

# Pemrograman Lanjut

## Lembar Kerja Praktikum Pemrograman Berorientasi Objek dan Struktur Data dengan Bahasa Pemrograman Java

Diadaptasi untuk Kurikulum 2024

### Penyusun

Raden Ajeng Reina Shafira Gayatri

NIM 245150407141004



Program Studi Sistem Informasi

Departemen Sistem Informasi

Fakultas Ilmu Komputer

Universitas Brawijaya

2025

# Contents

Modul 5: Class Abstract dan Interface .....	3
Percobaan.....	3
Class Abstract .....	3
Interface.....	13
Instruksi Pengisian Lembar Kerja.....	24

# Modul 5: Class Abstract dan Interface

## Percobaan

### Class Abstract

#### *Data dan Analisis Hasil Percobaan*

1. Jalankan program melalui file Shopipedia.java. Catat apa yang dikeluarkan dari program!

```
Top up: 10000.0
ShopiPay balance: 10500.0
Pay 2000.0 using ShopiPay
ShopiPay balance: 8500.0
Pay 3000.0 using ShopiPay
ShopiPay balance: 5500.0
Top up: 1000.0
ShopiPay balance: 6550.0
Not enough balance.
ShopiPay balance: 6550.0
-----
Top up: 10000.0
Opo balance: 10000.0
Pay 2000.0 using Opo
Opo balance: 8200.0
Pay 3000.0 using Opo
Opo balance: 5500.0
Top up: 1000.0
Opo balance: 6500.0
Pay 7000.0 using Opo
Opo balance: 200.0
```

2. Perhatikan kode program pada Shopipedia.java. Terdapat dua orang Customer yang melakukan transaksi dengan aktivitas/cara yang sama dan dengan jumlah nominal yang sama. Namun, ketika bertransaksi, Miki menggunakan e-Money berjenis ShopiPay dan Dono menggunakan e-Money berjenis Opo. Sehingga, ketika digunakan bertransaksi dengan cara dan nominal yang sama, luaran yang diperlihatkan oleh program berbeda.

- a. Cek kode program Opo.java dan ShopiPay.java. Bagian mana dari kedua kode program yang menyebabkan perbedaan tersebut?

**Jawaban:**

Perbedaan terjadi pada method **topUp()** dan **pay()** di class Opo dan ShopiPay, dikarenakan keduanya memiliki logika transaksi yang berbeda. :

**ShopiPay.java**

- topUp: Mendapatkan bonus 5% dari jumlah top-up

```
this.balance += (amount + (0.05 * amount));
```

- pay: Tidak ada diskon, bayar sesuai nominal

```
this.balance -= amount;
```

**Opo.java**

- topUp: Tidak ada bonus

```
this.balance += amount;
```

- pay: Mendapat diskon 10%, bayar menjadi lebih murah

```
double amountToPay = amount - (0.1 * amount);
```

**b.** E-money mana yang lebih menguntungkan ketika digunakan bertransaksi? Mengapa?

**Jawaban:**

Lebih menguntungkan ShopiPay dikarenakan ShopiPay memberikan bonus 5% setiap kali top-up sehingga saldo yang didapat lebih besar.

Contoh perbandingan:

Top up Rp10.000:

- ShopiPay jadi Rp10.500 (karena bonus 5%).
- Opo tetap Rp10.000.

Saat keduanya melakukan pembayaran sebanyak Rp2.000:

- ShopiPay dipotong Rp2.000, sisa Rp8.500.
- Opo hanya dipotong Rp1.800 (diskon 10%), sisa Rp8.200.

**3.** Sebuah produk e-Money baru bernama Kris diluncurkan. Bonus 5% diberikan ketika melakukan deposit dan diskon belanja 3% diberikan ketika pembayaran.

