Overriding
Overloading
Polymorphism

Topik

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

Overriding

- Subclass yang berusaha memodifikasi tingkah laku yang diwarisi dari superclass.
- Tujuan: subclass memiliki tingkah laku yang lebih spesifik.
- Dilakukan dengan cara mendeklarasikan kembali method milik parent class di subclass.

Overriding

- Deklarasi method pada subclass harus sama dengan yang terdapat di super class. Kesamaan pada:
  - Nama
  - Return type (untuk return type : class A atau merupakan subclass dari class A)
  - Daftar parameter (jumlah, tipe, dan urutan)
- Method pada parent class disebut overriden method
- Method pada subclass disebut overriding method.

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");
        }
}
```

Aturan Overriding

- 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 (dijelaskan pada materi Exception).

Overloading

- Menuliskan kembali method dengan nama yang sama pada suatu class.
- Tujuan : memudahkan penggunaan/pemanggilan method dengan fungsionalitas yang mirip.

Aturan Pendeklarasian 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 Pada Overloading

- 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.

Contoh Overloading

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

Contoh

```
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) {
        ...
    }
}
```

return type	nama method	daftar parameter
void	Gambar	(int t1)
void	Gambar	(int t1, int t2)
void	Gambar	(int t1, int t2, int t3)
void	Gambar	(int t1, int t2, int t3, int t4)
↓	↓	↓
sama	sama	berbeda

- 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.

```
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  }
```

Mengenal Var Args (Variable Argument Lists)

Method dengan var args

- Jika terdapat beberapa parameter dalam method, maka var args harus menjadi parameter terakhir.
- Dan hanya boleh ada satu var args dalam sebuah method

Legal:

```
void doStuff(int... x) { }  // expects from 0 to many ints
                            // as parameters
void doStuff2(char c, int... x)  { }  // expects first a char,
                                      // then 0 to many ints
void doStuff3(Animal... animal) { }   // 0 to many Animals
```

Illegal:

```
void doStuff4(int x...) { }              // bad syntax
void doStuff5(int... x, char... y) { }   // too many var-args
void doStuff6(String... s, byte b) { }   // var-arg must be last
```

Overloading

- Ada 3 faktor yang perlu diperhatikan dalam overloading
  - *Widening conversion* adalah merubah tipe data suatu variabel ke tipe data yang ukuran bit nya lebih besar dari aslinya.
  - Autoboxing
  - Var-args

Overloading

```
class EasyOver {
  static void go(int x) { System.out.print("int "); }
  static void go(long x) { System.out.print("long "); }
  static void go(double x) { System.out.print("double "); }

  public static void main(String [] args) {
    byte b = 5;
    short s = 5;
    long l = 5;
    float f = 5. 0f;

    go(b);
    go(s);
    go(l);
    go(f);
  }
}
```

Overloading

- Output

```
int int long double
```

Overloading

```
class AddBoxing {
  static void go(Integer x) { System.out.println("Integer"); }
  static void go(long x) { System.out.println("long"); }

  public static void main(String [] args) {
    int i = 5;
    go(i);            // which go() will be invoked?
  }
}
```

- Bagaimana outputnya ?
- Kompiler akan mendahulukan widening coversion daripada autoboxing
- Output : long

Overloading

```
class AddVarargs {
  static void go(int x, int y) { System.out.println("int,int");}
  static void go(byte... x) { System.out.println("byte ... "); }
  public static void main(String[] args) {
    byte b = 5;
    go(b,b);          // which go() will be invoked?
  }
}
```

- Kompiler akan mendahulukan model lama dibandingkan dengan model baru sehingga :
  - Widening didahulukan dibandingkan boxing
  - Widening didahulukan dibandingkan var args

- Output : int,int

Overloading

```
class BoxOrVararg {
  static void go(Byte x, Byte y)
                  { System.out.println("Byte, Byte"); }
  static void go(byte... x) { System.out.println("byte... "); }

  public static void main(String [] args) {
    byte b = 5;
    go(b,b);          // which go() will be invoked?
  }
}
```

- Output :
- Byte, Byte

Overloading dengan menggabungkan Widening and Boxing

