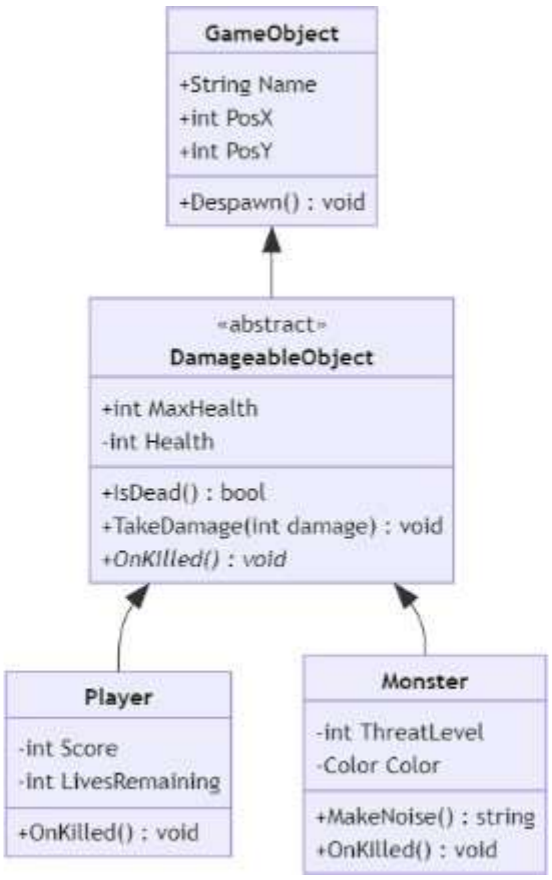
QUIZ QUESTIONS 2
OBJECT-BASED PROGRAMMING
PRACTICUM

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 :

I. This system is designed to organize various objects in the game that can be destroyed or interacted with by the player. In this system, there are four main classes: GameObject, DamageableObject, Player, and Monster.

   The GameObject class acts as a base class that contains common attributes and methods used by all objects in the game. DamageableObject is an abstract class that is derived from GameObject, with additional features to handle damage and death status of objects. Meanwhile, Player and Monster are derived classes from DamageableObject that have special attributes and functions according to their respective roles.

II. Class Diagram :

a. GameObject

- Atribut:
  - Name : String – Name of the object.
  - PosX : int – X position of the object.
  - PosY : int – Y position of the object.
- Methods:
  - Despawn() : Removes an object from the game.

b. DamageableObject (Abstract Class)

- Attributes:
  - MaxHealth : int – Maximum amount of health.
  - Health : int – Current amount of health.
- Abstract Methods:
  - TakeDamage(int damage) : Reduces the Health value.
  - OnKilled() : Defines the action when the object dies.
- Non-Abstract Methods:
  - IsDead() : Checks whether the Health value is zero or less.

c. Player

- Attributes:
  - Score : int – Player score.
  - LivesRemaining : int – Number of player lives remaining.
- Methods:
  - OnKilled() : Implements the logic when the player runs out of lives.

d. Monster

➢ Attributes:

- ThreatLevel : int – Monster threat level.
- Color : String – Monster color.

➢ Methods:

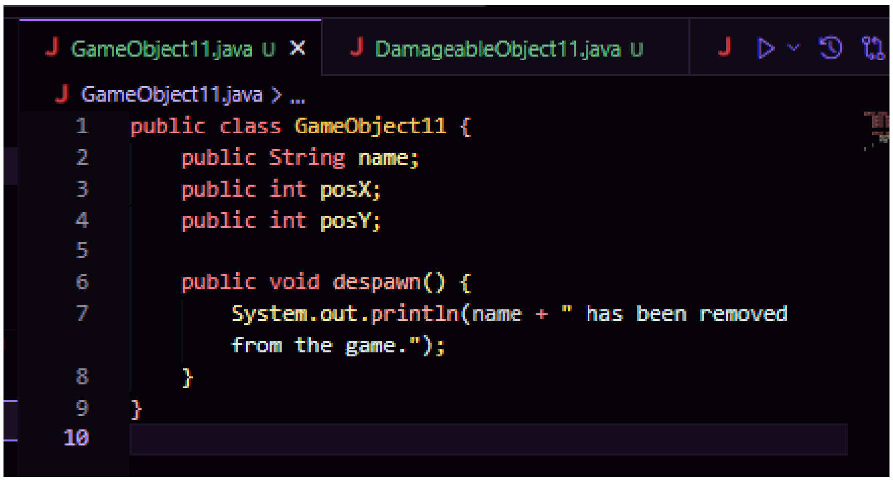
- MakeNoise() : Generates monster sound.
- OnKilled() : Implements the logic when the monster dies.

III. Relationship Between Classes:

- GameObject is the parent class for DamageableObject.
- DamageableObject is an abstract class that is the basis for Player and Monster.
- The Player and Monster classes have specific implementations for the OnKilled() abstract method according to their respective functions.