**a.** Implementasikan kode programnya!

Membuat class baru bernama Kris.java:

```
public class Kris extends Emoney {
    public Kris() {
        this.name = "Kris";
    }

    @Override
    public void topUp(double amount) {
        System.out.println("Top up: " + amount);
        this.balance += amount + (0.05 * amount); // Bonus 5%
        this.balance();
    }

    @Override
```

```

    public void pay(double amount) {
        double amountToPay = amount - (0.03 * amount); // Diskon
3%
        if (this.balance > amountToPay) {
            this.balance -= amountToPay;
            System.out.printf("Pay " + amount);
            System.out.println(" using " + this.name);
        } else {
            System.out.println("Not enough balance.");
        }
        this.balance();
    }
}

```

Menambahkan baris berikut di dalam Class Shopipedia.java:

```

System.out.println("-----");
Kris krisCard = new Kris();
Customer rina = new Customer(krisCard);
rina.deposit(10000);
rina.pay(2000);
rina.pay(3000);
rina.deposit(1000);
rina.pay(7000);

```

b. Apakah E-money Kris ini lebih menguntungkan dari E-money ShopiPay dan Opo? Buktikan dalam program!

```
public class Shopipedia {  
    public static void main(String[] args) {  
        // === ShopiPay bonus top-up 5% ===  
        ShopiPay shopiCard = new ShopiPay();  
        Customer miki = new Customer(shopiCard);  
        miki.deposit(10000);    // Bonus 5% → saldo: 10500  
        miki.pay(2000);        // Tanpa diskon → saldo: 8500  
        miki.pay(3000);        // Tanpa diskon → saldo: 5500  
        miki.deposit(1000);    // Bonus 5% → +1050 → saldo: 6550  
        miki.pay(7000);        // Gagal (saldo < 7000) → saldo tetap:  
6550  
  
        System.out.println("-----");  
  
        // === Opo bonus Diskon pembayaran 10%===  
        Opo opoCard = new Opo();  
        Customer dono = new Customer(opoCard);  
        dono.deposit(10000);    // Tanpa bonus → saldo: 10000  
        dono.pay(2000);        // Diskon 10% → bayar 1800 → saldo: 8200  
        dono.pay(3000);        // Diskon 10% → bayar 2700 → saldo: 5500  
        dono.deposit(1000);    // Tanpa bonus → saldo: 6500  
        dono.pay(7000);        // Diskon 10% → bayar 6300 → saldo: 200  
  
        System.out.println("-----");  
  
        // === Kris Dapat bonus top-up 5% & Diskon pembayaran 3% ===  
        Kris krisCard = new Kris();  
        Customer rina = new Customer(krisCard);  
        rina.deposit(10000);    // Bonus 5% → saldo: 10500  
        rina.pay(2000);        // Diskon 3% → bayar 1940 → saldo: 8560  
        rina.pay(3000);        // Diskon 3% → bayar 2910 → saldo: 5650  
        rina.deposit(1000);    // Bonus 5% → +1050 → saldo: 6700
```

```
        rina.pay(7000);          // Gagal (diskon 3% → 6790 > 6700) → saldo
tetap: 6700
    }
}
```

Kris lebih menguntungkan karena:

- Dapat bonus top-up 5% (seperti ShopiPay)
- Dapat diskon 3% saat pembayaran
- Saldo akhirnya paling besar (6700) dibanding ShopiPay (6550) dan Opo (200).

**4.** Sebagaimana aturan deposit dan belanja pada eksperimen nomor 3, implementasikan objek E money Kris dalam program tanpa membuat class baru yang menjadi subclass dari class E money.

```
public class Shopipedia {
    public static void main(String[] args) {
        ShopiPay shopiCard = new ShopiPay();
        Customer miki = new Customer(shopiCard);
        miki.deposit(10000);
        miki.pay(2000);
        miki.pay(3000);
        miki.deposit(1000);
        miki.pay(7000);

        System.out.println("-----");
        Opo opoCard = new Opo();
        Customer dono = new Customer(opoCard);
        dono.deposit(10000);
        dono.pay(2000);
        dono.pay(3000);
        dono.deposit(1000);
        dono.pay(7000);
    }
}
```



```

System.out.println("-----");
// Anonymous class for Kris
Emoney krisCard = new Emoney() {
    {
        this.name = "Kris";
    }

    @Override
    public void topUp(double amount) {
        System.out.println("Top up: " + amount);
        this.balance += amount + (0.05 * amount); // Bonus 5%
        this.balance();
    }

    @Override
    public void pay(double amount) {
        double amountToPay = amount - (0.03 * amount); // Discount

        if (this.balance > amountToPay) {
            this.balance -= amountToPay;
            System.out.printf("Pay " + amount);
            System.out.println(" using " + this.name);
        } else {
            System.out.println("Not enough balance.");
        }
        this.balance();
    }
};

Customer rina = new Customer(krisCard);
rina.deposit(10000);
rina.pay(2000);
rina.pay(3000);

```

3%

```
        rina.deposit(1000);  
        rina.pay(7000);  
    }  
}
```

**5.** Buat sebuah studi kasus/skenario lain di mana class abstract tepat untuk diterapkan dalam menyelesaikan masalah studi kasus/skenario tersebut dan jelaskan pula mengapa demikian!

**Studi Kasus:** Sistem Transportasi Umum

Aplikasi mengelola Bus, Kereta, dan Pesawat, yang memiliki:

- Nama transportasi
- Metode tampilkanInfo()
- Metode hitungBiaya(jarak), dengan logika berbeda tiap jenis

**Solusi:**

Menggunakan abstract class Transportasi sebagai dasar, lalu subclass seperti Bus, Kereta, dan Pesawat mengimplementasikan logika masing-masing.

Kenapa pakai abstract class?

- Ada fitur umum (nama & info)
- Tiap jenis wajib punya hitungBiaya(), meski logikanya beda
- Menjaga konsistensi antar jenis transportasi
- Mudah dikembangkan (misal menambahkan transportasi **Kapal** tanpa mengubah struktur lama)

Transportasi.java

```
public abstract class Transportasi {  
    protected String nama;  
  
    public Transportasi(String nama) {  
        this.nama = nama;  
    }  
  
    public abstract double hitungBiaya(double jarak);  
}
```

```
    public void tampilkanInfo() {  
        System.out.println("Jenis transportasi: " + this.nama);  
    }  
}
```

## Bus.java

```
public class Bus extends Transportasi {  
    public Bus() {  
        super("Bus");  
    }  
    @Override  
    public double hitungBiaya(double jarak) {  
        return jarak * 1000; // Rp 1000 per km  
    }  
}
```

## Kereta.java

```
public class Kereta extends Transportasi {  
    public Kereta() {  
        super("Kereta");  
    }  
    @Override  
    public double hitungBiaya(double jarak) {  
        return 5000 + (jarak * 800); // Rp 800 per km + biaya dasar  
    }  
}
```

## Pesawat.java

```
public class Pesawat extends Transportasi {  
    public Pesawat() {  
        super("Pesawat");  
    }  
    @Override  
    public double hitungBiaya(double jarak) {  
        return jarak * 3000; // Rp 3000 per km  
    }  
}  
rina.pay(7000);
```

## AplikasiTransportasi.java

```
public class AplikasiTransportasi {  
    public static void main(String[] args) {  
        Transportasi bus = new Bus();  
        Transportasi kereta = new Kereta();  
        Transportasi pesawat = new Pesawat();  
  
        double jarak = 50;  
  
        bus.tampilkanInfo();  
        System.out.println("Biaya: Rp " + bus.hitungBiaya(jarak));  
        System.out.println();  
  
        kereta.tampilkanInfo();  
        System.out.println("Biaya: Rp " + kereta.hitungBiaya(jarak));  
        System.out.println();  
  
        pesawat.tampilkanInfo();  
        System.out.println("Biaya: Rp " + pesawat.hitungBiaya(jarak));  
    }  
}
```

## Interface

Gunakan interface **Impostor.java**, **ICrew.java**, class **Impostor.java**, **Crew.java**, dan **AmongUsGame.java** untuk menyelesaikan percobaan ini. Program dijalankan melalui file **AmongUsGame.java**.

### Data dan Hasil Analisis

1. Jalankan program melalui file **AmongUsGame.java**. Catat apa yang dikeluarkan dari program!

```
Crew Brian is doing work.
Crew Cindy is doing work.
Crew David is doing work.
Impostor Jacky is doing work.
Cindy has been killed!
David found a corpse and calls a meeting. Let's find the impostor!
Brian is not the impostor.
David is not the impostor.
Jacky is the impostor!
```

2. Dari kode program yang disediakan, terdapat class Crew yang menjadi class bagi objek-objek Crew yang dapat melakukan perilaku sebagaimana dideklarasikan dalam kontrak interface ICrew. Walau demikian, dapatkah class Crew meng-override method dari interface Impostor, sehingga objek dari class Crew dapat melakukan hal-hal sebagaimana kontrak dalam interface Impostor? Mengapa demikian?

