Temu-07

Polymorphism

Ali Tarmuji, S.T., M.Cs.
alitarmuji@tif.uad.ac.id

Bahan Diskusi

- Polymorphism
- Overriding
- Overloading
- Constructor overloading
- Virtual Method Invocation
- Polymorphic arguments
- Operator instanceof
- Casting & Convertion Objects
- Method Final
- Beberapa fungsi terkait

Apa itu Polimorfisme?

- Umum: Polymorphism adalah kemampuan untuk mempunyai beberapa bentuk yang berbeda.
- OOP: Polimorfisme merupakan kemampuan sebuah kelas bertingkah laku atau memiliki sifat seperti kelas turunannya.
- Penggunaan polimorfisme yang paling umum dalam OOP terjadi ketika referensi kelas induk digunakan untuk merujuk ke objek kelas anak.
- Dengan menggunakan polimorfisme, sebuah kelas dapat disamarkan menjadi seperti kelas parentnya.

Polymorphism:

- Objek Java apa pun yang dapat melewati lebih dari satu tes IS-A dianggap polimorfik.
- Di Java, semua objek Java bersifat polimorfik karena objek apa pun akan lulus tes IS-A untuk tipenya sendiri dan untuk objek kelas.
- Penting untuk diketahui bahwa satu-satunya cara yang mungkin untuk mengakses suatu objek adalah melalui variabel referensi.
- Variabel referensi bisa hanya satu jenis.
- Setelah dideklarasikan, tipe variabel referensi tidak dapat diubah.
- Variabel referensi dapat dipindahkan ke objek lain asalkan tidak dinyatakan final.
- Jenis variabel referensi akan menentukan metode yang dapat digunakan pada objek.
- Variabel referensi dapat merujuk ke objek apa pun dari tipe yang dideklarasikan atau subtipe dari tipe yang dideklarasikan.
- Variabel referensi dapat dideklarasikan sebagai tipe kelas atau antarmuka.
- Satu obyek hanya boleh mempunyai satu bentuk saja.
- Yaitu bentuk yang diberikan ketika obyek dibuat.
- Reference variabel bisa menunjuk ke bentuk yang berbeda.

Misal: Manager adalah Employee

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

        void infoNama(){
                System.out.println("Nama" + nama);
        }
}

public class Manajer extends Employee {
        public String departemen;
}

Employee emp = new Manager();
```

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

Apa itu Overiding?

- Overiding merupakan kemampuan **mendeklarasikan ulang** prosedur atau fungsi yang sudah ada di kelas *parent*.
- Deklarasi method pada subclass harus sama dengan yang terdapat di super class. Kesamaan pada:
  - Nama
  - Return type
  - Daftar parameter (jumlah, tipe, dan urutan)
- Method pada parent class disebut overriden method
- Method pada subclass disebut overriding method.
- Aturan:
  - Mode akses overriding method harus sama atau lebih luas dari pada overriden method.
  - Subclass hanya boleh meng-override method superclass satu kali saja, tidak boleh ada lebih dari satu method pada kelas yang sama yang sama persis.
  - Overriding method tidak boleh throw checked exceptions yang tidak dideklarasikan oleh overridden method.

Contoh Overiding

```
public class Parent {
      public void tampilInfo(){
      System.out.println("Parent");
      }
}

public class Child extends Parent {
      public void tampilInfo(){
            System.out.println("Child");
      }

}
```

Contoh Overiding

```
Child anak = new Child();
anak.tampilInfo();

// output
Child

Parent anak = new Parent();
anak.tampilInfo();

// output
Parent
```

Contoh Overriding

```
public class Employee {
   protected String name;
   protected double salary;
   protected Date birthDate;

   public String getDetails() {
      return "Name: " + name + "\n" +
             "Salary: " + salary;
   }
}


public class Manager extends Employee {
   protected String department;

   public String getDetails() {
      return "Name: " + name + "\n" +
             "Salary: " + salary + "\n" +
             "Manager of: " + department;
   }
}
```

Contoh Overriding

```
public class Animal {
        public void SetVoice() {
        System.out.println("Blesepblesep");
        }
}

public class Dog extends Animal {
        public void SetVoice() {
                System.out.println("Hug hug");
        }
}
```

Apa itu Overloading

- Overloading merupakan kemampuan untuk mendefinisikan beberapa method dengan nama yang sama, tapi daftar parameternya berbeda.
- Menuliskan kembali method dengan nama yang sama pada suatu class.
- Tujuan: memudahkan penggunaan/pemanggilan method dengan fungsionalitas yang mirip.
- Menuliskan kembali method dengan nama yang sama pada suatu class.
- Tujuan : memudahkan penggunaan/pemanggilan method dengan fungsionalitas yang mirip.
- Aturan deklarasi method overloading:
  - Nama method harus sama
  - Daftar parameter harus berbeda
  - Return type boleh sama, juga boleh berbeda

Daftar Parameter Pada Overloading

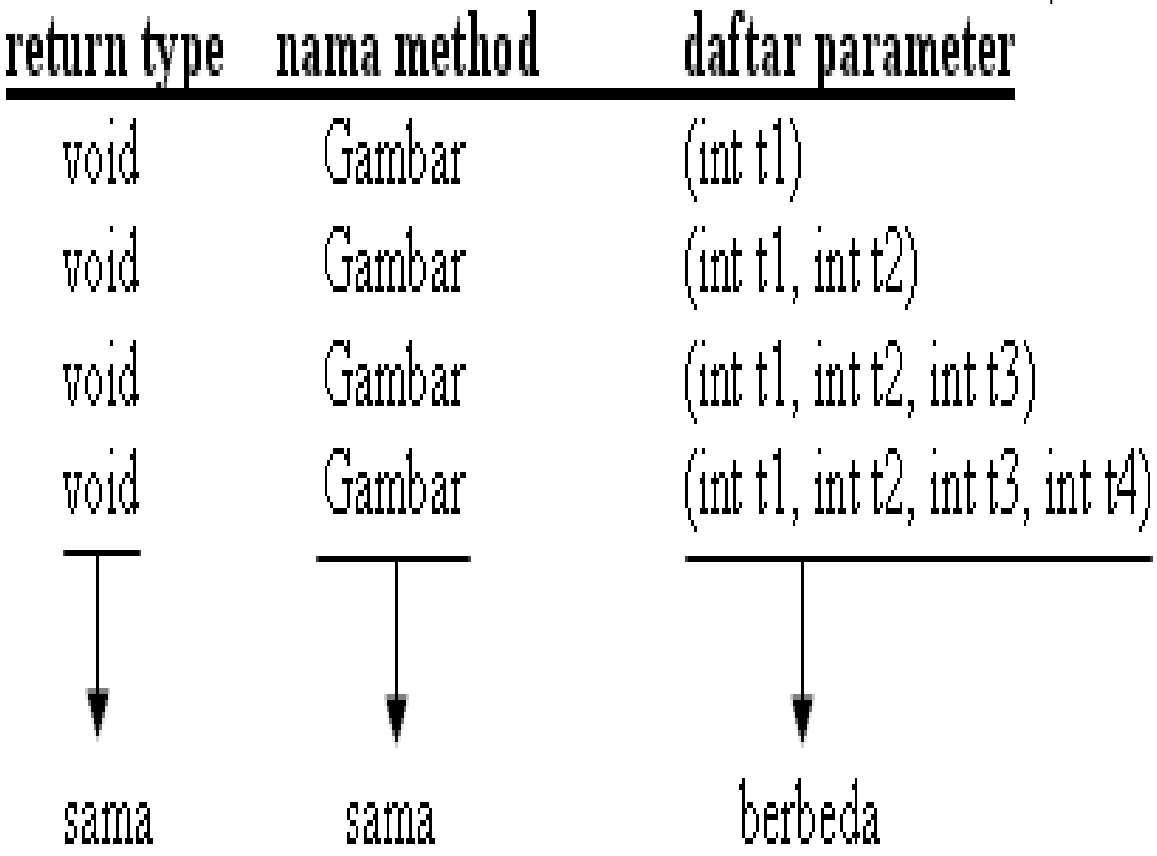
- Perbedaan daftar parameter bukan hanya terjadi pada perbedaan banyaknya parameter, tetapi juga urutan dari parameter tersebut.
- Misalnya saja dua buah parameter berikut ini :
  - function_member(int x, String n)
  - function_member(String n, int x)
- Dua parameter tersebut juga dianggap berbeda daftar parameternya.
- Daftar parameter tidak terkait dengan penamaan variabel yang ada dalam parameter.
- Misalnya saja 2 daftar parameter berikut :
  - function_member(int x)
  - function_member(int y)
- Dua daftar parameter diatas dianggap sama karena yang berbeda hanya penamaan variabel parameternya saja.

- Overloading juga bisa terjadi antara parent class dengan subclass-nya jika memenuhi ketiga syarat overload.
- Misalnya saja dari class Bentuk pada contoh sebelumnya kita turunkan sebuah class baru yang bernama WarnaiBentuk.
- Contoh:

```
public void println(int i)
public void println(float f)
public void println(String s)
```

Contoh overloading

```
public class Bentuk {
    ...
    public void Gambar(
       int t1) {
            ...
    }
    public void Gambar(
       int t1, int t2) {
            ...
    }
    public void Gambar(
       int t1, int t2, int t3) {
            ...
    }
    public void Gambar(
        int t1, int t2, int t3, int t4) {
            ...
    }
}
```



Contoh overloading

```
public class WarnaiBentuk extends Bentuk {
    public void Gambar(String warna, int t1, int t2, int3) {
            …
    }
    public void Gambar(String warna, int t1, int t2, int3, int
    t4) {
            …
    }
    …
}
```

Constructor Overloading

- As with methods, constructors can be overloaded.
- Example:

```
public Employee(String name, double salary, Date DoB)
public Employee(String name, double salary)
public Employee(String name, Date DoB)
```

- Argument lists *must* differ.
- You can use the `this` reference at the first line of a constructor to call another constructor.

Constructor Overloading

```
1   public class Employee {
2     private static final double BASE_SALARY = 15000.00;
3     private String name;
4     private double salary;
5     private Date   birthDate;
6
7     public Employee(String name, double salary, Date DoB) {
8       this.name = name;
9       this.salary = salary;
10      this.birthDate = DoB;
11    }
12    public Employee(String name, double salary) {
13      this(name, salary, null);
14    }
15    public Employee(String name, Date DoB) {
16      this(name, BASE_SALARY, DoB);
17    }
18    public Employee(String name) {
19      this(name, BASE_SALARY);
20    }
21    // more Employee code...
22  }
```

Memanggil parent class construktor

```
1   public class Manager extends Employee {
2     private String department;
3
4     public Manager(String name, double salary, String dept) {
5       super(name, salary);
6       department = dept;
7     }
8     public Manager(String n, String dept) {
9       super(name);
10      department = dept;
11    }
12    public Manager(String dept) {
13      department = dept;
14    }
15  }
```

Contoh Overloading

```
public class Parent {

    public void jumlah(){
        int hasil = 2 + 2;
        return hasil;
    }

    public void jumlah(int x, int y){
        int hasil = x + y;
        return hasil;
    }

}
```

Virtual Method Invocation

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

```
class Employee{}
class Manager extends Employee{}
…
Employee emp = new Manager();
emp.getDetails();
```

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 method milik Manager
- 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 menjalankan konstruktor Manager (subclass).

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 {
   …
}

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

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

Operator instanceof

- Pernyataan instanceof sangat berguna untuk mengetahui tipe asal dari suatu polymorphic arguments
- Kata kunci instanceof digunakan untuk mengecek apakah sebuah objek merupakan kelas tertentu.
- Hasil dari penggunaan instanceof adalah boolean.
- Sintaks menggunakan instanceof

```
boolean hasil = objek instanceof NamaKelas;

if(objek instanceof NamaKelas){
   // jika objek adalah NamaKelas
}
```

Contoh menggunakan instanceof

```
public class Employee extends Object
public class Manager extends Employee
public class Engineer extends Employee
-------------------------------------


public void doSomething(Employee e) {
  if (e instanceof Manager) {
    // Process a Manager
  } else if (e instanceof Engineer) {
    // Process an Engineer
  } else {
    // Process any other type of Employee
  }
}
```

```
…
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);
    }
}
```

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.
- Misal:

```
…
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
```

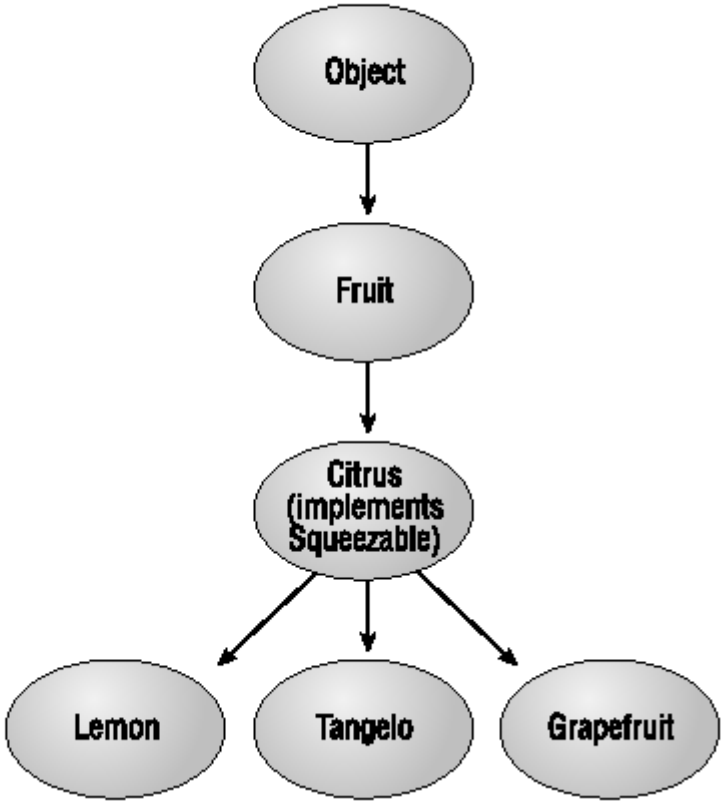
Contoh 3:

```
Grapefruit g = new Grapefruit();
```

```
Squeezable squee = g;// No problem
```

```
Grapefruit g2 = squee;  // Error
```

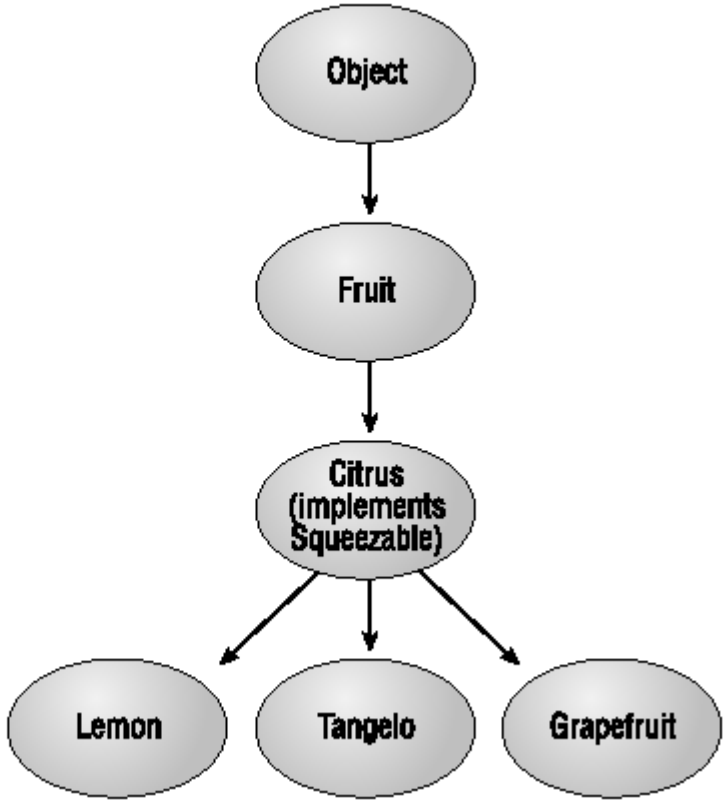
A simple class hierarchy



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:

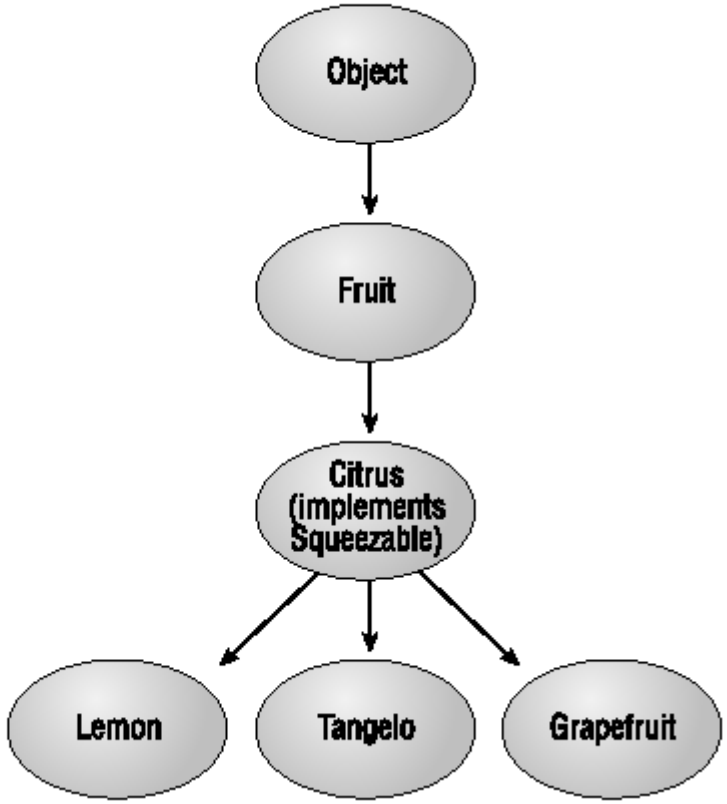
```
Vector myVec = new Vector();

