**Day 4**

**Java programing**

**Polymorphism:**

* Polymorphism in JAVA refers to ability of a single actionto behave differently based on the context

**Types of polymorphism:**

* Compile time polymorphism
* Runtime polymorphism

**Compile time polymorphism:**

* Achieved through method overloading
* Resolved during compilation

**Runtime polymorphism:**

* Achieved through method overriding

**Eg 1:**

**class** Shoppingitems{

**void** additems(String item\_name,**int** price){

System.***out***.println("itemname is:"+item\_name+"price is:"+price);

}

**void** additems (String item\_name,**int** price,**int** item\_quantity){

**int** totalprice=price\*item\_quantity;

System.***out***.println("item name is:"+item\_name+"quantity:"+item\_quantity+"totalprice"+totalprice);

}

**void** additems(String item\_name,**int** price,**int** item\_quantity,**int** discount){

}

}

**public** **class** Sample {

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

Shoppingitems ref=**new** Shoppingitems();

ref.additems("laptop", 50000);

ref.additems("keyboard",1000,3);

}

}

**Eg 2:**

**import** Payment.Creditupi;

**import** Payment.Gpay;

**import** Payment.Topayvia;

**class** Topayvia{

**void** pay(**int** amount) {

System.***out***.println("in hand account:"+amount);

}

}

**class** Creditupi **extends** Topayvia{

**void** pay(**int** amount ) {

System.***out***.println("upipayment:"+amount);

}

}

**class** Gpay **extends** Topayvia{

**void** pay(**int** amount) {

System.***out***.println("Gpay:"+amount);

}

}

**class** Paytm **extends** Topayvia{

**void** pay(**int** amount) {

System.***out***.println("Paytm:"+amount);

}

}

**public** **class** Payment{

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

Topayvia ref;

ref= **new** Creditupi();

ref.pay(2000);

ref=**new** Gpay();

ref.pay(3000);

ref=**new** Paytm();

ref.pay(3000);

}

}

**Eg 3:**

**package** Vehicle;

**abstract** **class** Vehicle{

**abstract** **void** Startsusing();

}

**class** Bike **extends** Vehicle{

**void** Startsusing() {

System.***out***.println("Bike Starts With Kick!.");

}

}

**public** **class** Vehicle {

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

Bike ref=**new** Bike();

ref.Startsusing();

}

}

**Abstraction:**

Abstraction is the process of hiding implementation details and showing only the essential features of the object

**Key features of abstraction:**

* Abstract methos does not contain body
* Implemented using abstract classes and interfaces in java
* Helps focus on **“what an object”** does rather than **“how it does it”**
* If you know the implementation partially , use an abstract class
* If you dont know the implementation at all use an interface

**Interface:**

* Only public ,static,final (constants) variables are allowed.
* All interfaces method are public and abstract by default

**When we use interface?**

* When multiple class need a common behaivour.
* When you want to achieve full abstrasction.
* When you want to decouple dependencies between components.

**Syntax:**

Interface interfacename{

Static int a=10;

Final int b=10;

Default void method name()

}

Abstract void methodname() {

}

}

**Eg 1 :**

**package** Interface;

**interface** Superclass1{

**default** **void** sending() {

System.***out***.println("Superclass 1");

}

}

**interface** Superclass2{

**default** **void** sended() {

System.***out***.println("Superclass 2");

}

}

**class** subclass **implements** Superclass1,Superclass2{

**public** **void** sending() {

System.***out***.println("superclass 1");

}

**public** **void** sended() {0

System.***out***.println("superclass 2");

}

}

**public** **class** Interface {

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

subclass s = **new** subclass();

s.sended();

s.sending();

}

}

**Exception handling:**

* Exception handling is a mechanism to handle run time errors in java ensuring the program can recover or provide meaningful error messages
* It uses try,catch,throw,throws,and finally keywords to manage exceptions
* **Try:** block of code where exceptions might occur
* **Catch:** block of handle exceptions
* **Throw:** explicitly throws an exception
* **Throws:**declares exceptions that a method might throw
* **Finally:**block that executes regardless of whether an exception occured or not

**Eg 1:**

**package** Exception ;

**public** **class** Exception{

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

**try** {

**int** num=10/0;//ArithmeticException

}

**catch**(ArithmeticException e) {

System.***out***.println("Cannot divide by zero:"+e.getMessage());

}

**finally** {

System.***out***.println("Execution completed.");

}

}

}