Polymorohism in Java

Adharul Muttaqin – Teknik Elektro Universitas Brawijaya Malang

Di Java polymorphism dapat diartikan:

-. Kemampuan suatu refference variable untuk bertindak (melakukan method, mempunyai ciri) sesuai dengan object yang dipegangnya.

-. Memungkinkan berbagai object dari berbagai subclass diperlakukan sebagai sebuah object super class. Super class yang dimaksud akan melakukan tindakan sesuai dengan object subclass tersebut.

Polymorphism dapat diterjemahkan pula sebagai "sebuah method yang sama namanya (homonim) tetapi mempunyai tingkah laku yang berbeda". Komputer membedakan method berdasrkan signature method (**saat compile**) atau berdasarkan reference object (**saat runtime**).

Pada contoh berikut polymorphism ditunjukkan dengan menggunakan berbagai method **`tambah`**. Komputer membedakan mana method yang dipanggil berdasarkan signature dari method yaitu parameter inputnya (jumlah parameter input, type data parametr input, dan urutan type data).

```
public class PenjumlahDemo
{
public static void main(String [] args)
{
Penjumlah obj = new Penjumlah();
System.out.println(obj.tambah(2,5)); // int, int
System.out.println(obj.tambah(2, 5, 9)); // int, int, int
System.out.println(obj.tambah(3.14159, 10)); // double, int
} // end main
}// end PenjumlahDemo

public class Penjumlah
{
public int tambah(int x, int y)
{
return x + y;
}// end tambah(int, int)
public int tambah(int x, int y, int z)
{
return x + y + z;
}// end tambah(int, int, int)
public int tambah(double pi, int x)
{
return (int)pi + x;
}// end tambah(double, int)
}// end Penjumlah
```

Bentuk polymorphism ini dinamakan **early-binding (**atau **compile-time) polymorphism** karena komputer mengetahui mana method `tambah` yang dipanggil setelah mengcompile byte code .

Jadi setelah proses compile ketika code sudah menjadi byte code, komputer akan "tahu" method mana yang akan dieksekusi. Jika hanya ada dua actual parameter dengan tipe int maka method `tambah` yang memiliki dua parameter input. Bentuk ini yang kita kenal dengan **overloaded method.**

Bentuk polymrphism yang lain adalah **late-binding (run time) polymorphism** karena saat compile komputer tidak tahu method mana yang harus dipanggil, dan baru akan diketahui saat run-time. Run-time polymorphism dapat diterapkan melalui **overridden method.** Run time polymorphism dapat dilakukan dalam dua bentuk yaitu dengan abstract base class dan dengan interface. Run time polymorphism juga mengacu pada istilah **dynamic binding.**

Jenis-jenis Run-Time Polymorphism

Ada lima kategoru run time polymorphism :

1. Polymorphic assignment statements
2. Polymorphic Parameter Passing
3. Polymorphic return types
4. Polymorphic (Generic) Array Types
5. Polymorphic exception handling

Pada tulisan ini, kategori yang ke-5 tidak dibahas.

Polymorphic Assignment Statements

Sebelum kita bahas mengenai penerapan polymorphism dengan pada proses pembelian nilai (assignment), mari kita lihat deklarasi berikut ini:

```
int x = 5;
double y = x; // hasilnya adalah y yang diberi nilai 5.0
```

Deklarasi di atas merupakan contoh "type broadening" atau membuat menjadi type yang lebih luas. Variable x bernilia 5 (integer), karena integer (int) adalah bagian dari bilangan real (double) maka x atau 5 dapat di*assign* ke variabel y.
Pada contoh lain:

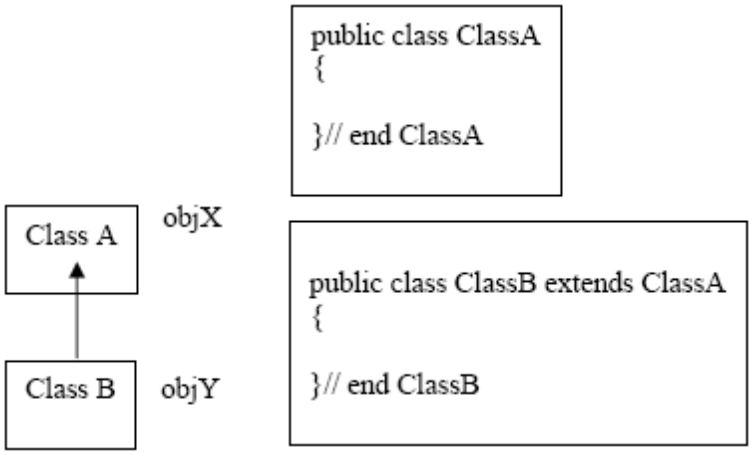
```
double x = 3.14;
int y = x;
```

akan mendapatkan error pada saat compile "Possible loss of precision". JVM mengetahui bahwa akan terjadi pembulatan nilai 3.14 pada kasus di atas dan menganggap kita telah membuat suatu kesalahan. Agar JVM tidak menganggap ini adalah suatu kesalahan kita dapat menggunakan "**type casting"** .

```
double x = 3.14;
int y = (int) x;
```

baris ke dua, (int) x, berarti membuat x yang tadinya bertipe double menjadi bertipe integer.

Kita sekarang kembali ke topik polymorphism. Apa yang kita contohkan di atas adalah tentang variabel **primitive,** sekarang yang akan kita bicarakan adalah **object reference variable.** Perhatikan gambar pada contoh berikut, kita akan dapat menerjemahkan relasi antara objX dan objY yaitu menggunakan relasi "is-a". ClassB mewarisi ClassB. ObjY mempunyai type ClassA, tetapi tidak sebaliknya objX **tidak** mempunyai tipe ClassB. Mengapa? Karena ClassB "adalah" ClassA.




```
public class PolymorphicAssignment
{
public static void main(String [] args)
{
      ClassA obj1 = new ClassA();
      ClassA obj2 = new ClassA();
      ClassB obj3 = new ClassB();
      obj1 = obj2; // 1) no problem here...same data types
      obj1 = obj3; // 2) obj3 is a type of ClassA…ok
      obj3 = obj2; // 3) "incompatible types" compile message
      obj3 = obj1; // 4) still incompatible, karena nilai obj3
```

```
                    // disimpan di obj1 (lihat nomor 2)telah kehilangan
                    // identitas ClassB-nya.
      obj3 = (ClassB)obj1; // 5) the ClassB identity of the object
      // referenced by obj1 has been retrieved!
      // This is called "downcasting"
      obj3 = (ClassB)obj2; // 6) This compiles but will not run.
      // ClassCastException run time error
      // Unlike obj1 the obj2 object ref. variable
      // never was a ClassB object to begin with
} // end main
}// end class
```

Tugas: Cobalah kode di atas (ClassA, ClassB, dan Class PolymorphicAssignment). Mana yang error mana yang tidak? Mengapa?

Pada code di atas,
1) sudah jelas, kedua obj1 dan obj2 adalah bertype ClassA.
2) dapat bekerja karena obj3 adalah sebuah variable yang merefer ke sebuah object dengan type ClassB, dan Class B adalah ClassA juga.
3) tidak akan dicompile, karena mencoba mengasign variable dengan type ClassA dengan nilai variable dengan type ClassB. Ini sama dengan ketika kita mengasign 5.0 ke type integer.
4) lebih kompleks. Kita tahu dari (2) bahwa obj1 secara aktual merefer ke nilai ClassB. Akan tetapi, informasi milik ClassB tidak lagi disimpan pada variable yang merefer ke object ClassA.
5) Menggunakan type casting. Untuk mengembalikan variable object ClassA menjadi variable object ClassB.
6) Secara syntax sama dengan (5) tetapi akan mendapatkan "ClassCastException" karena obj2 tidak pernah menjadi (atau bertype) ClassB.

Contoh:

```
import java.util.Random;
public class PolyAssign
{
public static void main(String [] args)
{
      Shape shp = null;
      Random r = new Random();
      int flip = r.nextInt(2);
      if (flip == 0)
      shp = new Triangle();
      else
      shp = new Rectangle();
      System.out.println("Area = " +
                  shp.area(5,10));
} // end main
}// end class
```

```
abstract class Shape
{
public abstract double area(int,int);
} // end Shape

public class Triangle extends Shape
{
public double area(int b, int h)
{
return 0.5 * b * h;
}
}// end Triangle.

public class Rectangle extends Shape
{
public double area(int b, int h)
{
return b * h;
}
}// end Rectangle
```

Output dipilih seacara random: pakah Area=25 (triangle) atau Area=50 (rectangle)

Di sini kita lihat polymorphism saat run time. JVM tidak tahu isi variable “shp” sampai run-time, yang saat itu memilih method mana yang sesuai dengan object yang diassign.

2. Polymorphic Parameter Passing

Contoh lain implementasi polymorphism adalah dengan melakukan passing paramter actual sebuah variable object sebagai parameter input (formal parameter) sebuah method. Perhatikan contoh berikut untuk memahami type polymorphic ini.

Class **Sayap, Roket**, dan **Kaki** merupakan pewaris dari Class AlatGerak. Sedangkan Class **Flashroket** merupakan anak dari Class Roket. Alat gerak mempunyai method bergerak untuk menunjukkan cara bergerak. Method-method ini dioverride oleh anak-anak classnya.
KuraKura mempunyai alat gerak. Diharapkan kura-kura mampu bergerak sesuai dengan alat gerak yang digunakan. Perhatikan bahwa method setAlatGerak() memberikan kemampuan mengasign suatu nilai dengan type AlatGerak. KuraKura mempunyai method bergerak() untuk menunjukkan caranya bergerak. Didalam method ini dipanggil method bergerak() yang dimiliki oleh class AlatGerak.

```
class AlatGerak{
    void bergerak(){
        System.out.println("    Saya mampu bergerak");
    }
}
```

```
class Sayap extends AlatGerak{
    Sayap(){
        System.out.println("- Persiapan Sayap");
    }
    void bergerak(){
        System.out.println("    Saya bisa terbang");
    }
}
```

```
class Roket extends AlatGerak{
    Roket(){
        System.out.println("- Persiapan Roket");
    }

    void bergerak(){
        System.out.println("    Saya bisa terbang antar benua");
    }
}
```

```
class Kaki extends AlatGerak{
    Kaki(){
        System.out.println("- Persiapan kaki");
    }
    void bergerak(){
        System.out.println("    Saya bisa jalan-jalan meski lambat");
    }
}
```

```
class FlashRoket extends Roket{
    FlashRoket(){
        System.out.println("- Persiapan FlashRoket");
```

```
    }

    void bergerak(){
        System.out.println("    Saya bisa terbang antar benua secepat kilat");
    }
}
```

```
class KuraKura {
    private AlatGerak alatGerak=new AlatGerak();
    KuraKura(){
        System.out.println("Hai saya kura - kura");
    }

    public void bergerak() {
        alatGerak.bergerak();
    }

    public void setAlatGerak(AlatGerak alatGerak) {
        this.alatGerak = alatGerak;
        System.out.println("Sekarang saya pakai " + alatGerak.toString());
    }
}
```

```
class KuraKuraTest{
   public static void main(String[] args){
   KuraKura turtle=new KuraKura();
   turtle.bergerak();

   Sayap sayap=new Sayap();
   Roket roket=new Roket();
   Kaki kaki=new Kaki();
   FlashRoket fRoket=new FlashRoket();
   turtle.setAlatGerak(sayap);
   turtle.bergerak();
   turtle.setAlatGerak(roket);
   turtle.bergerak();
   turtle.setAlatGerak(kaki);
   turtle.bergerak();
   turtle.setAlatGerak(fRoket);
   turtle.bergerak();
   }
}
```

```
Hai saya kura - kura
   Saya mampu bergerak

- Persiapan Sayap
- Persiapan Roket
- Persiapan kaki
- Persiapan Roket
- Persiapan FlashRoket
Sekarang saya pakai Sayap@a90653
   Saya bisa terbang
Sekarang saya pakai Roket@de6ced
   Saya bisa terbang antar benua
Sekarang saya pakai Kaki@c17164
   Saya bisa jalan-jalan meski lambat
Sekarang saya pakai FlashRoket@1fb8ee3
   Saya bisa terbang antar benua secepat kilat
```

Perhatikan method setAlatGerak() yang dimiliki oleh class KuraKura. Method itu mempunyai formal parameter (parameter input) dengan type AlatGerak yang berfungsi untuk mengubah private variable alatGerak yang bertype sama yaitu AlatGerak. Perhatikan pula method bergerak() pada class KuraKura, yang didalamnya memanggil method bergerak() yang dimiliki object alatGerak.

Sekarang perhatikan pada class KuraKuraTest untuk menguji class-class yang kita buat. Perhatikan bahwa:

1. Method setAlatGerak dapat menerima berbagai type yaitu Sayap, Roket, Kaki, dan FlashRoket Hal ini legal karena pada kenyataannya Sayap, Roket, Kaki dan FlashRoket **ADALAH** AlatGerak.

2. Saat method bergerak() milik object trutle dipanggil, method yang dipanggil adalah sesuai dengan method yang dimiliki oleh type yang diberikan.

3. Polymorphic Return Types

Saat memberikan nilai kembalian pada suatu method kita tahu bahwa type yang dikembalikan harus kompatibel antara variable yang menerima nilai dan variable yang memberikan nilai.

Contoh:

`int x = retMethod();` dan `public int retMethod(){ return 5;}`

Kita dapat menulisnya dengan :

`double x = retMethod();` and `public int retMethod(){ return 5;}`

karena nilai 5 yang dikembalikan juga bertype duoble yang diperluas menjadi 5.0.

Code berikut:

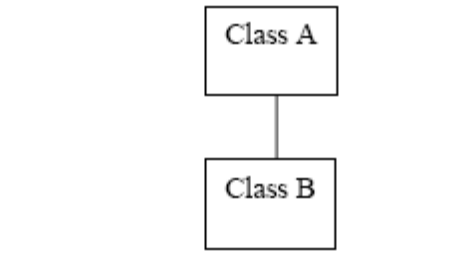
`int x = retMeth2();` with `public double retMeth2() { return 5;}`

memberikan error "`type compatibility`" meskipun 5.0 adalah nilai asalnya. Kecuali kita melakukan type casting:

```
int x = (int) retMeth2();
```

Contoh diatas dapat kita bawa untuk menjelaskan polimorphism tipe 3 yaitu polymorphism pada type kembalian.

Perhatikan dan jalankan contoh berikut. Hilangkan tanda angka 1.), 2.) dst untuk mencobanya.




```
public class ClassA
{

}// end ClassA
```

```
public class ClassB extends ClassA
{

}// end ClassB
```

```
public class PolymorphicReturnTypes
{
   public static void main(String [] args)
    {
       ClassA obj1 = new ClassA();
       ClassA obj2 = new ClassA();
       ClassB obj3 = new ClassB();

       1.) obj1 = method1();
       2.) obj1 = method2();
       3.) //obj3 = method1();  // incompatible types
       4.) //obj3 = method3();  // incompatible...why?
       5.) obj3 = (ClassB) method3();
       6.) //obj3 = (ClassB) method1();

    }  // end main

    public static ClassA method1()  {  return new ClassA(); }

    public static ClassB method2()  {  return new ClassB(); }

    public static ClassA method3()  {  return new ClassB(); }

}// end class
```

1. Mengapa 1.) dapat dicompile dan dieksekusi?
2. Mengapa 2.) dapat dicompile dan dieksekusi?
3. Mengapa 3.) gagal dicompile dan dieksekusi?
4. mengapa 4.) gagal dicompile dan dieksekusi? Apa bedanya dengan 3.)
5. Bagian 5.) mirip dengan 4.). Bagaimana bisa sukses sedangkan 4.) gagal?
6. Bgian 6.) mirip dengan 5.). Bagian 6.) dapat dicompile tapi tidak dapat dieksekusi. Mengapa?

4. Polymorphic (Generic) Array Types

Seperti ilustrasi sebelumnya,

```
int  x = 5;
double[] y = new double[3];
y[0]=x; // hasilnya adalah y index ke 0 bernilai 5.0
y[2]=3.14; // hasilnya adalah y index ke 2 bernilai 3.14
```

Deklarasi di atas merupakan contoh "type broadening" atau membuat menjadi type yang lebih luas. Variable x bernilia 5 (integer), karena integer (int) adalah bagian dari bilangan real (double) maka x atau 5 dapat di*assign* ke variabel y[0].

Hal ini juga berlaku pada variable refference ke object. Jadi kita dapat membuat variable dengan type base class kemudian masing-masing anggota array tersebut diassign suatu nilai yang merefer ke object dengan type child class. Perhatikan contoh berikut.

Contoh:
Lihat model Alat gerak pada contoh sebelumnya. Kita dapat mendeklarasikan sebuah variable **array** suatu tipe (array of type). Sedangkan isi dari array tersebut dapat berbeda-beda asalkan masih mempunyai type yang merupakan extend dari AlatGerak.

```
class TestAlatGerak {

Public void static main(String arg[]) {
   AlatGerak[] daftarAlatGerak=new AlatGerak[3];

   daftarAlatGerak[0]=sayap;
   daftarAlatGerak[1]=kaki;
   daftarAlatGerak[2]=roket;

   for (int i=0 ;i<3;i++)
      daftarAlatGerak[i].bergerak();
   }

}
```