Assignment 4

Hitarth Patel   
150096724046

Jensen Huang

1) Explain the following terms ((Definition and explanation how it is implemented in Java, rules and syntax, and a program to demonstrate the concept))

a) Class

b) Object

c) Inheritance: Single Inheritance, Multilevel inheritance, hierarchical inheritance, hybrid inheritance

d) Polymorphism

e) Compile time polymorphism and method overloading

f) Runtime polymorphism and method overriding

g) Constructor

h) Constructor overloading

i) super

j) this

k) getters and setters

l) access modifiers

→

a) Class

A class is a blueprint or template that defines the properties and behavior of an object. It's essentially a design pattern or template that defines the characteristics and actions of an object.

Example:

class Car {

private String color;

private int speed;

public Car(String *color*, int *speed*) {

this.color = *color*;

this.speed = *speed*;

}

public void **accelerate**() {

speed++;

}

public String **getColor**() {

return color;

}

}

b) Object

An object is an instance of a class, and it has its own set of attributes (data) and methods (functions).

Example:

Car myCar = new Car("Red", 60);

c) Inheritance

Inheritance is a mechanism where one class can inherit the properties and behavior of another class.

Types of Inheritance:

* Single Inheritance: One child class inherits from one parent class.
* Multilevel Inheritance: A child class inherits from a parent class, which in turn inherits from another parent class.
* Hierarchical Inheritance: Multiple child classes inherit from one parent class.
* Hybrid Inheritance: Combination of multiple inheritance types.

Example:

class Inheritance{

*// Single Inheritance*

public class Animal {

public void **eat**() {

System.out.println("Eating...");

}

}

public class Dog extends Animal {

public void **bark**() {

System.out.println("Barking...");

}

}

*// Multilevel Inheritance*

public class Mammal extends Animal {

public void **walk**() {

System.out.println("Walking...");

}

}

public class Cat extends Mammal {

public void **meow**() {

System.out.println("Meowing...");

}

}

*// Hierarchical Inheritance*

public class Vehicle {

public void **move**() {

System.out.println("Moving...");

}

}

public class Car extends Vehicle {

public void **accelerate**() {

System.out.println("Accelerating...");

}

}

public class Truck extends Vehicle {

public void **load**() {

System.out.println("Loading...");

}

}

*// Hybrid Inheritance*

public class ElectricCar extends Car {

public void **charge**() {

System.out.println("Charging...");

}

}

}

d) Polymorphism

Polymorphism is the ability of an object to take on multiple forms. This can be achieved through method overloading or method overriding.

*// Method Overloading*

public class Calculator {

public int **add**(int *a*, int *b*) {

return *a* + *b*;

}

public double **add**(double *a*, double *b*) {

return *a* + *b*;

}

}

*// Method Overriding*

public class Shape {

public void **draw**() {

System.out.println("Drawing a shape...");

}

}

public class Circle extends Shape {

@Override

public void **draw**() {

System.out.println("Drawing a circle...");

}

}

//Example:

Calculator calculator = new Calculator();

System.out.println(calculator.add(2, 3)); *// Output: 5*

System.out.println(calculator.add(2.5, 3.7)); *// Output: 6.2*

Shape shape = new Circle();

shape.draw(); *// Output: Drawing a circle…*

e) Compile-time Polymorphism and Method Overloading

Method overloading is a form of compile-time polymorphism where multiple methods with the same name can be defined, but with different parameter lists.

public class Calculator {

public int add(int *a*, int *b*) {

return a + b;

}

public double add(double *a*, double *b*) {

return a + b;

}

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

return a + b + c;

}

}

//Example:

Calculator calculator = new Calculator();

System.out.println(calculator.add(2, 3)); *// Output: 5*

System.out.println(calculator.add(2.5, 3.7)); *// Output: 6.2*

System.out.println(calculator.add(2, 3, 4)); *// Output: 9*

f) Runtime Polymorphism and Method Overriding

Method overriding is a form of runtime polymorphism where a subclass provides a different implementation of a method that is already defined in its superclass.

public class Sh**ape** {

public voiddra*w*() {

System.out.println("Drawing a shape...");

}

}

public class Circle extends Shape {

@Override

public void **dr**aw() *{*

System.*o*ut.*p*rin*t*ln("Drawing a circle...");

}

}

//Example:

Shape shape = new Circle();

shape.draw(); *// Output: Drawing a circle…*

g) Constructor

A constructor is a special method that is used to initialize objects when they are created.

