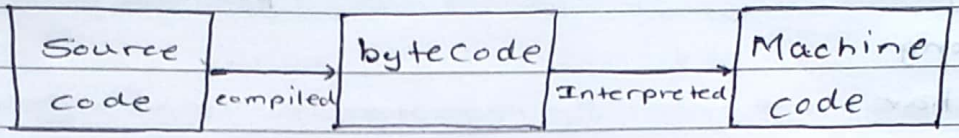
Page No.
Date

* Java *

- It is purely Object Based.

- How Java works?



- **Basic Structure of a Java Program :-**

```
package com.company;  ⟶ Group of classes
                        Enterpoint into the
public class main {     applications.
    public static void main (String [] args) {
        System.out.println("Hello World");
    }
}
```

• Variables and Datatypes *

- Variable is a container that stores a value.

eg :-

```
int number = 8;
```

- `int` → data type
- `number` → variable name
- `8` → Value it stores.

- **rules:-**
  1. Must not start with a digit
  2. Name is case sensetive
  3. Should not be a keyword
  4. White space not allowed.

Page No.
Date

- Data types:-

- Primitive
  1. byte
  2. int
  3. float
  4. long
  5. char
  6. double
  7. bool
  8. short

- Non - Primitive
  ↓
  Derived from primitive data type

* Program for addition of 3 numbers:-

```
public class Main{
    public static void name (String[] args){
        System.out.println("Addition is :");
        int num1 = 1;
        int num2 = 2;
        int num3 = 3;
        int sum = num1 + num2 + num3;
        System.out.println ("Sum);
    }
}
```

- Literals:-

- A constant value which can be assigned to the variable is called as a literal.

Page No.
Date

* Operators :-

- operators are used to program operations on variable and values.

- **Types of operator :-**

1. Arithmetic operators :- +, -, *, /, %, ++, --
2. Assignment operators :- =, +=
3. Comparision operator :- ==, >=, <=
4. Logical operator :- &&, ||, !
5. Bitwise operator :- &, | (operates bitwise).

+ Arithmatic operators cannot work with boolears.
+ % operator can work only float and doubles.

- **Precedence and associativity :-**

- Precedence means which operator will be operated first on behalf of the following table.

operator	Associativity	precedance
!, ++, --, &	Right to left	13
*, /, %	Left to right	12
+, -	Left to right	11
<<, >>	Left to right	10
<, <=, >, >=	Left to right	8

Page No.
Date

- Increment and Decrement operators:-

a++ , ++a ⟶ Increment operator
a-- , --a ⟶ Decrement operator

These will operate on all data type except boolean

a++ ⟶ first use the value and increment.
++a ⟶ first increment the value and use it.

* Strings:-

- Strings is ~~Java are~~ a sequence of characters.
- A string is instantiated as follows:-

```
String name;
name = new String ("Harry");
```

- String is a class but can be datatype.
- Strings are immutable and cannot be changed.

- Different ways to print in Java

1. System.out.print() ⟶ No newline at end
2. System.out.printf()
3. System.out.println() ⟶ Newline at end.
4. System.out.format()

format specifier.
- %d for int
- %f for float
- %c for char
- %s for string

Page No.
Date

String Methods:-

- String Methods operate on Java string
- They can be used to find length of the string convert to Lowercase, etc.

String = "Hello";

1. name.length() → Returns length of string.
2. name.toLower() → Returns new string which has all Lower cases.
3. name.toUpper() → Returns a new string which has all upper case.
4. name.trim() → Returns a string after removing all leading and trailing spaces.
5. name.substring(int start) → Returns a substring from start to the end substring.
6. name.substring(int start, int end) → Returns a substring from start index to end index.
7. name.replace('e', 'a') → Returns new string by replacing all 'e' by 'a'
8. name.startsWith("He") → Return Boolean expression.
9. name.endsWith("lo") → Return Boolean expression
10. name.charAt(2) → Returns character present at the given index

(11) name.indexOf("e",3) → returns the index of the given string starting from the index 3

(12) name.lastIndexOf("e",2) → returns the last index of the given string before index 2

(14) name.lastIndexOf("e") → returns the last index of the given string.

(15) name.equals("Hello") → Returns Boolean expression [case sensetive].

(16) name.equalsIgnoreCase("Hello") → Returns Boolean expression ignoring the case of characters.