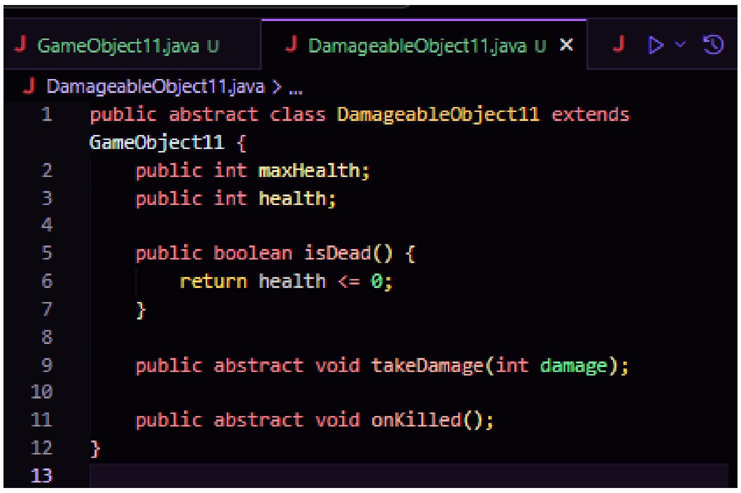
IV. Java Program Code

1. GameObject



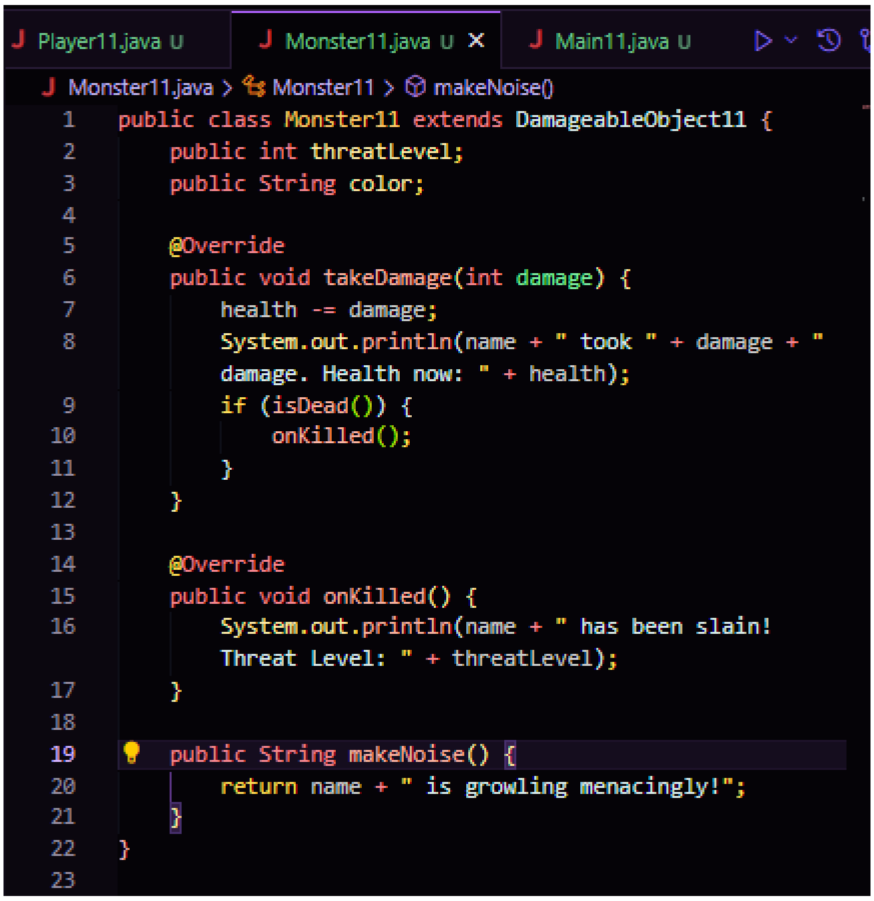
```java
public class GameObject11 {
    public String name;
    public int posX;
    public int posY;

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

2. DamageableObject



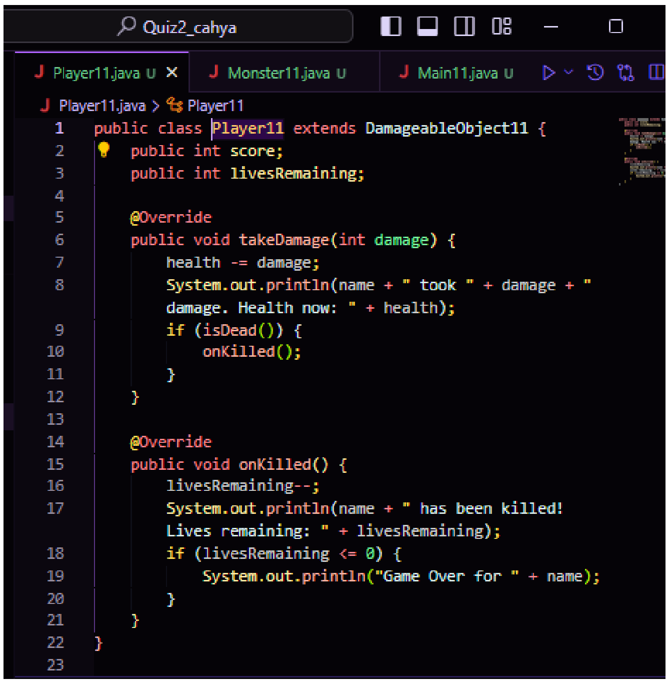
```java
public abstract class DamageableObject11 extends
GameObject11 {
    public int maxHealth;
    public int health;

    public boolean isDead() {
        return health <= 0;
    }

    public abstract void takeDamage(int damage);

