BAB 8
POLIMORFISME

A. POKOK BAHASAN

⊕ Konsep dasar polimorfisme

⊕ Virtual Method Invocation

⊕ Polymorphic arguments

⊕ Pernyataan instanceof

⊕ Casting object

B. TUJUAN BELAJAR

Dengan praktikum ini mahasiswa diharapkan dapat:

- ✓ Memahami dan menerapkan konsep polimorfisme dalam pemrograman
- ✓ Memahami proses terjadinya Virtual Method Invocation
- ✓ Memahami dan menerapkan polymorphic arguments dalam pemrograman
- ✓ Memahami penggunaan instanceof dan cara melakukan casting object

C. DASAR TEORI

Polymorphism (polimorfisme) adalah kemampuan untuk mempunyai beberapa bentuk class yang berbeda. Polimorfisme ini terjadi pada saat suatu obyek bertipe parent class, akan tetapi pemanggilan constructornya melalui subclass. Misalnya deklarasi pernyataan berikut ini:

dimana Manager() adalah kontruktor pada class Manager yang merupakan subclass dari class Employee.

```
Employee employee=new Manager();
//<nama class> <variable bebas> = new <kontruktor>();
```

Virtual Method Invocation (VMI) bisa terjadi jika terjadi polimorfisme dan overriding. Pada saat obyek yang sudah dibuat tersebut memanggil overridden method pada parent class, kompiler Java akan melakukan invocation (pemanggilan) terhadap overriding method pada subclass, dimana yang seharusnya dipanggil adalah overridden method. Berikut contoh terjadinya VMI:

```
class Parent {
int x = 5;
public void Info() {
System.out.println("Ini class Parent");
}
}
class Child extends Parent {
int x = 10;
public void Info() {
System.out.println("Ini class Child");
}
}
public class Tes {
public static void main(String args[]) {
Parent tes=new Child();
System.out.println("Nilai x = " +
tes.x); tes.Info();
}
}
```

Hasil dari running program diatas adalah sebagai berikut:

```
Nilai x = 5
Ini class Child
```

Polymorphic arguments adalah tipe suatu parameter yang menerima suatu nilai yang bertipe subclass-nya. Berikut contoh dari polymorphics arguments:

```
class Pegawai {
…
}
class Manajer extends Pegawai {
…
}
public class Tes {
public static void Proses(Pegawai peg) {
…
}
public static void main(String args[]) { Manajer man = new
Manajer(); Proses(man);
}
```

Pernyataan instanceof sangat berguna untuk mengetahui tipe asal dari suatu polymorphic arguments. Untuk lebih jelasnya, misalnya dari contoh program sebelumnya, kita sedikit membuat modifikasi pada class Tes dan ditambah sebuah class baru Kurir, seperti yang tampak dibawah ini:

```
…
class Kurir extends Pegawai {
…
}
public class Tes {
public static void Proses(Pegawai peg) {
if (peg instanceof Manajer) {
…lakukan tugas-tugas manajer…
} else if (peg instanceof Kurir) {
…lakukan tugas-tugas kurir…
} else {
…lakukan tugas-tugas lainnya…
}
}
public static void main(String args[]) { Manajer man =
new
Manajer();
Kurir kur = new Kurir(); Proses(man);
Proses(kur);
}
}
```

Seringkali pemakaian instanceof diikuti dengan casting object dari tipe parameter ke tipe asal. Misalkan saja program kita sebelumnya. Pada saat kita sudah melakukan instanceof dari tipe Manajer, kita dapat melakukan casting object ke tipe asalnya, yaitu Manajer. Caranya adalah seperti berikut:

```
…
if (peg instanceof Manajer) { Manajer man = (Manajer) peg;
…lakukan tugas-tugas manajer…
}
```

D. PERCOBAAN

Virtual Method Invocation

Buatlah 3 class dalam project kalian kemudia beri nama dan isi sebagai berikut :

Pegawai

```
public class Pegawai {
   private String name;
   private String address;
   private int number;
   public Pegawai(String name, String address, int
number){
      System.out.println("Menyusun Pegawai");
      this.name = name;
      this.address = address;
      this.number = number;
   }
   public void mailCheck(){
      System.out.println("Memeriksa Surat Untuk " +
this.name+ " " + this.address);
   }
   public String toString(){
      return name + " " + address + " " + number;
   }
   public String getName(){
      return name;
   }
   public String getAddress(){
      return address;
   }
   public void setAddress(String newAddress){
      address = newAddress;
   }
   public int getNumber(){
     return number;
   }
}
```

Gaji

```
public class Gaji extends Pegawai
{
   private double salary; //Gaji Tahunan
   public Gaji(String name, String address, int number,
double
       salary){
        super(name, address, number);
        setSalary(salary);
   }
   public void mailCheck(){
        System.out.println("Memeriksa kelas gaji dalam
surat ");
        System.out.println("Surat tertuju untuk " +
getName()
        + " dengan gaji " + salary);
   }
   public double getSalary(){
        return salary;
   }
   public void setSalary(double newSalary){
        if(newSalary >= 0.0){
           salary = newSalary;
        }
   }
   public double computePay(){
      System.out.println("Menghitung pembayan gaji untuk
" + getName());
      return salary/52;
   }
}
```

VirtualDemo

```
public class VirtualDemo{

  public static void main(String [] args){

    Gaji s = new Gaji("Wahyu", "KUBAR", 3, 5000.00);

    Pegawai e = new Gaji("Ini nama", "Samarinda", 2, 2500.00);

    System.out.println("Memanggil mailCheck Berdasarkan Referensi
Gaji --");

    s.mailCheck();

    System.out.println("\nMemanggil mailCheck Berdasarkan
Referensi Pegawai--");

    e.mailCheck();

  }

}
```

Hasil dari running program diatas adalah sebagai berikut:

```
Menyusun Pegawai
Memanggil mailCheck Berdasarkan Referensi Gaji --
Memeriksa kelas gaji dalam surat
Surat tertuju untuk Wahyu dengan gaji 5000.0

Memanggil mailCheck Berdasarkan Referensi Pegawai--
Memeriksa kelas gaji dalam surat
Surat tertuju untuk Ini nama dengan gaji 2500.0
```

Analisislah mengapa hasil akhirnya seperti ini !