Q1:

abstract class Animal {

public abstract void sound();

public void sleep() {

System.out.println("Animal is sleeping");

}

}

class Dog extends Animal {

public void sound() {

System.out.println("Bark");

}

public abstract void eat();

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.sound();

dog.sleep();

}

}

**Question**: Find and explain the error in the program. If there is no error, explain the output.

**Explanation**: The above program has an error, we can’t have abstract method within non-abstract class at line “public abstract void eat();”

**Mark**: 1 mark for stating there is an error, 1 mark for the correct explanation

Q2

abstract class Shape {

abstract double area();

}

class Circle extends Shape {

double radius;

Circle(double radius) {

this.radius = radius;

}

double area() {

return Math.PI \* radius \* radius;

}

}

public class Main {

public static void main(String[] args) {

Circle circle = new Circle(1);

System.out.println("Area of the circle: " + circle.area());

}

}

**Question**: Find and explain the error in the program. If there is no error, explain the output.

**Explanation**: There is no error in this program. Class circle extend abstract class shape and implemented the abstract method area(), the expected output of the program is “Area of the circle:” followed by the area of the circle

**Mark**: 1 mark for no error; 1 mark for correct expected output and explanation

abstract class Vehicle {

abstract void start();

}

class Car extends Vehicle {

void start() {

System.out.println("Car started");

}

void stop() {

System.out.println("Car stopped");

}

}

abstract class ElectricCar extends Car {

abstract void charge();

}

public class Q3 {

public static void main(String[] args) {

ElectricCar tesla = new ElectricCar();

tesla.start();

tesla.charge();

}

}

**Question**: Find and explain the error in the program. If there is no error, explain the output.

**Explanation**: There is an error at line “ElectricCar tesla = new ElectricCar();”, ElectricCar is an abstract class extends class Car; abstract class can’t be instantiated.

**Mark**: 1 mark for stating there is an error, 1 mark for the correct explanation

Q4:

abstract class Appliance {

abstract void turnOn();

void turnOff() {

System.out.println("Appliance turned off");

}

}

class WashingMachine extends Appliance {

void turnOn() {

System.out.println("Washing Machine turned on");

}

}

class Dishwasher extends Appliance {

void turnOn() {

System.out.println("Dishwasher turned on");

}

void start() {

System.out.println("Dishwasher started");

}

}

public class Main {

public static void main(String[] args) {

Appliance myAppliance = new WashingMachine();

myAppliance.turnOn();

myAppliance.turnOff();

myAppliance.start();

}

}

**Question**: Find and explain the error in the program. If there is no error, explain the output.

**Explanation**: There is an error at line “myAppliance.start();”, start method is only implemented in Dishwasher class which won’t be inherited by class WashingMachine since they are sibling.

**Mark**: 1 mark for stating there is an error, 1 mark for the correct explanation

Q5:

abstract class Printer {

abstract void print();

void scan() {

System.out.println("Scanning document");

}

}

class LaserPrinter extends Printer {

void print() {

System.out.println("Printing with laser printer");

}

}

class InkjetPrinter extends Printer {

void print() {

System.out.println("Printing with inkjet printer");

}

}

public class Main {

public static void main(String[] args) {

Printer printer = new LaserPrinter();

printer.print();

printer.scan();

}

}

**Question**: Find and explain the error in the program. If there is no error, explain the output.

**Explanation**: There is no error in the program; both subclasses overwrite the abstract method print and inherit the method scan from Printer. The expected output is “Printing with laser printer” follow by “Scanning document”

**Mark**: 1 mark for no error, 1 mark for the correct explanation and output

Q6:

interface Movable {

void move();

}

class Car implements Movable {

public void move() {

System.out.println("Car is moving");

}

void stop() {

System.out.println("Car stopped");

}

}

public class Main {

public static void main(String[] args) {

Movable myCar = new Car();

myCar.move();

myCar.stop();

}

}

**Question**: Find and explain the error in the program. If there is no error, explain the output.

**Explanation:** The Movable reference type only knows about the move() method, so calling stop() on myCar will result in a compilation error.

**Mark**: 1 mark for stating there is an error, 1 mark for the correct explanation

Q7:

interface Drawable {

void draw();

}

abstract class Shape implements Drawable {

abstract double area();

}

class Circle extends Shape {

double radius;

Circle(double radius) {

this.radius = radius;

}

public void draw() {

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

}

double area() {

return Math.PI \* radius \* radius;

}

}

public class Main {

public static void main(String[] args) {

Circle circle = new Circle(5);

circle.draw();

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

}

}

**Question**: Find and explain the error in the program. If there is no error, explain the output.

**Explanation:** There is no error in this program. The output will be "Drawing Circle" followed by the calculated area of the circle. This example demonstrates how a class can implement both an abstract class and an interface.

**Mark**: 1 mark for no error, 1 mark for the correct explanation and output

**Q8:**

interface Flyable {

void fly();

}

interface Movable {

void move();

}

class Airplane implements Flyable, Movable {

public void fly() {

System.out.println("Airplane is flying");

}

public void move() {

System.out.println("Airplane is moving on the runway");

}

}

public class Main {

public static void main(String[] args) {

Airplane airplane = new Airplane();

airplane.fly();

airplane.move();

}

}

**Question**: Find and explain the error in the program. If there is no error, explain the output.

**Explanation:** There is no error in this program. The output will be "Airplane is flying" followed by "Airplane is moving on the runway." This example shows how a class can implement multiple interfaces.

Mark: 1 mark for no error, 1 mark for the correct explanation and output

**Q9**

interface Walkable {

void walk();

}

class Person implements Walkable {

public void walk() {

System.out.println("Person is walking");

}

public void run() {

System.out.println("Person is running");

}

}

public class Main {

public static void main(String[] args) {

Walkable walker = new Person();

walker.walk();

walker.run();

}

}

**Question**: Find and explain the error in the program. If there is no error, explain the output.

**Explanation:** The Walkable reference type only knows about the walk() method. Trying to call run() on a Walkable reference will result in a compilation error.

**Mark**: 1 mark for stating there is an error, 1 mark for the correct explanation

**Q10**

interface Animal {

void eat();

void play();

}

class Dog implements Animal {

public void eat() {

System.out.println("Dog is eating");

}

public void play() {

System.out.println("Dog is playing");

}

void bark() {

System.out.println("Dog is barking");

}

}

public class Main {

public static void main(String[] args) {

Animal myPet = new Dog();

myPet.eat();

myPet.play();

myPet.bark();

}

}

**Question**: Find and explain the error in the program. If there is no error, explain the output.

**Explanation:** The Animal interface doesn't have a bark() method, so trying to call it on an Animal reference will result in a compilation error.

**Mark**: 1 mark for stating there is an error, 1 mark for the correct explanation