, late binding & overriding as well.	It is also known as static binding, early binding & overloading as well.
2.	Slower execution	faster execution

3. Overriding in run-time polymorphism have same method with same parameters or signature but associated in a class & its subclass

Overloading in compile time poly, where more than one methods share same name with diff parameters or signature and different return type

Abstraction

- Data abstraction is the process of hiding certain details and ~~sharing~~ showing only essential information to the user.
- It helps to reduce programming complexity & effort
- Abstraction can be achieved with either abstract classes or interfaces

Abstract Class

- A class that is defined using "abstract" keyword
- It can have abstract methods (without body) as well as concrete methods (with body)
- A normal class cannot have abstract methods
- An abstract class cannot be instantiated, i.e. we cannot create its object.

- An abstract class must be extended and its abstract methods must be overridden

Abstraction V/S Incapsulation

Abstraction is about hiding unwanted details while showing most essential information.	Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons
Abstraction allows focussing on what the information object must contain	Encap: means binding the code & data into a single unit
Abstraction solves the issues at the design level	Encap: solves it at implementation level

★ Reference variable can be created of abstract class

eg

if A is abstract & parent class,
B is child or inherited class,

```
B obj = new B();
A obj1 = new B();
```

Final Keyword

You can assign final keyword to variable, classes & methods.

1. A final variable cannot be changed
2. A final method cannot be overridden
3. A final class cannot be extended.

Final Variable

→ can only be initialized once.
→ We must initialize a final variable, otherwise compiler will throw compile time error
→ It is a good practice to name final var in all CAPS
→ A final var is called blank final variable, if it is not initialized while declaration
→ In case of reference final variable, internal state of the object pointed by that reference variable can be changed

Final Method

If there are final method in parent class then children class can't override them

Date / /

Interface

An interface defines a set of specification that other classes must implement

```
interface Polygon {
    public void getArea();
}
```

By default fons in interfaces are public & abstract

Here, Polygon is an interface. We have used the interface keyword to ~~create~~ declare an interface

We can implement interfaces in other classes. In Java we use implements keyword to implement interfaces

```
class Rectangle implements Polygon {
    public void getArea() {
        int len = 5;
        int breadth = 6;
        SysOut ("area =" + (len * breadth));
    }
}
```

Why use interface ?

→ Interface provides specification that a class (which implements it) must follow.

→ Similar to abstract classes, interfaces helps to achieve abstraction in Java

→ In Java, multiple inheritance is not possible by extending classes. However, a class can implement multiple interfaces. This allows us to get the functionality of multiple inheritance in Java. for eg:

```
public abstract interface line {

}
public abstract interface polygon {

}
class Rectangle implements line, polygon {

}
```

few important points

1. We can't instantiate an interface in Java.
2. Interface can't have constructors becoz we can't instantiate them, & they dont have method body
3. By default any attribute of interface is public, static and final, so we dont need to provide access modifiers to the attribute but if we do, compiler doesn't complain about it either
4. By default interface methods are implicitly abstract & public, it makes total sense becoz the method don't have body & so that subclasses can provid method implementation
5. An interface can't extends any class but it can extend another interface
6. A class implementing an interface must provide implementation for all of its method unless its an abstract class

Exception Handling

Exception is an error event that can happen during the execution of a program and disrupts its normal flow.
Exception in Java can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure, database server down etc

Java provides a robust and object oriented way to handle exception scenarios, known as Java Exception handling.

Throwing an Exception

→ Java being an OOPL, whenever an error occurs, creates an exception object. The exception object contains a lot of debugging info, such as method hierarchy, line number where the exception occured, type of exception etc.

→ The normal flow of the program halts and JRE tries to find someone that can handle the raised exception