Example:

public class Car {

private String color;

private int speed;

public Car(String *color*, int *speed*) {

this.color = *color*;

this.speed = *speed*;

}

}

h) Constructor Overloading

Constructor overloading is a technique where multiple constructors with different parameter lists can be defined.

public class Car {

private String color;

private int speed;

public Car() {

this.color = "Red";

this.speed = 60;

}

public Car(String *color*) {

this.color = *color*;

this.speed = 60;

}

public Car(String *color*, int *speed*) {

this.color = *color*;

this.speed = *speed*;

}

}

i) super

The super keyword is used to access the members of a superclass.

public class Subclass extends Superclass {

public void **methodName**() {

super.methodName();

}

}

//Example:

public class Animal {

public void **sound**() {

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

}

}

public class Dog extends Animal {

public void **sound**() {

super.sound();

System.out.println("The dog barks.");

}

}

j) this

The this keyword is used to refer to the current object.

public class ClassName {

public void **methodName**() {

this.memberName;

}

}

//Example:

public class Car {

private String color;

private int speed;

public Car(String *color*, int *speed*) {

this.color = *color*;

this.speed = *speed*;

}

}

k) Getters and Setters

Getters and setters are methods used to access and modify the properties of an object.

public class ClassName {

private DataType propertyName;

public DataType **getPropertyName**() {

return propertyName;

}

public void **setPropertyName**(DataType *propertyName*) {

this.propertyName = *propertyName*;

}

}

//Example:

public class Car {

private String color;

private int speed;

public String **getColor**() {

return color;

}

public void **setColor**(String *color*) {

this.color = *color*;

}

public int **getSpeed**() {

return speed;

}

public void **setSpeed**(int *speed*) {

this.speed = *speed*;

}

}

l) Access Modifiers

Access modifiers are keywords used to specify the accessibility of a class, method, or variable.

* public: accessible from anywhere
* private: accessible only within the same class
* protected: accessible within the same class and subclasses
* default (no modifier): accessible within the same package

public class ClassName {

*// public members*

private DataType propertyName;

*// private members*

protected DataType propertyName;

*// protected members*

default DataType propertyName;

*// default members*

}

*//Example:*

public class Car {

public String color;

private int speed;

protected String brand;

public void accelerate() {

*// public method*

}

private void brake() {

*// private method*

}

protected void turn() {

*// protected method*

}

}

Q1. WAP to create a class called Circle. It contains:

• private instance variable: radius (of the type double) with default value of 1.0.

• Two overloaded constructors - a default constructor with no argument, and a constructor which takes a double argument for radius.

• Two public methods: getRadius(), calculateArea(), calculateCircumference() which return the radius, calculate and return area, and circumference respectively.

Hint: Use Math.PI for calculating area and circumference

→

import java.util.Scanner;

public class Circle {

private double radius;

public Circle() {

this.radius = 1.0;

}

public Circle(double *radius*) {

this.radius = *radius*;

}

public double **getRadius**() {

return this.radius;

}

public double **calculateArea**() {

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

}

public double **calculateCircumference**() {

return 2 \* Math.PI \* this.radius;

}

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

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the radius of the circle (default is 1.0): ");

double radius = scanner.nextDouble();

Circle circle = new Circle(radius);

System.out.println("Radius: " + circle.getRadius());

System.out.println("Area: " + circle.calculateArea());

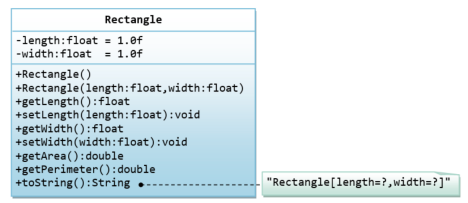
System.out.println("Circumference: " + circle.calculateCircumference());

scanner.close();

}

}

Q2. A class called Rectangle, which models a rectangle with a length and a width (in float), is designed as shown in the following class diagram. Write the Rectangle class as per UML diagram.