#### Jawaban:

Class **Crew** tidak bisa meng-override method dari interface **Impostor** karena tidak menggunakan interface tersebut. Sebuah class hanya bisa menimpa method dari interface yang memang ia pakai. Karena **Crew** hanya menggunakan **ICrew**, maka hanya method dari **ICrew** yang bisa di-override. Method **kill()** milik **Impostor**, jadi tidak bisa di-override oleh **Crew**.

3. Jika Jacky adalah seorang Impostor, karena Jacky adalah objek dari class Impostor, mengapa Jacky dapat melakukan pekerjaan doWork() sebagaimana layaknya Crew?

**Jawaban:**

Jacky bisa melakukan pekerjaan doWork() seperti Crew karena class Impostor mengimplementasikan interface ICrew. Karena ICrew memiliki method doWork(), dan Impostor mengimplementasikan ICrew, maka Impostor juga memiliki method doWork().

terlihat di baris:

```
public class Impostor implements IImpostor, ICrew {
```

dan

```
@Override  
public void doWork() {  
    System.out.println("Impostor " + this.name + " is doing work.");  
}
```

4. Di dalam program terdapat implementasi teknik polimorfisme. Dengan memperhatikan ciri-ciri dari penerapan teknik polimorfisme, di kode program baris mana teknik polimorfisme dengan menggunakan Interface diterapkan?

**Jawaban:**

Teknik polimorfisme dengan interface diterapkan pada baris:

```
public static void check(ICrew crew)
```

```
public static boolean isImpostor(ICrew crew)
```

```
AmongUsGame.check(brian);  
AmongUsGame.check(david);  
AmongUsGame.check(jacky);
```

Polimorfisme terjadi saat objek dari kelas berbeda diperlakukan sebagai objek dari tipe yang sama, yaitu interface ICrew. Kelas Crew dan Impostor sama-sama mengimplementasikan ICrew, sehingga method check bisa menerima keduanya. Meskipun jacky adalah objek Impostor, ia tetap bisa

digunakan dalam method yang parameternya ICrew karena dia mengimplementasikan interface tersebut. Ini menunjukkan polimorfisme melalui interface.

5. Jika pada baris ke-20 file AmongUsGame.java objek yang memanggil method callMeeting() adalah jacky,

**a. Dapatkah program dapat di-compile? Jika tidak, apa alasannya?**

Tidak bisa di-compile karena method callMeeting() tidak ada di class Impostor.

**b. Apakah pemanggilan method tersebut termasuk ke dalam penerapan teknik polimorfisme? Mengapa demikian?**

Bukan polymorphism, karena method tersebut tidak dideklarasikan di interface yang digunakan sebagai referensi objek.

6. Dari contoh kasus game “Among Us” yang dipraktikkan, ketika diperiksa, karakter Jacky ditemukan dan diketahui sebagai impostor. Melalui *statement* apa dan di baris kode program manakah yang membuktikan bahwa Jacky adalah seorang impostor? Mengapa demikian?

**Jawaban:**

Jacky terbukti sebagai impostor melalui statement berikut:

```
Crew brian, cindy, david;  
Impostor jacky;  
brian = new Crew("Brian");  
cindy = new Crew("Cindy");  
david = new Crew("David");  
jacky = new Impostor("Jacky");
```

```
System.out.println(crew.getName() + " is the impostor!");
```

Hal ini terjadi karena Jacky dikirim ke method check(jacky), lalu di dalam method isImpostor(crew) dilakukan pengecekan:

```
return (crew instanceof Impostor);
```

Karena objek jacky merupakan instance dari class Impostor, maka instanceof Impostor akan bernilai true, sehingga Jacky dikenali sebagai impostor.

7. Jika Impostor dan Crew keduanya dapat dikeluarkan dari pesawat luar angkasa, maka dengan menggunakan teknik interface, ubah dan tambahkan kode program sedemikian rupa sehingga objek Crew dan Impostor dapat dikeluarkan dari pesawat melalui method dengan nama kick(). Oleh karena objek Crew dan objek Impostor keduanya dapat dikeluarkan dari pesawat, gunakan teknik generalisasi untuk menyederhanakan deklarasi dan definisi dari interface-nya!

Tambah Kelas IEjectable.java

```
public interface IEjectable {  
    void kick();  
}
```

Update Crew.java

```
public class Crew implements ICrew, IEjectable {  
    private String name;  
  
    public Crew(String name) {  
        this.name = name;  
    }  
  
    @Override  
    public void doWork() {  
        System.out.println("Crew " + this.name + " is doing work.");  
    }  
  
    @Override  
    public String getName() {  
        return this.name;  
    }  
  
    public void callMeeting() {  
        System.out.print(this.name + " found a corpse ");  
        System.out.print("and calls a meeting. ");  
    }  
}
```



```
        System.out.println("Let's find the impostor!");
    }

    @Override
    public void kick() {
        System.out.println("Crew " + this.name + " has been kicked from
the spaceship.");
    }
}
```

### Update Impostor.java

```
public class Impostor implements IImpostor, ICrew, IEjectable {
    private final String name;

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

    @Override
    public void kill(ICrew crew) {
        System.out.println(crew.getName() + " has been killed!");
    }

    @Override
    public void doWork() {
        System.out.println("Impostor " + this.name + " is doing work.");
    }

    @Override
    public String getName() {
        return this.name;
    }
}
```

```
    }

    @Override
    public void kick() {
        System.out.println("Impostor " + this.name + " has been kicked
from the spaceship.");
    }
}
```

### Update main class

```
public class AmongUsGame {

    public static void main(String[] args) {
        Crew brian, cindy, david;
        Impostor jacky;

        brian = new Crew("Brian");
        cindy = new Crew("Cindy");
        david = new Crew("David");
        jacky = new Impostor("Jacky");

        brian.doWork();
        cindy.doWork();
        david.doWork();
        jacky.doWork();
        jacky.kill(cindy);

        david.callMeeting();

        AmongUsGame.check(brian);
        AmongUsGame.check(david);
    }
}
```

```

        AmongUsGame.check(jacky);

        // Tambahan: Tendang keluar dari pesawat
        kickOut(brian);
        kickOut(jacky);
    }

    public static void check(ICrew crew) {
        if (AmongUsGame.isImpostor(crew))
            System.out.println(crew.getName() + " is the impostor!");
        else
            System.out.println(crew.getName() + " is not the impostor.");
    }

    public static boolean isImpostor(ICrew crew) {
        return (crew instanceof Impostor);
    }

    // Tambahan: Method kickOut
    public static void kickOut(IEjectable person) {
        person.kick();
    }
}

```

8. Ubah kode program untuk mengakomodasi logic, di mana seorang Crew yang terbunuh tidak dapat melakukan/memanggil method callMeeting(). Laporkan keluaran programnya!

Tambah atribut isAlive di Crew.java

```

public class Crew implements ICrew, IEjectable {
    private String name;
    private boolean isAlive = true; // Tambahan
}

```

```
public Crew(String name) {
    this.name = name;
}

@Override
public void doWork() {
    System.out.println("Crew " + this.name + " is doing work.");
}

@Override
public String getName() {
    return this.name;
}

public void callMeeting() {
    if (!isAlive) {
        System.out.println(this.name + " is dead and cannot call a
meeting.");
        return;
    }

    System.out.print(this.name + " found a corpse ");
    System.out.print("and calls a meeting. ");
    System.out.println("Let's find the impostor!");
}

public void die() {
    this.isAlive = false;
}

@Override
public void kick() {
    System.out.println("Crew " + this.name + " has been kicked from
the spaceship.");
}
```

```
}
```

### Update Impostor.java

```
public class Impostor implements IImpostor, ICrew, IEjectable {
    private final String name;

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

    @Override
    public void kill(ICrew crew) {
        System.out.println(crew.getName() + " has been killed!");
        if (crew instanceof Crew) {
            ((Crew) crew).die(); // Casting dan mematikan crew
        }
    }

    @Override
    public void doWork() {
        System.out.println("Impostor " + this.name + " is doing work.");
    }

    @Override
    public String getName() {
        return this.name;
    }

    @Override
    public void kick() {
```

```
        System.out.println("Impostor " + this.name + " has been kicked  
from the spaceship.");  
    }  
}
```