```
class WidenAndBox {
  static void go(Long x) { System.out.println("Long"); }

  public static void main(String [] args) {
    byte b = 5;
    go(b);            // must widen then box - illegal
  }
}
```

- Output

```
WidenAndBox.java:6: go(java.lang.Long) in WidenAndBox cannot be
applied to (byte)
```

- Kenapa ?
- Proses widening hanya berlaku untuk tipe data primitif

Overloading dengan menggabungkan Widening and Boxing

```
class BoxAndWiden {
  static void go(Object o) {
    Byte b2 = (Byte) o;          // ok - it's a Byte object
    System.out.println(b2);
  }

  public static void main(String [] args) {
    byte b = 5;
    go(b);        // can this byte turn into an Object ?
  }
}
```

- Output:
- 5

Overloading digabungkan dengan var args

```
class Vararg {
  static void wide_vararg(long... x)
              { System.out.println("long..."); }
  static void box_vararg(Integer ... x)
              { System.out.println("Integer..."); }
  public static void main(String [] args) {
    int i = 5;
    wide_vararg(5,5);     // needs to widen and use var-args
    box_vararg(5,5);      // needs to box and use var-args
  }
}
```

- Output:

```
long...
Integer...
```

Aturan Overloading

- Primitive widening conversion didahulukan dalam overloading dibandingkan boxing dan var args
- Kita tidak dapat melakukan proses widening dari tipe wrapper ke tipe wrapper lainnya (mengubah Integer ke Long)
- Kita tidak dapat melakukan proses widening dilanjutkan boxing (dari int menjadi Long)
- Kita dapat melakukan boxing dilanjutkan dengan widening (int dapat menjadi Object melalui Integer)
- Kita dapat menggabungkan var args dengan salah satu yaitu widening atau boxing

Polymorphism

Polymorphism

- Polymorphism adalah kemampuan untuk mempunyai beberapa bentuk yang berbeda.
- Polimorfisme ini terjadi pada saat suatu obyek bertipe parent class, akan tetapi pemanggilan constructornya melalui subclassnya.

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;
}
```

Contoh

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

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

Polymorphism: ingat !!

- Satu obyek hanya boleh mempunyai satu bentuk saja.
- Yaitu bentuk yang diberikan ketika obyek dibuat.
- Reference variabel bisa menunjuk ke bentuk yang berbeda.

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.
- Virtual Method Invocation (Pemanggilan Method secara Virtual)

Contoh Virtual Method Invocation

```
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

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 menjalankan konstruktor Manager (subclass).

```
public class A {
    int x = 5 ;

     public A() {
          System.out.println("Konstruktor A");
     }

    public void fungsiA(){
          System.out.println("FungsiA Class A");
    };
}
```

```
class B extends A{
    int x = 10 ;

    public B() {
        System.out.println("Konstruktor B");
    }

    public void fungsiA(){
        System.out.println("FungsiA Class B");
    };
    public static void main(String args[]){
        A a = new B();
        System.out.println(a.x);
        a.fungsiA();
    }
}
```

```
Konstruktor A
Konstruktor B
5
FungsiA Class B
```

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`**.

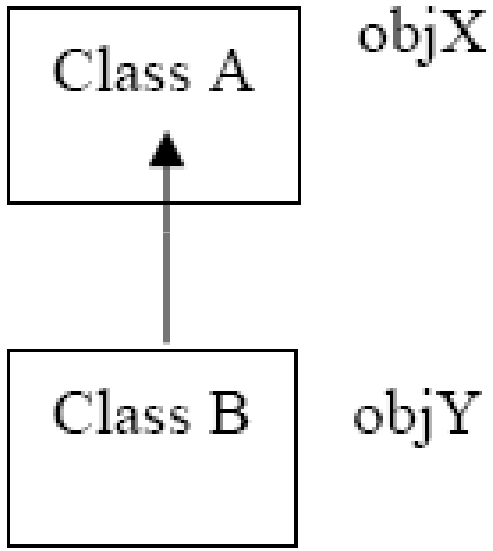
Macam-macam Polymorphism

- Polymorphic assignment statements
- Polymorphic Argument
- Polymorphic return types

Polymorphic assignment statements Contoh 1