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:

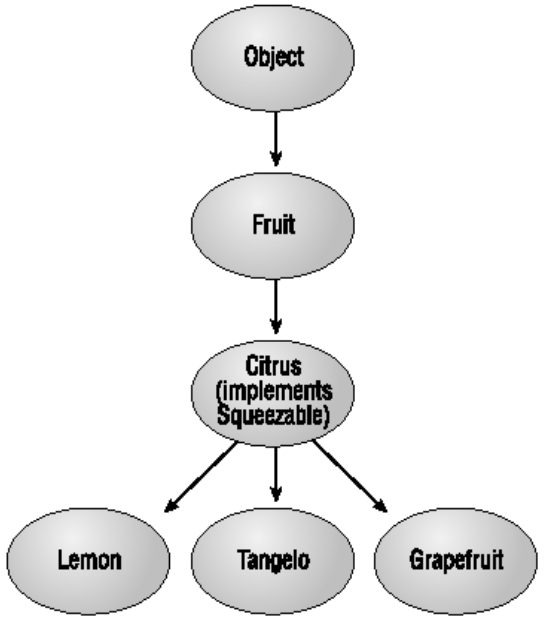
```
Lemon lem = new Lemon();
Citrus cit = lem; // No problem
```

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.

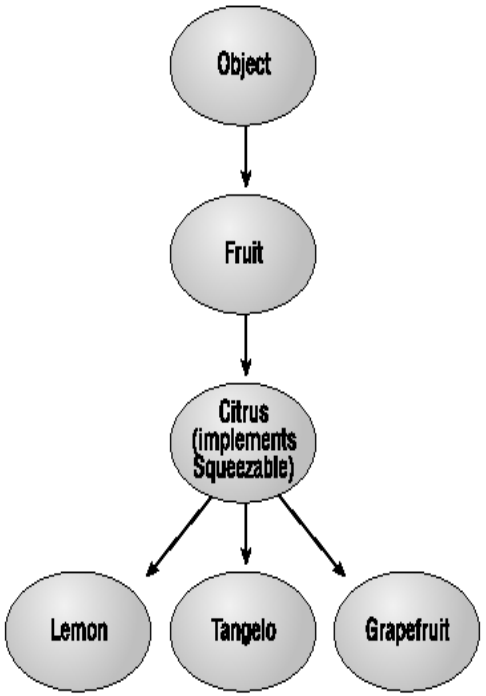
A simple class hierarchy



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



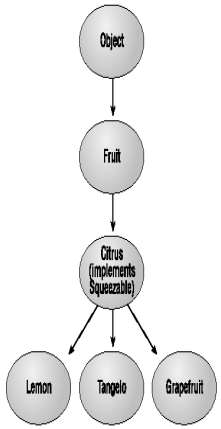
- Kompile → ok, kompiler tidak bisa mengetahui pegang oleh c.
- Runtime → error → class c adalah Grapefruit