    public abstract void onKilled();
}
```

3. Monster



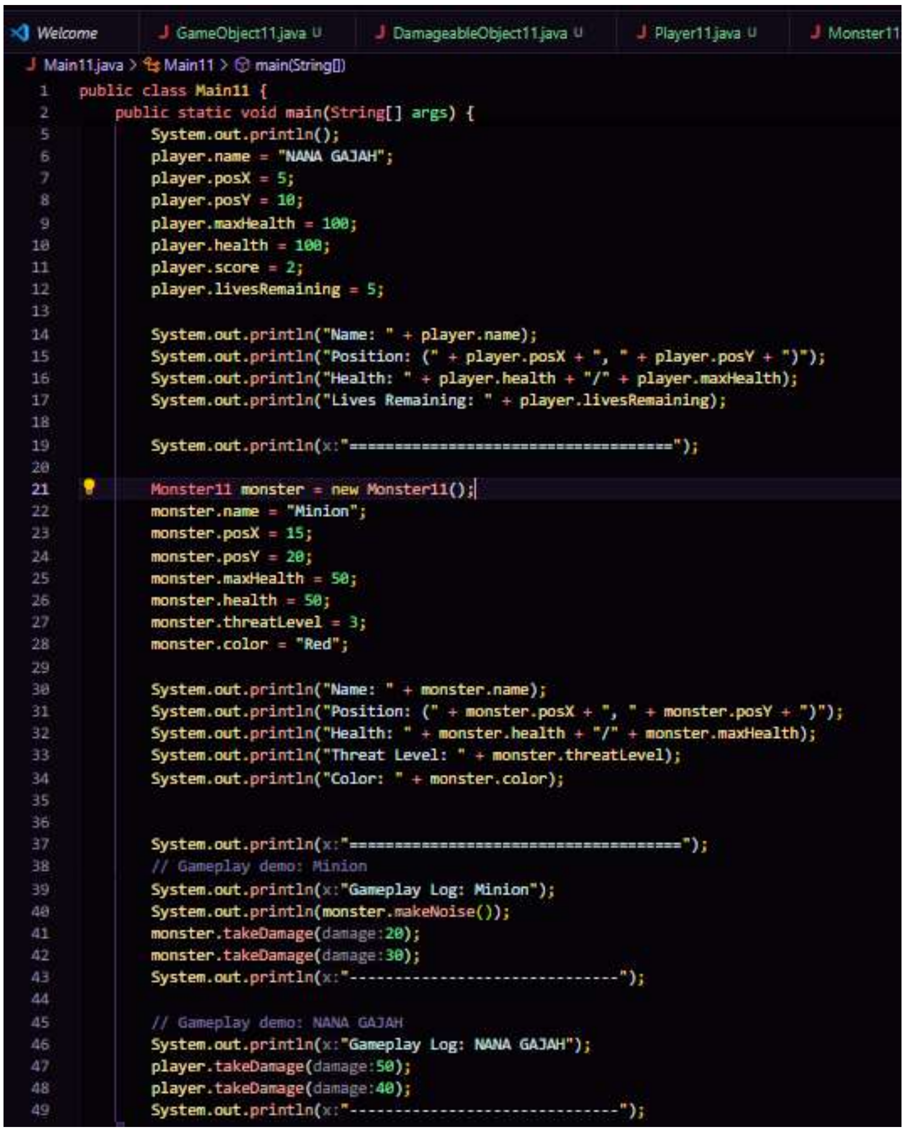
```java
public class Monster11 extends DamageableObject11 {
    public int threatLevel;
    public String color;

    @Override
    public void takeDamage(int damage) {
        health -= damage;
        System.out.println(name + " took " + damage + "
        damage. Health now: " + health);
        if (isDead()) {
            onKilled();
        }
    }

    @Override
    public void onKilled() {
        System.out.println(name + " has been slain!
        Threat Level: " + threatLevel);
    }

    public String makeNoise() {
        return name + " is growling menacingly!";
    }
}
```

4. Player



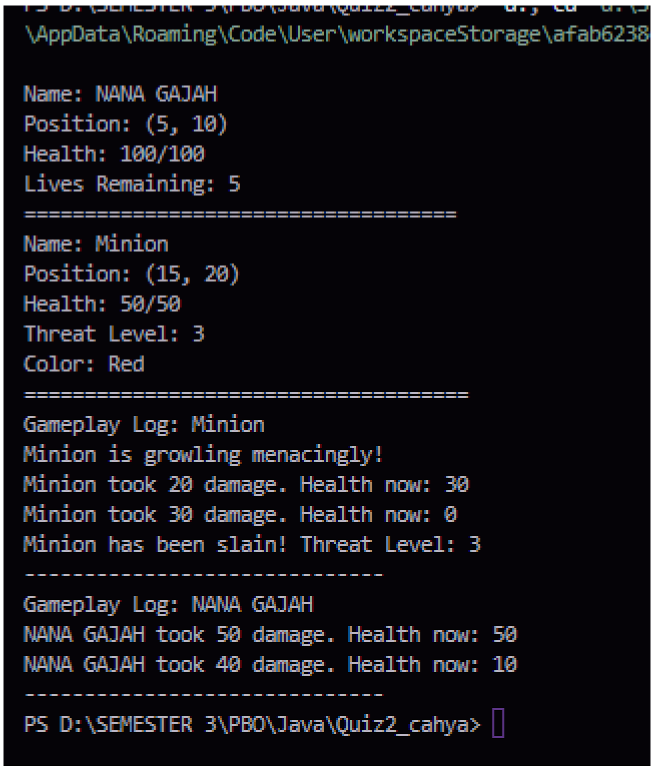
```java
public class Player11 extends DamageableObject11 {
    public int score;
    public int livesRemaining;

    @Override
    public void takeDamage(int damage) {
        health -= damage;
        System.out.println(name + " took " + damage + "
        damage. Health now: " + health);
        if (isDead()) {
            onKilled();
        }
    }

    @Override
    public void onKilled() {
        livesRemaining--;
        System.out.println(name + " has been killed!
        Lives remaining: " + livesRemaining);
        if (livesRemaining <= 0) {
            System.out.println("Game Over for " + name);
        }
    }
}
```

5. Main


Welcome
GameObject11.java U
DamageableObject11.java U
Player11.java U
Monster11
Main11.java > Main11 > main(String[])

```java
public class Main11 {
    public static void main(String[] args) {
        System.out.println();
        player.name = "NANA GAJAH";
        player.posX = 5;
        player.posY = 10;
        player.maxHealth = 100;
        player.health = 100;
        player.score = 2;
        player.livesRemaining = 5;

        System.out.println("Name: " + player.name);
        System.out.println("Position: (" + player.posX + ", " + player.posY + ")");
        System.out.println("Health: " + player.health + "/" + player.maxHealth);
        System.out.println("Lives Remaining: " + player.livesRemaining);

        System.out.println(x:"====================================");

        Monster11 monster = new Monster11();
        monster.name = "Minion";
        monster.posX = 15;
        monster.posY = 20;
        monster.maxHealth = 50;
        monster.health = 50;
        monster.threatLevel = 3;
        monster.color = "Red";

        System.out.println("Name: " + monster.name);
        System.out.println("Position: (" + monster.posX + ", " + monster.posY + ")");
        System.out.println("Health: " + monster.health + "/" + monster.maxHealth);
        System.out.println("Threat Level: " + monster.threatLevel);
        System.out.println("Color: " + monster.color);


        System.out.println(x:"====================================");
        // Gameplay demo: Minion
        System.out.println(x:"Gameplay Log: Minion");
        System.out.println(monster.makeNoise());
        monster.takeDamage(damage:20);
        monster.takeDamage(damage:30);
        System.out.println(x:"------------------------------");

        // Gameplay demo: NANA GAJAH
        System.out.println(x:"Gameplay Log: NANA GAJAH");
        player.takeDamage(damage:50);
        player.takeDamage(damage:40);
        System.out.println(x:"------------------------------");
```

6. Output



```
PS D:\SEMESTER 3\PBO\Java\Quiz2_cahya> d:; cd d:\S
\AppData\Roaming\Code\User\workspaceStorage\afab6238

