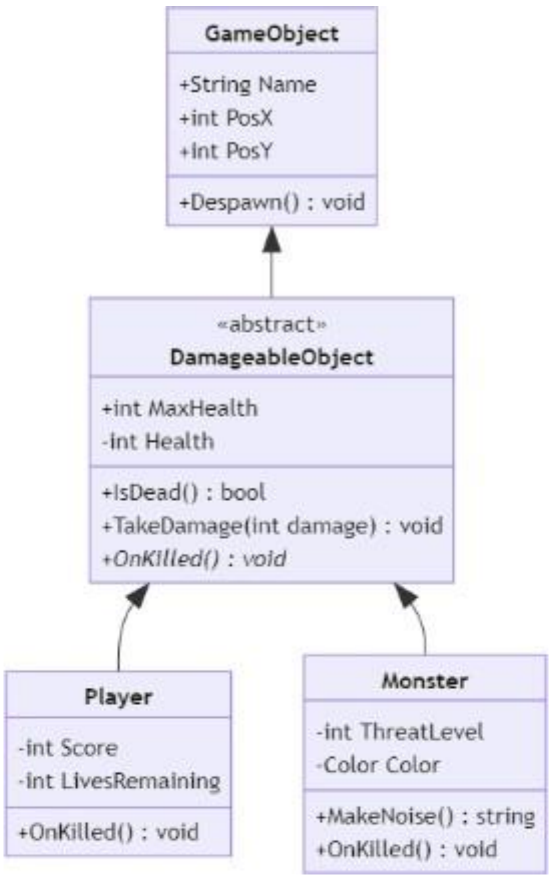
QUIZ QUESTIONS 2
OBJECT-BASED PROGRAMMING PRACTICUM

1. Identify the following Abstract method and Class usage, explain the purpose of the diagram class and create the program code to the demo to display it.



- Analisis Diagram

  Class GameObject:

  kelas ini berfungsi untuk menyimpan atribut umum seperti String Name, int PosX, int PosY

  Class DamageableObject:

  Kelas ini merepresentasikan objek seperti MaxHealth dan Health mengelola status Health dari object, IsDead() mengecek apakah objek sudah mati (Boolean), TakeDamage(int) ini untuk mengurangi nilai health berdasarkan int damage yang diterima, OnKilled(), ini adalah metod abstrak yang akan diimplementasikan oleh child classnya

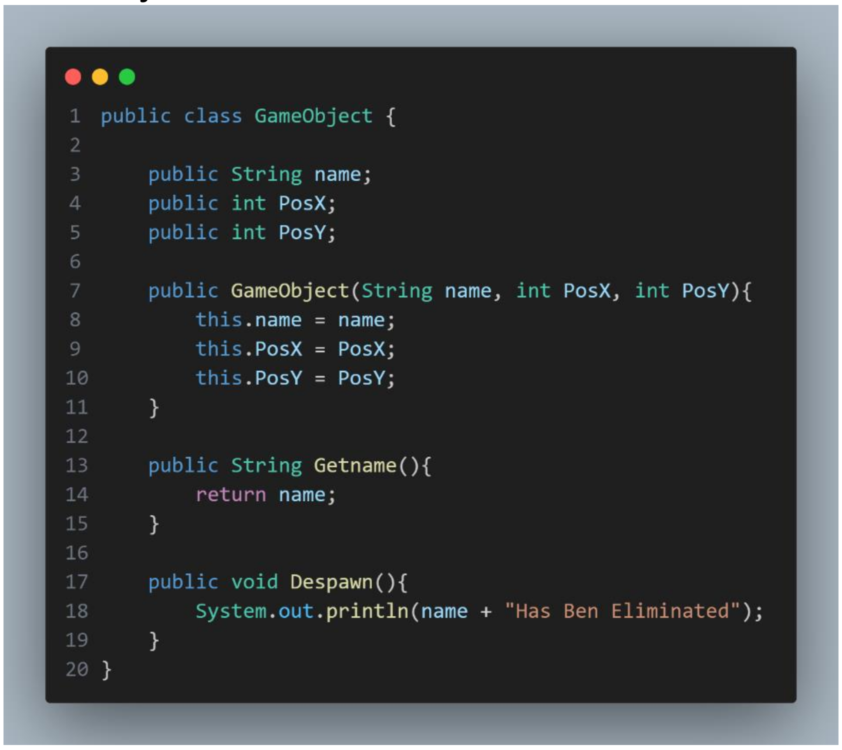
Class Player:

Class ini memiliki atribut tambahan seperti Score dan LiverRemaining, pada kelas ini implementasi dari abstrak class yaitu method OnKilled()

Class Monster:

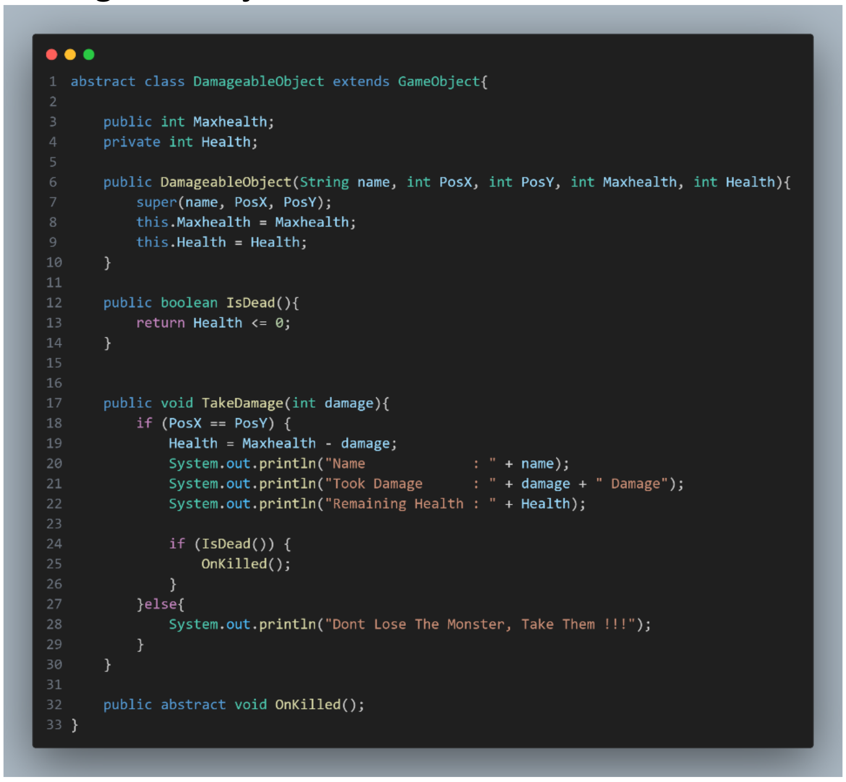
Class ini memiliki atribut tambahan sepert ThreatLevel, dan color, pengimplementasian dari abstak method yaitu OnKilled() dan memiliki method tambahan yaitu MakeNoise

- Implementasian Code:
  - **GameObject**



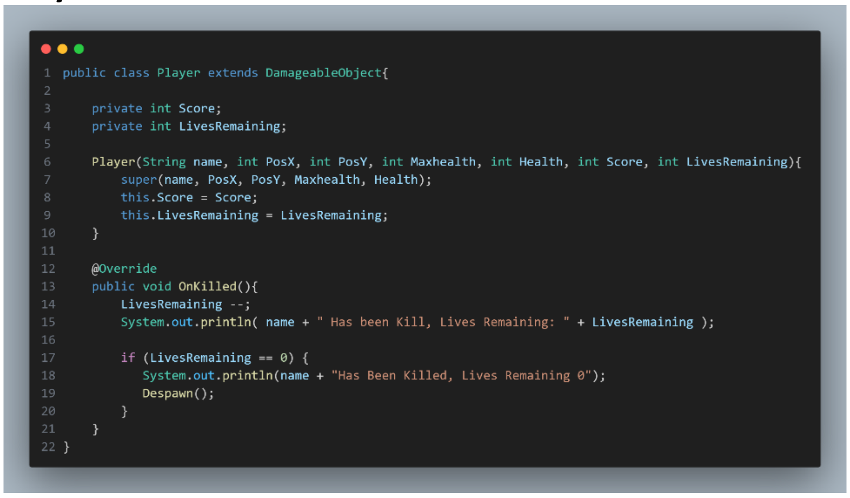
```java
public class GameObject {

    public String name;
    public int PosX;
    public int PosY;

    public GameObject(String name, int PosX, int PosY){
        this.name = name;
        this.PosX = PosX;
        this.PosY = PosY;
    }

    public String Getname(){
        return name;
    }

    public void Despawn(){
        System.out.println(name + "Has Ben Eliminated");
    }
}
```