→

import java.util.Scanner;

public class Rectangle {

private float length;

private float width;

public Rectangle() {

this.length = 1.0f;

this.width = 1.0f;

}

public Rectangle(float *length*, float *width*) {

this.length = *length*;

this.width = *width*;

}

public float **getLength**() {

return this.length;

}

public float **getWidth**() {

return this.width;

}

public double **calculateArea**() {

return this.length \* this.width;

}

public double **calculatePerimeter**() {

return 2 \* (this.length + this.width);

}

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

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the length of the rectangle (default is 1.0): ");

float length = scanner.nextFloat();

System.out.println("Enter the width of the rectangle (default is 1.0): ");

float width = scanner.nextFloat();

Rectangle rectangle = new Rectangle(length, width);

System.out.println("Length: " + rectangle.getLength());

System.out.println("Width: " + rectangle.getWidth());

System.out.println("Area: " + rectangle.calculateArea());

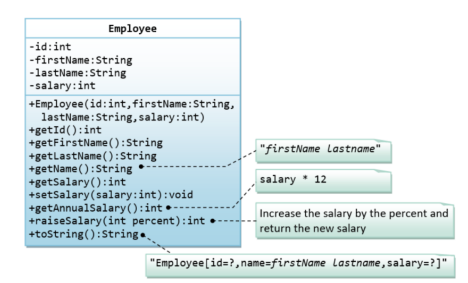
System.out.println("Perimeter: " + rectangle.calculatePerimeter());

scanner.close();

}

}

Q3. A class called Employee, which models an employee with an ID, name and salary, is designed as shown in the following class diagram. The method raiseSalary(percent) increases the salary by the given percentage. Write the Employee class and its driver class.



→

import java.util.Scanner;

public class Employee {

private int id;

private String firstName;

private String lastName;

private int salary;

public Employee(int *id*, String *firstName*, String *lastName*, int *salary*) {

this.id = *id*;

this.firstName = *firstName*;

this.lastName = *lastName*;

this.salary = *salary*;

}

public Employee() {}

public int **getId**() {

return id;

}

public void **setId**(int *id*) {

this.id = *id*;

}

public String **getFirstName**() {

return firstName;

}

public void **setFirstName**(String *firstName*) {

this.firstName = *firstName*;

}

public String **getLastName**() {

return lastName;

}

public void **setLastName**(String *lastName*) {

this.lastName = *lastName*;

}

public String **getName**() {

return firstName + " " + lastName;

}

public int **getSalary**() {

return salary;

}

public void **setSalary**(int *salary*) {

this.salary = *salary*;

}

public int **getAnnualSalary**() {

return salary \* 12;

}

public void **raiseSalary**(double *percent*) {

this.salary += this.salary \* (*percent* / 100);

}

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

Scanner scanner = new Scanner(System.in);

Employee employee = new Employee();

System.out.println("Enter Employee ID: ");

employee.setId(scanner.nextInt());

scanner.nextLine(); *// Consume newline left-over*

System.out.println("Enter Employee First Name: ");

employee.setFirstName(scanner.nextLine());

System.out.println("Enter Employee Last Name: ");

employee.setLastName(scanner.nextLine());

System.out.println("Enter Employee Monthly Salary: ");

employee.setSalary(scanner.nextInt());

System.out.println("Employee ID: " + employee.getId());

System.out.println("Employee Name: " + employee.getName());

System.out.println("Employee Monthly Salary: " + employee.getSalary());

System.out.println("Employee Annual Salary: " + employee.getAnnualSalary());

System.out.println("Enter percentage to raise salary: ");

double percent = scanner.nextDouble();

employee.raiseSalary(percent);

System.out.println("Employee Monthly Salary after raise: " + employee.getSalary());

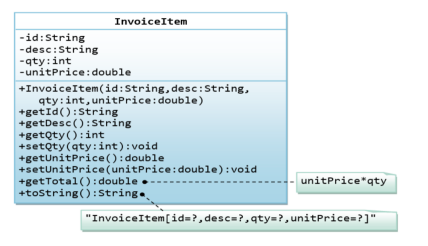
System.out.println("Employee Annual Salary after raise: " + employee.getAnnualSalary());

scanner.close();

}

}

Q4. A class called InvoiceItem, which models an item of an invoice, with ID, description, quantity and unit price, is designed as shown in the following class diagram. It has a method getTotal which calculates total value (total=quantity\*unit price). Write the InvoiceItem class and it’s driver class.