- Escape Sequence characters :-

- Sequence of characters after backslash '\'

eg: \n → new line
\t → tab
\' → single quote.
\\ → backslash.

- Conditions in Java :-

(1) If - Else Statement
(2) Switch Statement.

① If - Else Statement :-

- Syntax :-

```
if (condition - to - be - checked) {
        statement - if - condition - true;
    }
else {
        statement - else - condition - false;
    }
```

- Code Example :-

```
int a = 17;
if (a > 18) {
        System.out.println("Yes");
}
else
{
        System.out.println("No");
}
```

⇒ Output :- No.

- Note :- Else block is optional.

- Relational operator in Java :-

- Relational operators are used to evaluate conditions inside the if ~~else~~ statement.
- Some of Relational operators are :-

==, >=, >, <, <=, !=

The condition can either be true or false.

Page No.
Date

- **Logical Operators :-**

&& → AND
|| → OR
! → NOT

} Most commonly used logical operators in Java.

① **AND operator:-**

Y && Y = Y
Y && N = N
N && Y = N
N && N = N

② **OR operator:-**

Y && Y = Y
Y && N = Y
N && Y = Y
N && N = N

③ **Not operator:-**

! Y = N
! N = Y

- **Switch case Control Instruction:-**

- Switch case is used when we have to make a choice betⁿ number of alternatives for a given variable.

```
Switch (var) {
    case c1:
        //code;
        break;
    case c2:
        //code;
        break;
    case c3:
        //code;
        break;
    default:
        //code;
}
```

- Var can be an integer, character or String in Java.

Loops :-

- Types :-

1. while loop
2. do-while loop
3. for loop.

1. while loop :

```
while (boolean expression)
    {
        //statement
    }
```

→ This keeps executing as long as condition is true.

- If the condition never becomes false, the while loop keeps getting executed. It is known as infinite loop.

Page No.
Date

② do-while loop:-

- This loop is similar to a while loop except the fact that it is guaranteed to execute at least once.

```
do {
    //code
} while (Condition);
```

while → checks the condition & executes it.
do-while → Executes the code & then check condition.

③ for Loop:-

- A for loop is usually used to execute a piece of code for specific interval of times.

- ```
for (initialize; check_bool_exp_; update) {
        //code
}
```

• Decrementing for loop:-

```
for (i=7; i>=0 , i--) {
    System.out.println(i);
}
```

→ O/P:-
7
6
5
4
3
2
1
0
```

Page No.
Date

• **break statement:-**

- The break statement is used to exit the loop irrespective of whether the condition is true or false.

• **Continue Statement:-**

- The continue statement is used to immediately move to the next iteration of the loop
- The control is taken to the next iteration thus skipping everything below "continue" inside the loop for that iteration.

• **In a Nut Shell:-**

+ break statement completely exits the loop.
- Continue statement skips the particular iteration of the Loop.

* **Arrays:-**

- Array is a collection of similar types of data.
- dataType [ ] ArrayName;

eg:-

```
int [ ] marks = new int [5]
```

- Accessing Array Elements:-

Array elements can be accessed as follows.

```
marks [0] = 100
marks [2] = 90
    ⋮
marks [4] = 60.
```

Page No.
Date

- So in nutshell, this is how array works:-

1) int [ ] marks; → Declaration
marks = new int [5]; → Memory allocation.

2) int [ ] marks = new int [5] → Declare + memory allocation

3) int [ ] marks = {100, 70, 80, 90, 75} → Declare and Initialize

- **Array Length:-**

- Arrays have a length property which gives the length of the array.

marks.length

- **Displaying an Array:-**

- A array can be displayed using a for loop:

```
for (int i = 0; i < marks.length ; i++)
{
    Sout (marks[i]); ⟹ Array Traversal.
}
```

- **for-each loop in Java:-**

- Array elements can also be traversed as follows:-

```
for (int element : Arr){
    Sout (element);
}
```

Page No.
Date

• Multi-dimentional array:-

- Multidimentional arrays are array of arrays.

*

+ Multidimentional 2D-array:-

- A 2D-array can be created as follows.

int [][] flats = new int [2] [3];

- similarly 3D-array can be created as:-

int [][][] flats = new int [1] [2] [3] flats

* Methods in Java:- [DRY]

- we can use methods to avoid repeating the logic.

- Syntax of Method:- [using static].

