UNIT:1

Dashrath Nandan

☆ Java: Java is a high-level object-oriented programming language. It was developed by Sun Microsystem in 1995. James Gosling is known as father of Java.

Platform: Any hardware or software environment in which a program runs. Java has JRE and API.

* Features of Java:

i) Simple: Java Syntax is simple, clean and easy.
ii) Robust: It uses strong memory managment.
iii) Secured: No explicit pointers.
iv) Object-Oriented: Object, class, Inheritance, Polymorphism, Abstraction, Encapsulation.
v) Platform Independent (vi) Portable (vii) High performance

☆	C++	Java
1.	C++ is platform dependent.	Java is platform Independent.
2.	It support Operator Overloading.	Doesn't support operator Overloading
3.	Support multiple Inheritance.	Doesn't Support Multiple Inheritance
4.	Support Pointers.	Java has restricted pointer support.
5.	It doesn't have buit-in support for threads.	It has buit-in thread support.
6.	It support structure and union.	Java doesn't support Structures and union.

- Object: An object is an instance of a class.
- Class: A class is defined as blueprint of object.
- Variable: It is a container which holds the value.
  - ↳ Local Variable, Instance and Static Variable.

* Java Identifiers: Names used for classes, variables and methods are called identifiers.
  - ↳ All identifiers should begin with letter, currency character or _
  - ↳ A keyword cannot be used as identifiers.
  - ↳ Identifiers are case sensitive.

  Eg: age, $salary, _value

* Java Keyword: In Java, a keyword is a reserved words that have a predefined meaning.

  Eg: abstract, if, else, do, int, short, long, float, for, ...

* Tokens: A token is the smallest element of a program that is a specified meaning to compiler. It include keyword, variable, constant, etc.

* Java Data Type: Data type specify the different sizes and values that can be stored in variable.

  1. → Primitive Data Type:
  2. → Non-primitive data type.

* Primitive D.T.	Non-Primitive DT
i) These are predefined.	→ Created by programmer.
ii) It cannot used to call method.	→ It can be used to call method to perform some operation.
iii) It has always a value.	→ It can be null.
iv) The size depends on data-type.	→ They have all the same size
Eg: int, char, bool, byte, float, long, short, double.	Eg: String, array, etc.

* Packages: A package is a pack (group) of classes, interfaces and other packages.
  (i) Built-in package: The already defined packages like - java.io.*, java.lang.*
  (ii) User-defined package: The packages created by user.

- Sub packages: A package inside another package is known as subpackage.
- Advantages of using packages: Reusability, Better Optimization, Name conflicts.

★ Access Modifiers: An access modifiers restricts the access of a class, constructor, data member and method in another class.

TYPES:

1.) Default AM: The scope is limited to the package only.
2.) Public AM: These can be accessed from anywhere.
3.) Protected AM: These can be accessed within the package and outside pkg. through child class.
4.) Private AM: These can be accessed within the class only.

Example:

Private AM

```
Class Dn{
  Private String name;}
Public class main{
  PSVM (String [] args){
  Dn d = new Dn();
  d.name = "Dashrath";}}
```

Output → This will generate Error b/c we are Private Variable of Data class from main class.

Protected AM

```
Class Animal{
  Protected void display(){
    SOP("Animal");}}
Class Dog extends Animal{
  PSVM (String [] args){
    Dog dog = new Dog();
    dog.display();}}
```

Output: Animal

☆ **Class**: A class is a group of object which have common properties. It is blueprint of object.

* A class in java contains :-
  - Fields
  - Methods
  - Constructors
  - Blocks
  - Nested class & Interface.