Name: NANA GAJAH
Position: (5, 10)
Health: 100/100
Lives Remaining: 5
====================================
Name: Minion
Position: (15, 20)
Health: 50/50
Threat Level: 3
Color: Red
=====================================
Gameplay Log: Minion
Minion is growling menacingly!
Minion took 20 damage. Health now: 30
Minion took 30 damage. Health now: 0
Minion has been slain! Threat Level: 3
------------------------------
Gameplay Log: NANA GAJAH
NANA GAJAH took 50 damage. Health now: 50
NANA GAJAH took 40 damage. Health now: 10
------------------------------
PS D:\SEMESTER 3\PBO\Java\Quiz2_cahya>
```

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 :

I. This class diagram explains that Order is a base abstract class for various types of orders, with an orderDate attribute and an abstract confirmOrder() method, while MailOrder, WebOrder, and WhatsappOrder are subclasses that each implement the trackStatus() method (for MailOrder and WebOrder) or just confirmOrder() (for WhatsappOrder).

II. Class Diagram:

a. Order (Abstract Class)

➢ Attributes:
  - orderDate : String – Order date.

➢ Methods:

- confirmOrder() : Abstract method to confirm an order.

b. MailOrder (Class)

➢ Atribut:
  - (Mewarisi atribut dari Order)

➢ Methods:
  - confirmOrder() : Mengonfirmasi pesanan melalui surat.
  - trackStatus() : Melacak status pesanan.

c. WebOrder (Class)

➢ Atribut:
  - (Inherit attributes from Order)

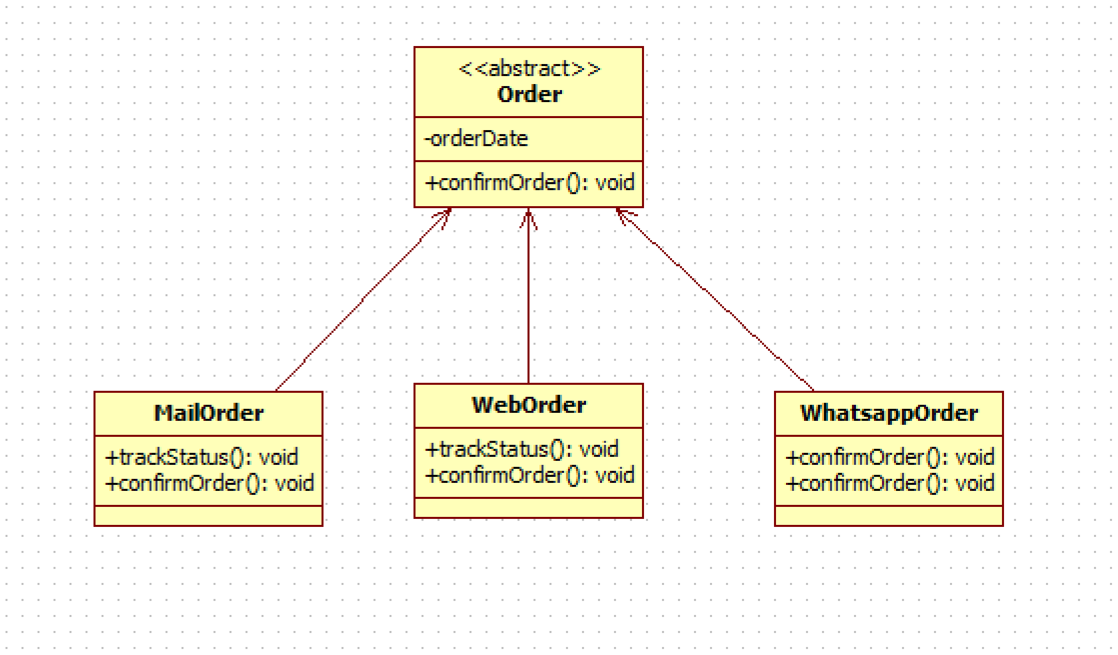
➢ Methods:
  - confirmOrder() : Confirms orders via website.
  - trackStatus() : Tracks order status.

d. WhatsappOrder (Class)

➢ Attributes:
  - (Inherit attributes from Order)

➢ Methods:
  - confirmOrder() : Confirms orders via WhatsApp.

III. Class Diagram



IV. Explanation:

The Order class is abstract, containing an orderDate attribute and an abstract method confirmOrder(). Its subclasses MailOrder, WebOrder, and WhatsappOrder provide implementations for the confirmOrder()method. Additionally, MailOrder and WebOrder include a trackStatus() method, whereas WhatsappOrder solely implements the confirmOrder() method.

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.

Answer :

I. This system is designed to manage various types of buildings and calculate their construction costs efficiently. There are three main classes: Building, House, Building, and Mall.
Building is an abstract class that stores general information such as the name and area of the building, and a method for calculating construction costs.
House, Building, and Mall are subclasses that each add more specific information, such as the number of rooms, the number of floors, and the number of shops.

II. Class

a. Building (Abstract Class)

➢ Attributes :
  - name: Name of the building
  - area: Area of the building in m²

➢ Methods:
  - calculateCost(): Abstract method for calculating construction costs
  - getName(): Gets the name of the building
  - getArea(): Gets the area of the building

b. House (Building Subclass)

- Attributes:
    - numberofRooms: Number of rooms in the house
- Methods:
    - calculateCost(): Calculate the construction cost of a house based on the area
    - infoHouse(): Display the number of rooms in the house

c. Building (Building Subclass)

- Attributes:
    - numberOfFloors: Number of floors in the building
- Methods:
    - calculateCost(): Calculate the construction cost of a building based on the area and floors
    - infoGedung(): Display the number of floors in the building

d. Mall (Building Subclass)

- Attributes:
    - numberOfStores: Number of stores in the mall

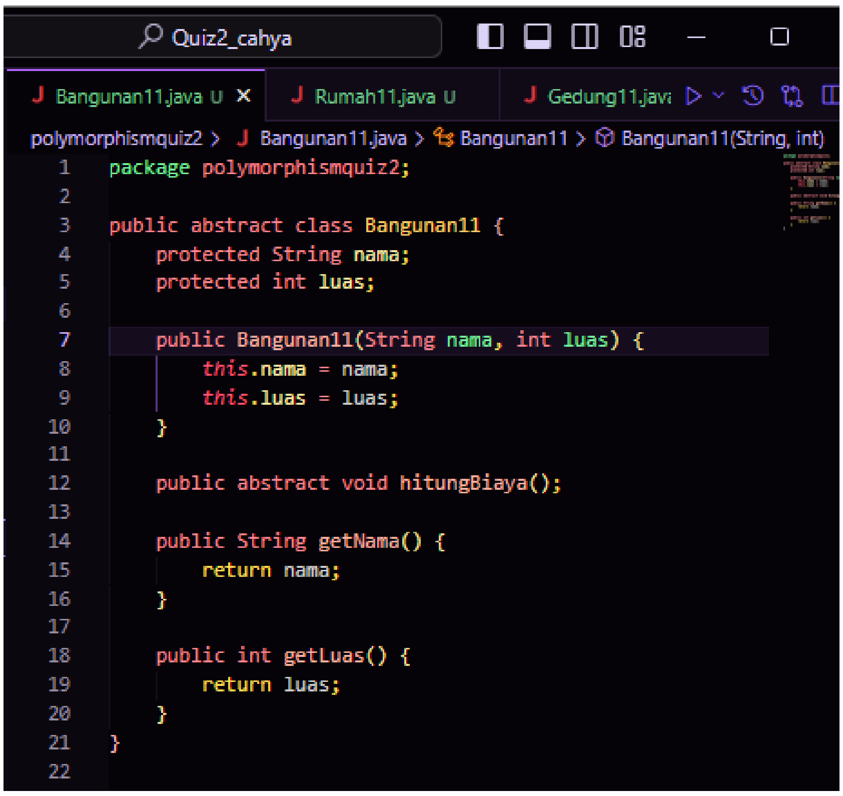
- Methods:
    - calculateCost(): Calculate the construction cost of a mall based on the area
    - infoMall(): Display the number of mall stores

III. Relationship Between Classes

- Building is an abstract parent class with name and area attributes, and methods hitungBiaya(), getNama(), and getLuas().
- Rumah, Gedung, and Mall are subclasses of Bangunan that add specific attributes and implement the hitungBiaya() method to calculate construction costs according to their type.

IV. Java Program Code

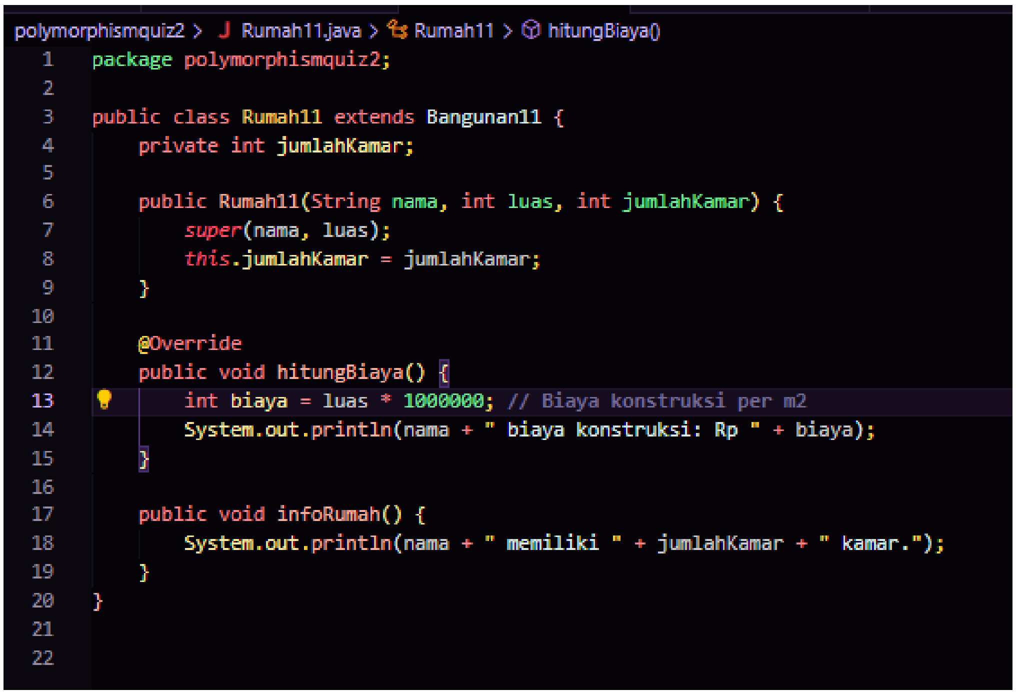
1. Bangunan



Quiz2_cahya
Bangunan11.java U
Rumah11.java U
Gedung11.java
polymorphismquiz2 > Bangunan11.java > Bangunan11 > Bangunan11(String, int)

```java
package polymorphismquiz2;

public abstract class Bangunan11 {
    protected String nama;
    protected int luas;

    public Bangunan11(String nama, int luas) {
        this.nama = nama;
        this.luas = luas;
    }

    public abstract void hitungBiaya();

    public String getNama() {
        return nama;
    }

    public int getLuas() {
        return luas;
    }
}
```

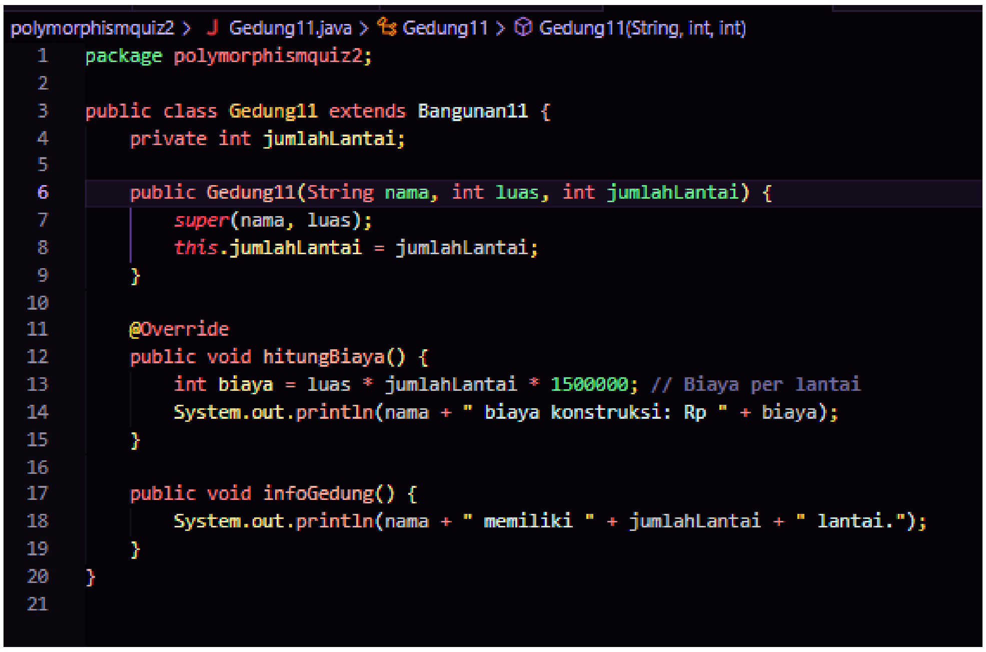
2. Rumah



polymorphismquiz2 > Rumah11.java > Rumah11 > hitungBiaya()

