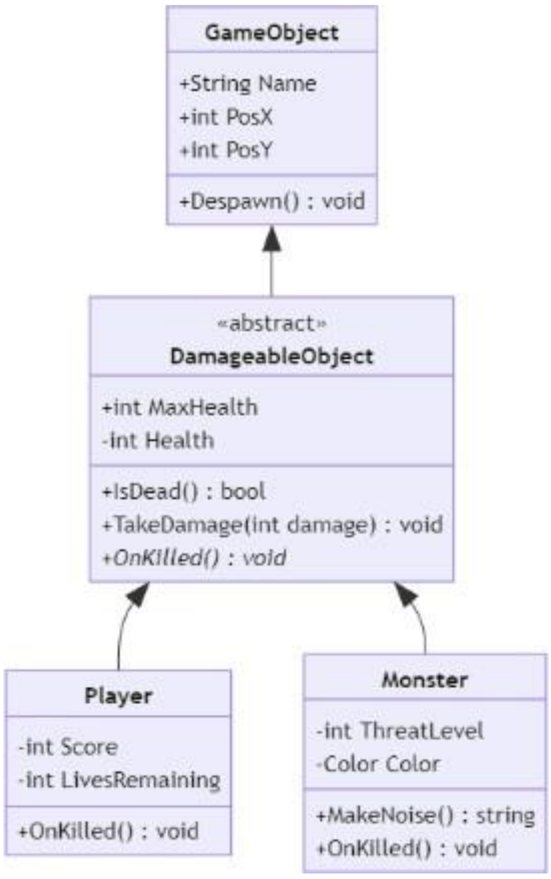
Name	: Evan Diantha Fafian
Class	: SIB 2G
Absent	: 09
NIM	: 2341760163

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.



Answer :

Diagram Explanation

- Class GameObject

Functions: A base class that provides generic attributes and functions for objects in the game, such as Name, position (PosX, PosY), and Despawn() method.

- Abstract Class DamageableObject

Function: Represents objects that have the property of being able to take damage.This class is abstract because it has an abstract method (OnKilled) that must be implemented by its child classes. In addition, it provides MaxHealth and Health attributes, as well as methods such as IsDead() and TakeDamage().

- Class Player

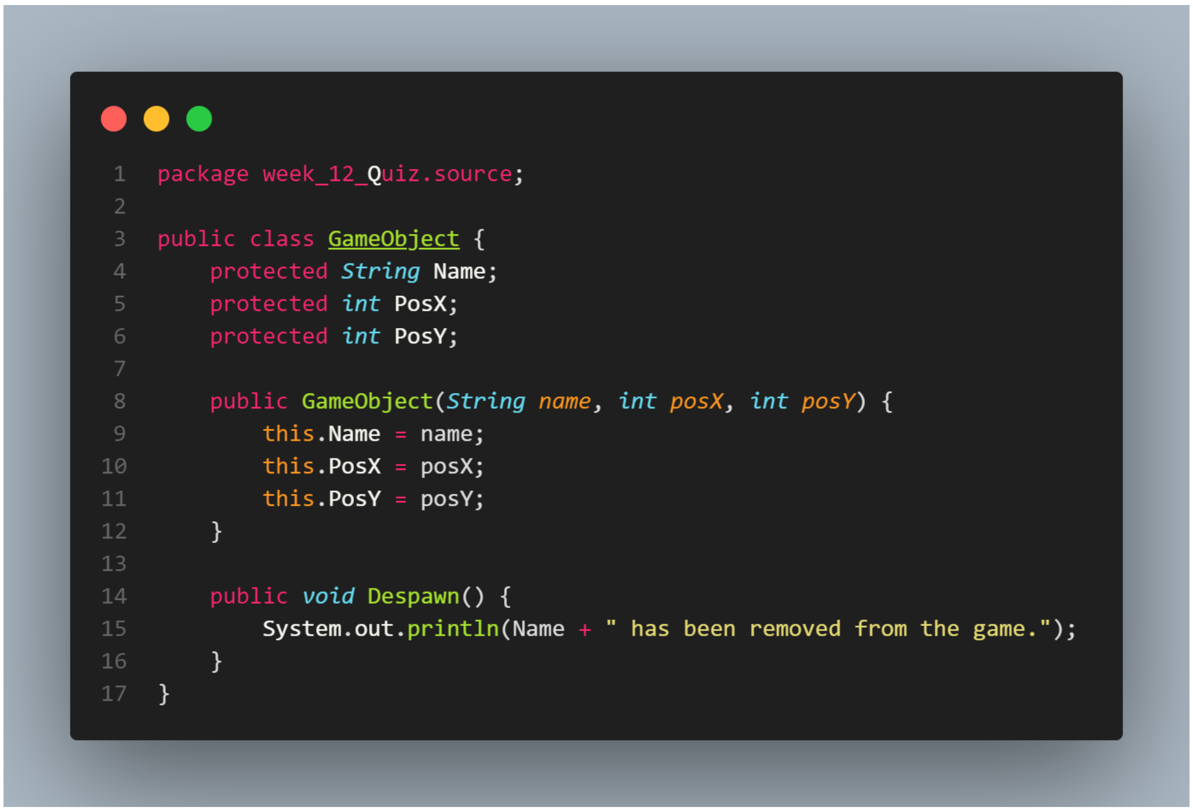
Function: A player-specific derivative of DamageableObject. Adds Score and LivesRemaining attributes, and implements the OnKilled() method.

- Class Monster

Function: Derived from the enemy-specific DamageableObject. Added ThreatLevel and Color attributes, as well as additional MakeNoise() method and OnKilled() implementation.