```
dataType name ( ) {
        // Method body
}
```

- foll. method returns sum of 2 numbers:-

```
int mySum (int a, int b) :{
        int c = a+b;
        return c;
}
```

(int: return type; return c; ⟶ Return value)

// + Calling a Method:-

- A method can be called by creating an object of the class in which the method exists followed by the method call.

Page No.
Date

Calc obj = new Calc(); → Object creation

obj.mysum(a, b); → Method call upon an object.

+ Void return type:-

- when we don't want our method to return anything we use void as the return type.

+ Static Keyword:-

- Static keyword is used to associate a method of a given class with the class rather than the object.

- Static method in a class is shared by all the objects.

Note:- In case of Arrays, the reference is passed. Same is the case for object passing to methods.

• Method Overloading:-

- Two or more methods can have same name but different parameters. Such methods are called overloaded method.

```
void food()
void food(int a)
void food(int a, int b)
```
} Overloading function food.

Page No.
Date

Note:- Method overloading cannot be performed by changing the return type of methods.

- Variable Arguments:- (Varargs)

- A function with vararg can be created in Java using the foll. syntax:-

```
public static void food (int...arr)
{
    //arr is available here as int[] arr
}
```

- food can be called with zero or more arguments like this
food(7) food(7, 8, 9) food(1, 2, 7, 8, 9)

- We can also create a function bar like this.

```
public static void bar (int a, int arr).
{
    //Code
}
```

→ Atleast one integer is required no cc.

bar can be called as bar(1), bar(1, 2), bar(1, 2, 7, 11)

- Recursion:-

- A function in Java can call itself. Such calling of function by itself is called recursion.

- Eg:- factorial of a number:-

factorial(n) = n * factorial(n-1).

Page No.
Date

* Introduction to OOPs *

- Object oriented programming tries to map code instructions with real world making the code short and easier to understand

- **Class :-**
- A class is a blueprint for creating objects.

- **Object :-**
- An object is an instance of an class
- When a class is defined, a template (info) is defined.
- Memory is allocated only after object instantiation.

- **OOPs Terminology :-**

1) **Abstraction :-**
- Hiding internal details
- Show only essential info.

2) **Encapsulation :-**
- The act of putting various components together (in a capsule)
- In Java, encapsulation simply means that the sensitive data can be hidden from the users.

Page No.
Date

3). Inheritance:-

- The act of deriving new things from existing things.

4). Polymorphism:-

- One entity, many forms.

• Writing a custom class:-

```
public class Employee {
    int id;
    String name;
}
```

Object in Real world = Properties + Behaviour
Object in OOPs = Attributes + Methods

* Access Modifiers and Constructors:-

• Access Modifier:-

- Specifies where a method is accessible.
- Types:-

① Private
② Default
③ Protected
④ Public.

• Getters and setters:-

- Getter - Returns the value [accessor]
- Setter - Sets/updates the value [mutator]

Page No.
Date

• Constructors in Java:-

- A member function used to initialize an object while creating it.

```
Employee gg = new Employee();
gg.SetName("GG");
```

In order to write our own constructor, we define a method with name same as class name

