Bab 7
Polimorfisme

Definisi Polimorfisme

Kata "polimorfisme" berasal dari bahasa Yunani yang berarti "banyak bentuk". Dalam konteks OOP, polimorfisme memungkinkan suatu entitas (seperti method atau function) untuk berperilaku dengan cara yang berbeda tergantung pada konteks di mana entitas(seperti method atau function) tersebut digunakan . “Bentuk” di sini dapat kita artikan: isinya berbeda (overriding), parameternya berbeda dan tipe datanya berbeda (overloading). Polymorphism digunakan untuk mengimplementasi suatu fungsi dari sebuah induk class , baik fungsi yang abstract maupun sudah terdefinisi, untuk diimplementasikan sesuai dengan relevansi (memberikan implementasi fungsi yang sesuai dengan karakteristik atau tujuan sendiri) suatu class.

Contoh Polimorfisme dalam kehidupan

Manusia dapat memakan semua jenis buah. Tetapi cara memakannya berbeda. Apel dapat dimakan, Pisang juga dapat dimakan(apel dan pisang termasuk golongan buah), hanya saja caranya berbeda. Pisang harus dikupas dulu.

Dalam hal tersebut, manusia memiliki method makanBuah() dengan parameter object buah. parameter ini dapat menerima object yang merupakan sub-class dari buah Apel, Pisang. Barulah pada method makanBuah dideteksi,apakah buah tersebut Apel atau Pisang atau yang lain. Kemudian diperlakukan dengan berbeda (Apel bisa langsung dimakan, Pisang dikupas dahulu,dls).

Jenis-jenis Polimorfisme

Polymorphisme pada java terdiri dari 2 jenis, yaitu Static Polymorphism dan Dynamic Polymorphism. Static Polymorphism adalah Polymorphism yang dilakukan pada waktu compile (compile time), sedangkan Dynamic Polymorphism adalah Polymorphism yang dilakukan pada waktu berjalannya program (run time).

1. Compile-time: Tipe compile-time atau static terjadi ketika metode dijalankan pada waktu kompilasi (compile-time). Tipe ini terjadi saat menjalankan metode overloading.
2. Run time: Tipe runtime atau dynamic merujuk pada metode dijalankan tepat saat kode yang dapat dieksekusi mulai dijalankan. Runtime berlangsung saat metode overriding.

Static Polimorfisme

Pada static polimorfisme mengunakan konsep overloading, overloading sendiri memiliki ciri ciri bahwa pada suatu class terdapat method dengan nama yang sama , lalu tipe data dan parameter berbeda, inilah yang dinamakan method overloading.

```
Public nama_class(){
     double nama_method(double d){
           //isi
     }
     double nama_method(double d, double f){
           //isi
     }
}}
```

Dynamic Polimorfisme

Polimorfisme dinamik menggunakan konsep method overrdiding dalam penerapannya dan dilakukan disaat run time(selesai menjalankan program).pada polimorfisme dinamis biasanya terjadi saat kita menggunakan pewarisan (inheritance) dan implementasi interface.

Super Class

```
 Public Super_Class(){
     Void nama_method(){
       isi
}}
```

Dynamic Polimorfisme

Sub Class 1

```
Public Sub_Class1 extend Super_Class(){
@override
    Void nama_method(){Isi2
    }
}
```

Sub Class 2

```
 Public Sub_Class2 extend
 Super_Class(){@override
     Void
       nama_method()
       {Isi3
     }
 }
```

```
public class main {
    public static void main(String[] args) {
        Super_Class induk; // instansiasi super class

        induk = new Sub_Class1();
        // memanggil method dari class Sub_Class1
        induk.nama_method();

        induk = new Sub_Class2();
        // memanggil method dari class Sub_Class2
        induk.nama_method();
    }
}
```

Latihan Polimorfisme

```
public class Buah {
    public String nama;

    public void setNamaBuah(){
        this.nama = "Apel,mangga,pisang,jambu,dll";
    }

     void makanBuah() {
        System.out.println( x:"Makan buah secara umum.");
    }
}
```