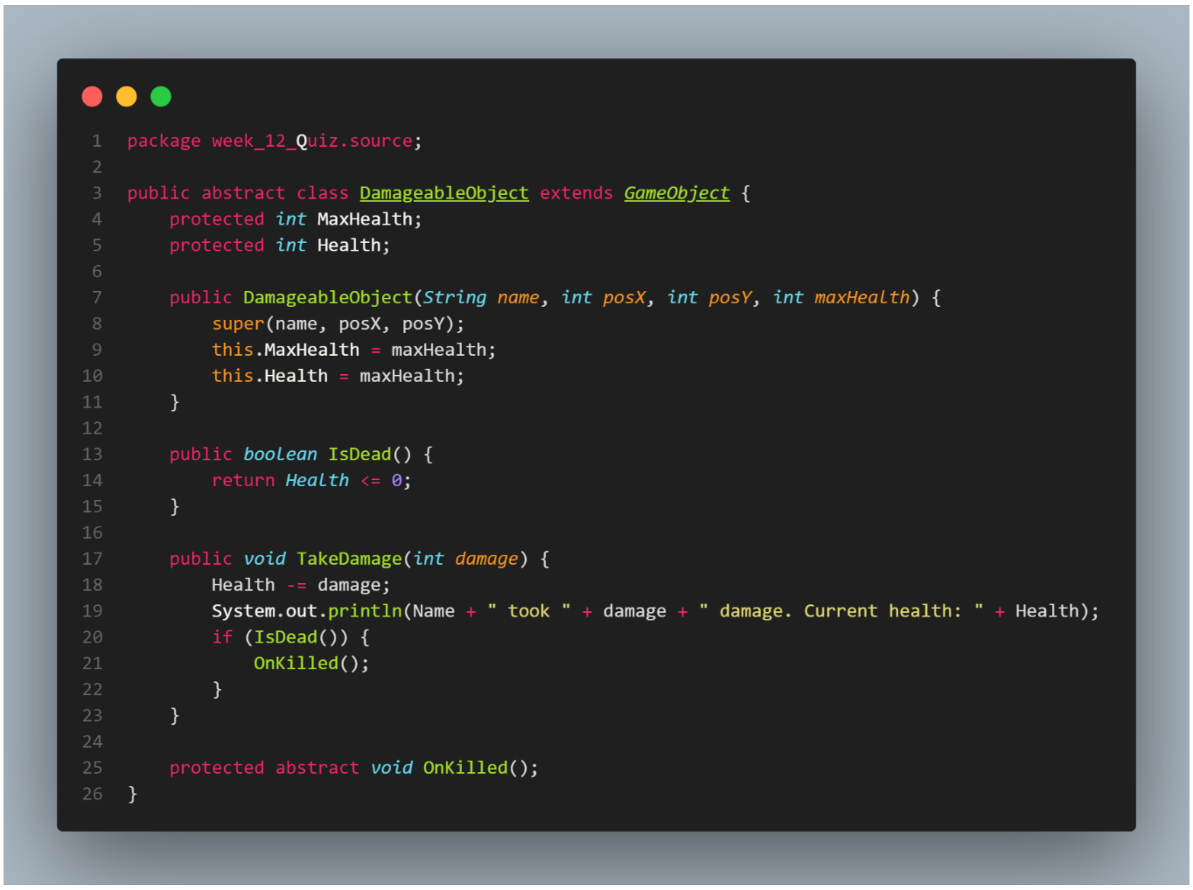
Program Implementation

- GameObject



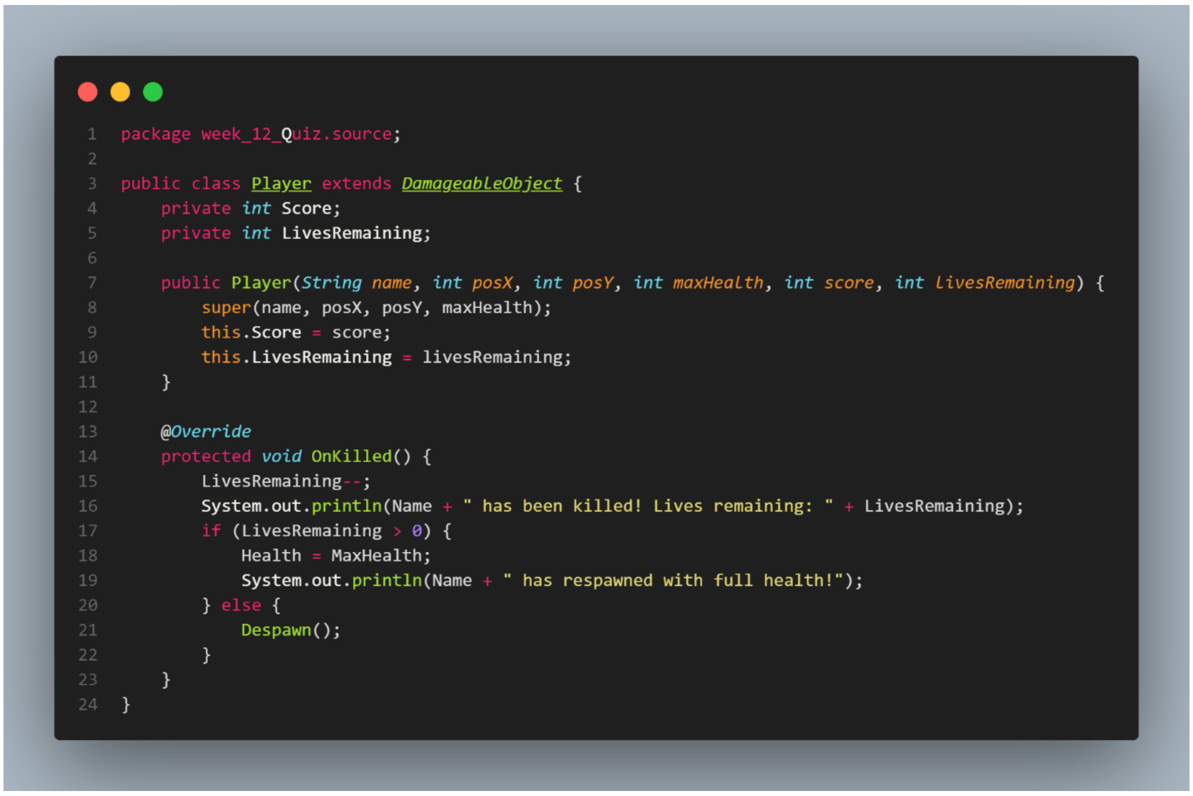
```java
package week_12_Quiz.source;

public class GameObject {
    protected String Name;
    protected int PosX;
    protected int PosY;

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

    public void Despawn() {
        System.out.println(Name + " has been removed from the game.");
    }
}
```