```
public class ClassA
{
}// end ClassA
```

```
public class ClassB extends ClassA
{
}// end ClassB
```



Polymorphic assignment statements Contoh 1

```
public class PolymorphicAssignment
{
public static void main(String [] args)
{
    ClassA obj1 = new ClassA();
    ClassA obj2 = new ClassA();
    ClassB obj3 = new ClassB();
    1) obj1 = obj2; // no problem here...same data types
    2) obj1 = obj3; // obj3 is a type of ClassA…ok
    3) //obj3 = obj2; // "incompatible types" compile message
    4) //obj3 = obj1; // still incompatible as the obj3 value
                        // stored in obj1 (see line 2 above)
                         // has lost its ClassB identity
```

Polymorphic assignment statements Contoh 1

```
5) obj3 = (ClassB)obj1; // the ClassB identity of the object
                        // referenced by obj1 has been retrieved!
                        // This is called "downcasting"
6) obj3 = (ClassB)obj2; // 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
```

Polymorphic assignment statements Contoh 1

- Di contoh program sebelumnya. Object obj1 dan obj2 mempunyai tipe Class A. Sehingga pada baris 1 proses assigment berhasil karena mempunyai tipe yang sama.
- Pada baris 2 berhasil karena obj3 adalah object dengan tipe class B, sedangkan class B merupakan subclass dari class A. Proses assigment berhasil karena mengkopikan object dari class B ke class A.
- **Baris ke 3 tidak bisa dicompile karena menkopikan object class A ke object class B. Object dari parent class dikopikan ke subclass.**
- **Line 4 is more complicated. We know from line 2 that obj1 actually does reference a ClassB value. However, that ClassB information is now no longer accessible as it is stored in a ClassA object reference variable. Line 5 restores the ClassB class identity before the assignment to ClassB object reference variable obj3 with a type cast. Life is good again. Line 6 is syntactically equivalent to line 5 and will actually compile because of it, but will result in a "ClassCastException" at run time because obj2 never was ClassB data to begin with.**

Polymorphic assignment statements Contoh 2

```
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
```

```
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
```

Area = 25 (area triangle)
or Area = 50 (area rect)

Polymorphic assignment statements Contoh 2

- Ini adalah contoh run-time polymorphism. JVM tidak mengetahui nilai dari variabel “shp” pada saat compile. Pada saat runtime, JVM memilih method area() yang sesuai dengan object dari shp.

Polymorphic Arguments

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

Polymorphic Arguments Contoh 1

Because a Manager is an Employee:

```
// In the Employee class
public TaxRate findTaxRate(Employee e) {
}
// Meanwhile, elsewhere in the application class
Manager m = new Manager();
:
TaxRate t = findTaxRate(m);
```

```
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

Operator 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.

```
...
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

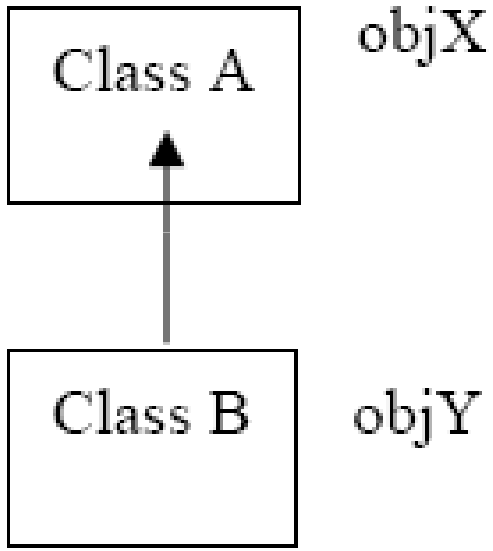
```
...
public class Tes {
    public static void ProsesManajer() {
        ...lakukan tugas-tugas manajer...
    }