```java
package polymorphismquiz2;

public class Rumah11 extends Bangunan11 {
    private int jumlahKamar;

    public Rumah11(String nama, int luas, int jumlahKamar) {
        super(nama, luas);
        this.jumlahKamar = jumlahKamar;
    }

    @Override
    public void hitungBiaya() {
        int biaya = luas * 1000000; // Biaya konstruksi per m2
        System.out.println(nama + " biaya konstruksi: Rp " + biaya);
    }

    public void infoRumah() {
        System.out.println(nama + " memiliki " + jumlahKamar + " kamar.");
    }
}
```

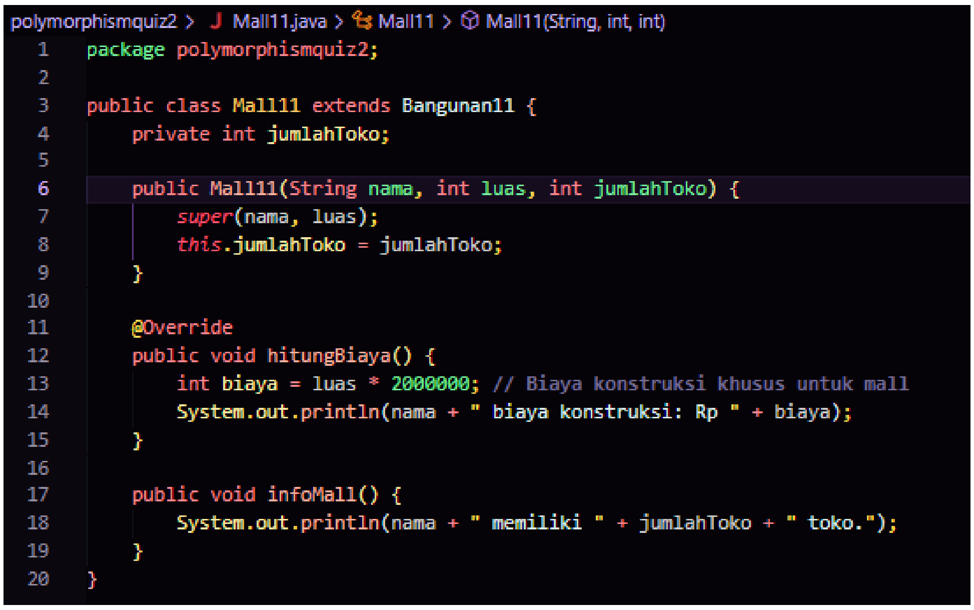
3. Gedung



polymorphismquiz2 > Gedung11.java > Gedung11 > Gedung11(String, int, int)

```java
package polymorphismquiz2;

public class Gedung11 extends Bangunan11 {
    private int jumlahLantai;

    public Gedung11(String nama, int luas, int jumlahLantai) {
        super(nama, luas);
        this.jumlahLantai = jumlahLantai;
    }

    @Override
    public void hitungBiaya() {
        int biaya = luas * jumlahLantai * 1500000; // Biaya per lantai
        System.out.println(nama + " biaya konstruksi: Rp " + biaya);
    }

    public void infoGedung() {
        System.out.println(nama + " memiliki " + jumlahLantai + " lantai.");
    }
}
```

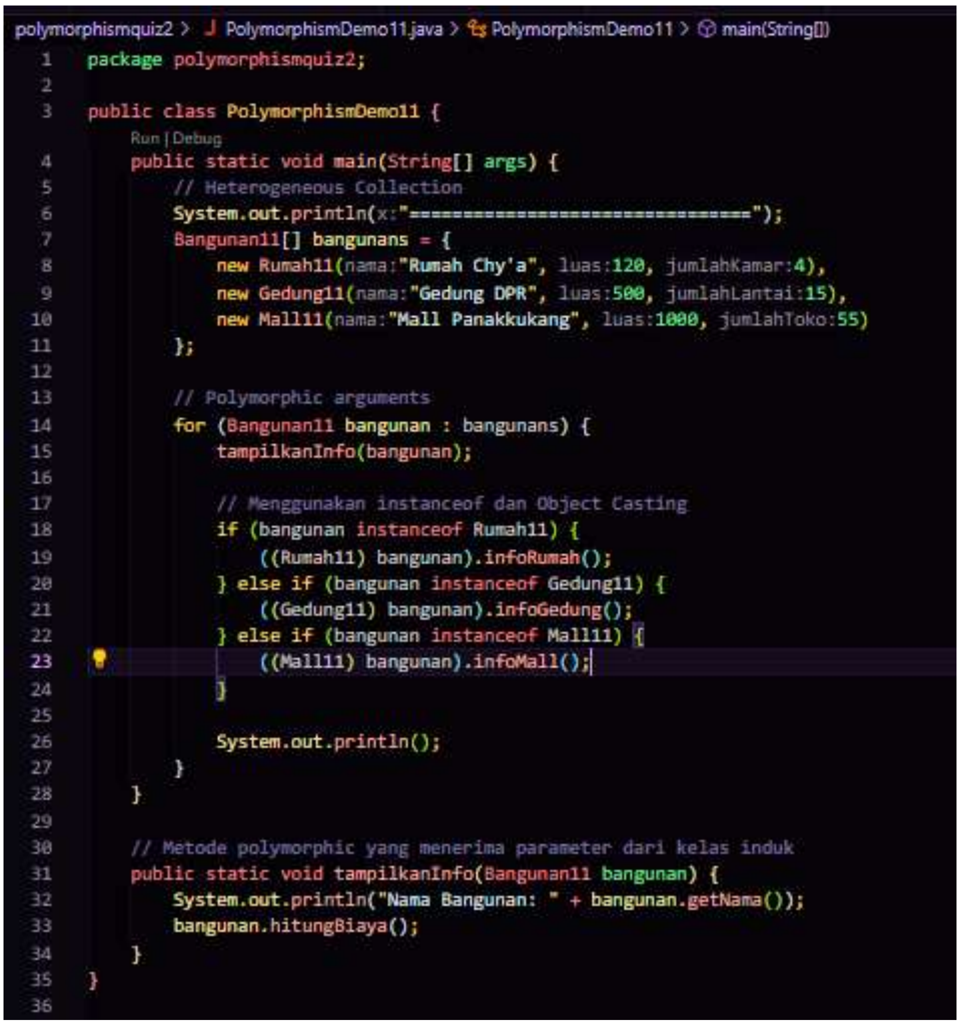
4. Mall



polymorphismquiz2 > Mall11.java > Mall11 > Mall11(String, int, int)