→

import java.util.Scanner;

public class InvoiceItem {

private String id;

private String description;

private int quantity;

private double unitPrice;

public InvoiceItem(String *id*, String *description*, int *quantity*, double *unitPrice*) {

this.id = *id*;

this.description = *description*;

this.quantity = *quantity*;

this.unitPrice = *unitPrice*;

}

public InvoiceItem() {}

public String **getId**() {

return id;

}

public void **setId**(String *id*) {

this.id = *id*;

}

public String **getDescription**() {

return description;

}

public void **setDescription**(String *description*) {

this.description = *description*;

}

public int **getQuantity**() {

return quantity;

}

public void **setQuantity**(int *quantity*) {

this.quantity = *quantity*;

}

public double **getUnitPrice**() {

return unitPrice;

}

public void **setUnitPrice**(double *unitPrice*) {

this.unitPrice = *unitPrice*;

}

public double **getTotal**() {

return quantity \* unitPrice;

}

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

InvoiceItem item = new InvoiceItem();

Scanner scanner = new Scanner(System.in);

System.out.println("Enter Invoice Item ID: ");

item.setId(scanner.nextLine());

System.out.println("Enter Invoice Item Description: ");

item.setDescription(scanner.nextLine());

System.out.println("Enter Invoice Item Quantity: ");

item.setQuantity(scanner.nextInt());

scanner.nextLine(); *// Consume newline left-over*

System.out.println("Enter Invoice Item Unit Price: ");

item.setUnitPrice(scanner.nextDouble());

scanner.nextLine(); *// Consume newline left-over*

System.out.println("ID: " + item.getId());

System.out.println("Description: " + item.getDescription());

System.out.println("Quantity: " + item.getQuantity());

System.out.println("Unit Price: " + item.getUnitPrice());

System.out.println("Total: " + item.getTotal());

scanner.close();

}

}

Q5. A class called Author is designed to model a book's author. It contains:

1) Three private instance variables: name (String), email (String), and gender (char of either 'm' or 'f');

2) One constructor to initialize the name, email and gender with the given values; a) public Author (String name, String email, char gender) {......}(There is no default constructor for Author, as there are no defaults for name, email and gender.) 3) public getters/setters: getName(), getEmail(), setEmail(), and getGender(); (There are no setters for name and gender, as these attributes cannot be changed.)

4) A toString() method that returns "Author[name=?,email=?,gender=?]", e.g., “Author[name=Abc ,email=Abc@gmail.com, gender=m]".

→

import java.util.Scanner;

public class Author {

private String name;

private String email;

private char gender;

public Author(String *name*, String *email*, char *gender*) {

this.name = *name*;

this.email = *email*;

this.gender = *gender*;

}

public String **getName**() {

return name;

}

public String **getEmail**() {

return email;

}

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

this.email = *email*;

}

public char **getGender**() {

return gender;

}

public String **toString**() {

return "Author[name=" + name + ",email=" + email + ",gender=" + gender + "]";

}

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

Scanner scanner = new Scanner(System.in);

System.out.println("Enter Author's Name: ");

String name = scanner.nextLine();

System.out.println("Enter Author's Email: ");

String email = scanner.nextLine();

System.out.println("Enter Author's Gender (m/f): ");

char gender = scanner.next().charAt(0);

Author author = new Author(name, email, gender);

System.out.println(author.toString());

System.out.println("Enter new Email: ");

String newEmail = scanner.next();

author.setEmail(newEmail);

System.out.println("Updated Author's Information: ");

System.out.println(author.toString());

scanner.close();

}

}

Q6. WAP to create a class time having default constructor, parameterized constructor whose specifications are as follows:

1) Instance variable: hr, min, sec

2) Constructors:

i) default (with no parameters passed; should initialize the represented time to 12:0:0) ii) a constructor with three parameters: hours, minutes, and seconds.

iii) a constructor with one parameter: the value of time in seconds since midnight (it should be converted into the time value in hours, minutes, and seconds)

3) Instance methods:

i) setClock() with one parameter seconds since midnight (to be converted into the time value in hours, minutes, and seconds as above).

ii) tick() with no parameters that increments the time stored in a Clock object by one second.

iii) tickDown() which decrements the time stored in a Clock object by one second.

