Polymorphism

class Animal {

public void animalSound() {

System.out.println("The animal makes a sound");

}

}

class Pig extends Animal {

public void animalSound() {

System.out.println("The pig says: wee wee");

}

}

class Dog extends Animal {

public void animalSound() {

System.out.println("The dog says: bow wow");

}

}

class Main {

public static void main(String[] args) {

Animal myAnimal = new Animal();

Animal myPig = new Pig();

Animal myDog = new Dog();

myAnimal.animalSound();

myPig.animalSound();

myDog.animalSound();

}

}

Output:

The animal makes a sound  
The pig says: wee wee  
The dog says: bow wow

Graphical user interface, text, application

Description automatically generated

Example 2

class Polygon {

// method to render a shape

public void render() {

System.out.println("Rendering Polygon...");

}

}

class Square extends Polygon {

// renders Square

public void render() {

System.out.println("Rendering Square...");

}

}

class Circle extends Polygon {

// renders circle

public void render() {

System.out.println("Rendering Circle...");

}

}

class Main {

public static void main(String[] args) {

// create an object of Square

Square s1 = new Square();

s1.render();

// create an object of Circle

Circle c1 = new Circle();

c1.render();

}

}

Output

Rendering Square...

Rendering Circle...

Text

Description automatically generated

Method overriding

//Java Program to demonstrate the real scenario of Java Method Overriding

//where three classes are overriding the method of a parent class.

//Creating a parent class.

**class** Bank{

**int** getRateOfInterest(){**return** 0;}

}

//Creating child classes.

**class** SBI **extends** Bank{

**int** getRateOfInterest(){**return** 8;}

}

**class** ICICI **extends** Bank{

**int** getRateOfInterest(){**return** 7;}

}

**class** AXIS **extends** Bank{

**int** getRateOfInterest(){**return** 9;}

}

//Test class to create objects and call the methods

**class** Test2{

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

SBI s=**new** SBI();

ICICI i=**new** ICICI();

AXIS a=**new** AXIS();

System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());

System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());

System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());

}

}

Output:

SBI Rate of Interest: 8

ICICI Rate of Interest: 7

AXIS Rate of Interest: 9

Graphical user interface, application

Description automatically generated

### Example 1: Polymorphism using method overriding

class Language {

public void displayInfo() {

System.out.println("Common English Language");

}

}

class Java extends Language {

@Override

public void displayInfo() {

System.out.println("Java Programming Language");

}

}

class Main {

public static void main(String[] args) {

// create an object of Java class

Java j1 = new Java();

j1.displayInfo();

// create an object of Language class

Language l1 = new Language();

l1.displayInfo();

}

}

**Output**:

Java Programming Language

Common English Language

Graphical user interface, text, application

Description automatically generated

Method Overloading: changing no. of arguments

**class** Adder{

**static** **int** add(**int** a,**int** b){**return** a+b;}

**static** **int** add(**int** a,**int** b,**int** c){**return** a+b+c;}

}

**class** TestOverloading1{

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

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}

}

Output:

22

33

Graphical user interface, text, application

Description automatically generated

### 2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in [data type](https://www.javatpoint.com/java-data-types). The first add method receives two integer arguments and second add method receives two double arguments.

**class** Adder{

**static** **int** add(**int** a, **int** b){**return** a+b;}

**static** **double** add(**double** a, **double** b){**return** a+b;}

}

**class** TestOverloading2{

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

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(12.3,12.6));

}

}

Output:

22

24.9

### Graphical user interface, text, application Description automatically generated

### Example 3: Polymorphism using method overloading

class Pattern {

// method without parameter

public void display() {

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

System.out.print("\*");

}

}

// method with single parameter

public void display(char symbol) {

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

System.out.print(symbol);

}

}

}

class Main {

public static void main(String[] args) {

Pattern d1 = new Pattern();

// call method without any argument

d1.display();

System.out.println("\n");

// call method with a single argument

d1.display('#');

}

}

**Output**:

\*\*\*\*\*\*\*\*\*\*

##########

Graphical user interface, text, application

Description automatically generated

Simple Example of Encapsulation in Java

Let's see the simple example of encapsulation that has only one field with its setter and getter methods.

*File: Student.java*

//A Java class which is a fully encapsulated class.

//It has a private data member and getter and setter methods.

**package** com.javatpoint;

**public** **class** Student{

//private data member

**private** String name;

//getter method for name

**public** String getName(){

**return** name;

}

//setter method for name

**public** **void** setName(String name){

**this**.name=name

}

}

*File: Test.java*

//A Java class to test the encapsulated class.

**package** com.javatpoint;

**class** Test{

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

//creating instance of the encapsulated class

Student s=**new** Student();

//setting value in the name member

s.setName("vijay");

//getting value of the name member

System.out.println(s.getName());

}

}

Compile By: javac -d . Test.java

Run By: java com.javatpoint.Test

Output:

vijay

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Read-Only class

//A Java class which has only getter methods.

**public** **class** Student{

//private data member

**private** String college="AKG";

//getter method for college

**public** String getCollege(){

**return** college;

}

}

Now, you can't change the value of the college data member which is "AKG".

s.setCollege("KITE");//will render compile time error

Write-Only class

//A Java class which has only setter methods.

**public** **class** Student{

//private data member

**private** String college;

//getter method for college

**public** **void** setCollege(String college){

**this**.college=college;

}

}

Now, you can't get the value of the college, you can only change the value of college data member.

System.out.println(s.getCollege());//Compile Time Error, because there is no such method

System.out.println(s.college);//Compile Time Error, because the college data member is private.

//So, it can't be accessed from outside the class

Another Example of Encapsulation in Java

Let's see another example of encapsulation that has only four fields with its setter and getter methods.

*File: Account.java*

//A Account class which is a fully encapsulated class.

//It has a private data member and getter and setter methods.

**class** Account {

//private data members

**private** **long** acc\_no;

**private** String name,email;

**private** **float** amount;

//public getter and setter methods

**public** **long** getAcc\_no() {

**return** acc\_no;

}

**public** **void** setAcc\_no(**long** acc\_no) {

**this**.acc\_no = acc\_no;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** **float** getAmount() {

**return** amount;

}

**public** **void** setAmount(**float** amount) {

**this**.amount = amount;

}

}

*File: TestAccount.java*

//A Java class to test the encapsulated class Account.

**public** **class** TestEncapsulation {

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

    //creating instance of Account class

    Account acc=**new** Account();

    //setting values through setter methods

    acc.setAcc\_no(7560504000L);

    acc.setName("Sonoo Jaiswal");

    acc.setEmail("sonoojaiswal@javatpoint.com");

    acc.setAmount(500000f);

    //getting values through getter methods

    System.out.println(acc.getAcc\_no()+" "+acc.getName()+" "+acc.getEmail()+" "+acc.getAmount());

}

}

Output:

7560504000 Sonoo Jaiswal sonoojaiswal@javatpoint.com 500000.0

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated