DAY 4

/\*

Copy Constructor: It's a parameterized constructor. It copies data of one object to another.

class Test() {

Test(Test ob) { } // Copy Constructor

}

\*/

/\*

class Test {

int x;

public Test() {

x = 30;

}

public Test(int x) {

this.x = x;

}

public Test(Test ob) {

x = ob.x; // x = ob2.x

}

void show() {

System.out.println("Data: " + x);

}

}

class TestDrive {

public static void main(String[] args) {

Test ob1 = new Test();

Test ob2 = new Test(25);

Test ob3 = ob2; //Shallow Copy

Test ob4 = new Test(ob2); // Deep Copy (Copy Constructor Invoked)

ob1.show();

ob2.show();

ob4.show();

ob3.x = 45;

ob2.show();

ob4.show();

}

}

\*/

/\*

Features of OOP:

1) Encapsulation: Binding up of data member and method together into a single unit.

2) Inheritance: Creating a new class from an existing class. New clas is called as

Child/Derived/Sub class and existing class is called as Parent/Base/Super class.

In the process, Child class will inherit properties from the Parent class.

3) Polymorphism: Existing in different forms.

4) Abstraction: Providing necessary features while hiding implementation details.

E.g.: ADT: Abstract Data Type (Stack, Queue)

\*/

/\*

Inheritance: "IS A" relationship

Advantages: Code Reusability

Types of Inheritance:

Single

Multilevel

Hierarchical

Multiple

Hybrid - Combination of above type of inheritances

\*/

/\*

//Single

class Animal {

void run() {

System.out.println("Running");

}

}

class Lion extends Animal{

void roar() {

System.out.println("Roaring");

}

public static void main(String[] args) {

Lion l = new Lion();

Animal a = new Animal();

l.roar();

l.run();

a.run();

// a.roar(); // Error: Parent object can't access Child members

}

}

\*/

/\*

// Multilevel

class Animal {

void run() {

System.out.println("Running");

}

}

class Lion extends Animal{

void roar() {

System.out.println("Roaring");

}

}

class Cub extends Lion {

void play() {

System.out.println("Playing");

}

public static void main(String[] args) {

Cub c = new Cub();

c.play();

c.roar();

c.run();

}

}

\*/

/\*

// Hierarchical Inheritance

class Animal {

void run() {

System.out.println("Running");

}

}

class Lion extends Animal{

void roar() {

System.out.println("Roaring");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Barking");

}

}

class Drive {

public static void main(String[] args) {

Lion l = new Lion();

Dog d = new Dog();

l.roar();

l.run();

d.bark();

d.run();

// l.bark(); // Error

// d.roar(); // Error

}

}

\*/

// Multiple Inheritance

/\*

class A {

void show() {

}

}

class B {

void show() {

}

}

class C extends A, B {

void fun() {

}

public static void main(String[] args) {

C ob = new C();

ob.show(); // Abiguity in method invocation

// Java doesn't support multiple inheritance due to this.

}

}

// Diamond Problem

// super, super()

\*/

/\*

//super & super()

class A {

int x;

public A() {

System.out.println("Class A Constructor");

}

void showA() {

System.out.println("Class A Data: " + x);

}

}

class B extends A{

int y;

public B() {

//super(); //super() invokes Parent constructor

System.out.println("Class B Constructor");

}

void showB() {

System.out.println("Class B Data: " + y);

super.showA();

}

}

class Driver {

public static void main(String[] args) {

B ob = new B();

ob.showB();

}

}

\*/

/\*

Types of Polymorphism:

1) Compile Time or Static - Static Binding or Early Binding is involved

=> Method Overloading

=> Constructor Overloading

2) Runtime or Dynamic - Dynamic Binding or Late Binding involved

=> Method Overriding

Method Overloading: multiple methods with same name but different parameters

(different no of parameters or different types of parameters)

\*/

/\*

// Method overloading is not possibel by changing return type

class Demo {

void add(int a, int b) {

System.out.println("Sum: " + (a + b));

}

int add(int a, int b) {

return a + b;

}

public static void main(String[] args) {

Demo ob = new Demo();

int res = ob.add(25, 35);

System.out.println("Sum: " + res);

}

}

\*/

/\*

// Method Overloading

class Demo {

int add(int a, int b, int c) {

return a + b + c;

}

int add(int a, int b) {

return a + b;

}

public static void main(String[] args) {

Demo ob = new Demo();

int res = ob.add(25, 35);

// int res = ob.add(25, 35, 40);

System.out.println("Sum: " + res);

}

}

\*/