```
public Employee() {
    name = "Your Name";
}
```

Note:- ① Constructors can take parameters without being overloaded.

② There can be more than two overloaded constructors.

* Inheritance:-

- Inheritance is used to borrow properties and methods from an existing class.

• Declaring Inheritance in Java:-

- Inheritance in Java is declared using extends keyword.

- when a class inherits from a superclass, it ~~helps~~ inherits parts of superclass method and field Java doesn't support multiple inheritance i.e 2 classes cannot be super-classes for a subclass.

• Constructor in Inheritance:

- when a derived class is extended from a base class, the constructor of the base class is executed first followed by the constructor of the derived class.
- for the foll. inheritance hierarchy, the constructors are executed in the order

C_1 → Parent
↓
C_2 → child
↓
C_3 → Grand-child

Constructor execute in top to bottom order.

• Constructors during Constructor overloading:-

- when there are multiple constructors in the parent class, the constructor without any parameters from the parent class, we can use super keyword.

super (a, b); → calls the constructor from parent class which takes 2 variables.

- **this keyword:**

- this is a way for us to reference an object of the class which is being created / referenced.

- this.area = 2 → this is a refence to current object.

- **Super keyword:-**

- A reference variable used to refer immediate parent class object.
- → Can be used to refer immidiate parent class instance variable.
- → Can be used to invoke parent class method.
- → Can be used to invoke parent class constructors.

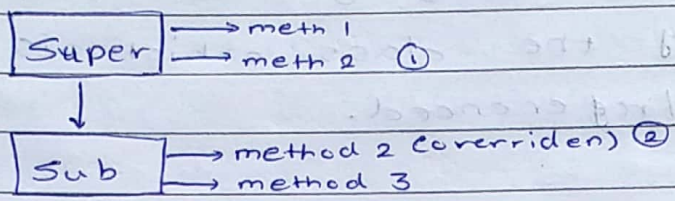
- **Method overiding:-**

- If the child class implements the same method present in the parent class again, it is known as method overriding.
  - ↳ Redefining method of super class (in subclass).
- when an object of subclass is created and the overrided method is called, the method which has been implemented in the sub-class is called and it code is executed.

Page No.
Date

• **Dynamic Method Dispatch:-**

- Consider the foll. inheritance hierarchy:-



- Scenario 1 → Super obj = new Sub() → //Allowed.
  obj.meth2() → ② is called
  obj.meth3() → Not allowed.

- Scenario 2 → Sub obj = new Super(); → Not Allowed

* **Abstract Classes and Interfaces:-**

+ **Abstract Method:-**

- A method that is declared without an implementation
- abstract void moveTo (double x, double y)

•

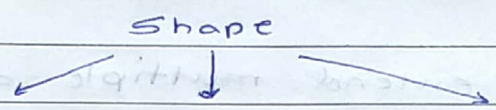
+ **Abstract Class:-**

- If a class includes abstract methods, then the class itself must be declared abstract.

```
public abstract class PhoneModel {
    abstract void switchOff();
    //more code
}
```

- When an abstract class is subclassed, the subclass usually provides implementations for all of the methods in parent class. If it doesn't it must be declared abstract.

An eg:-



- Note:- It is possible to create reference of an abstract class.
  :- It is not possible to create an object of an abstract class.

- We can also assign reference of an abstract class to the object of a concrete subclass.

- Interfaces in Java:-

- In Java interface is a group of related methods with empty body.
- eg:-

```
interface Bicycle {
    void applyBrake(int decrement);
    void speedUp(int increment);
}
class AvonCycle implements Bicycle {
        int speed = 7;
        void applyBrake(int decrement){
            speed = speed - decrement};
        }
        void speedUp(int increment) {
            speed = speed + increment;
        }
}
```

Page No.
Date

• Abstract class vs Interfaces :-

- We can't extend multiple abstract classes but we can implement multiple interfaces at a time.
- Interfaces are meant for dynamic method dispatch and run time polymorphism.

• Is Multiple Inheritance present in Java :-

- Multiple inheritance face problems when there exist methods with same signature in both the classes.
- Due to such problems, Java does not support multiple inheritance directly but the similar concept can be acheived using interfaces.
- A class can implement multiple Interfaces and extent a class at the same time

+ Note: ① Interface in Java is a bit like the class but with a significant difference.
② An interface can only have method signatures, constant fields and default methods.
③ The class implementing an interface needs to define the methods (not fields).
④ You can create a reference of Interfaces but not the object.
⑤ Interface methods are public by default.

Page No.
Date

• Default Methods :-

- An interface can have static and default methods.
- Default method enable us to add new functionality to existing interfaces.
- This feature was introduced in Java 8 to ensure backward compatibility while updating an Interface.
- Classes implementing the interface need not implement the default methods.
- Interfaces can also include private methods for default methods to use.

• Inheritance in Interfaces :-

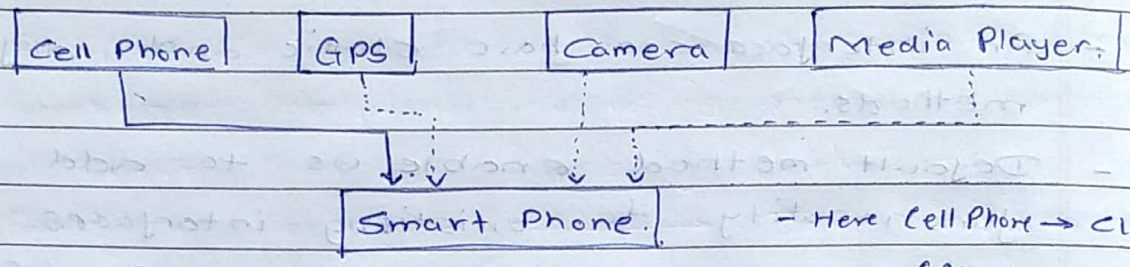
- Interfaces can extend another interfaces.

```
public interface Interface1 {
        void meth1();
}
public interface Interface2 extends Interface1 {
        void meth2();
}
```

* Note :- Remember that interface cannot implement another interface, only classes can do that.

Page No.
Date

- **Polymorphism using Interfaces:-**



- Here Cell Phone → Class

GPS, Camera, Media P } Interface

Similar to dynamic method dispatch in Inheritance

↳ GPS g = new SmartPhone(); → Can only use GPS methods

SmartPhone S = new SmartPhone(); → Can only use SmartPhone methods

- Implementing an Interface forces method Implementation.

* **Packages:-**

+ Interpreter vs Compiler

- Interpreter translates one statement at a time into machine code.
- Compiler scans the entire program and translates whole of it into machine code.

- **Interpreter:-**

- One statement at a time.
- Interpreter is needed everytime.
- Partial execution if error
- Easy for programmers.

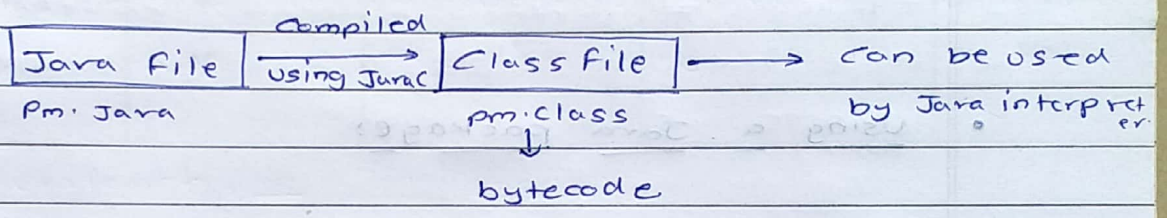
Page No.
Date

- **Compiler:-**

- Entire program at a time
- Once compiled it is not needed.
- No execution if any error occurs.
- Usually not as easy as interpreted ones.

- **Is Java compiled or interpreted ?**

- Java is a hybrid language → both compiled as well as interpreted.



- A JVM can be used to interprete this bytecode.
- This bytecode can be taken to any platform for execution.
- Hence Java is platform independent

- **Executing a Java program:-**

Javac PM.Java → Compiled

Java PM class → Interpreted.

Page No.
Date

• Packages in Java:-

- A package is used to group related classes
- Packages help in avoiding name conflicts.

There are 2 types of packages:

① Built-in packages ⟶ Java API
② User-defined packages ⟶ custom Packages.

Song.mp3 photo1.jpg		songs
photo2.jpg song1.mp3	⟶ organized	photos
Video.mp4, Video2.mp4	as folders	videos.

• Using a Java package:-

- import java.lang* ⟶ Imports everything from Java.lang.
- import java.lang.String ⟶ " String "

• Creating a package:-

Javac pm.Java ⟶ Creates pm class.
Java -d. pm.Java ⟶ Creates a package folder.