➢ DamageableObject



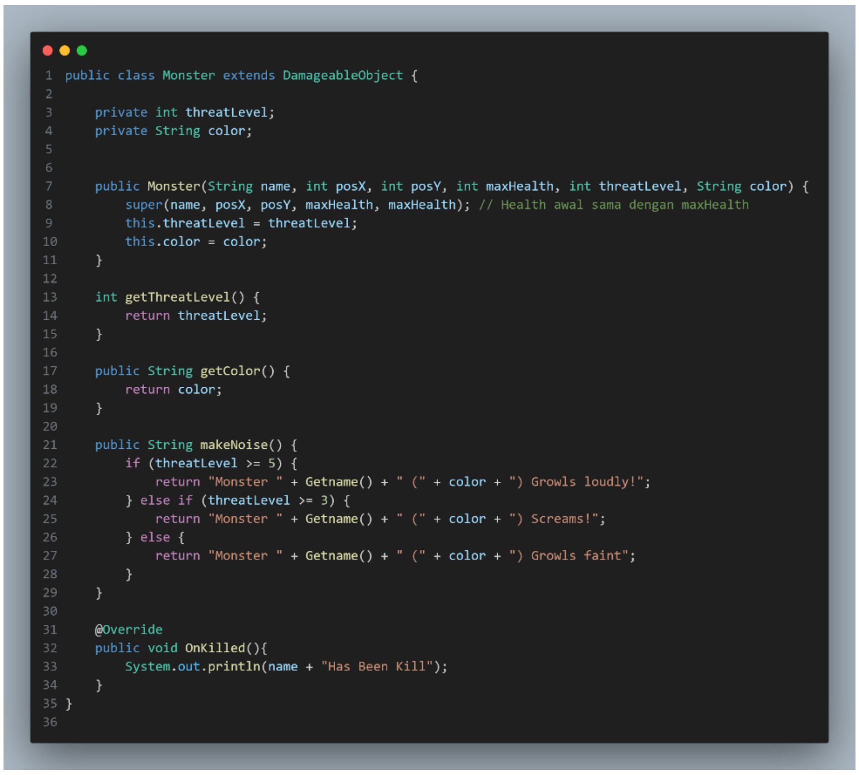
```java
abstract class DamageableObject extends GameObject{

    public int Maxhealth;
    private int Health;

    public DamageableObject(String name, int PosX, int PosY, int Maxhealth, int Health){
        super(name, PosX, PosY);
        this.Maxhealth = Maxhealth;
        this.Health = Health;
    }

    public boolean IsDead(){
        return Health <= 0;
    }


    public void TakeDamage(int damage){
        if (PosX == PosY) {
            Health = Maxhealth - damage;
            System.out.println("Name             : " + name);
            System.out.println("Took Damage      : " + damage + " Damage");
            System.out.println("Remaining Health : " + Health);

            if (IsDead()) {
                OnKilled();
            }
        }else{
            System.out.println("Dont Lose The Monster, Take Them !!!");
        }
    }

    public abstract void OnKilled();
}
```

➢ Player



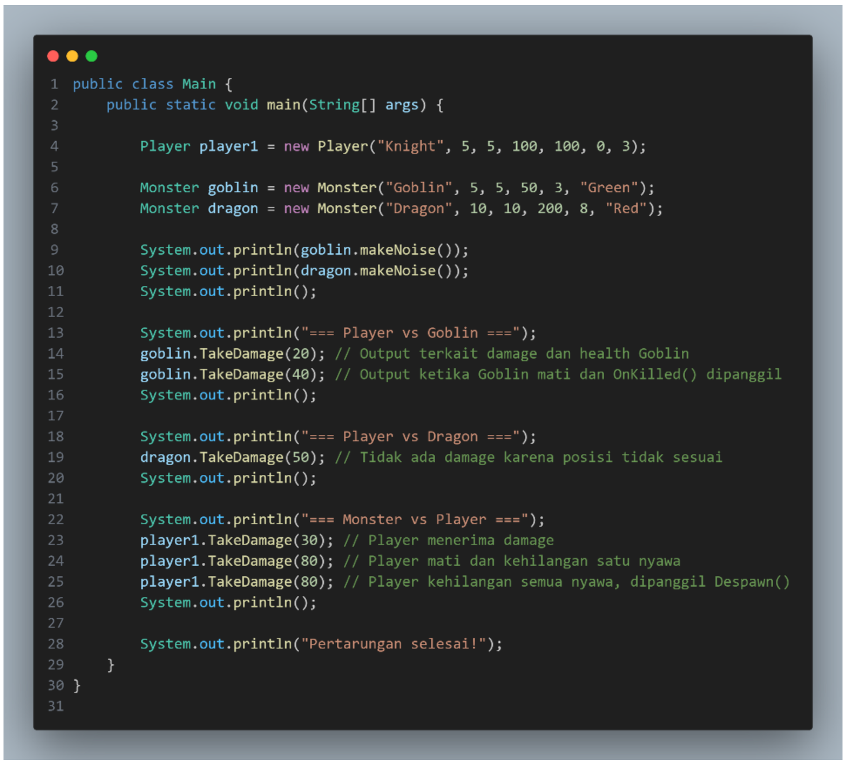
```java
public class Player extends DamageableObject{

    private int Score;
    private int LivesRemaining;

    Player(String name, int PosX, int PosY, int Maxhealth, int Health, int Score, int LivesRemaining){
        super(name, PosX, PosY, Maxhealth, Health);
        this.Score = Score;
        this.LivesRemaining = LivesRemaining;
    }

    @Override
    public void OnKilled(){
        LivesRemaining --;
        System.out.println( name + " Has been Kill, Lives Remaining: " + LivesRemaining );

        if (LivesRemaining == 0) {
            System.out.println(name + "Has Been Killed, Lives Remaining 0");
            Despawn();
        }
    }
}
```