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

Polymorphic Arguments
Contoh 2

```
public class ClassA
{
}// end ClassA
```

```
public class ClassB extends ClassA
{
}// end ClassB
```



Polymorphic Arguments Contoh 2

```
public class PolymorphicParameterPassing
{
public static void main(String [] args)
{
      ClassA obj1 = new ClassA();
      ClassA obj2 = new ClassA();
      ClassB obj3 = new ClassB();
      1) method1(obj1);
      2) method1(obj3);
      3) //method2(obj1);
      4) obj1 = obj3;
      5) //method2(obj1);
      6) method2((ClassB) obj1);
      7) // method2((ClassB) obj2);
} // end main
public static void method1(ClassA formal) {}
public static void method2(ClassB formal) {}
}// end class
```

Penjelasan

- In **line 1, at left, an object reference variable of ClassA type is passed to method1 and received as a ClassA object reference variable. Actual and formal parameter types are the same. Life is good! Line 2 shows a ClassB object reference variable passed to and received as a ClassA type variable. This is okay, as a ClassB type variable "is-a" type of ClassA variable. Line 3 fails, as you are passing a superclass type variable to be received as a subclass type. It seems as though line 5 should work, as obj1 received the value of a ClassB variable, but it doesn't work unless the ClassB identity is restored through a type cast as shown in line 6. Line 7 will compile, as it is syntactically the same as line 6, but line 7 will result in a "type cast exception" upon program execution.**

Polymorphic assignment statements Contoh 3

```
import java.util.Random;
public class PolyParam
{
        public static void main(String [] args)
        {
            Shape shp;
            Shape tri  = new Triangle();
            Shape rect = new Rectangle();
            Random r = new Random();
            int flip = r.nextInt(2);
            if (flip == 0)
               shp = tri;
            else
               shp = rect;

            printArea(shp);   // output will vary
        }  // end main

        public static void printArea(Shape s)
        {
                System.out.println("area = " + s.area(5, 10));
        }// end printArea()
}  //    end class
```

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

public class Triangle extends Shape
{
        public double area(int x, int y)
        {
           return 0.5 * x * y;
        }

}// end Triangle

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

Polymorphic Return Types Contoh 1

```
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
```

Polymorphic Return Types Contoh 2

```
import java.util.Random;

public class PolyReturn
{
        public static void main(String [] args)
        {
            Shape shp = retMethod();
            System.out.println(shp.area(5, 10));
        }  // end main

        public static Shape retMethod()
        {
                Random r = new Random();
                int flip = r.nextInt(2);
                if (flip == 0)
                    return new Triangle();
                else
                    return new Rectangle();
        }// end retMethod()
}// end class
```