- DamageableObject



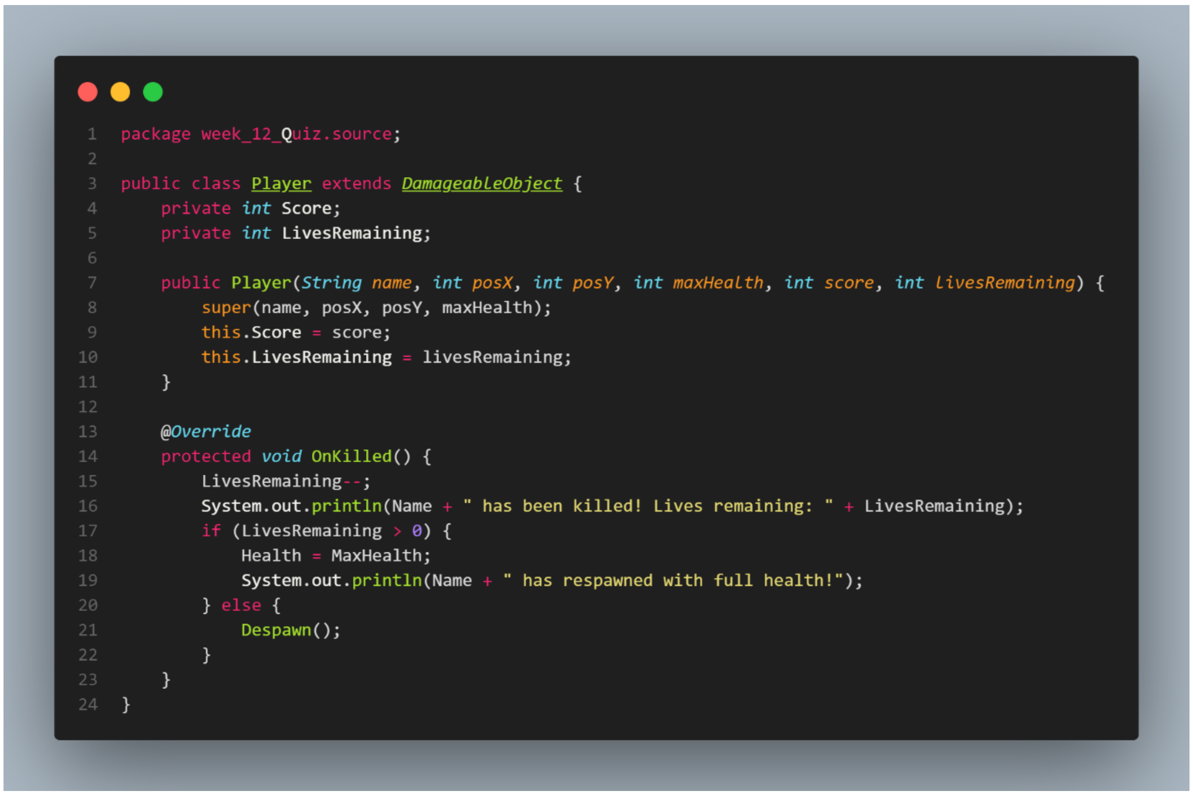
```java
package week_12_Quiz.source;

public abstract class DamageableObject extends GameObject {
    protected int MaxHealth;
    protected int Health;

    public DamageableObject(String name, int posX, int posY, int maxHealth) {
        super(name, posX, posY);
        this.MaxHealth = maxHealth;
        this.Health = maxHealth;
    }

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

    public void TakeDamage(int damage) {
        Health -= damage;
        System.out.println(Name + " took " + damage + " damage. Current health: " + Health);
        if (IsDead()) {
            OnKilled();
        }
    }

    protected abstract void OnKilled();
}
```

- Player



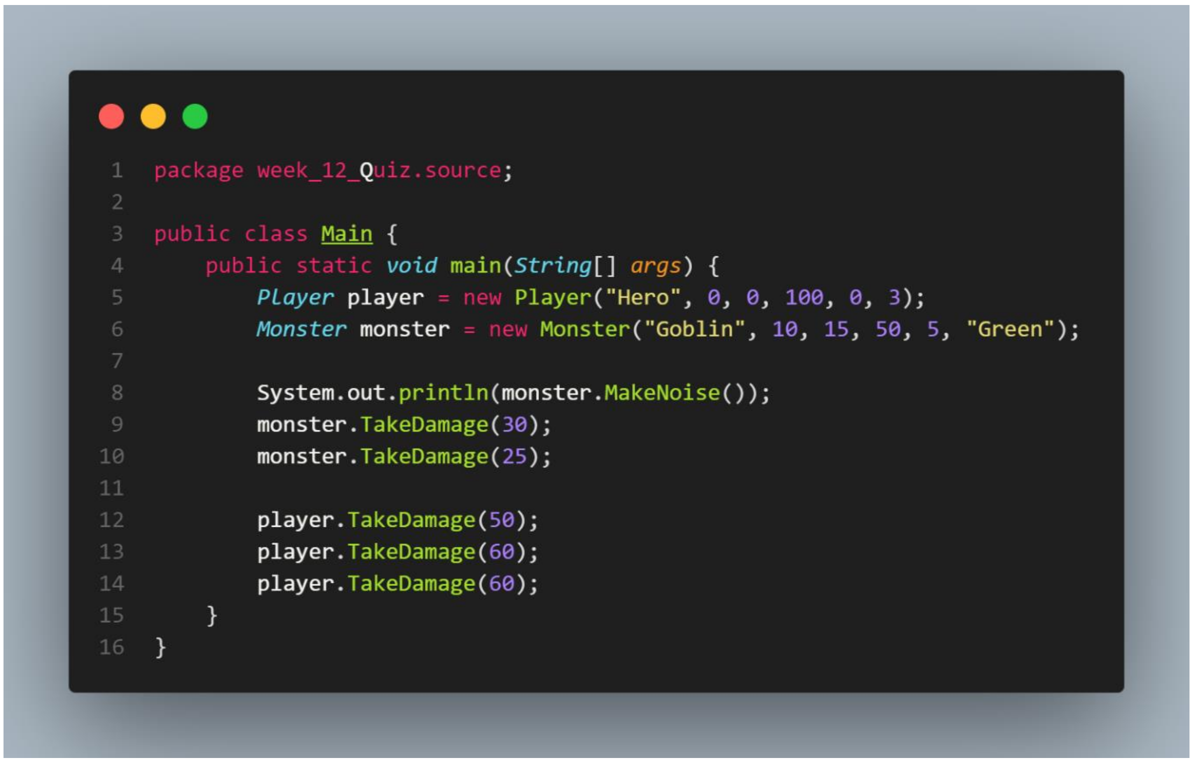
```java
package week_12_Quiz.source;

public class Player extends DamageableObject {
    private int Score;
    private int LivesRemaining;

    public Player(String name, int posX, int posY, int maxHealth, int score, int livesRemaining) {
        super(name, posX, posY, maxHealth);
        this.Score = score;
        this.LivesRemaining = livesRemaining;
    }

    @Override
    protected void OnKilled() {
        LivesRemaining--;
        System.out.println(Name + " has been killed! Lives remaining: " + LivesRemaining);
        if (LivesRemaining > 0) {
            Health = MaxHealth;
            System.out.println(Name + " has respawned with full health!");
        } else {
            Despawn();
        }
    }
}
```