* Static class : A static class cannot be instantiated i.e. we can't create objects of a static class. we can only access its member using class name itself

Example: Nested classes

```
class OuterClass{
    int x = 10;
  class InnerClass{
    int y = 5;
  }}
class Main{
  PSVM (String[] args){
  OuterClass O1 = new OuterClass();
OuterClass.InnerClass I1 = O1.new InnerClass();
  SOP(I1.y + O1.x); }}
```

Output : 15

Static Nested class

```
class TestOuter{
Static int data = 30;
Static class Inner{
void msg(){
  S.out("data is" + data);}}
Public static void main(String[] args)
  { TestOuter.Inner Obj =
    new TestOuter;
Obj.msg();}}
```

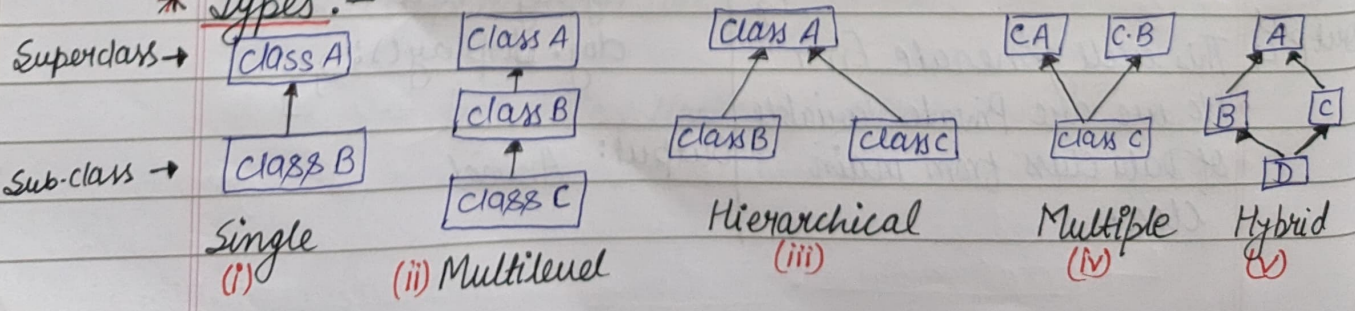
Output :
data is 30

☆ **Inheritance in Java**:

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviour of a parent object.

Why ? ⇒ For Method Overriding ; For Code Reusability.

* Types :-



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

Examples:

Common code

```
class Animal{
  void eat(){
  S.out("Eating");}}
class Dog extends Animal{
  void bark(){
  S.out("barking");}}
class main{ PSVM(SO){
  Dog d = new Dog();
    d.bark();
    d.eat();}}
```

Output: barking
eating

Single Inheritance

```
Class Animal{
  Common code
}
Class BabyDog etends Dog{
  Void weep(){
  Sout("weeping");}}
Class Main{
  PSVM(String[]args)
  {
  BabyDog bd = new BabyDog();
    bd.weep();
    bd.bark();
    bd.eat();
```

Output: weeping
barking
eating

Multilevel Inheritance

```
Class Animal {
  Common code
}
class Cat extends
        Animal
{  void meow(){
  Sout("meowing")}
class main{
  PSVM(E]){
  Cat C = new Cat();
    C.meow();
    C.eat();
}}
```

Output: Meowing
Eating

Hierarchical

Note: To reduce the complexity and simplify the language, Multiple Inheritance is not supported in java.

Ex: The C class inherits A and B class. If A and B have the same method and if we call it from child class object, there will be ambiguity to call method of A or B class.

★ Abstraction:

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

There are two ways to achieve abstraction:

(i) Abstract class (0 to 100%)
(ii) Interface (100%)

Abstract	Interface
i) The abstract keyword is used.	The Interface keyword is used.
ii) Doesn't support Inheritance.	Support inheritance
iii) Can have abstract as well as non abstract methods.	Can only have abstract method.
iv) Can have static method, and constructors.	Cannot have static method and constructors.

ex:
```
abstract class Animal {
    abstract void eat() }
class Dog extends Animal {
    void eat() { sop ("Eating"); }
PSVM (String [] args) {
    Animal obj = new Dog();
    obj.eat();
}
}
```

ex:
```
Interface A { void a() }
abstract class B implements A {
    Public void a() { sop ("I am a"); }}
class main {
    PSVM (String [] args) {
    B b = new B();
    b.a(); } }
```

★ Polymorphism:

Polymorphism means "Many form" and it occurs when we have many class that are releated by inheritance.