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
```

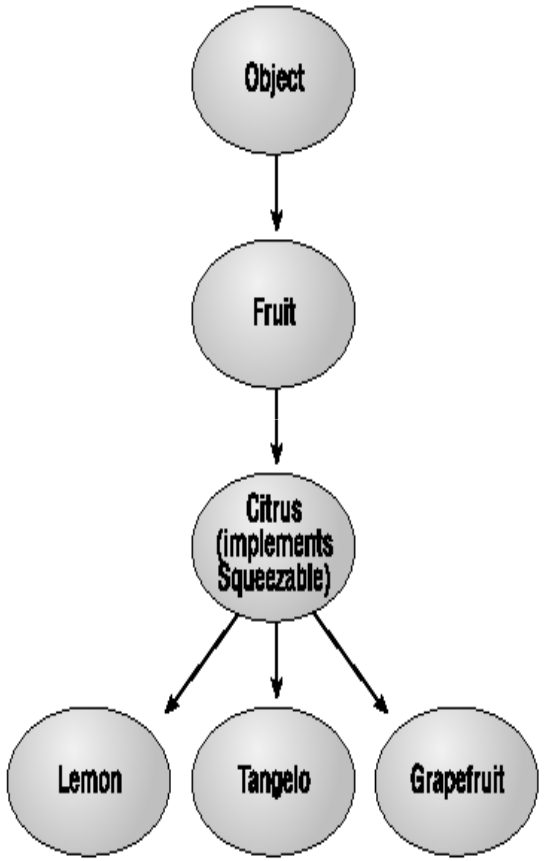
Contoh 2:

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

Contoh 3:

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

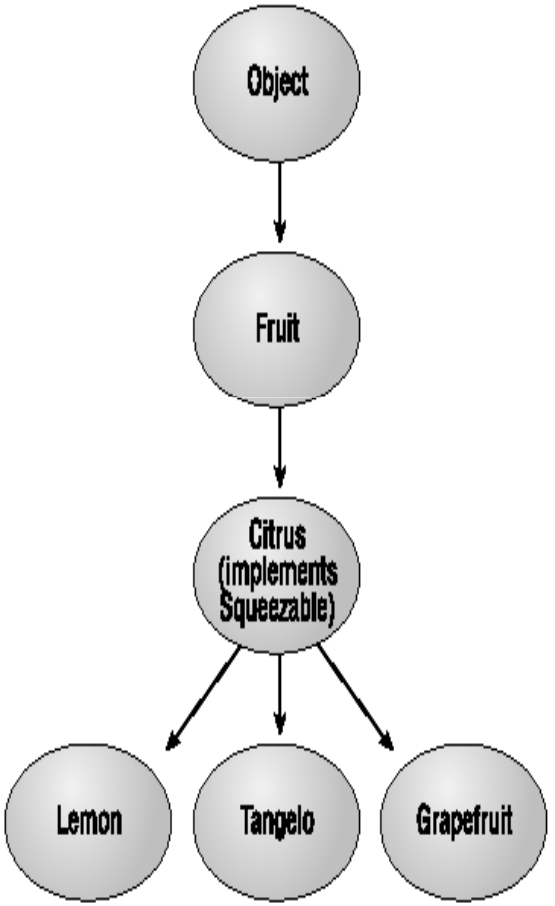
A simple class hierarchy



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.

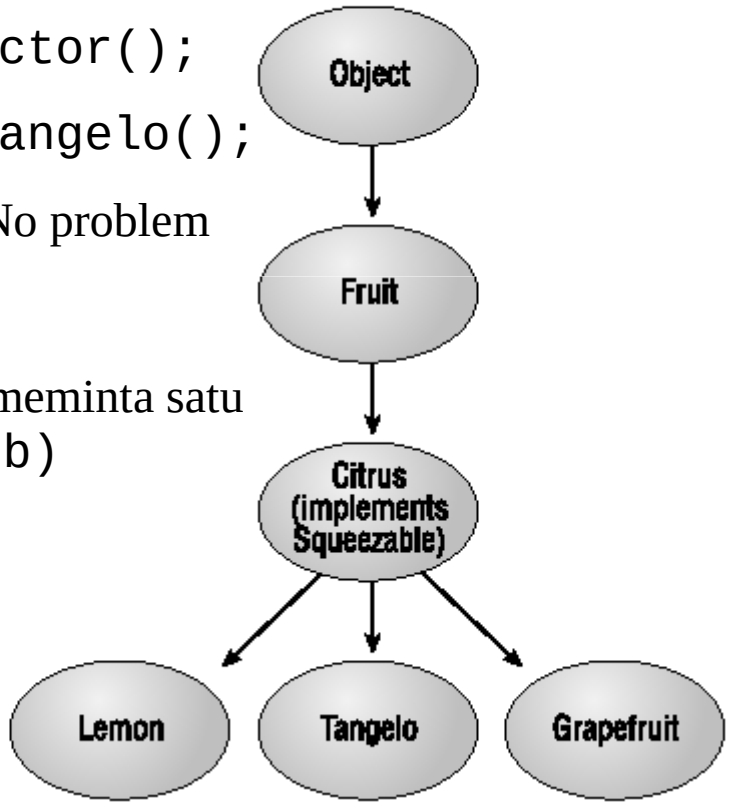
Object Method-Call Conversion

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:

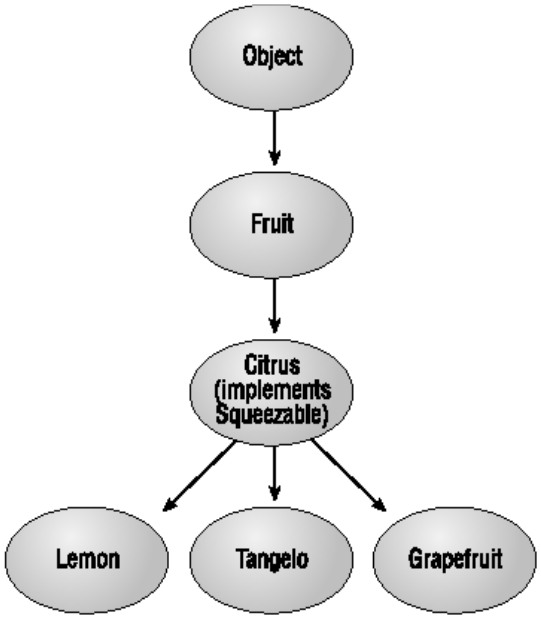
```
Lemon lem = new Lemon();
Citrus cit = lem; // No problem