iv) displaytime() displays the time in the format hr: min:sec e.g: 05:45:23

→

public class Time {

private int hr;

private int min;

private int sec;

public Time() {

this.hr = 12;

this.min = 0;

this.sec = 0;

}

public Time(int *hr*, int *min*, int *sec*) {

this.hr = *hr*;

this.min = *min*;

this.sec = *sec*;

}

public Time(int *seconds*) {

this.hr = *seconds* / 3600;

this.min = (*seconds* % 3600) / 60;

this.sec = *seconds* % 60;

}

public void **setClock**(int *seconds*) {

this.hr = *seconds* / 3600;

this.min = (*seconds* % 3600) / 60;

this.sec = *seconds* % 60;

}

public void **tick**() {

this.sec++;

if (this.sec == 60) {

this.sec = 0;

this.min++;

if (this.min == 60) {

this.min = 0;

this.hr = (this.hr + 1) % 24;

}

}

}

public void **tickDown**() {

this.sec--;

if (this.sec == -1) {

this.sec = 59;

this.min--;

if (this.min == -1) {

this.min = 59;

this.hr = (this.hr - 1 + 24) % 24;

}

}

}

public void **displayTime**() {

System.out.printf("%02d:%02d:%02d\n", hr, min, sec);

}

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

Time time = new Time();

time.displayTime();

time.tick();

time.displayTime();

time.tickDown();

time.displayTime();

}

}

Q7. Write a Java class Complex for dealing with complex number. Your class must have the following features:

1) Instance variables :

a) real for the real part of type double

b) imag for imaginary part of type double.

2) Constructor:

a) public Complex (): A default constructor, it should initialize the number to 0, 0) b) public Complex (double real, double imag: A constructor with parameters, it creates the complex object by setting the two fields to the passed values.

3) Instance methods:

a) public Complex add (Complex n): This method will find the sum of the current complex number and the passed complex number. The methods returns a new Complex number which is the sum of the two.

b) public Complex subtract (Complex n): This method will find the difference of the current complex number and the passed complex number. The methods returns a new Complex number which is the difference of the two.

c) public void setReal(double real): Used to set the real part of this complex number. d) public void setImag(double image): Used to set the imaginary part of this complex number.

e) public double getReal(): This method returns the real part of the complex number f) public double getImag(): This method returns the imaginary part of the complex number

g) public String toString(): This method allows the complex number to be easily printed out to the screen

4) Write a separate class ComplexDemo with a main() method and test the Complex class methods.

→

import java.util.Scanner;

public class Complex {

private double real;

private double imag;

public Complex() {

this.real = 0;

this.imag = 0;

}

public Complex(double *real*, double *imag*) {

this.real = *real*;

this.imag = *imag*;

}

public Complex **add**(Complex *n*) {

return new Complex(this.real + *n*.getReal(), this.imag + *n*.getImag());

}

public Complex **subtract**(Complex *n*) {

return new Complex(this.real - *n*.getReal(), this.imag - *n*.getImag());

}

public void **setReal**(double *real*) {

this.real = *real*;

}

public void **setImag**(double *imag*) {

this.imag = *imag*;

}

public double **getReal**() {

return this.real;

}

public double **getImag**() {

return this.imag;

}

public String **toString**() {

return this.real + " + " + this.imag + "i";

}

}

class ComplexDemo {

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

Scanner scanner = new Scanner(System.in);

System.out.println("Enter real part of first complex number: ");

double real1 = scanner.nextDouble();

System.out.println("Enter imaginary part of first complex number: ");

double imag1 = scanner.nextDouble();

System.out.println("Enter real part of second complex number: ");

double real2 = scanner.nextDouble();

System.out.println("Enter imaginary part of second complex number: ");

double imag2 = scanner.nextDouble();

Complex c1 = new Complex(real1, imag1);

Complex c2 = new Complex(real2, imag2);

System.out.println("First complex number: " + c1.toString());

System.out.println("Second complex number: " + c2.toString());

Complex sum = c1.add(c2);

System.out.println("Sum: " + sum.toString());

Complex difference = c1.subtract(c2);

System.out.println("Difference: " + difference.toString());

scanner.close();

}

}

Q8. Create a SumEx1class and overload sum() method for:

• two integers

• three integers

• two double

• three double

→

import java.util.Scanner;

public class SumEx1 {

public int **sum**(int *a*, int *b*) {

return *a* + *b*;