➢ Monster



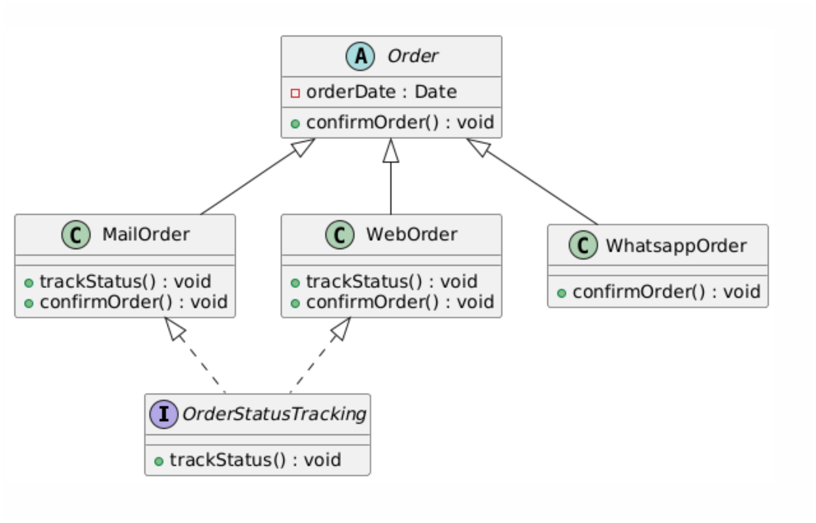
```java
public class Monster extends DamageableObject {

    private int threatLevel;
    private String color;


    public Monster(String name, int posX, int posY, int maxHealth, int threatLevel, String color) {
        super(name, posX, posY, maxHealth, maxHealth); // Health awal sama dengan maxHealth
        this.threatLevel = threatLevel;
        this.color = color;
    }

    int getThreatLevel() {
        return threatLevel;
    }

    public String getColor() {
        return color;
    }

    public String makeNoise() {
        if (threatLevel >= 5) {
            return "Monster " + Getname() + " (" + color + ") Growls loudly!";
        } else if (threatLevel >= 3) {
            return "Monster " + Getname() + " (" + color + ") Screams!";
        } else {
            return "Monster " + Getname() + " (" + color + ") Growls faint";
        }
    }

    @Override
    public void OnKilled(){
        System.out.println(name + "Has Been Kill");
    }
}

```

➢ Main



```java
public class Main {
    public static void main(String[] args) {

        Player player1 = new Player("Knight", 5, 5, 100, 100, 0, 3);

        Monster goblin = new Monster("Goblin", 5, 5, 50, 3, "Green");
        Monster dragon = new Monster("Dragon", 10, 10, 200, 8, "Red");

        System.out.println(goblin.makeNoise());
        System.out.println(dragon.makeNoise());
        System.out.println();

        System.out.println("=== Player vs Goblin ===");
        goblin.TakeDamage(20); // Output terkait damage dan health Goblin
        goblin.TakeDamage(40); // Output ketika Goblin mati dan OnKilled() dipanggil
        System.out.println();

        System.out.println("=== Player vs Dragon ===");
        dragon.TakeDamage(50); // Tidak ada damage karena posisi tidak sesuai
        System.out.println();

        System.out.println("=== Monster vs Player ===");
        player1.TakeDamage(30); // Player menerima damage
        player1.TakeDamage(80); // Player mati dan kehilangan satu nyawa
        player1.TakeDamage(80); // Player kehilangan semua nyawa, dipanggil Despawn()
        System.out.println();

        System.out.println("Pertarungan selesai!");
    }
}