↳ we can keep adding classes to a package like this.

- we can also create inner packages by adding "package.inner" as package name.
- These packages once created can be used by other classes.

Page No.
Date

• Access Modifiers in Java:-

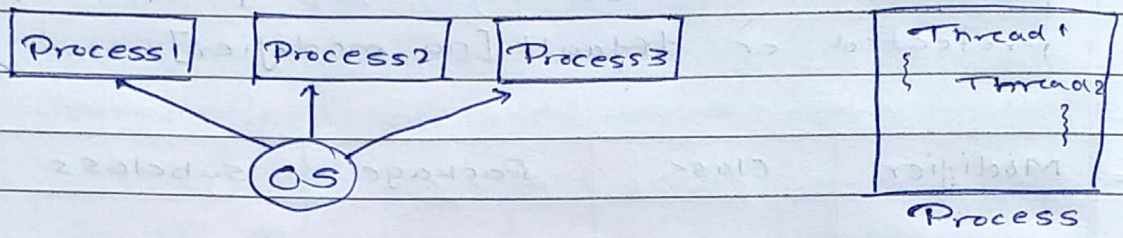
- Access modifiers determine whether other classes can use a particuler field or invoke a particular method can be public, private, protected or default [no modifier].

Modifier	Class	Package	Subclass	World.
Public	Y	Y	Y	Y
Private	Y	N	N	N
Protected	Y	Y	Y	N
Default	Y	Y	N	N.

Page No.
Date

* Multithreading :-

- Multiprocessing and multithreading both are used to acheive multitasking.

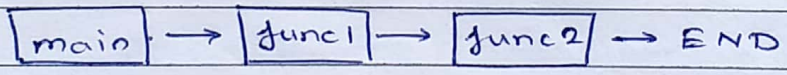


In a nutshell :-

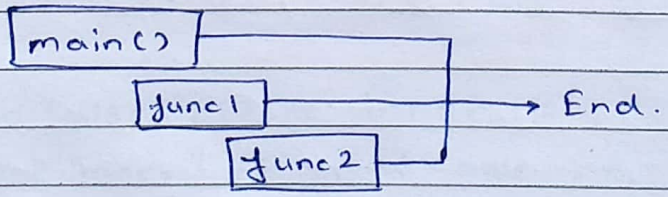
- Threads use shared memory area.
- Threads ⇒ faster content switching.
- A thread is light weight whereas a process is heavy weight.

• Flow of control in Java :-

1) without threading :-



2) with threading :-



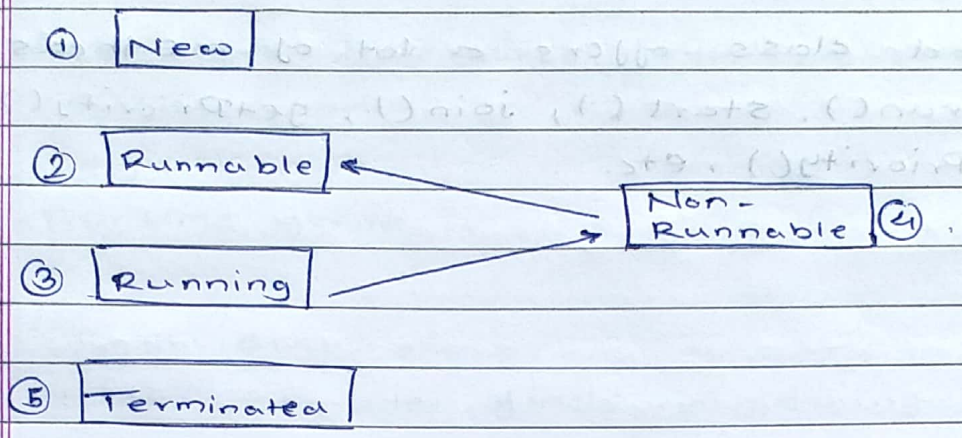
• Creating a thread :-

There are 2 ways to create a thread in Java

1) By extending thread class
2) By implimenting runnable interface

Page No.
Date

- **Life cycle of a Thread:-**



① New ⇒ Instance of thread created which is not yet started by invoking start().

② Runnable ⇒ After invokation start() and before it is selected to be run by the scheduler.

③ Running ⇒ After thread schedular has selected it.

④ Non-runnable ⇒ Thread alive, not eligible to run.

⑤ Terminated ⇒ run() method has exited.

- **The Thread class:-**

Below are commonly used constructors of thread class :-

① Thread()
② Thread(String name)
③ Thread(Runnable r)
④ Thread(Runnable r, String name)

Page No.
Date

- **Methods of thread class :-**

- Thread class offers a lot of methods such as run(), start(), join(), getPriority(), setPriority(), etc.

Page No.
Date

* Errors and Exceptions:-

- There are three types of errors in Java.

1) Syntax error
2) Logical error
3) Runtime error.

+ Syntax error:-

- When compiler finds something wrong with our program, it throw a syntax error.

