[01]

Super class – A

Sub class – B,C

Claas A{

}

Class B implements A{}

Class C implements A{}

Class D implements C{}

Class E implements C{}

[02]

Inheritance in java is a core concept that requires the properties of one class to another class like a guardian. For example the relationship between father and son. Or also we can say that the properties derived from one class to another class are a term inheritance.

[03]

Inheritance helps to prevent user private details from others we can hide valuable and private details

[04]

**Student**

public class Student {

}

public class GraduateStudent extends Student {

}

public class UndergraduateStudent extends Student {

}

**Shape**

public abstract class Shape {

public abstract double getArea();

public abstract double getPerimeter();

}

public class Circle extends Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double getArea() {

return Math.PI \* radius \* radius;

}

@Override

public double getPerimeter() {

return 2 \* Math.PI \* radius;

}

}

public class Triangle extends Shape {

private double base;

private double height;

public Triangle(double base, double height) {

this.base = base;

this.height = height;

}

@Override

public double getArea() { }

@Override

public double getPerimeter() {

return 3 \* base;

}

}

**Loan**

public class Loan {

}

public class CarLoan extends Loan {

}

public class HomeImprovementLoan extends Loan {

}

public class MortgageLoan extends Loan {

}

**Employee**

public class Employee {

}

public class Faculty extends Employee {

}

public class Staff extends Employee {

}

**BankAccount**

public class BankAccount {

protected double balance;

public BankAccount(double balance) {

this.balance = balance;

}

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

balance -= amount;

}

public double getBalance() {

return balance;

}

}

public class CheckingAccount extends BankAccount {

private double overdraftLimit;

public CheckingAccount(double balance, double overdraftLimit) {

super(balance);

this.overdraftLimit = overdraftLimit;

}

@Override

public void withdraw(double amount) {

if (balance - amount >= -overdraftLimit) {

balance -= amount;

} else {

System.out.println("Insufficient funds including overdraft limit.");

}

}

}

public class SavingsAccount extends BankAccount {

private double interestRate;

public SavingsAccount(double balance, double interestRate) {

super(balance);

this.interestRate = interestRate;

}

public void addInterest() {

balance += balance \* interestRate;

}

}

[05]

abstract class Shape {

abstract double getArea();

String getName() {

return this.getClass().getSimpleName();

}

}

class Circle extends Shape {

private double radius;

Circle(double radius) {

this.radius = radius;

}

@Override

double getArea() {

return Math.PI \* radius \* radius;

}

}

class Rectangle extends Shape {

private double width;

private double height;

Rectangle(double width, double height) {

this.width = width;

this.height = height;

}

@Override

double getArea() {

return width \* height;

}

}

class Triangle extends Shape {

private double base;

private double height;

Triangle(double base, double height) {

this.base = base;

this.height = height;

}

@Override

double getArea() {

return 0.5 \* base \* height;

}

}

class Sphere extends Shape {

private double radius;

Sphere(double radius) {

this.radius = radius;

}

@Override

double getArea() {

return 4 \* Math.PI \* radius \* radius;

}

}

class Cube extends Shape {

private double side;

Cube(double side) {

this.side = side;

}

@Override

double getArea() {

return 6 \* side \* side;

}

}

public class Main {

public static void main(String[] args) {

Shape[] shapes = new Shape[] {

new Circle(5),

new Rectangle(4, 6),

new Triangle(3, 7),

new Sphere(2.5),

new Cube(3)

};

for (Shape shape : shapes) {

System.out.println("Shape: " + shape.getName() + ", Area: " + shape.getArea());

}

}

}

[06]

Student

* UndergraduateStudent
  + Freshman
  + Sophomore
  + Junior
  + Senior
* GraduateStudent
  + MastersStudent
  + DoctoralStudent

[07]

class Point {

private int x, y;

public Point(int x, int y) {

this.x = x;

this.y = y;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

}

class Quadrilateral {

private Point p1, p2, p3, p4;

public Quadrilateral(Point p1, Point p2, Point p3, Point p4) {

this.p1 = p1;

this.p2 = p2;

this.p3 = p3;

this.p4 = p4;

}

public Point getP1() {

return p1;

}

public Point getP2() {

return p2;

}

public Point getP3() {

return p3;

}

public Point getP4() {

return p4;

}

}

class Trapezoid extends Quadrilateral {

public Trapezoid(Point p1, Point p2, Point p3, Point p4) {

super(p1, p2, p3, p4);

}

public double area() {

return 0.0;

}

}

class Parallelogram extends Quadrilateral {

public Parallelogram(Point p1, Point p2, Point p3, Point p4) {

super(p1, p2, p3, p4);

}

public double area() {

return 0.0;

}

}

class Rectangle extends Parallelogram {

public Rectangle(Point p1, Point p2, Point p3, Point p4) {

super(p1, p2, p3, p4);

}

public double area() {

// Area calculation logic

return 0.0;

}

}

class Square extends Rectangle {

public Square(Point p1, Point p2, Point p3, Point p4) {

super(p1, p2, p3, p4);

}

public double area() {

// Area calculation logic

return 0.0;

}

}

public class Main {

public static void main(String[] args) {

Point p1 = new Point(0, 0);

Point p2 = new Point(0, 2);

Point p3 = new Point(2, 2);

Point p4 = new Point(2, 0);

Square square = new Square(p1, p2, p3, p4);

System.out.println("Square area: " + square.area());

}

}

[08]

import javax.swing.JFrame;

class CalculatorView extends JFrame {

// Just a window

}

class Demo {

public static void main(String []args) {

CalculatorView v1 = new CalculatorView();

v1.setSize(300, 300);

v1.setTitle("Calculator");

v1.setDefaultCloseOperation(CalculatorView.EXIT\_ON\_CLOSE);

v1.setVisible(true);

}

}

[09]

 A. True.

 B. True.

 C. True.

 D. True.

 E. True.

[10]

Line 2,4

[11]

B , D

[12]

A, B, C, D.

[13]

A, B, C, D

[14]

Compilation fails. The class B does not have a constructor that matches the constructor of class A. The superclass A has a parameterized constructor A(int i), and since no constructor is defined in class B, Java tries to insert a default no-argument constructor in B which calls the no-argument constructor of A. But A does not have a no-argument constructor, causing a compile-time error.

[15]

Lines 9, 10, 12, 13, 15, 16, 17

[16]

B

[17]

D

[18]

B,C,E

[19]

not compile because the constructor Sub(int i) in class Sub does not explicitly call a constructor of the superclass Super. The superclass Super does not have a no-argument constructor, so the constructor in Sub must explicitly call super(i).

[20]

E,F

[21]

Super()

Sub()

Super(int)

Sub(int)

[22]

F

[23]

Super(int)

Sub(int)

[25]

A B D A B C E

[26]

A

/ \

B C

/ \ / \

D E F

|

G

[27]

class Vertebrate {

void move() {

System.out.println("move");

}

}

class Mammal extends Vertebrate {

@Override

void move() {

System.out.println("walks");

}

}

class Dog extends Mammal {

@Override

void move() {

System.out.println("walks on paws");

}

void accessAncestorMove() {

super.move(); // This calls Mammal's move method

}

}

public class Test {

public static void main(String[] args) {

Dog d = new Dog();

d.accessAncestorMove(); // prints "walks"

}

}

[29]

B,D,E,F

[30]

C

[31]

A

[32]

B,C

[33]

Regular Customer : null -> Panadura

Sub : null -> Panadura