- Monster



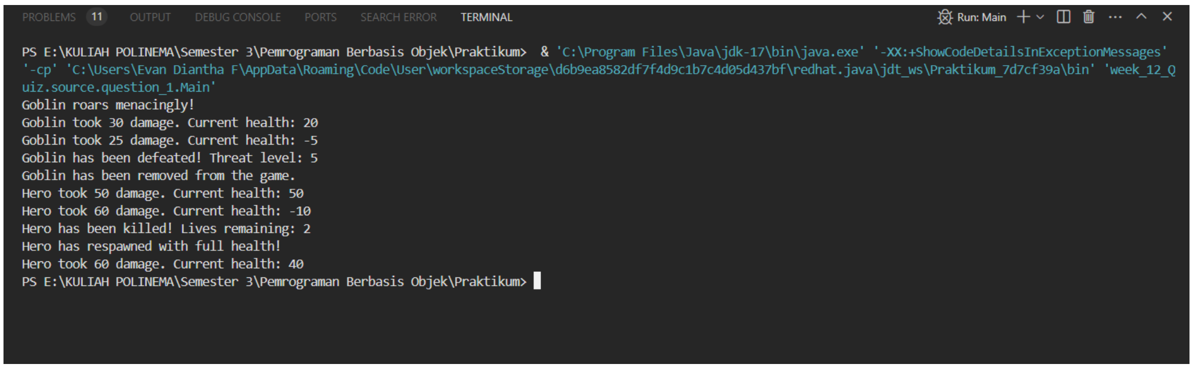
```java
package week_12_Quiz.source;

public class Player extends DamageableObject {
    private int Score;
    private int LivesRemaining;

    public Player(String name, int posX, int posY, int maxHealth, int score, int LivesRemaining) {
        super(name, posX, posY, maxHealth);
        this.Score = score;
        this.LivesRemaining = livesRemaining;
    }

    @Override
    protected void OnKilled() {
        LivesRemaining--;
        System.out.println(Name + " has been killed! Lives remaining: " + LivesRemaining);
        if (LivesRemaining > 0) {
            Health = MaxHealth;
            System.out.println(Name + " has respawned with full health!");
        } else {
            Despawn();
        }
    }
}
```

- Main



```java
package week_12_Quiz.source;

public class Main {
    public static void main(String[] args) {
        Player player = new Player("Hero", 0, 0, 100, 0, 3);
        Monster monster = new Monster("Goblin", 10, 15, 50, 5, "Green");

        System.out.println(monster.MakeNoise());
        monster.TakeDamage(30);
        monster.TakeDamage(25);

        player.TakeDamage(50);
        player.TakeDamage(60);
        player.TakeDamage(60);
    }
}
```

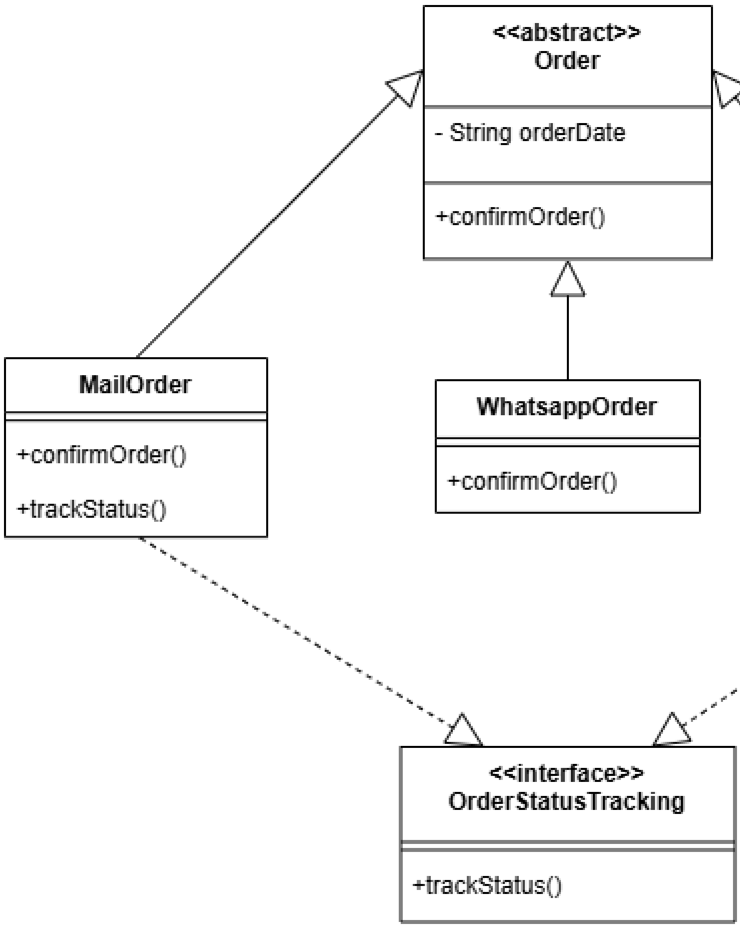
- Output



```
PS E:\KULIAH POLINEMA\Semester 3\Pemrograman Berbasis Objek\Praktikum>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages'
'-cp' 'C:\Users\Evan Diantha F\AppData\Roaming\Code\User\workspaceStorage\d6b9ea8582df7f4d9c1b7c4d05d437bf\redhat.java\jdt_ws\Praktikum_7d7cf39a\bin' 'week_12_Q
uiz.source.question_1.Main'
Goblin roars menacingly!
Goblin took 30 damage. Current health: 20
Goblin took 25 damage. Current health: -5
Goblin has been defeated! Threat level: 5
Goblin has been removed from the game.
Hero took 50 damage. Current health: 50
Hero took 60 damage. Current health: -10
Hero has been killed! Lives remaining: 2
Hero has respawned with full health!
Hero took 60 damage. Current health: 40
PS E:\KULIAH POLINEMA\Semester 3\Pemrograman Berbasis Objek\Praktikum>
```