```

2. A client of yours is a Seller who has a lot of media to accommodate orders from customers, but this Seller has difficulty in creating Order categories, he wants every order to have an order date and there must be a confirmation method for each category which is separated into 3 classes: MailOrder, WebOrder, WhatsappOrder. There is an "order status tracking" contract on the MailOrder and WebOrder classes
Help your client by describing his diagram classes that are easy for him to understand!

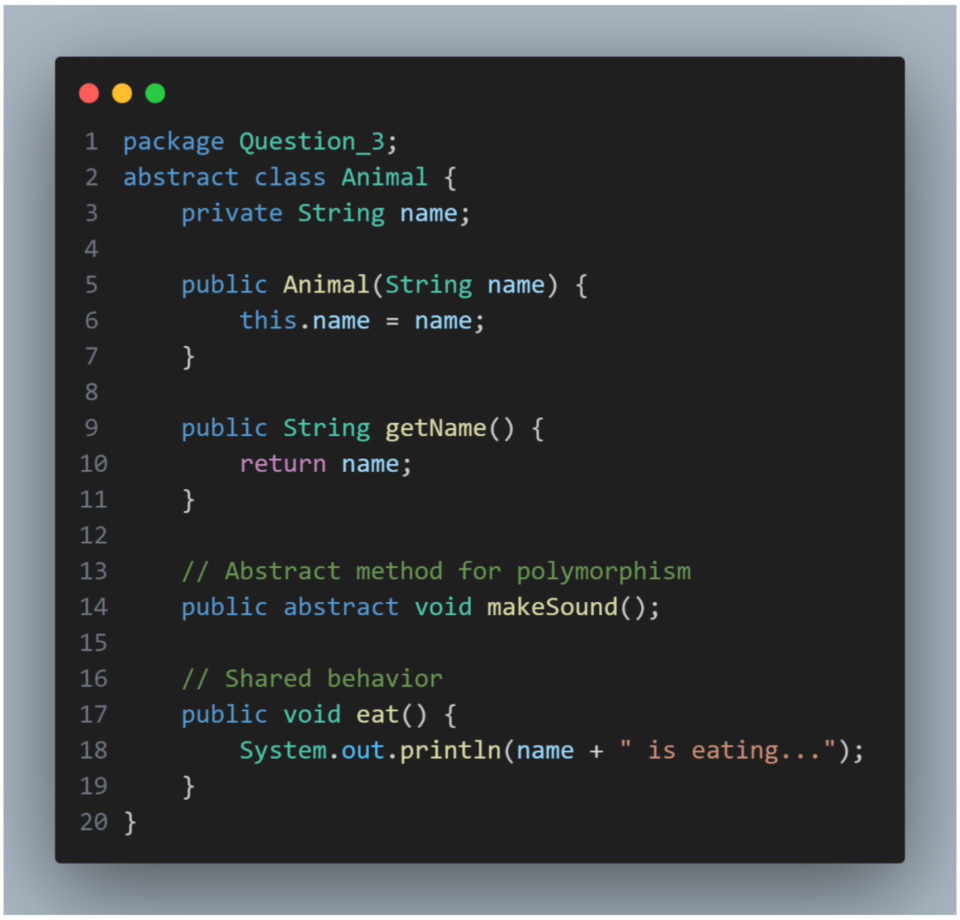
- 

Explanation of Diagram

- **Order Class**:
  - Acts as the parent class and enforces a common structure for all order types.
  - Includes a field for orderDate and an abstract method confirmOrder().
- **MailOrder and WebOrder Classes**:
  - Extend the Order class and implement the OrderStatusTracking interface.
  - Provide concrete implementations for confirmOrder() and trackStatus().
- **WhatsappOrder Class**:
  - Extends the Order class but does not implement OrderStatusTracking, as the requirement states that it does not need "order status tracking".
- **OrderStatusTracking Interface**:
  - Ensures that classes implementing it provide functionality for tracking order status.