Upcasting: If the reference variable of Parent class refers to the object of child class, it is known as upcasting.

* Types:

Method Overloading	Method Overriding
i) Compile-time polymorphism.	Run-time polymorphism.
ii) It is performed within class.	It occurs in two class.
iii) Parameters are different.	In this Name as well as parameter must be same.
iv) Inheritance is not possible.	Inheritance is possible.
Def: Class having multiple methods having same name but different in Parameter.	If subclass has the same method as declared in the parent class.

Method Overloading

```
Public class Sample {
Public static void add (int a, int b){
  Sop(a+b);}
Public void add (int a, int b, int c){
  Sop(a+b+c);}
PSVM (String[] args){
  Sample obj = new Sample();
    obj.add(1,2);
    obj.add(1,2,3); } }
```

Output: 3
6

Method Overriding

```
Public class Vechicle {
Void run(){ Sop("vechicle");}
class Bike extends Vechicle{
void run(){ Sop("Bike");}
PSVM(String[] args){
  Bike b = new Bike();
  Vechicle V = new Vechicle();
    b.run();
    V.run(); } }
```

Output: Bike
Vechicle

☆ Encapsulation:

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

Advantages:

- Make the class read-only or write-only.
- Provide control over the data.
- Way to achieve data hiding.
- Encapsulated class is easy to test.

Accessors (Getter): Method to retrieve the hidden data.
Mutators (Setter): Method to change hidden data.

Eg:

```
Package Dashrath;
Public class student{
  private String name;
public String getName(){
  return name;
}
```

getter method.

```
public void SetName(String Name){
  this.name = name;
}
}
```

Setter method.

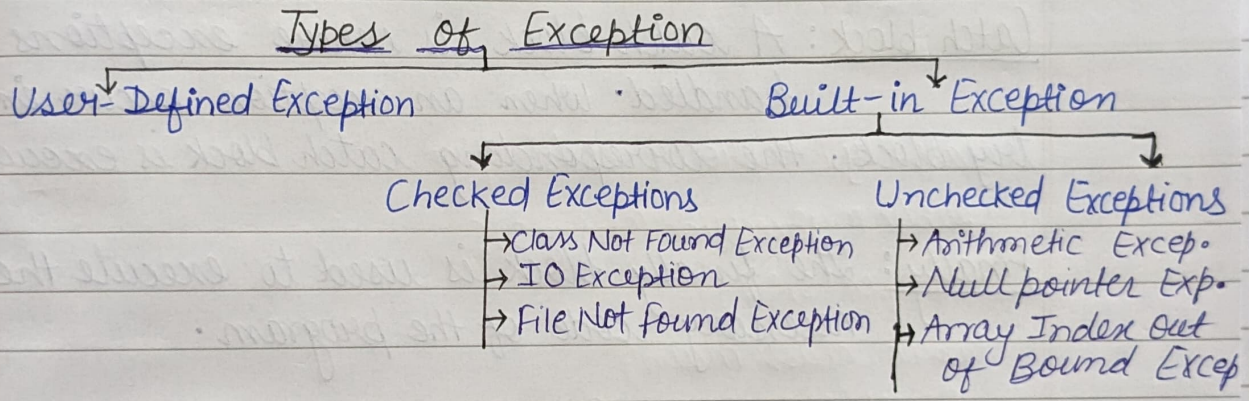
Dashrath Nandan

☆ Exception Handling:

The 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.

- Java exception handling is managed via five keywords: try, catch, throw, throws and finally.