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!

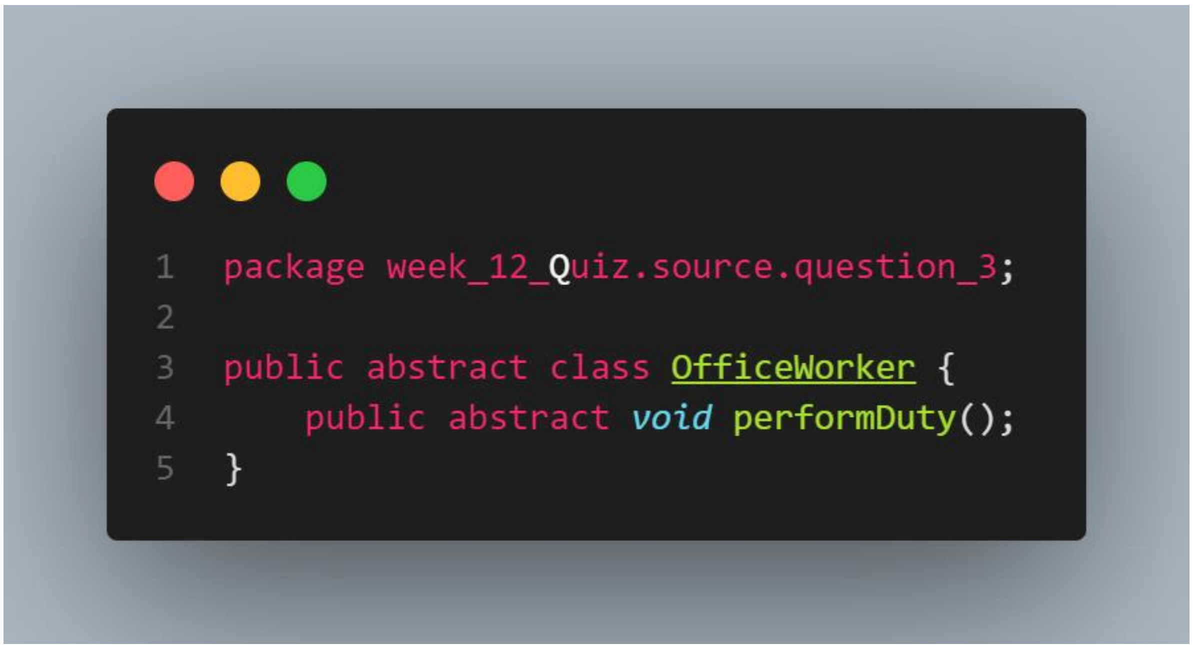
   Answer :



3. Give an example of program code using the concept of polymorphism (Heterogenous Collection, Object Casting, Polymorphic Arguments,

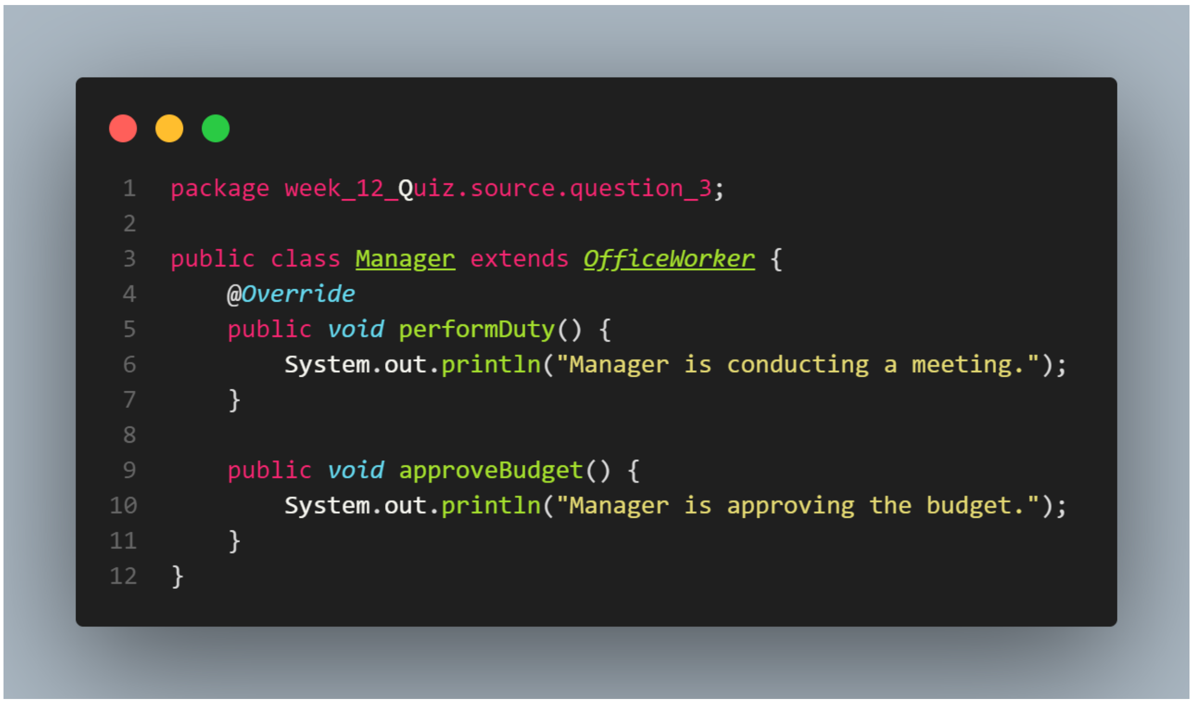
Answer:

- OfficeWorker



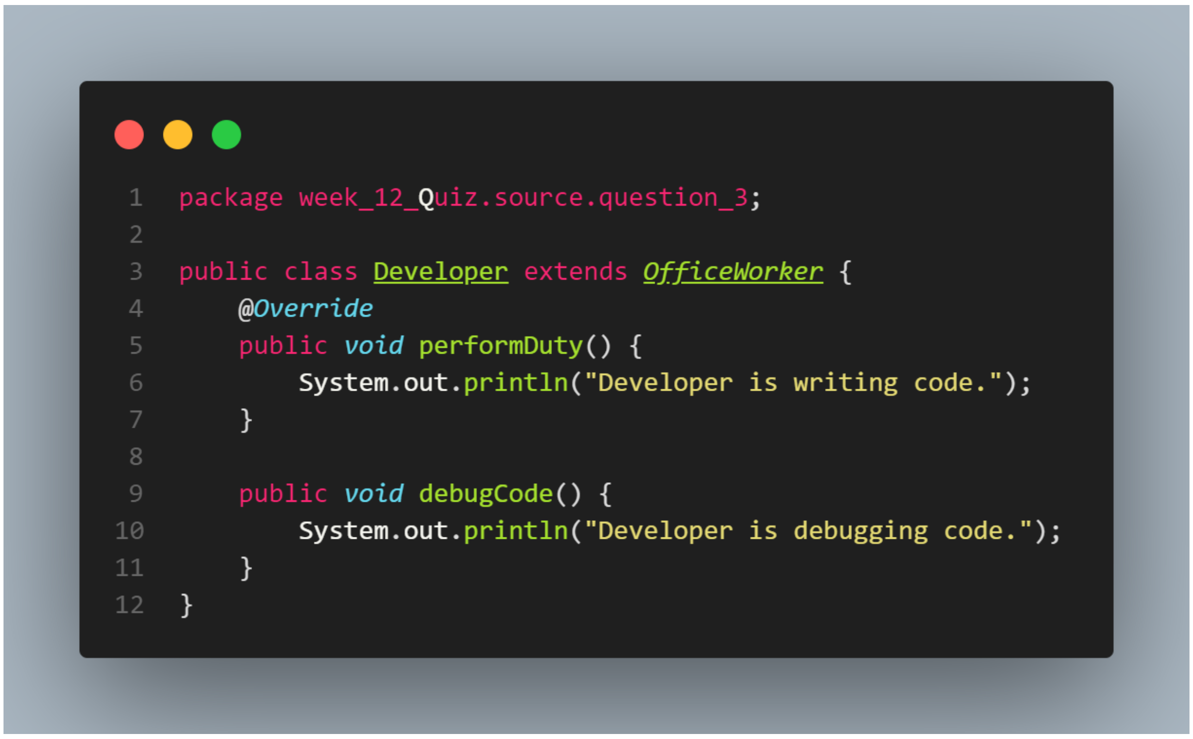
```java
package week_12_Quiz.source.question_3;

public abstract class OfficeWorker {
    public abstract void performDuty();
}
```

- Manager



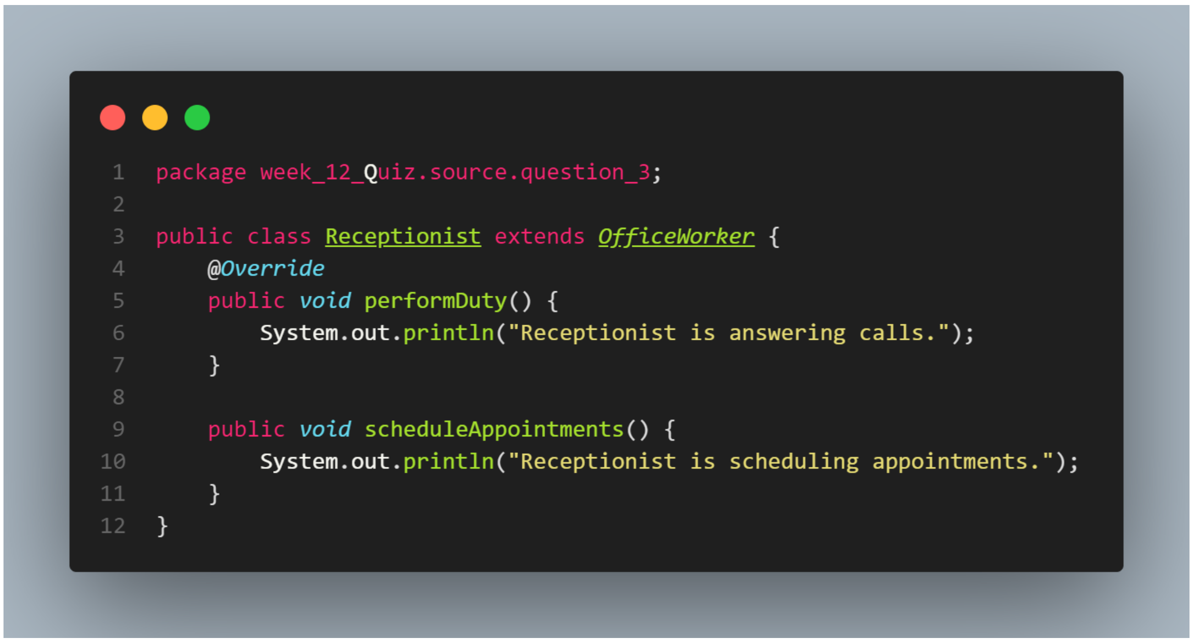
```java
package week_12_Quiz.source.question_3;

public class Manager extends OfficeWorker {
    @Override
    public void performDuty() {
        System.out.println("Manager is conducting a meeting.");
    }

    public void approveBudget() {
        System.out.println("Manager is approving the budget.");
    }
}
```