```

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.

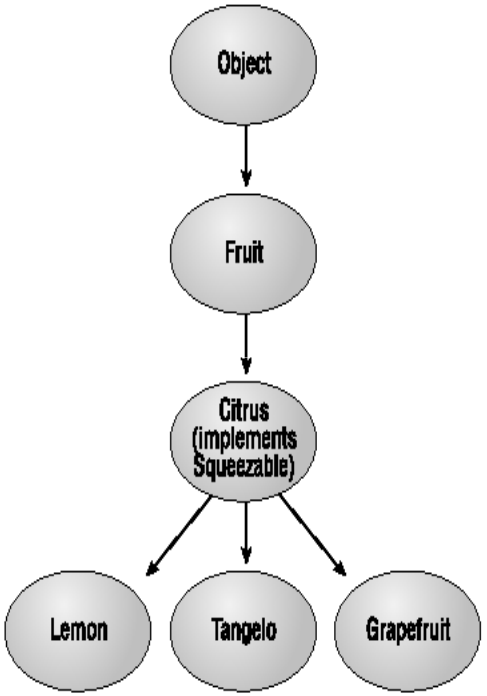
A simple class hierarchy



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

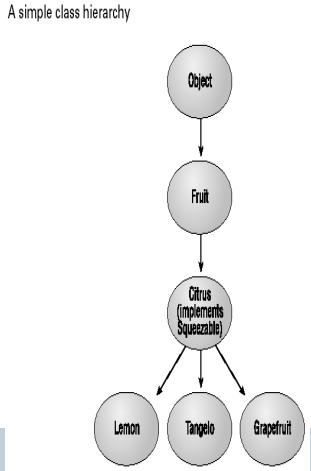
Example: Object is cast to an interface type.

```
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

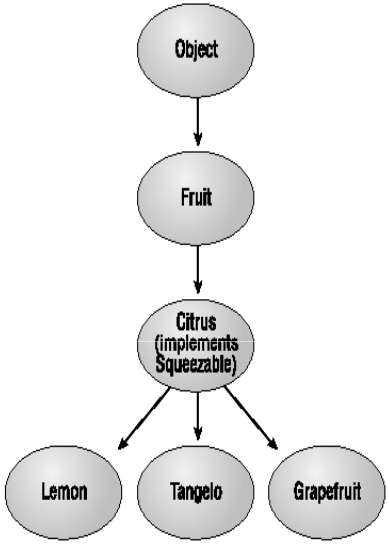
```
g1 = (Grapefruit) s;
```

- Pada saat runtime terjadi pengecekan.



Object Reference Casting

A simple class hierarchy



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)
```