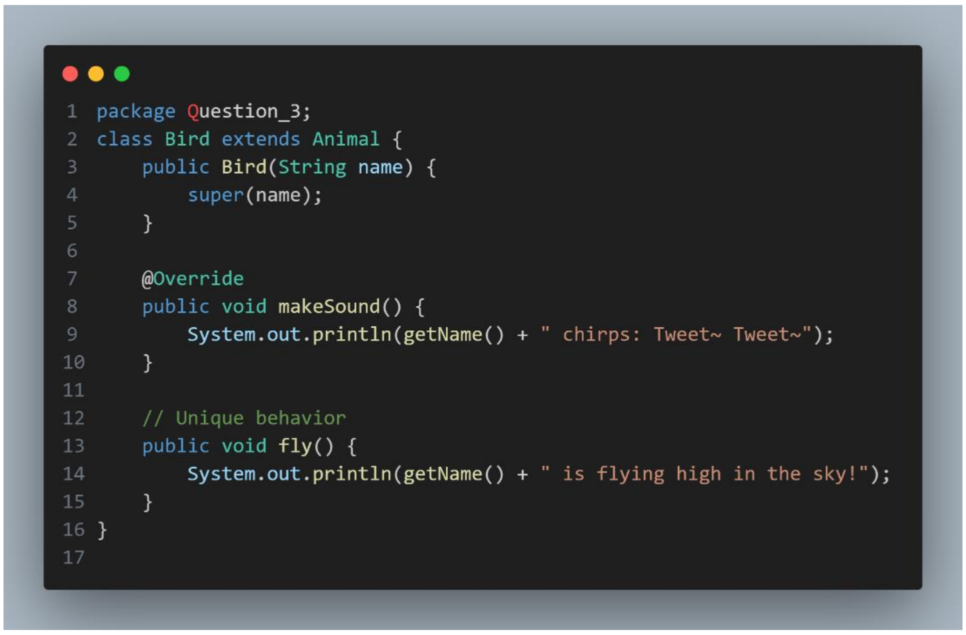
3. Give an example of program code using the concept of polymorphism (Heterogenous Collection, Object Casting, Polymorphic Arguments, InstanceOf) on 1 theme (for example, choose 1 theme: vehicle or electronic device or animal, etc... You can create any theme to apply the 4 points of polymorphism). Create interrelated java program code.

- **Class Animal**



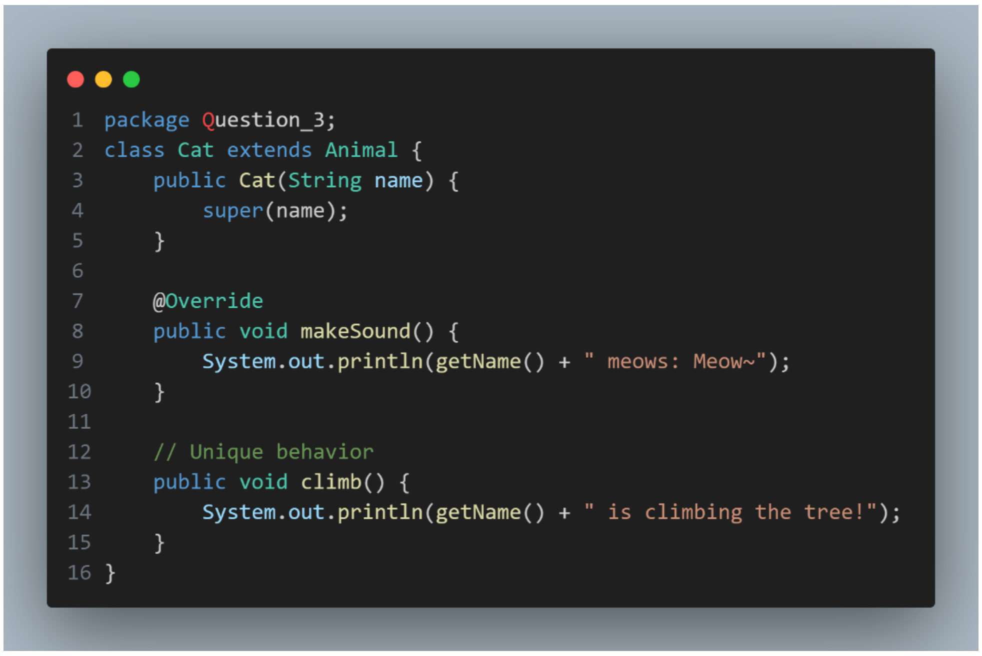
```java
package Question_3;
abstract class Animal {
    private String name;

    public Animal(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    // Abstract method for polymorphism
    public abstract void makeSound();

    // Shared behavior
    public void eat() {
        System.out.println(name + " is eating...");
    }
}
```

- **Class Bird**



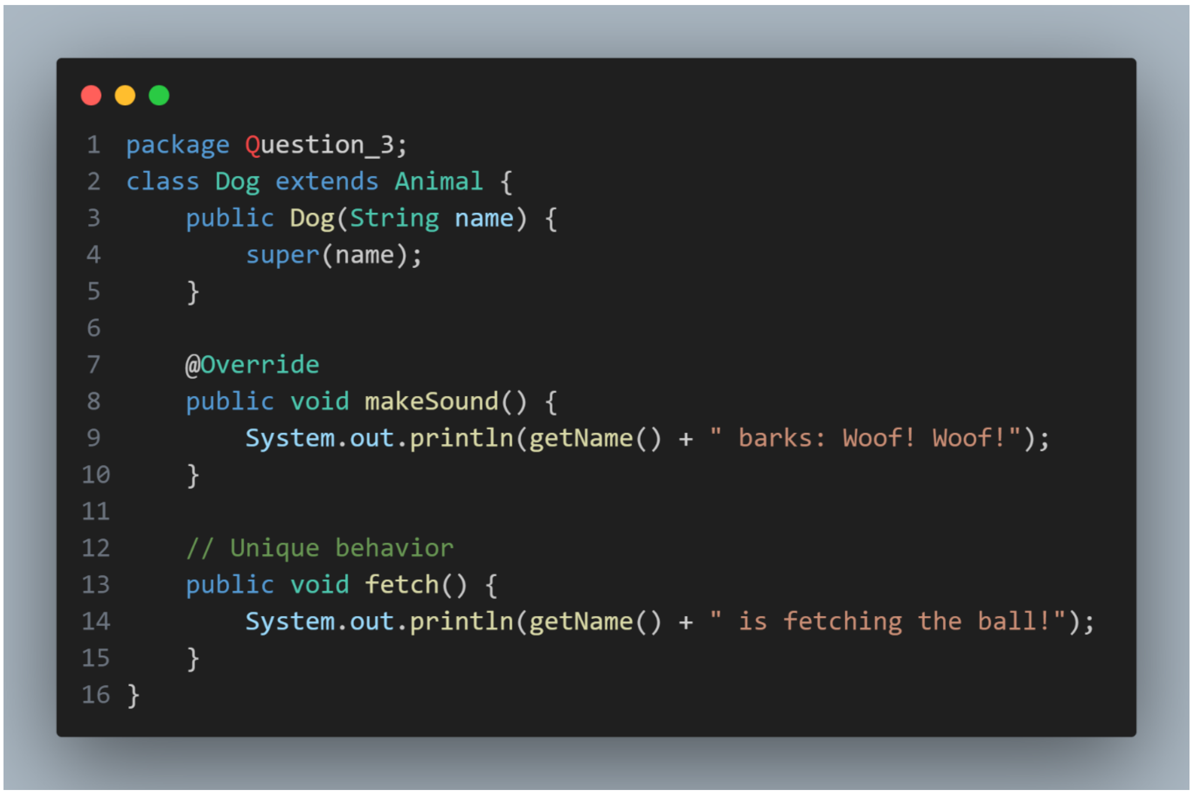
```java
package Question_3;
class Bird extends Animal {
    public Bird(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(getName() + " chirps: Tweet~ Tweet~");
    }

    // Unique behavior
    public void fly() {
        System.out.println(getName() + " is flying high in the sky!");
    }
}

```

➢ **Class Cat**



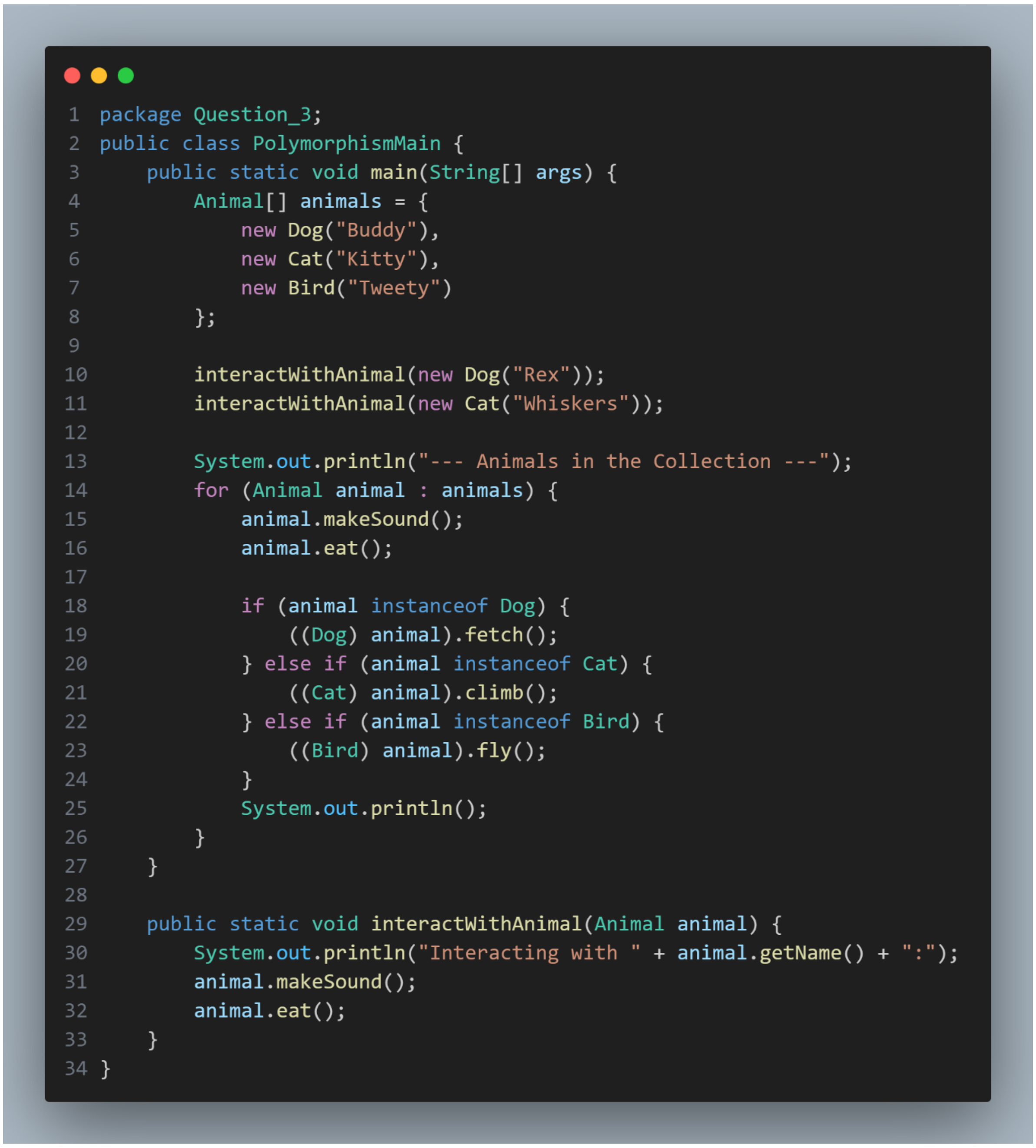
```java
package Question_3;
class Cat extends Animal {
    public Cat(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(getName() + " meows: Meow~");
    }

    // Unique behavior
    public void climb() {
        System.out.println(getName() + " is climbing the tree!");
    }
}
```

➢ **Class Dog**



```java
package Question_3;
class Dog extends Animal {
    public Dog(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(getName() + " barks: Woof! Woof!");
    }

    // Unique behavior
    public void fetch() {
        System.out.println(getName() + " is fetching the ball!");
    }
}
```

➢ **Class PolymorphismMain**



```java
package Question_3;
public class PolymorphismMain {
    public static void main(String[] args) {
        Animal[] animals = {
            new Dog("Buddy"),
            new Cat("Kitty"),
            new Bird("Tweety")
        };

        interactWithAnimal(new Dog("Rex"));
        interactWithAnimal(new Cat("Whiskers"));

        System.out.println("--- Animals in the Collection ---");
        for (Animal animal : animals) {
            animal.makeSound();
            animal.eat();

            if (animal instanceof Dog) {
                ((Dog) animal).fetch();
            } else if (animal instanceof Cat) {
                ((Cat) animal).climb();
            } else if (animal instanceof Bird) {
                ((Bird) animal).fly();
            }
            System.out.println();
        }
    }

    public static void interactWithAnimal(Animal animal) {
        System.out.println("Interacting with " + animal.getName() + ":");
        animal.makeSound();
        animal.eat();
    }
}
```

---- Good Luck ----