Latihan Polimorfisme

```java
public class Apel extends Buah {
    public Apel() {
        this.nama = "apel";
    }
    // implementasi makanBuah() khusus untuk Apel
    void makanBuah() {
        System.out.println(nama + "di makan dengan di kupas kulitnya atau dimakan langsung (digigit).");
    }
}
```

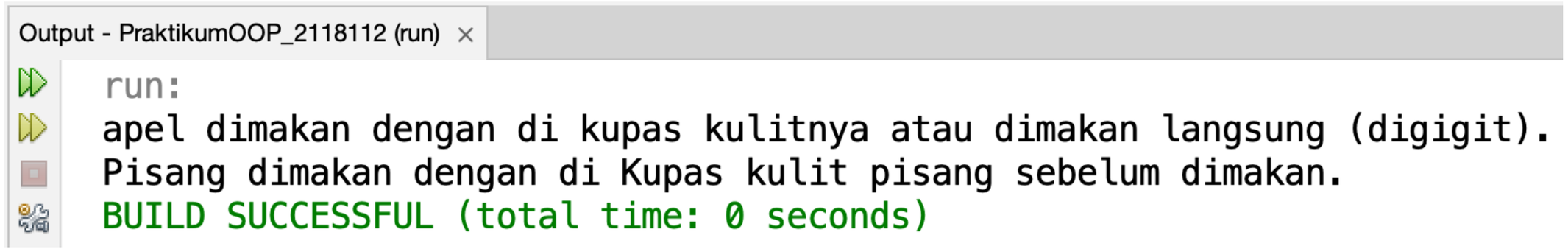
```java
public class Pisang extends Buah {
    public Pisang() {
        this.nama = "Pisang";
    }
    // implementasi makanBuah() khusus untuk Pisang
    void makanBuah() {
        System.out.println(nama + " dimakan dengan di Kupas kulit pisang sebelum dimakan.");
    }
}
```

Latihan Polimorfisme

```java
public class Manusia {
    // Metode makanBuah() yang memanfaatkan polimorfisme
    void makanBuah(Buah buah) {
        buah.makanBuah();  // Polimorfisme: memanggil metode makanBuah() sesuai dengan tipe objek
    }
}
```

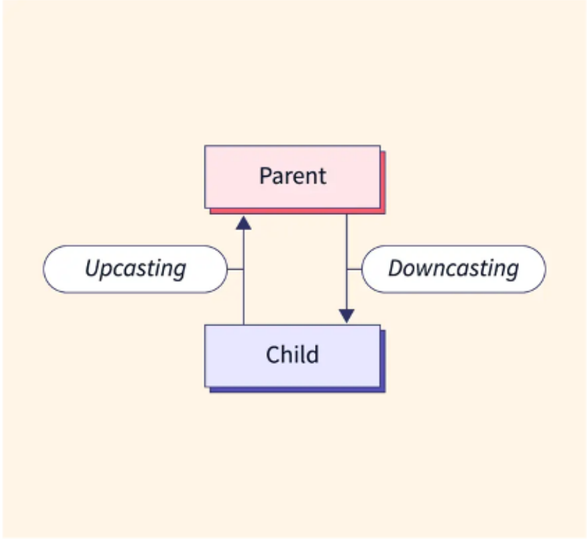
```java
public class Driveclass {
    public static void main(String[] args) {
        // Membuat objek Manusia
        Manusia manusia = new Manusia();

        // Membuat objek Apel dan Pisang
        Buah apel = new Apel();
        Buah pisang = new Pisang();

        // Memanggil metode makanBuah() pada Manusia untuk objek Apel dan Pisang
        manusia.makanBuah( buah:apel);    // Output: Makan apel dengan di gigit.
        manusia.makanBuah( buah:pisang);  // Output: Kupas pisang sebelum dimakan.
    }
}
```