### AmongUsGame.java

```
public class AmongUsGame {  
  
    public static void main(String[] args) {  
        Crew brian, cindy, david;  
        Impostor jacky;  
  
        brian = new Crew("Brian");  
        cindy = new Crew("Cindy");  
        david = new Crew("David");  
        jacky = new Impostor("Jacky");  
  
        brian.doWork();  
        cindy.doWork();  
        david.doWork();  
        jacky.doWork();  
  
        jacky.kill(cindy);  
        cindy.callMeeting();  
  
        david.callMeeting();  
  
        AmongUsGame.check(brian);  
        AmongUsGame.check(david);  
        AmongUsGame.check(jacky);  
  
        kickOut(brian);  
        kickOut(jacky);  
    }  
}
```

```
}

public static void check(ICrew crew) {
    if (AmongUsGame.isImpostor(crew))
        System.out.println(crew.getName() + " is the impostor!");
    else
        System.out.println(crew.getName() + " is not the impostor.");
}

public static boolean isImpostor(ICrew crew) {
    return (crew instanceof Impostor);
}

public static void kickOut(IEjectable person) {
    person.kick();
}
}
```

## Output

```
Crew Brian is doing work.
Crew Cindy is doing work.
Crew David is doing work.
Impostor Jacky is doing work.
Cindy has been killed!
Cindy is dead and cannot call a meeting.
David found a corpse and calls a meeting. Let's find the impostor!
Brian is not the impostor.
David is not the impostor.
Jacky is the impostor!
Crew Brian has been kicked from the spaceship.
Impostor Jacky has been kicked from the spaceship.
```

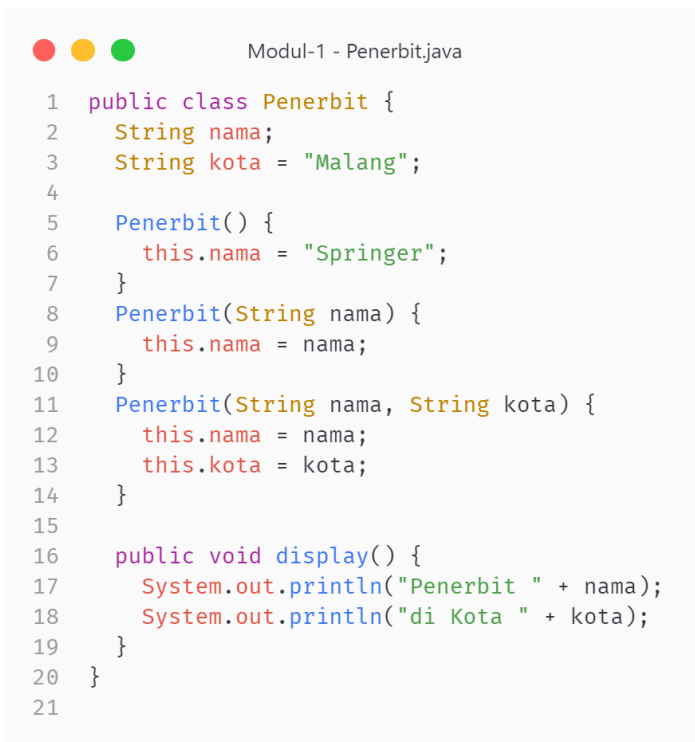
## Instruksi Pengisian Lembar Kerja

*Contoh Style Microsoft Word untuk isian data dan analisis hasil percobaan:*

Paragraf/uraian, menggunakan style Normal. Kode program/luaran program, menggunakan screenshot atau Coding Normal, seperti:

```
public class App {  
    public static void main(String[] args){  
        Mobil mobilA = new Mobil();  
    }  
}
```

Lalu dilanjutkan lagi dengan uraian paragraf menggunakan style Normal. Ukuran gambar screenshot dapat disesuaikan sedemikian rupa sehingga ukuran teks/coding di dalam screenshot sama atau lebih kecil dari ukuran font paragraf dengan style Normal. Jika dimungkinkan, ambil screenshot kode program dengan menggunakan Extension VSCode: CodeSnap agar rapi, seperti pada screenshot contoh kode program yang diperlihatkan berikut ini:



```
Modul-1 - Penerbit.java  
  
1  public class Penerbit {  
2      String nama;  
3      String kota = "Malang";  
4  
5      Penerbit() {  
6          this.nama = "Springer";  
7      }  
8      Penerbit(String nama) {  
9          this.nama = nama;  
10     }  
11     Penerbit(String nama, String kota) {  
12         this.nama = nama;  
13         this.kota = kota;  
14     }  
15  
16     public void display() {  
17         System.out.println("Penerbit " + nama);  
18         System.out.println("di Kota " + kota);  
19     }  
20 }  
21
```