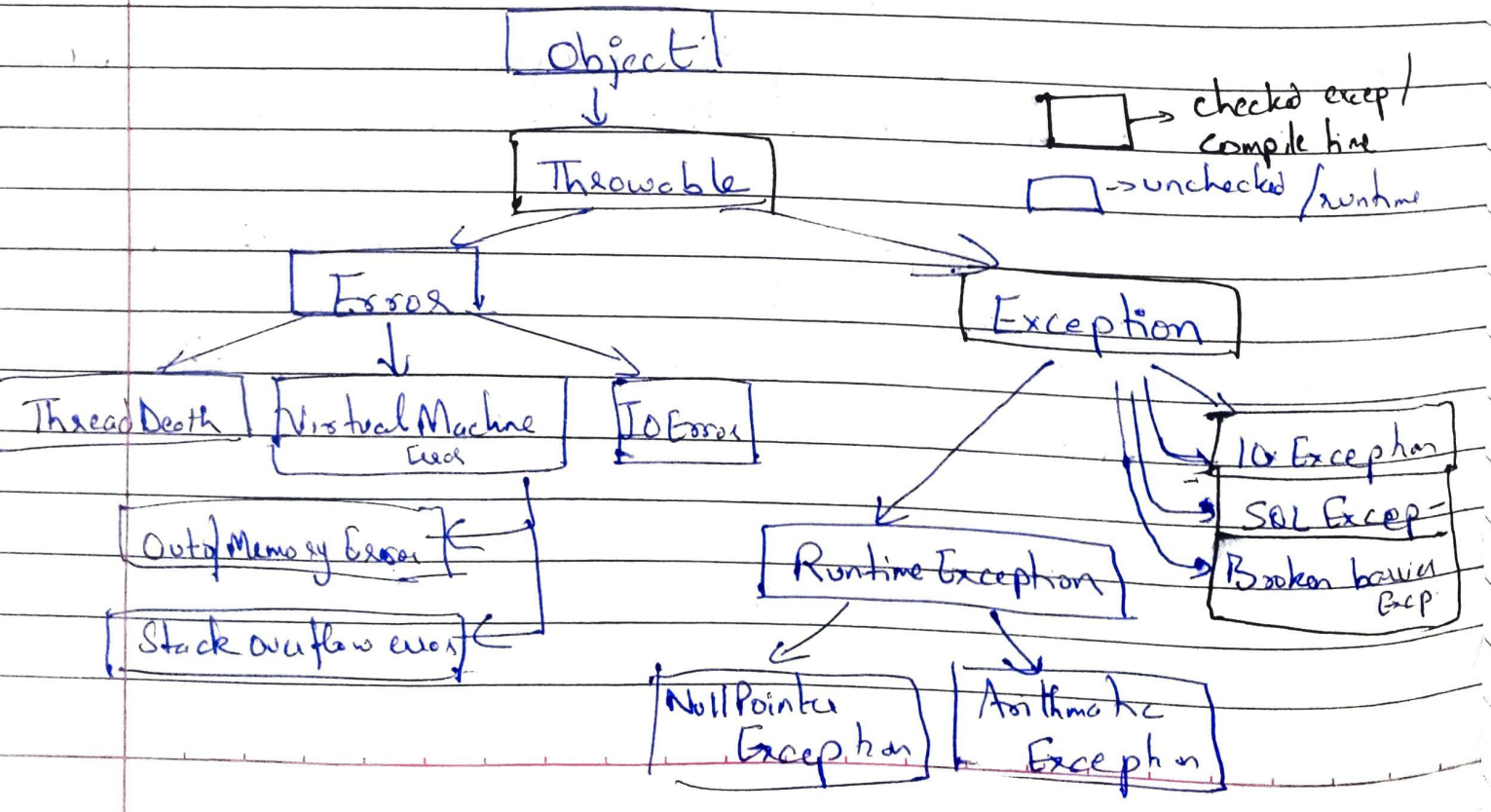
→ When the exception occurs in a method, the process of creating the exception object and handling it over to runtime environment is called "throwing the exception".

Catching the exception

→ Once runtime recieves the exception object, it tries to find the handler for the exception. Exception handler is the block of code that can process the exception object.

→ The logic to find except^n handler is simple - starting the search in the method where error occourred. If no appropriate handler found, then move to caller method & soon. So if methods call stack is A → B → C & exception is raised in method C, then the search for appropriate handler will move from C → B → A.

→ If appropriate exception handler is found, except^n object is passed to handler to process it. The handler is said to be "catching the exception".



finally keyword

It always run whether we find catch or not

```
try {
}
catch ( ArithmeticException e) {
}
catch ( ArrayIndexOutOfBoundsException e) {
}
finally {
    System.out.println("Sorry");
}
```

Always run

Throw & Throws

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code.

The throws keyword in Java is used in the signature of method to indicate that this method might throws one of the listed type exceptions. The called to these methods has to handle the exception using a try-catch block.