Hasil Polimorfisme



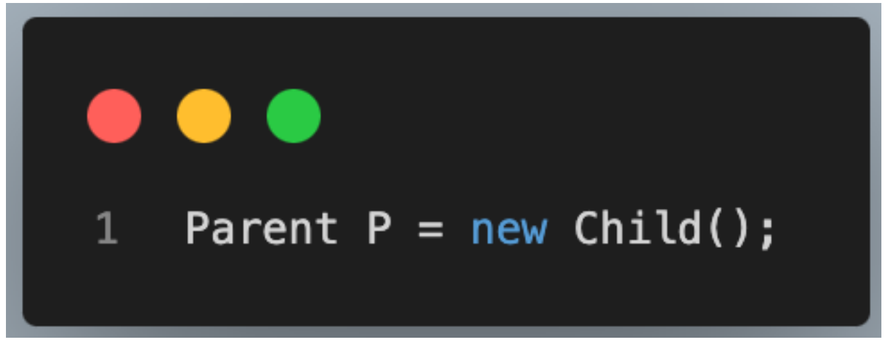
Casting

Casting adalah proses mengubah tipe data dari satu tipe ke tipe(attribut maupun method) lainnya. Casting diperlukan untuk mengubah atau menyesuaikan tipe data antara tipe data yang berbeda. Terdapat 2 jens casting yaitu Upcasting dan downcasting adalah dua konsep yang terkait dengan pewarisan (inheritance) pada pemrograman berorientasi objek yang memanfaatkan hubungan antara kelas-kelas dalam hierarki pewarisan. Kedua konsep ini akan memberikan fleksibilitas dan kemampuan untuk mengelola hubungan antara kelas-kelas dalam hierarki pewarisan.



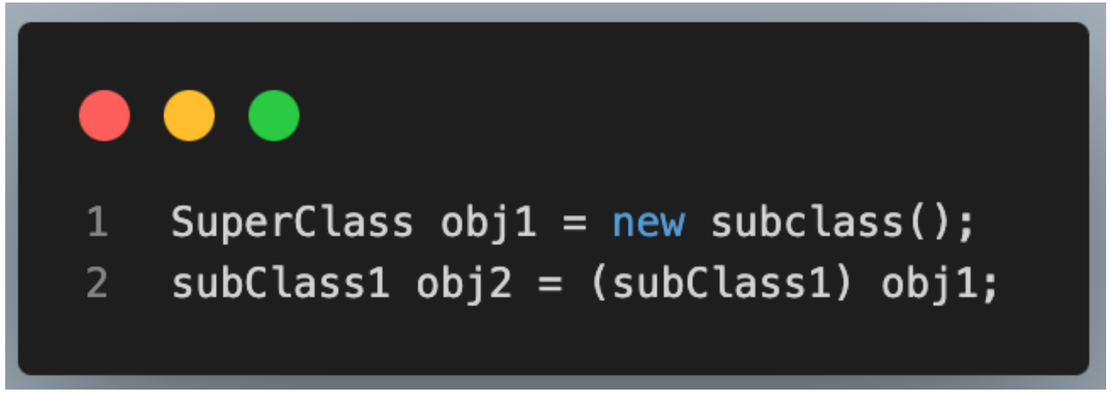
1. Upcasting

Upcasting adalah Proses mengkonversi objek, dari tipe sub class (class anak/turunan) ke tipe super class (class parent), ke arah atas pada hirarki pewarisan (inheritance). Dalam upcasting, objek dari kelas turunan dapat dianggap sebagai objek dari kelas induknya. Hal ini berarti object dari kelas anak dapat mengakses anggota kelas induk menggunakan objek kelas turunan setelah dilakukan upcasting.



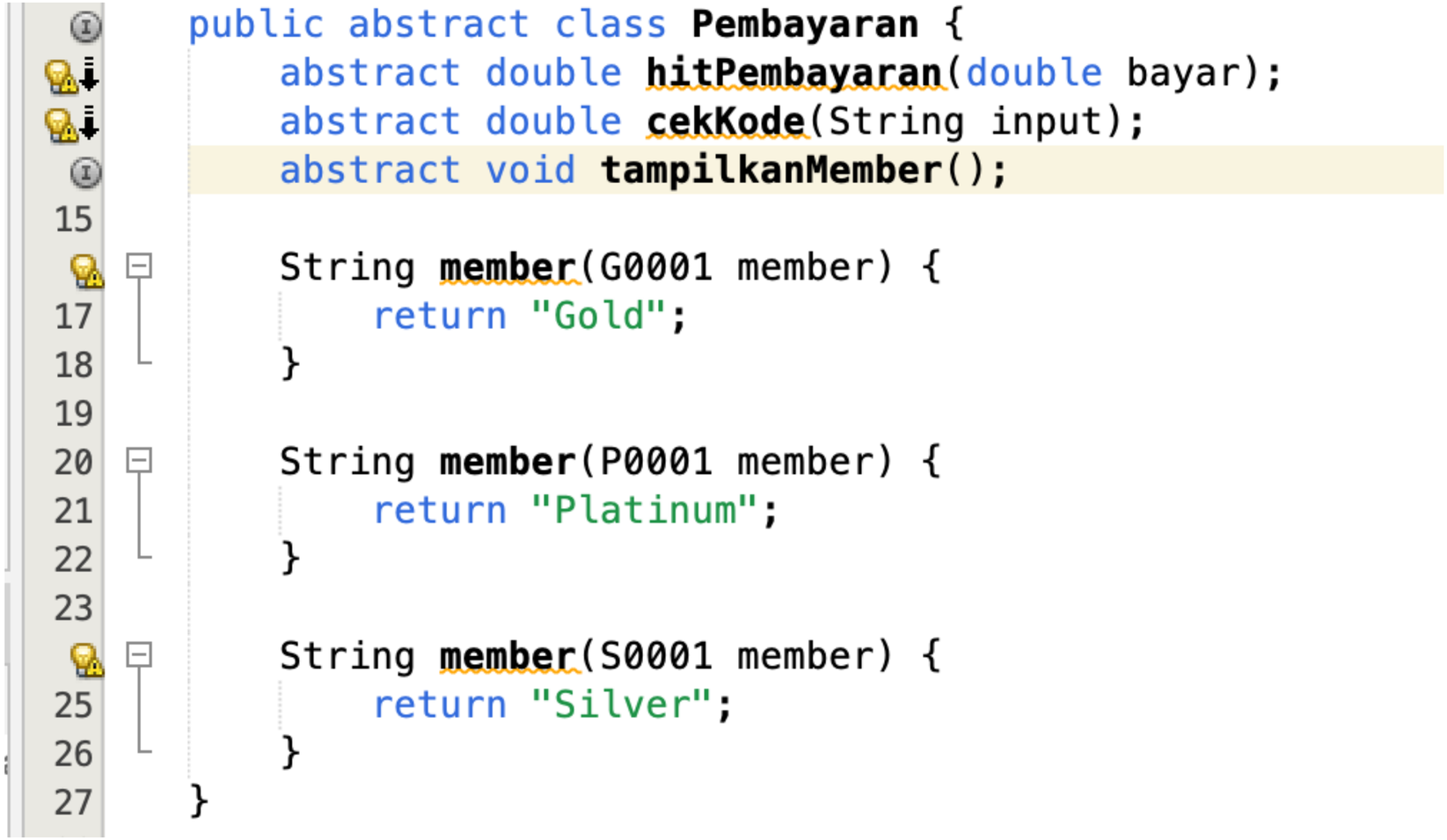
2. Downcasting

Downcasting adalah proses mengubah objek dari tipe induk ke tipe turunannya dalam hierarki pewarisan. Dalam downcasting, objek anak diubah kembali dari objek kelas induk menjadi objek kelas turunan agar dapat mengakses metode atau atribut khusus dari kelas turunan tersebut.



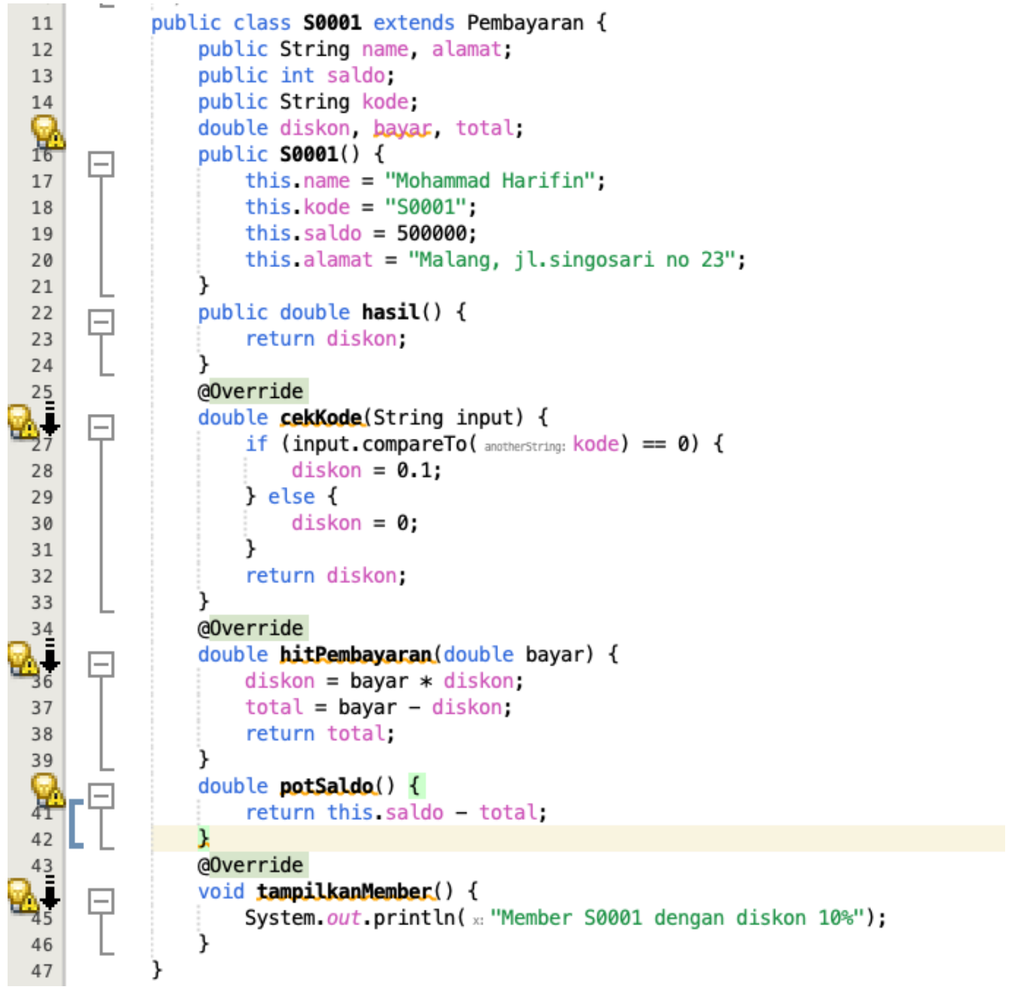
```
1   SuperClass obj1 = new subclass();
2   subClass1 obj2 = (subClass1) obj1;
```

Latihan Casting



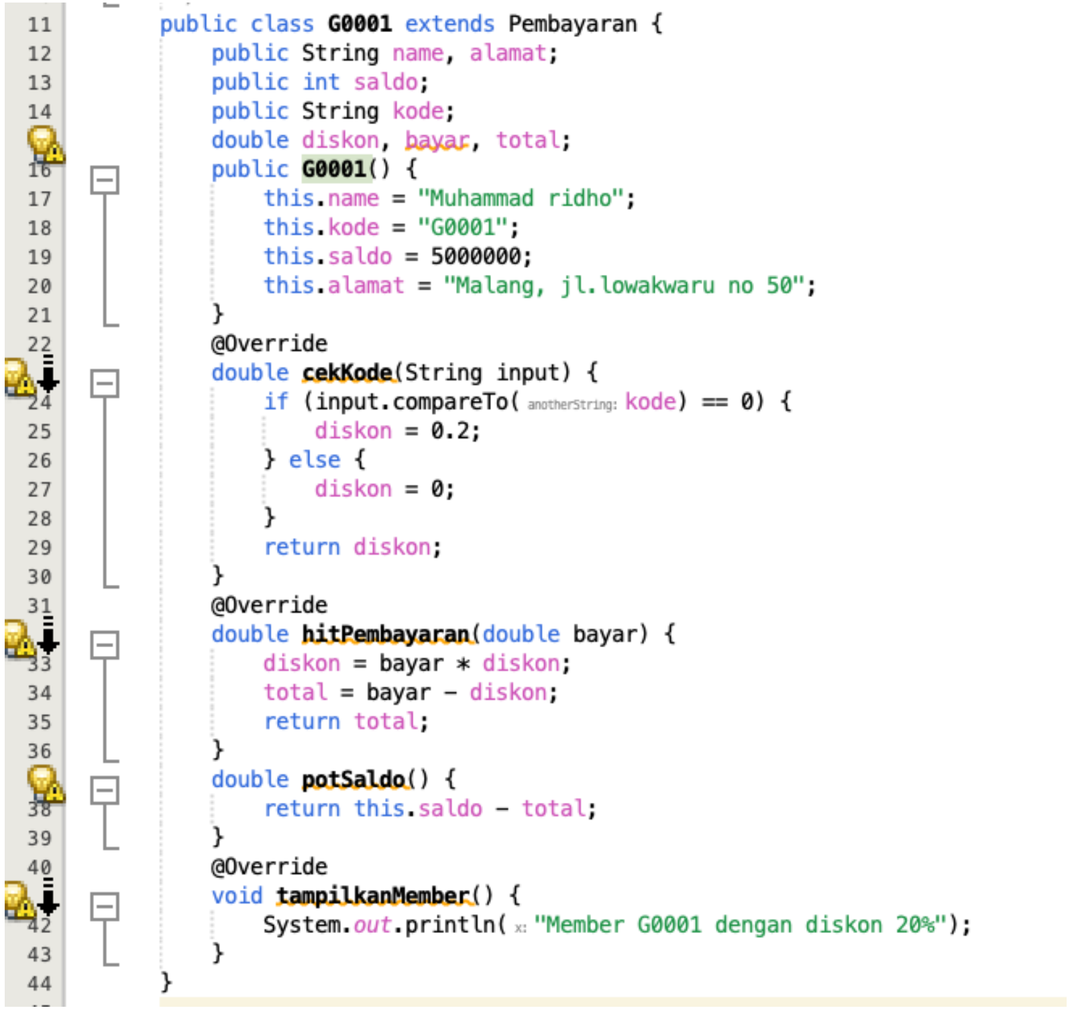
```
1  public abstract class Pembayaran {
2      abstract double hitPembayaran(double bayar);
3      abstract double cekKode(String input);
4      abstract void tampilkanMember();
15
16     String member(G0001 member) {
17         return "Gold";
18     }
19
20     String member(P0001 member) {
21         return "Platinum";
22     }
23
24     String member(S0001 member) {
25         return "Silver";
26     }
27 }
```