```java
package polymorphismquiz2;

public class Mall11 extends Bangunan11 {
    private int jumlahToko;

    public Mall11(String nama, int luas, int jumlahToko) {
        super(nama, luas);
        this.jumlahToko = jumlahToko;
    }

    @Override
    public void hitungBiaya() {
        int biaya = luas * 2000000; // Biaya konstruksi khusus untuk mall
        System.out.println(nama + " biaya konstruksi: Rp " + biaya);
    }

    public void infoMall() {
        System.out.println(nama + " memiliki " + jumlahToko + " toko.");
    }
}
```

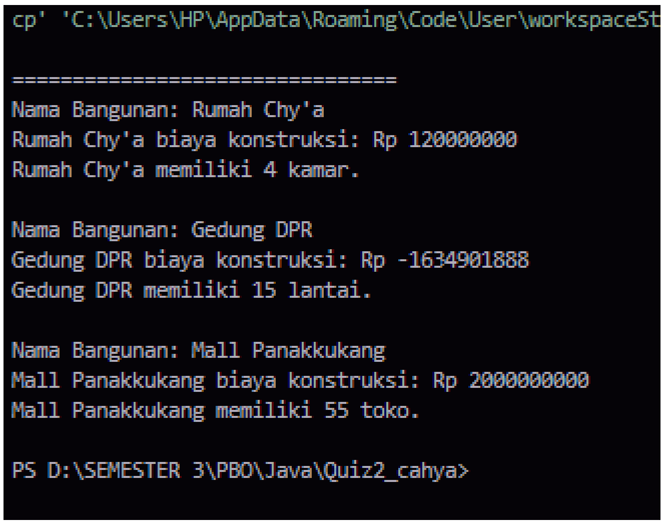
5. PolymorphismDemo



polymorphismquiz2 > PolymorphismDemo11.java > PolymorphismDemo11 > main(String[])

```java
package polymorphismquiz2;

public class PolymorphismDemo11 {
    Run | Debug
    public static void main(String[] args) {
        // Heterogeneous Collection
        System.out.println(x:"===============================");
        Bangunan11[] bangunans = {
            new Rumah11(nama:"Rumah Chy'a", luas:120, jumlahKamar:4),
            new Gedung11(nama:"Gedung DPR", luas:500, jumlahLantai:15),
            new Mall11(nama:"Mall Panakkukang", luas:1000, jumlahToko:55)
        };

        // Polymorphic arguments
        for (Bangunan11 bangunan : bangunans) {
            tampilkanInfo(bangunan);

            // Menggunakan instanceof dan Object Casting
            if (bangunan instanceof Rumah11) {
                ((Rumah11) bangunan).infoRumah();
            } else if (bangunan instanceof Gedung11) {
                ((Gedung11) bangunan).infoGedung();
            } else if (bangunan instanceof Mall11) {
                ((Mall11) bangunan).infoMall();
            }

            System.out.println();
        }
    }

    // Metode polymorphic yang menerima parameter dari kelas induk
    public static void tampilkanInfo(Bangunan11 bangunan) {
        System.out.println("Nama Bangunan: " + bangunan.getNama());
        bangunan.hitungBiaya();
    }
}
```

6. Output



```
cp' 'C:\Users\HP\AppData\Roaming\Code\User\workspaceSt

=================================
Nama Bangunan: Rumah Chy'a
Rumah Chy'a biaya konstruksi: Rp 120000000
Rumah Chy'a memiliki 4 kamar.

Nama Bangunan: Gedung DPR
Gedung DPR biaya konstruksi: Rp -1634901888
Gedung DPR memiliki 15 lantai.

Nama Bangunan: Mall Panakkukang
Mall Panakkukang biaya konstruksi: Rp 2000000000
Mall Panakkukang memiliki 55 toko.

PS D:\SEMESTER 3\PBO\Java\Quiz2_cahya>
```

V. Code Explanation

- Heterogeneous Collection
  - Bangunan[] Bangunans stores objects from the subclasses: Houses, Buildings, and Malls.
- Object Casting
  - Object type checking is done with instanceof, then the object can be converted (cast) to the appropriate type, such as House, Building, or Mall.
- Polymorphic Arguments
  - The method tampilkanInfo accepts a parameter from the parent type Building.
- InstanceOf
  - Used to check the object type, ensuring that the object being processed has the appropriate type before being converted to a specific type.

Good Luck ----