Tugas Teori 3

Pemrograman Berorientasi Objek

Disusun oleh:

Restu Akbar

231511088

Jurusan Teknik Komputer dan Informatika

Politeknik Negeri Bandung

Aggregation, Dependency, dan Inheritance

Kode:

```
public class main {
    public static void main(String[] args) {
        item item1 = new item("Kulkas", 100.0);
        order order1 = new order("order1", item1.getHarga(), item1.getNamaBarang());
        order1.displayItem();
        System.out.println();
        order1.takeOrder();
        System.out.println();
        rushOrder rushOrder = new rushOrder("rush1", 150.0, "TV", 20.0);
        System.out.println("Rush Order:");
        rushOrder.displayOrderDetails();
    }
}

class order {
    private String orderID;
    private double harga;
    private String namaBarang;
    public order(String orderID, double harga, String namaBarang) {
        this.orderID = orderID;
        this.namaBarang = namaBarang;
        this.harga = harga;
    }
    public void displayItem() {
        System.out.println("Item(s) bought: ");
        System.out.println("Item name: " + namaBarang);
        System.out.println("Amount: $" + harga);
    }
    account acc1 = new account("123", "andi", "ciwaruga");
    public void takeOrder () {
        System.out.println("Order ID: " + orderID);
        System.out.println("Order taken by:");
        System.out.println("ID: " + acc1.getID());
        System.out.println("Name: " + acc1.getNama());
        System.out.println("Address: " + acc1.getAlamat());
    }

}

class account {
    private String ID;
    private String nama;
    private String alamat;

    public account(String ID, String nama, String alamat) {
```

```
        this.ID = ID;
        this.nama = nama;
        this.alamat = alamat;
    }

    public String getID() {
        return ID;
    }
    public String getNama() {
        return nama;
    }
    public String getAlamat() {
        return alamat;
    }
}

class item {
    private String namaBarang;
    private double harga;
    public item (String namaBarang, double harga) {
        this.namaBarang = namaBarang;
        this.harga = harga;
    }
    public String getNamaBarang() {
        return namaBarang;
    }
    public double getHarga() {
        return harga;
    }
}

class rushOrder extends order {
    private double rushFee;
    private double total;
    public rushOrder(String orderID, double harga, String namaBarang, double rushFee) {
        super(orderID, harga, namaBarang);
        this.rushFee = rushFee;
        this.total = rushFee + harga;
    }
    public void displayOrderDetails() {
        super.displayItem();
        System.out.println("Rush Fee: $" + rushFee);
        System.out.println("Total Amount: $" + total);
    }
}
```

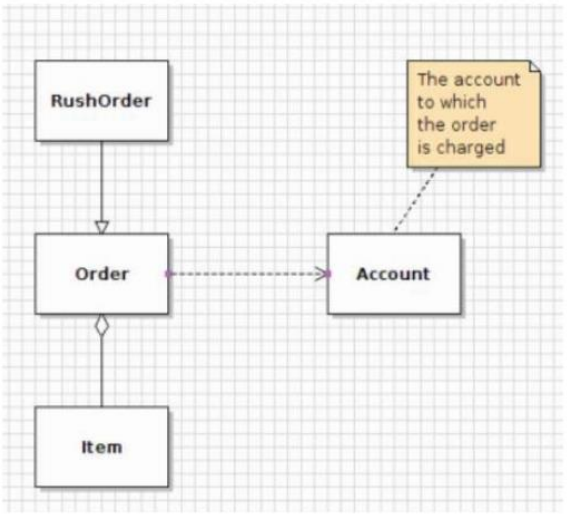
Output:

```
Item(s) bought:
Item name: Kulkas
Amount: $100.0

Order ID: order1
Order taken by:
ID: 123
Name: andi
Address: ciwaruga

Rush Order:
Item(s) bought:
Item name: TV
Amount: $150.0
Rush Fee: $20.0
Total Amount: $170.0
```

Diagram hubungan antar kelas



Relationship	UML Connector
Inheritance	——▷
Interface implementation	- - - - - -▷
Dependency	- - - - - ->
Aggregation	◇——
Association	——
Directed association	——>

1. Dependency

Dalam OOP (Object-Oriented Programming) di Java, **dependensi** mengacu pada hubungan antara dua kelas, di mana satu kelas bergantung pada kelas lain untuk menjalankan state atau behaviournya. Ini terjadi ketika satu objek menggunakan (atau bergantung pada) objek lain untuk menjalankan program tertentu. Dependensi ini biasanya muncul melalui parameter metode, pembuatan objek baru, atau penggunaan metode dari objek lain.

pada contoh kode, kelas order bergantung pada kelas account

main:

```
order order1 = new order("order1", item1.getHarga(), item1.getNamaBarang());
order1.displayItem();
System.out.println();
order1.takeOrder();
```

Kelas-kelas:

```
class order {
    private String orderID;
    private double harga;
    private String namaBarang;
    public order(String orderID, double harga, String namaBarang) {
        this.orderID = orderID;
        this.namaBarang = namaBarang;
        this.harga = harga;
    }
    public void displayItem() {
        System.out.println("Item(s) bought: ");
        System.out.println("Item name: " + namaBarang);
        System.out.println("Amount: $" + harga);
    }
    account acc1 = new account("123", "andi", "ciwaruga");
    public void takeOrder () {
        System.out.println("Order ID: " + orderID);
        System.out.println("Order taken by:");
        System.out.println("ID: " + acc1.getID());
        System.out.println("Name: " + acc1.getNama());
        System.out.println("Address: " + acc1.getAlamat());
    }

}

class account {
    private String ID;
    private String nama;
    private String alamat;

    public account(String ID, String nama, String alamat) {
        this.ID = ID;
        this.nama = nama;
        this.alamat = alamat;
    }

    public String getID() {
        return ID;
    }
    public String getNama() {
        return nama;
    }
    public String getAlamat() {
        return alamat;
    }
}
```

Kelas order memiliki method takeOrder yang membutuhkan atribut-atribut yang dimiliki kelas account untuk ditampilkan. Ciri dari dependensi adalah saat di mana satu kelas bergantung pada kelas lain, jika kelas account dihilangkan atau dihapus, maka method takeOrder di kelas order tidak akan bisa melakukan tugasnya karena tidak memiliki atribut yang diperlukan dari kelas account

2. Aggregation

Aggregation dalam OOP (Object-Oriented Programming) adalah jenis hubungan antara dua kelas yang menggambarkan konsep "has-a" atau kepemilikan, di mana satu objek mengandung objek lain sebagai bagian dari dirinya, tetapi objek yang terkandung ini tetap dapat exist keberadaannya secara independen dari objek induknya.

Pada contoh kode digambarkan dengan hubungan antara order dan item. Item terkandung dalam order karena ketika kita order barang pasti butuh data tentang item apa yang di order. Tetapi item ini tetap dapat ada atau berdiri jikalau tidak diorder sekalipun (independent terhadap kelas induk).

Main:

```
item item1 = new item("Kulkas", 100.0);
order order1 = new order("order1", item1.getHarga(), item1.getNamaBarang());
order1.displayItem();
System.out.println();
order1.takeOrder();
```

Kelas-kelas:

```
class order {
    private String orderID;
    private double harga;
    private String namaBarang;
    public order(String orderID, double harga, String namaBarang) {
        this.orderID = orderID;
        this.namaBarang = namaBarang;
        this.harga = harga;
    }
    public void displayItem() {
        System.out.println("Item(s) bought: ");
        System.out.println("Item name: " + namaBarang);
        System.out.println("Amount: $" + harga);
    }
    account acc1 = new account("123", "andi", "ciwaruga");
    public void takeOrder () {
        System.out.println("Order ID: " + orderID);
        System.out.println("Order taken by:");
        System.out.println("ID: " + acc1.getID());
        System.out.println("Name: " + acc1.getNama());
        System.out.println("Address: " + acc1.getAlamat());
    }
}
```

```
class item {
    private String namaBarang;
    private double harga;
    public item (String namaBarang, double harga) {
        this.namaBarang = namaBarang;
        this.harga = harga;
    }
    public String getNamaBarang() {
        return namaBarang;
    }
    public double getHarga() {
        return harga;
    }
}
```

Kelas order memiliki aggregation dengan kelas item di mana kelas order membutuhkan data dari kelas item untuk ditampilkan melalui parameter-parameter constructornya kelas order. Argument yang dipassing merupakan data-data yang dimiliki oleh kelas item seperti harga barang dan nama barang yang diorder.

3. Inheritance

Inheritance adalah konsep di mana satu kelas (subclass/child) mewarisi properti dan metode dari kelas lain (superclass/parent). Dalam contoh ini, kelas rushOrder mewarisi semua atribut dan metode dari kelas order, tetapi bisa menambahkan atau memodifikasi beberapa fitur yang lebih spesifik.

Main:

```
rushOrder rushOrder = new rushOrder("rush1", 150.0, "TV", 20.0);
System.out.println("Rush Order:");
rushOrder.displayOrderDetails();
```

Kelas-kelas:

```
class order {
    private String orderID;
    private double harga;
    private String namaBarang;
    public order(String orderID, double harga, String namaBarang) {
        this.orderID = orderID;
        this.namaBarang = namaBarang;
        this.harga = harga;
    }
    public void displayItem() {
        System.out.println("Item(s) bought: ");
        System.out.println("Item name: " + namaBarang);
        System.out.println("Amount: $" + harga);
    }
    account acc1 = new account("123", "andi", "ciwaruga");
    public void takeOrder () {
```

```
        System.out.println("Order ID: " + orderID);
        System.out.println("Order taken by:");
        System.out.println("ID: " + acc1.getID());
        System.out.println("Name: " + acc1.getNama());
        System.out.println("Address: " + acc1.getAlamat());
    }
}
class rushOrder extends order {
    private double rushFee;
    private double total;
    public rushOrder(String orderID, double harga, String namaBarang, double rushFee) {
        super(orderID, harga, namaBarang);
        this.rushFee = rushFee;
        this.total = rushFee + harga;
    }
    public void displayOrderDetails() {
        super.displayItem();
        System.out.println("Rush Fee: $" + rushFee);
        System.out.println("Total Amount: $" + total);
    }
}
```

Kelas rushOrder mewarisi atribut-atribut dan method dari kelas order yang didapat dengan extends dan metode super(). Kelas rushOrder akan memiliki atribut yang sama persis dari kelas induknya seperti konstruksi orderID, harga dan nama barang. Kelas rushOrder juga bisa mewarisi metode induknya seperti dalam hal ini adalah displayItem().