}

public int **sum**(int *a*, int *b*, int *c*) {

return *a* + *b* + *c*;

}

public double **sum**(double *a*, double *b*) {

return *a* + *b*;

}

public double **sum**(double *a*, double *b*, double *c*) {

return *a* + *b* + *c*;

}

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

SumEx1 sumEx1 = new SumEx1();

Scanner scanner = new Scanner(System.in);

System.out.println("Enter two integers: ");

int a = scanner.nextInt();

int b = scanner.nextInt();

System.out.println("Sum of two integers: " + sumEx1.sum(a, b));

System.out.println("Enter three integers: ");

a = scanner.nextInt();

b = scanner.nextInt();

int c = scanner.nextInt();

System.out.println("Sum of three integers: " + sumEx1.sum(a, b, c));

System.out.println("Enter two doubles: ");

double d = scanner.nextDouble();

double e = scanner.nextDouble();

System.out.println("Sum of two doubles: " + sumEx1.sum(d, e));

System.out.println("Enter three doubles: ");

d = scanner.nextDouble();

e = scanner.nextDouble();

double f = scanner.nextDouble();

System.out.println("Sum of three doubles: " + sumEx1.sum(d, e, f));

scanner.close();

}

}

Q9. WAP to implement Box class.

• Inherit Box class in BoxWt whose

o instance variable is weight and

o method is print\_BoxWt()

o constructors: default, parameterized and BoxWt(BoxWt ob)

o Use super() to invoke superclass constructors.

• WAP to demonstrate multilevel inheritance by creating a BoxColor class and inheriting BoxWt class. BoxColor class has an instance variable color of String type.

→

import java.util.Scanner;

class Box {

protected double width;

protected double height;

protected double depth;

public Box() {

this.width = 0;

this.height = 0;

this.depth = 0;

}

public Box(double *width*, double *height*, double *depth*) {

this.width = *width*;

this.height = *height*;

this.depth = *depth*;

}

public double **getWidth**() {

return width;

}

public double **getHeight**() {

return height;

}

public double **getDepth**() {

return depth;

}

}

class BoxWt extends Box {

protected double weight;

public BoxWt() {

super();

this.weight = 0;

}

public BoxWt(double *width*, double *height*, double *depth*, double *weight*) {

super(*width*, *height*, *depth*);

this.weight = *weight*;

}

public BoxWt(BoxWt *ob*) {

super(*ob*.getWidth(), *ob*.getHeight(), *ob*.getDepth());

this.weight = *ob*.weight;

}

public void **print\_BoxWt**() {

System.out.println("Width: " + getWidth() + ", Height: " + getHeight() + ", Depth: " + getDepth() + ", Weight: " + weight);

}

}

class BoxColor extends BoxWt {

private String color;

public BoxColor(double *width*, double *height*, double *depth*, double *weight*, String *color*) {

super(*width*, *height*, *depth*, *weight*);

this.color = *color*;

}

public void **print\_BoxColor**() {

System.out.println("Width: " + getWidth() + ", Height: " + getHeight() + ", Depth: " + getDepth() + ", Weight: " + weight + ", Color: " + color);

}

}

public class Bx {

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

Scanner scanner = new Scanner(System.in);

System.out.println("Enter width for BoxWt: ");

double widthWt = scanner.nextDouble();

System.out.println("Enter height for BoxWt: ");

double heightWt = scanner.nextDouble();

System.out.println("Enter depth for BoxWt: ");

double depthWt = scanner.nextDouble();

System.out.println("Enter weight for BoxWt: ");

double weightWt = scanner.nextDouble();

BoxWt boxWt = new BoxWt(widthWt, heightWt, depthWt, weightWt);

boxWt.print\_BoxWt();

System.out.println("Enter width for BoxColor: ");

double widthColor = scanner.nextDouble();

System.out.println("Enter height for BoxColor: ");

double heightColor = scanner.nextDouble();

System.out.println("Enter depth for BoxColor: ");

double depthColor = scanner.nextDouble();

System.out.println("Enter weight for BoxColor: ");

double weightColor = scanner.nextDouble();

System.out.println("Enter color for BoxColor: ");

String color = scanner.next();

BoxColor boxColor = new BoxColor(widthColor, heightColor, depthColor, weightColor, color);

boxColor.print\_BoxColor();

scanner.close();