- Developer



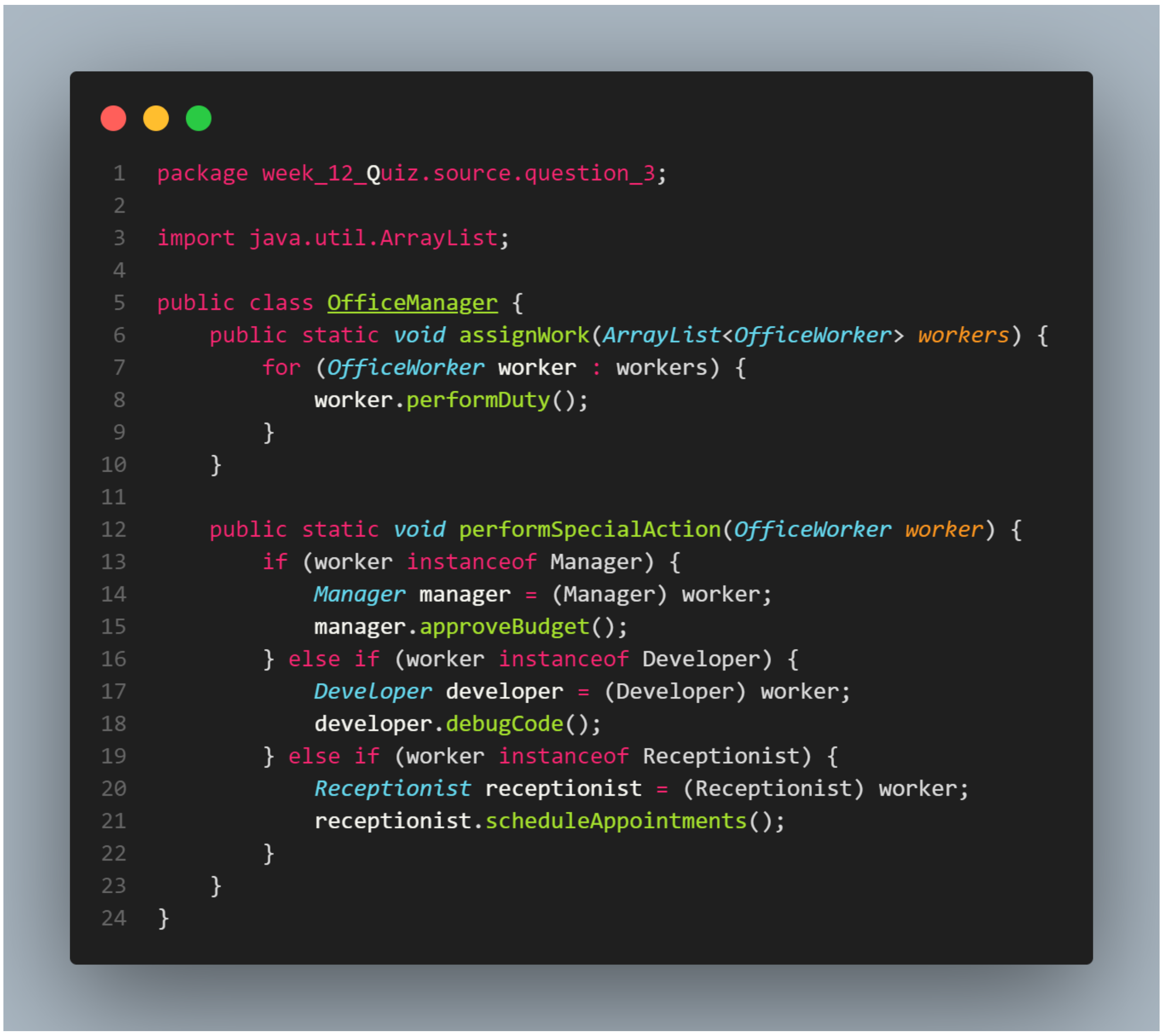
```java
package week_12_Quiz.source.question_3;

public class Developer extends OfficeWorker {
    @Override
    public void performDuty() {
        System.out.println("Developer is writing code.");
    }

    public void debugCode() {
        System.out.println("Developer is debugging code.");
    }
}
```