Latihan Casting



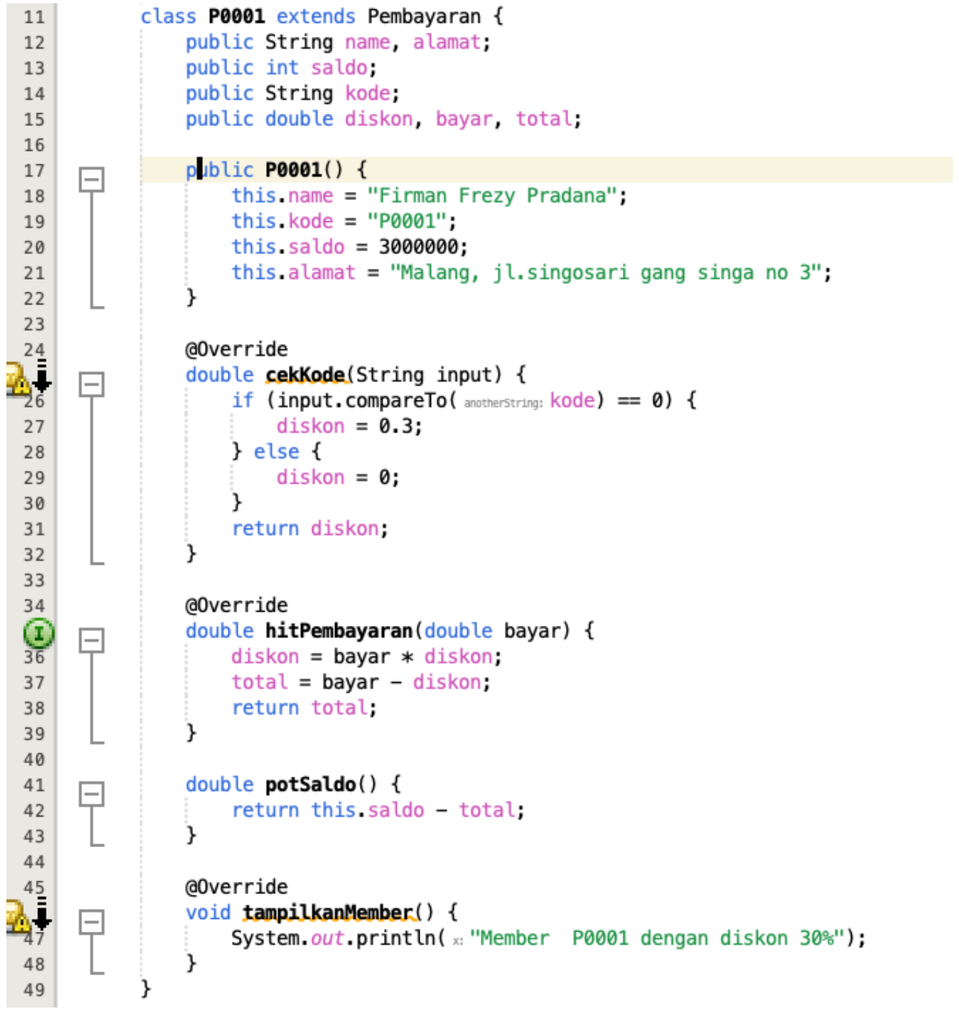
```java
public class S0001 extends Pembayaran {
    public String name, alamat;
    public int saldo;
    public String kode;
    double diskon, bayar, total;
    public S0001() {
        this.name = "Mohammad Harifin";
        this.kode = "S0001";
        this.saldo = 500000;
        this.alamat = "Malang, jl.singosari no 23";
    }
    public double hasil() {
        return diskon;
    }
    @Override
    double cekKode(String input) {
        if (input.compareTo( anotherString: kode) == 0) {
            diskon = 0.1;
        } else {
            diskon = 0;
        }
        return diskon;
    }
    @Override
    double hitPembayaran(double bayar) {
        diskon = bayar * diskon;
        total = bayar - diskon;
        return total;
    }
    double potSaldo() {
        return this.saldo - total;
    }
    @Override
    void tampilkanMember() {
        System.out.println( x: "Member S0001 dengan diskon 10%");
    }
}
```

Latihan Casting



```java
public class G0001 extends Pembayaran {
    public String name, alamat;
    public int saldo;
    public String kode;
    double diskon, bayar, total;
    public G0001() {
        this.name = "Muhammad ridho";
        this.kode = "G0001";
        this.saldo = 5000000;
        this.alamat = "Malang, jl.lowakwaru no 50";
    }
    @Override
    double cekKode(String input) {
        if (input.compareTo( anotherString: kode) == 0) {
            diskon = 0.2;
        } else {
            diskon = 0;
        }
        return diskon;
    }
    @Override
    double hitPembayaran(double bayar) {
        diskon = bayar * diskon;
        total = bayar - diskon;
        return total;
    }
    double potSaldo() {
        return this.saldo - total;
    }
    @Override
    void tampilkanMember() {
        System.out.println( x: "Member G0001 dengan diskon 20%");
    }
}
```