}

}

Q15. Create a class Animal with:

1) instance variables:

a) boolean vegetarian

b) String food

c) int numOfLegs

2) Create a no-argument constructor and parameterized constructor. (Use “this”) 3) Create getters and setters

4) Create a toString() method for animal class

5) Create a subclass Cat with instance variable:

a) String color

b) Create a no-argument constructor and parameterized constructor which has all four parameters (Use this and super)

c) Create a toString() method for Cat class

6) Create a subclass Cow with instance variable:

a) String breed

b) Create a no-argument constructor and parameterized constructor which has all four parameters. (Use this and super)

c) Create a toString() method for Cow class

→

import java.util.Scanner;

public class Animal {

private boolean vegetarian;

private String food;

private int numOfLegs;

public Animal() {

this.vegetarian = false;

this.food = "Unknown";

this.numOfLegs = 0;

}

public Animal(boolean *vegetarian*, String *food*, int *numOfLegs*) {

this.vegetarian = *vegetarian*;

this.food = *food*;

this.numOfLegs = *numOfLegs*;

}

public boolean **isVegetarian**() {

return vegetarian;

}

public void **setVegetarian**(boolean *vegetarian*) {

this.vegetarian = *vegetarian*;

}

public String **getFood**() {

return food;

}

public void **setFood**(String *food*) {

this.food = *food*;

}

public int **getNumOfLegs**() {

return numOfLegs;

}

public void **setNumOfLegs**(int *numOfLegs*) {

this.numOfLegs = *numOfLegs*;

}

public String **toString**() {

return "Animal{" +

"vegetarian=" + vegetarian +

", food='" + food + '\'' +

", numOfLegs=" + numOfLegs +

'}';

}

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

Scanner scanner = new Scanner(System.in);

System.out.println("Is the animal vegetarian? (true/false): ");

boolean vegetarian = scanner.nextBoolean();

System.out.println("What does the animal eat? ");

String food = scanner.next();

System.out.println("How many legs does the animal have? ");

int numOfLegs = scanner.nextInt();

Animal animal = new Animal(vegetarian, food, numOfLegs);

System.out.println(animal.toString());

scanner.close();

}

}

class Cat extends Animal {

private String color;

public Cat() {

super();

this.color = "Unknown";

}

public Cat(boolean *vegetarian*, String *food*, int *numOfLegs*, String *color*) {

super(*vegetarian*, *food*, *numOfLegs*);

this.color = *color*;

}

public String **getColor**() {

return color;

}

public void **setColor**(String *color*) {

this.color = *color*;

}

public String **toString**() {

return "Cat{" +

"color='" + color + '\'' +

", " + super.toString() +

'}';

}

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

Scanner scanner = new Scanner(System.in);

System.out.println("Is the cat vegetarian? (true/false): ");

boolean vegetarian = scanner.nextBoolean();

System.out.println("What does the cat eat? ");

String food = scanner.next();

System.out.println("How many legs does the cat have? ");

int numOfLegs = scanner.nextInt();

System.out.println("What color is the cat? ");

String color = scanner.next();

Cat cat = new Cat(vegetarian, food, numOfLegs, color);

System.out.println(cat.toString());

scanner.close();

}

}

class Cow extends Animal {

private String breed;

public Cow() {

super();

this.breed = "Unknown";

}

public Cow(boolean *vegetarian*, String *food*, int *numOfLegs*, String *breed*) {

super(*vegetarian*, *food*, *numOfLegs*);

this.breed = *breed*;

}

public String **getBreed**() {

return breed;

}

public void **setBreed**(String *breed*) {

this.breed = *breed*;

}

public String **toString**() {

return "Cow{" +

"breed='" + breed + '\'' +

", " + super.toString() +

'}';

}

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

Scanner scanner = new Scanner(System.in);

System.out.println("Is the cow vegetarian? (true/false): ");

boolean vegetarian = scanner.nextBoolean();

System.out.println("What does the cow eat? ");

String food = scanner.next();

System.out.println("How many legs does the cow have? ");

int numOfLegs = scanner.nextInt();

System.out.println("What breed is the cow? ");

String breed = scanner.next();

Cow cow = new Cow(vegetarian, food, numOfLegs, breed);

System.out.println(cow.toString());

scanner.close();

}

}