- Receptionist



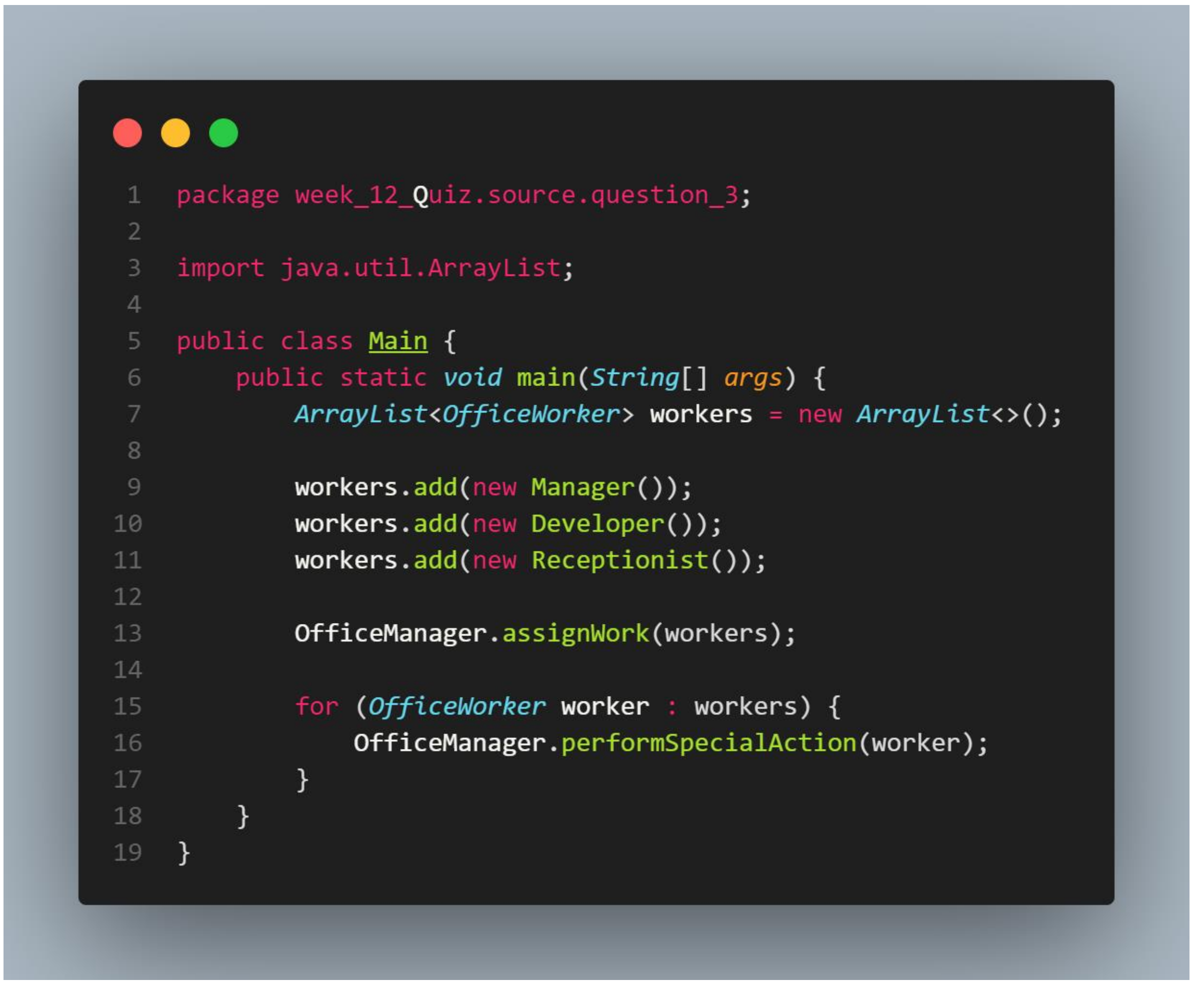
```java
package week_12_Quiz.source.question_3;

public class Receptionist extends OfficeWorker {
    @Override
    public void performDuty() {
        System.out.println("Receptionist is answering calls.");
    }

    public void scheduleAppointments() {
        System.out.println("Receptionist is scheduling appointments.");
    }
}
```

- OfficeManager



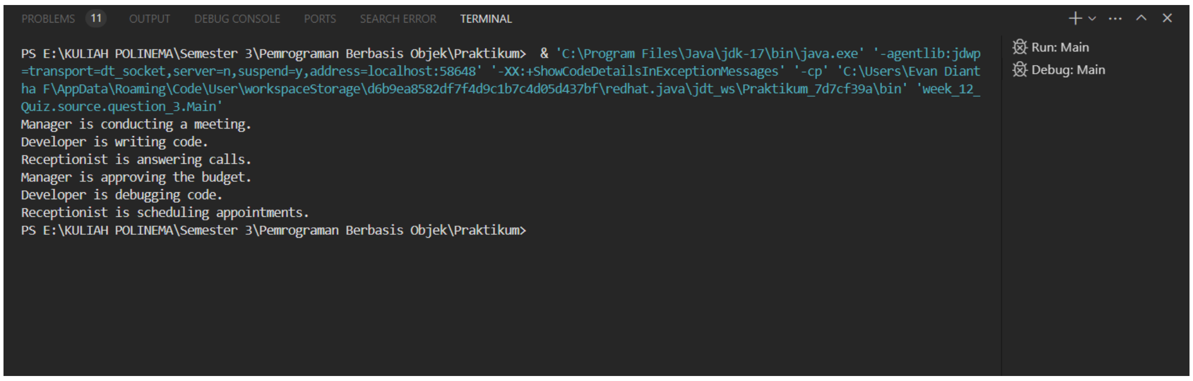
```java
package week_12_Quiz.source.question_3;

import java.util.ArrayList;

public class OfficeManager {
    public static void assignWork(ArrayList<OfficeWorker> workers) {
        for (OfficeWorker worker : workers) {
            worker.performDuty();
        }
    }

    public static void performSpecialAction(OfficeWorker worker) {
        if (worker instanceof Manager) {
            Manager manager = (Manager) worker;
            manager.approveBudget();
        } else if (worker instanceof Developer) {
            Developer developer = (Developer) worker;
            developer.debugCode();
        } else if (worker instanceof Receptionist) {
            Receptionist receptionist = (Receptionist) worker;
            receptionist.scheduleAppointments();
        }
    }
}
```

- Main



```java
package week_12_Quiz.source.question_3;

import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<OfficeWorker> workers = new ArrayList<>();

        workers.add(new Manager());
        workers.add(new Developer());
        workers.add(new Receptionist());

        OfficeManager.assignWork(workers);

        for (OfficeWorker worker : workers) {
            OfficeManager.performSpecialAction(worker);
        }
    }
}
```

- Output



```
PS E:\KULIAH POLINEMA\Semester 3\Pemrograman Berbasis Objek\Praktikum>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-agentlib:jdwp
=transport=dt_socket,server=n,suspend=y,address=localhost:58648' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Evan Diant
ha F\AppData\Roaming\Code\User\workspaceStorage\d6b9ea8582df7f4d9c1b7c4d05d437bf\redhat.java\jdt_ws\Praktikum_7d7cf39a\bin' 'week_12_
Quiz.source.question_3.Main'
Manager is conducting a meeting.
Developer is writing code.
Receptionist is answering calls.
Manager is approving the budget.
Developer is debugging code.
Receptionist is scheduling appointments.
PS E:\KULIAH POLINEMA\Semester 3\Pemrograman Berbasis Objek\Praktikum>
```

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.

GitHub :

https://github.com/rankadian/PEMROGRAMAN-BERBASIS-OBJEK.git

---- Good Luck ----