Latihan Casting

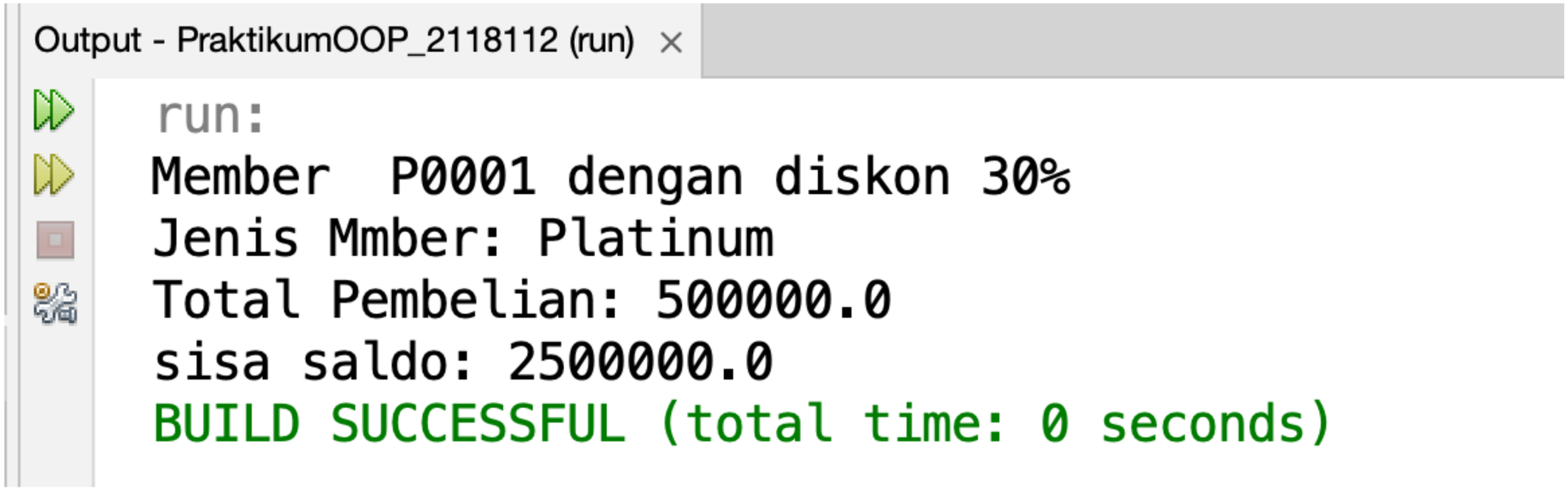


```
11  class P0001 extends Pembayaran {
12      public String name, alamat;
13      public int saldo;
14      public String kode;
15      public double diskon, bayar, total;
16
17      public P0001() {
18          this.name = "Firman Frezy Pradana";
19          this.kode = "P0001";
20          this.saldo = 3000000;
21          this.alamat = "Malang, jl.singosari gang singa no 3";
22      }
23
24      @Override
25      double cekKode(String input) {
26          if (input.compareTo( anotherString: kode) == 0) {
27              diskon = 0.3;
28          } else {
29              diskon = 0;
30          }
31          return diskon;
32      }
33
34      @Override
35      double hitPembayaran(double bayar) {
36          diskon = bayar * diskon;
37          total = bayar - diskon;
38          return total;
39      }
40
41      double potSaldo() {
42          return this.saldo - total;
43      }
44
45      @Override
46      void tampilkanMember() {
47          System.out.println( x: "Member  P0001 dengan diskon 30%");
48      }
49  }
```

Latihan Casting

```
11  public class Drive_pembayaran {
12      public static void main(String[] args) {
13          // Upcasting: Objek dari kelas turunan diubah menjadi objek dari kelas induk
14          Pembayaran payment;
15          payment = new P0001();
16          //tampilan output dari Upcasting
17          payment.tampilkanMember();
18
19          // Downcasting : Objek dari kelas induk diubah kembali menjadi objek dari kelas turunan
20          //Cek apakah P0001 merupakan turunan dari payment.
            if (payment instanceof P0001) {
22              P0001 p0001 = (P0001) payment; //Downcasting dengan menggunakan (P0001)
23
24              // Mengakses metode yang hanya ada di kelas P0001
25              System.out.println("Jenis Mmber: " + p0001.member( member: p0001));
26              System.out.println("Total Pembelian: " + p0001.hitPembayaran( bayar: 500000));
27              System.out.println("sisa saldo: " + p0001.potSaldo());
28          }
29      }
30  }
```

Hasil Casting



"Open NetBeans, and let's write code Java"