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)
```

A simple class hierarchy



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

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

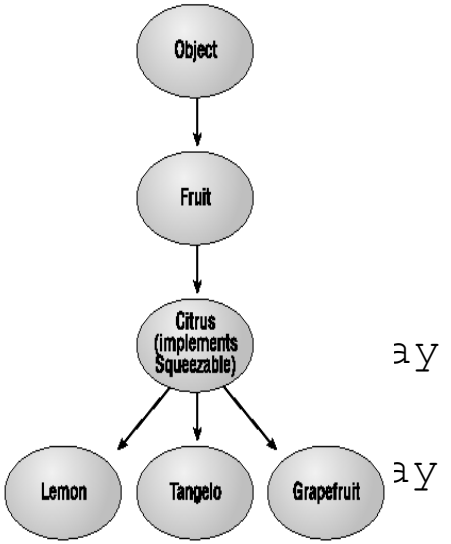
- Pada saat runtime terjadi pengecekan.

Object Reference Casting

Example: array.

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

A simple class hierarchy



Apa itu Metode Final?

- Metode final merupakan prosedur dan fungsi yang tidak dapat lagi di-override pada kelas turunannya.
- Untuk membuat sebuah prosedur dan fungsi final, dapat menggunakan kata kunci **final**.
- Contoh:

```
public class NamaKelas {

    public final void prosedurFinal(){
    }

    public final TipeData fungsiFinal(){
          return hasil;
    }

}
```

Apa itu Kelas Final?

- Kelas Final merupakan kelas yang tidak dapat diturunkan oleh kelas lain.
- Untuk membuat sebuah kelas menjadi final, dapat menggunakan kata kunci **final**.
- Contoh:

```
public final class NamaKelas {
    // isi kelas
}
```

Apa itu Kelas Abstrak?

- Kelas Abstrak merupakan kelas yang tidak dapat diinstansiasi menjadi Objek.
- Kelas Abstrak hanya bisa diwarisi oleh kelas lain.
- Untuk membuat kelas abstrak dapat menggunakan kata kunci **abstract**.
- Contoh:

```
public abstract class KelasAbstrak{
   // isi kelas
}
```

Apa itu Metode Abstrak?

- Metode abstrak merupakan prosedur atau fungsi yang belum dideklarasikan.
- Metode abstrak harus dideklarasikan pada kelas turunan.
- Metode abstrak hanya dapat dibuat pada kelas abstrak.
- Contoh:

```
public abstract class KelasAbstrak {
   public abstract void prosedurAbstrak();
   public abstract TipeData fungsiAbstrak();
}
```

Apa itu Kelas Tanpa Nama?

- Kelas tanpa nama merupakan kelas yang dideklarasikan saat pembuatan objek.
- Kelas tanpa nama hanya dapat digunakan sekali dan oleh satu objek.
- Contoh:

```
KelasAbstrak objek = new KelasAbstrak(){
    public void prosedurAbstrak(){
    }
    public TipeData fungsiAbstrak(){
    }
};
```

Apa itu equals()?

- Fungsi equals() merupakan fungsi yang digunakan untuk membandingkan dua buah objek.
- Membandingkan objek tidak dapat dilakukan dengan menggunakan operator perbandingan ==
- Disarankan untuk melakukan override terhadap metode equals()
- Contoh:

```
public class NamaKelas {
    private String id;
    public boolean equals(Object o){
        // pengecekan kesamaan
    }
}
```

Apa itu hashCode()

- hashCode() merupakan fungi yang menghasilkan nilai integer unik untuk tiap objek.
- Fungsi hashCode() erat kaitannya dengan fungsi equals()
- Ketentuan:
  - Jika dua buah objek dibandingkan dengan fungsi equals() menghasilkan nilai **true**, maka hasil fungsi hashCode() untuk kedua objek tersebut **harus sama** pula.
  - Jika dua buah objek dibandingkan dengan fungsi equals() menghasilkan nilai **false**, maka hasil fungsi hashCode() untuk kedua objek tersebut **boleh sama ataupun tidak**.

Contoh:

```
public class NamaKelas {
        private String id;
        public int hashCode(){
                //membuat hashcode
        }
}
```

Apa itu toString()

- Fungsi toString() merupakan fungsi yang menghasilkan nilai String untuk sebuah objek.
- Fungsi toString() akan dipanggil saat kita menuliskan output menggunakan System.out.println();
- Contoh:

```
public class NamaKelas {
    public String toString(){
        // isi fungsi
    }
}
```