Pemrograman Berbasis Obyek

Polymorphism

Oleh Politeknik Elektronika Negeri Surabaya

2017

Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer

Konten

- Polymorphism
- Virtual Method Invocation
- Polymorphic arguments
- Operator instanceof
- Casting & Convertion Objects

Polymorphism

- Polymorphism adalah kemampuan obyek untuk mempunyai beberapa bentuk class yang berbeda.
- Polimorfisme terjadi pada saat suatu obyek bertipe parent class, akan tetapi konstruktor yang dipanggil adalah konstruktor subclass

Misal

```
public class Employee {
        public String nama;
        public String gaji;

        void info(){
                System.out.println("Nama" + nama);
            System.out.println("Gaji" + gaji);
        }
}

public class Manajer extends Employee {
        public String departemen;

         void info(){
                System.out.println("Nama" + nama);
                System.out.println("Gaji" + gaji);
                System.out.println("Departemen" + departemen);
        }
}
```

Contoh

Employee emp = new Manager();

- Reference variabel dari emp adalah Employee.
- Bentuk emp adalah Employee.

Virtual Method Invocation

- Virtual method invocation merupakan suatu hal yang sangat penting dalam konsep polimorfisme.
- Syarat terjadinya VMI adalah sebelumnya sudah terjadi **polymorphism** 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.

Contoh Virtual Method Invocation

```
class Employee{}

class Manager extends Employee{}

Class Tes{
        …
        Employee emp = new Manager();
        emp.info();
}
```

Virtual Method Invocation

Yang terjadi pada contoh:

- Obyek e mempunyai behavior yang sesuai dengan runtime type bukan compile type.
- Ketika compile time e adalah Employee.
- Ketika runtime e adalah Manager.
- Jadi :
  - emp hanya bisa mengakses variabel milik Employee.
  - emp hanya bisa mengakses non private method milik Employee
  - Jika emp memanggil method Employee yang di-override oleh Manager, maka method yang dijalankan adalah method milik Manager (VMI)

Virtual Method Invocation

- Bagaimana dengan konstruktor yang dijalankan?
- Pada pembentukan

```
Employee e = new Manager();
```

- Pertama kali akan menjalankan konstruktor Manager, ketika ketemu super() maka akan menjalankan konstruktor Employee (superclass), setelah semua statement dieksekusi baru kemudian melanjutkan menjalankan konstruktor Manager (subclass).

Heterogeneous Collections

- Collections of objects with the same class type are called *homogenous* collections.

```
MyDate[]  dates = new MyDate[2];
dates[0] = new MyDate(22, 12, 1964);
dates[1] = new MyDate(22, 7, 1964);
```

- Collections of objects with different class types are called *heterogeneous* collections.

```
Employee [] staff = new Employee[1024];
staff[0] = new Manager();
staff[1] = new Employee();
staff[2] = new Engineer();
```

Virtual Method Invocation pada C++

Pada method yang akan dilakukan VMI harus ditandai dengan kata **`virtual`**.

Polymorphic Arguments

Polymorphic arguments adalah tipe data suatu argumen pada suatu method yang bisa menerima suatu nilai yang bertipe subclass-nya.

```
class Pegawai {
    …
}

class Manajer extends Pegawai {
    …
}

class Engineer extends Pegawai {
    …
}

public class Tes {
    public static void gajiPeg(Pegawai peg) {
                …
    }

    public static void main(String args[]) {
                Manajer m = new Manajer();
                gajiPeg(m);
                Engineer e = new Engineer();
                gajiPeg(e);
    }
}
```

Operator instanceof

Pernyataan instanceof sangat berguna untuk mengetahui tipe asal dari suatu polymorphic arguments

```
…
class Engineer extends Pegawai {
      …
}

public class Tes {
      public static void gajiPeg(Pegawai peg) {
                  if (peg instanceof Manajer) {
                              Manager m = (Manager) peg;
                              … lakukan kalkulasi gaji manajer…
                  } else if (peg instanceof Engineer) {
                              Engineer eng = (Engineer) peg;
                              lakukan kalkulasi gaji Engineer …
                  } else {
                               … lakukan kalkulasi gaji lainnya…
                  }
      }

      public static void main(String args[]) {
                  Manajer man = new Manajer();
                  Engineer eng = new Engineer();
                  gajiPeg (man);
                  gajiPeg (eng);
      }
}
```

Casting object

- Seringkali pemakaian instanceof diikuti dengan casting object dari tipe parameter ke tipe asal.

- Tanpa adanya casting obyek, maka nilai yang akan kita pakai setelah proses instanceof masih bertipe parent class-nya, sehingga jika ia perlu dipakai maka ia harus di casting dulu ke tipe subclass-nya.

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

Kenapa diperlukan polymorphic arguments?

- Mengefisienkan pembuatan program
- Misal Employee mempunyai banyak subclass.
- Maka kita harus mendefinisikan semua method yang menangani behavior dari masing-masing subclass.
- Dengan adanya polymorphic arguments kita cukup mendefinisikan satu method saja yang bisa digunakan untuk menangani behavior semua subclass.

Tanpa polymorphic arguments

```
…
public class Tes {
    public static void ProsesManajer() {
            …lakukan tugas-tugas manajer…
    }

    public static void ProsesKurir() {
            …lakukan tugas-tugas kurir…
    }
    …
}
```

Object Reference Conversion

- Pada object reference bisa terjadi:
  - Assignment conversion
  - Method-call conversion
  - Casting
- Pada object references tidak terdapat arithmetic promotion karena references tidak dapat dijadikan operan arithmetic.
- Reference conversion terjadi pada saat kompile

Object Reference Assignment Conversion

- Terjadi ketika kita memberikan nilai object reference kepada variabel yang tipenya berbeda.
- Three general kinds of object reference type:
  - A **class** type, such as Button or Vector
  - An **interface** type, such as Runnable or LayoutManager
  - An **array** type, such as int[][] or TextArea[]

- Contoh:

```
1. Oldtype x = new Oldtype();
2. Newtype y = x; // reference assignment conversion
```

Converting OldType to NewType

	OldType is a class	**OldType is an interface**	**OldType is an array**
NewType is a class	Oldtype must be a subclass of Newtype	NewType must be Object	NewType must be Object
NewType is an interface	OldType must implement interface NewType	OldType must be a subinterface of NewType	NewType must be Clonable or serializable
NewType is an array	Compile error	Compile error	OldType must be an array of some object reference

```
Oldtype x = new Oldtype();
Newtype y = x; // reference assignment conversion
```

The rules for object reference conversion

- Interface hanya dapat di konversi ke interface atau Object.

  Jika NewType adalah interface, maka NewType ini harus merupakan superinterface dari OldType.
- Class hanya bisa dikonversi ke class atau interface.

  Jika dikonversi ke class, NewType harus merupakan superclass dari OldType.

  Jika dikonversi ke interface, OldType (class) harus mengimplementasikan (NewType) interface
- Array hanya dapat dikonversi ke Object, interface Cloneable atau Serializable, atau array.

  Hanya array of object references yang dapat dikonversi ke array, dan old element type harus convertible terhadap new element type.

Contoh 1 :

```
Tangelo tange = new Tangelo();
Citrus cit = tange; // No problem
```

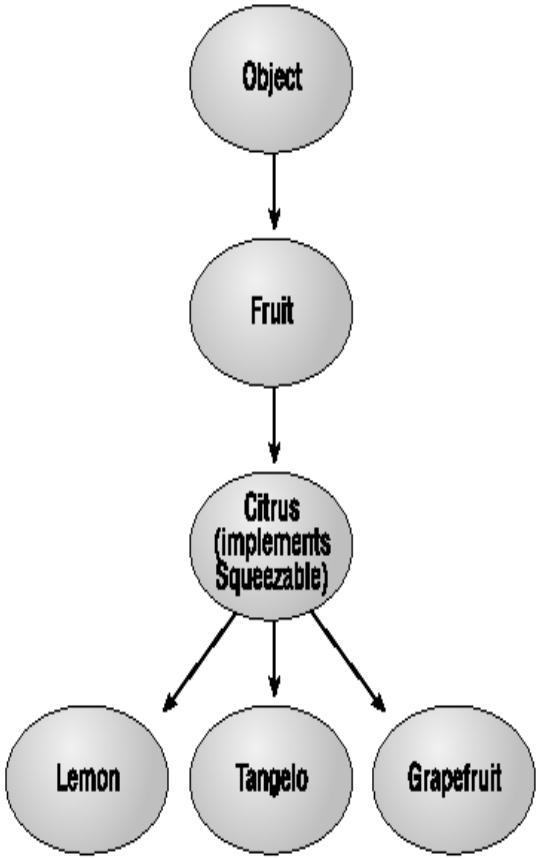
Contoh 2:

```
Citrus cit = new Citrus();
Tangelo tange = cit; // compile error
```

Contoh 3:

```
Grapefruit g = new Grapefruit();
Squeezable squee = g;// No problem
Grapefruit g2 = squee; // Error
```

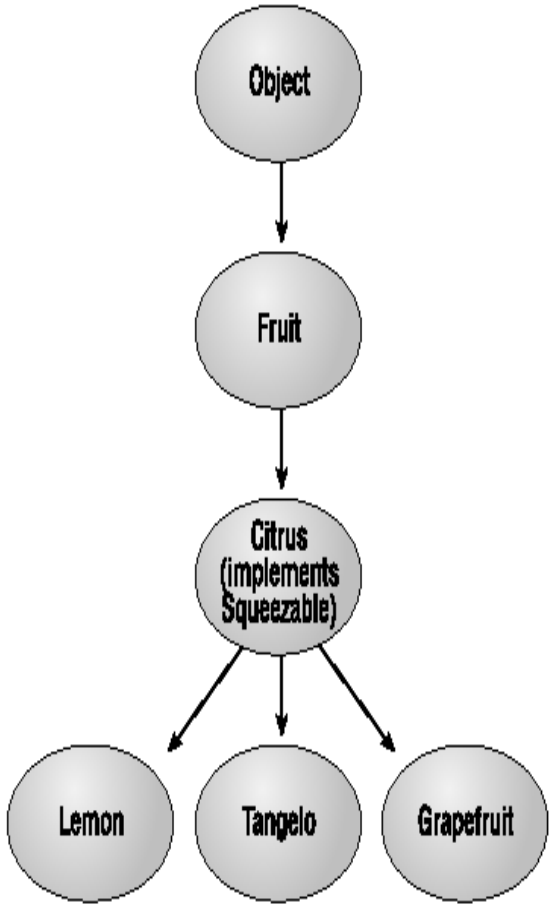
A simple class hierarchy



Contoh 4 :

```
Fruit fruits[];
Lemon lemons[];
Citrus citruses[] = new Citrus[10];
For (int I=0; I<10; I++) {
  citruses[I] = new Citrus();
}
fruits = citruses;  // No problem
lemons = citruses; // Error
```

A simple class hierarchy



Object Method-Call Conversion

- Aturan object reference method-call conversion sama dengan aturan pada object reference assignment conversion.
- Converting to superclass → permitted.
- Converting to subclass → not permitted.

Object Method-Call Conversion

Contoh:

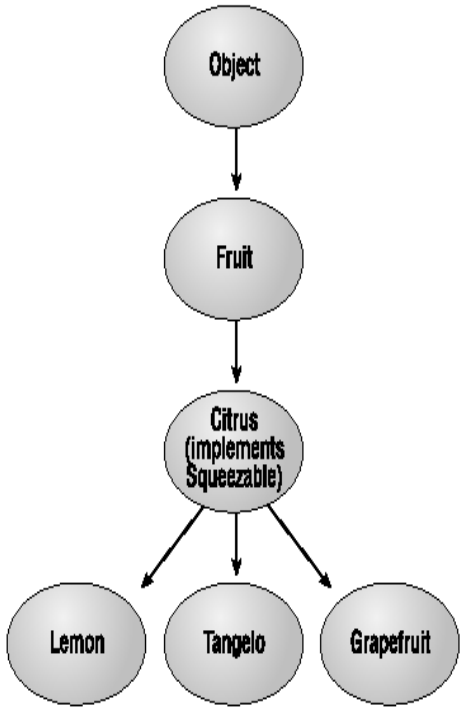
```
Vector myVec = new Vector();

Tangelo tange = new Tangelo();

myVect.add(tange); // No problem
```

Note: method add pada vector meminta satu parameter → `add(Object ob)`

A simple class hierarchy



Object Reference Casting

- Is like primitive casting
- Berbagai macam konversi yang diijinkan pada object reference assignment dan method call, diijinkan dilakukan eksplisit casting.

Contoh:

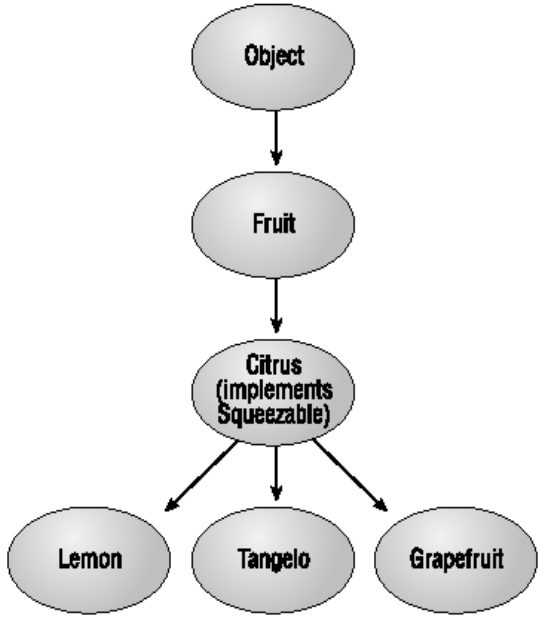
```
Lemon lem = new Lemon();
Citrus cit = lem;  // No problem
```

Sama dengan:

```
Lemon lem = new Lemon();
Citrus cit = (Citrus) lem;  // No problem
```

- The cast is **legal** but **not needed.**
- The power of casting appears when you explicitly cast to a type that is not allowed by the rules of implicit conversion.

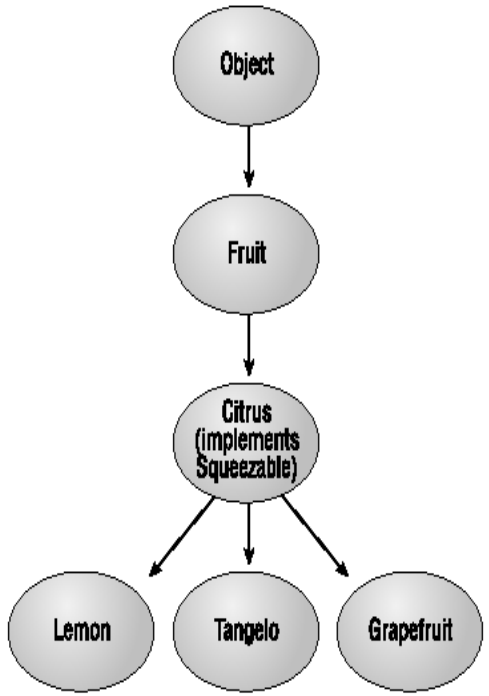
A simple class hierarchy



Object Reference Casting

```
1. Grapefruit g, g1;
2. Citrus c;
3. Tangelo t;
4. g = new Grapefruit();
        // Class is Grapefruit
5. c = g;
        // Legal assignment conversion,
        // no cast needed
6. g1 = (Grapefruit)c;
        // Legal cast
7. t = (Tangelo)c;
        // Illegal cast
        // (throws an exception)
```

A simple class hierarchy



- Kompile → ok, kompiler tidak bisa mengetahui object reference yang di pegang oleh c.
- Runtime → error → class c adalah Grapefruit

Object Reference Casting

Example: Object is cast to an interface type.

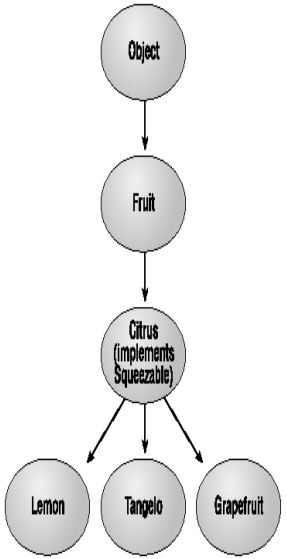
```
1. Grapefruit g, g1;
2. Squeezable s;
3. g = new Grapefruit();
4. s = g;       // Convert Grapefruit to Squeezable (OK)
5. g1 = s;      // Convert Squeezable to Grapefruit
                // (Compile error)
```

- Implicitly converting an interface to a class is never allowed
- Penyelesaian : gunakan eksplisit casting

```
g1 = (Grapefruit) s;
```

- Pada saat runtime terjadi pengecekan.

A simple class hierarchy

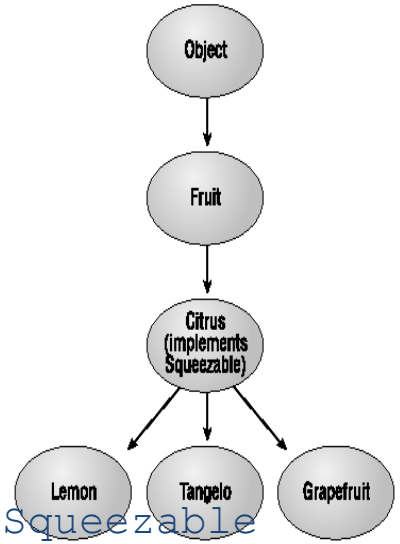


Object Reference Casting

Example: array.

```
1. Grapefruit g[];
2. Squeezable s[];
3. Citrus c[];
4. g = new Grapefruit[500];
5. s = g;          // Convert Grapefruit array to Squeezable
   array (OK)
6. c = (Citrus[])s; // Convert Squeezable array to Citrus
   array (OK)
```

A simple class hierarchy



Tugas

1. Apakah yang dimaksud dengan polimorfisme ?
2. Jelaskan proses terjadinya Virtual Method Invocation !
3. Apakah yang dimaksud dengan polymorphic arguments ?
4. Apakah kegunaan kata kunci instanceof ?

1. Oracle Java Documentation, The Java™ Tutorials, https://docs.oracle.com/javase/tutorial/, Copyright © 1995, Oracle 2015.
2. Tita Karlita, Yuliana Setrowati, Rizky Yuniar Hakkun, Pemrograman Berorientasi Obyek, PENS-2012
3. Sun Java Programming, Sun Educational Services, Student Guide, Sun Microsystems, 2001.
4. John R. Hubbard, Programming With Java, McGraw-Hill, ISBN: 0-07-142040-1, 2004.
5. Patrick Niemeyer, Jonathan Knudsen, Learning Java, O'reilly, CA, ISBN: 1565927184, 2000.
6. Philip Heller, Simon Roberts, Complete Java 2 Certification Study Guide, Third Edition, Sybex, San Francisco, London, ISBN: 0-7821-4419-5, 2002.
7. Herbert Schildt, The Complete Reference, Java™ Seventh Edition, Mc Graw Hill, Osborne, ISBN: 978-0-07-163177-8, 2007