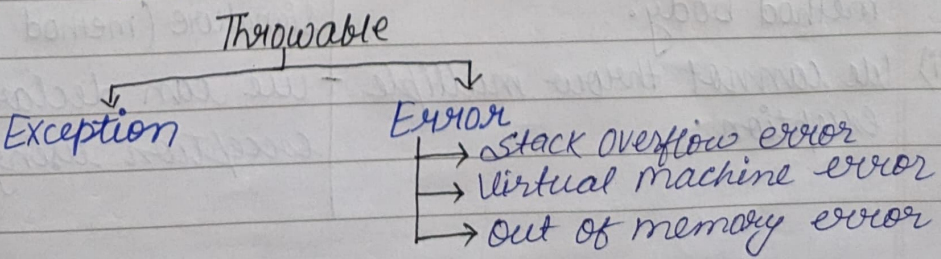
✱ Exception: An exception is an abnormal condition.
The parent class of the exception class is java.lang.Exception.



(i) Checked Exception: Checked exception are called compile-time exception because these exceptions are checked at compile time by compiler.

(ii) Unchecked Exception: The class that inherits the runtime exception.

✱ Error: Error is irrecoverable.



*	Error	Exception
i)	Classified as an Unchecked type.	→ classified as checked and unchecked.
ii)	It belong to java.lang.error.	→ It belong to java.lang.Exception
iii)	It is irrecoverable.	→ It is recoverable.
iv)	It can't be occur at compile time	→ It can occur at run time , compile time both
v)	IO Error, Out of Memory Error, etc.	→ NullPointer Exception, etc.

★ Try block : The try block contains set of statements where an exception can occur. It must be followed by catch block or finally block or both.

Catch block : A catch block is where exceptions are handled. When an exception occur in try block, the corresponding catch block is executed.

Finally : The finally block is used to execute the necessary code of the program.

Throw : Throw keyword is used to throw exception explicitly.

Throws : It is used for handling checked exception.

*	Throw	throws
i)	Used to throw exception explicitly.	Used to declare an exception possible during execution.
ii)	It is declared inside a method body.	It is used with method signature (method declaration).
iii)	We cannot throw multiple exception.	We can declare multiple exception using throws.

* Final: Final is a keyword.
Finally: Finally is a block. It will execute whether Excep. occur or not.
Finalize: Finalize is a method. It is used to prefor clean up processing just before object is garbage collected.

Eg:

```
Public class ErrorExample{
PSVM(String[]args){
 recursive method(10);}
public Void Method(int i){
 while(i!=0){
  i=i+1;
recursive method(i);}}
}
```

```
public class ExceptionExample{
 PSVM(String[] args){
   int x=100;
   int y=0;
   int z=x/y;
    Sout(z);}}
```

Output: java.lang. Arithmetic exception; /zero

Eg

```
public class Exception_Handling{
 PSVM(String[]args){
 try{
    int a[]=new int[5];
     a[5]=30/0;}
 catch(Arithmetic Exception e){
    S.out("Arithmetic Exception Occurs");}
 catch(Exception e){
     S.out("Parent Exception Occurs");}
 finally{
        S.out("Finally block is always executed");}
   S.out("rest of code");
  }
}
```

always followed by catch block, finally block or both.

Eg

```
Public class Test Throw{
Static void validate(int age){
 if(age<18)
 throw new Arithmetic Exception("Not Valid")
 else
 S.out("Welcome to vote");}
PSVM(String[] args){
  Validate(13);
  S.out("rest of code");
}}
```

```
import java.io.Exception;
class Test Throws{
void.   () throw IO Exception{
 throw new IO Exception("Error")}
Void n() throws IO Exception{
      m();}
Void P(){
try{ n();}
catch (Exception e){
 S.out("Exception handled");}
PSVM(String args[]){
Test throws Obj = new Test Throws();
  Obj.P();}
```

output: Exception handled

Eg User-defined

```
class My Exception extends Exception
{
 Public MyException(String S){
   Super(S);}}
 Public class main{
   PSVM(String[] args){
   try{
       throw new MyException("dn@nandan");}
 catch(My Exception e){
    S.OP("caught");
    Sop(e.getmessage());}}}
```

Output: caught
dn@nandan