```
int a = 9 ——→ No semicolon, syntax error
a = a+3;
d = 4; ——→ Variable not declared, syntax error.
```

+ Logical error:-

- A Logical error or a bug occurs when a program compiles and runs but does the wrong thing.

→ message delivered wrongly.
→ wrong time of chats being displayed.
→ incorrects redirects!

+ Runtime error:-

- Java may sometimes encounter an error while the program is running. These are also called as exceptions.
- These are encountered due to circumstances like bad input and (or) resource constraints.

- Syntax errors are logical errors which are encountered by the programmer whereas runtime errors are encountered by the users.

- **Exceptions in Java:-**

- An exception is an event that occurs when a program is executed disrupting the normal flow of instructions.
There are mainly 2-types of exception in Java:-

1) Checked exception. ⟶ Compile time exceptions.
2) Unchecked exception. ⟶ Runtime exceptions.

- **Commonly occuring exceptions:-**

- Null-pointer exception.
- Arithmetic exception.
- ArrayIndexOutofBound Exception.
- Illegal argument exception.
- Number format exception.

- **try-catch block in Java:-**

- In Java, exceptions are managed using try-catch blocks.
Syntax:

```
try {
  // code to try }
catch (Exception e) {
  // code if exception
}
```

Page No.
Date

• Handling Specific exceptions :-

- In Java, we can handle specific exceptions by typing multiple catch block.

```
try {
    // Code
}
Catch (IOException e) {  ⟶ Handles all exception of
    // code.                type IO Exception.
}
Catch (Arithmetic Exception e) { → Handles all exce-
    //code                    ption of type Arithmatic e.
}
Catch ( Exception e) {  ⟶ Handles all other
    // code                  exceptions.
}
```

• Nested try-catch :-

- We can nest multiple try-catch blocks as follows:

```
try {
    try {
        }
    catch (En-e) {
        }
}
catch (En. e){  ⟶ Nested try-catch
    }              block.
```

Page No.
Date

- **Exception class in Java:-**

- We can write our custom exceptions using Exception class in Java.

```
public class MyException extends Exception {
        //overriden methods
    }
```

The exception class has followed important methods.

① String toString() ⟶ executed when sout(e) is ran
② void printStackTrace() ⟶ prints stack trace.
③ String getMessage() ⟶ prints the exception message.

- **The Throw keyword :-**

- The Throw keyword is used to throw an exception explicitly by the programmer.

```
if (b == 0) {
    throw new ArithmeticException("Div by 0");
    }
else {
    return a/b;
}
```

In a similar manner, we can throw user defined exceptions:-

throw new MyException("Exception thrown");

- The throws exception :-

- The Java throws keyword is used to declare an exception. This gives an information to the Programmer that there might be an exception so, its better to be prepared with a try catch block!

```
public void calculate (int a, int b) throws IOException {
    //code
}
```

- Java finally block :-

- finally block contains the code which is always executed whether the exception is handled or not.
- It is used to execute code containing instruction to release the system resources, close a connection, etc.

Page No.
Date

• Annotations in Java:-

- Used to provide extra information about a program annotations provides metadata to class.

+ Foll. are some common annotations built into Java.

① @Override → Used to mark overidden elements in the child class.

② @SuppressWarnings → Used to suppress the gererated warning by the compiler.

③ @deprecated → Used to mark deprecated methods

④ @FunctionalInterface → Used to ensure an interface is a functional interface.

• Lambda Expressions:-

- Lambda expressions let us express instances of single method classes more compactly.
- Anonymous classes are used to implement a base class without giving it a name.
- For classes with a single method, even anonymous classes get slightly exessive.

• Java Generics:-

- Similar to C++ Templates (but not the same).

ArrayList<Integer> a = - - - -

→ If we write it like this then we don't have to write int everytime

Page No.
Date

If we write,

ArrayList a = new ArrayList();

a.add(75)

int anum = (int) a.get(0) → we have to do this.

int anum = a.get(0) → we can't do this.

Hence, generics aim to reduce bugs and enhance type safety.

Note: Type Parameter in Java generics cannot be a primitive data type.

- File-Handling in Java:-

- Reading from and writing to files is an important aspect of any programming language.

- We can use the file class in Java to create a file object.

→ create NewFile() method → creates a file object.

→ for reading files we can use the same Scanner class and supply it a file object.

→ To delete a file in Java we can use File object's delete() method.