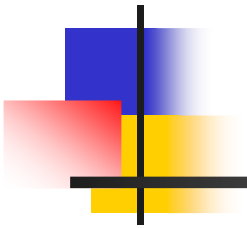
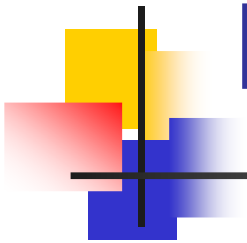


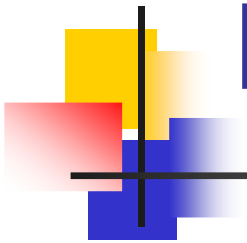
Generics

Riza Satria Perdana
IF 2210 Pemrograman Berorientasi Objek
STEI – ITB



Pengantar

- Pada proyek pengembangan perangkat lunak sering muncul bug. Dengan perencanaan, programming, dan testing yang baik akan mereduksi munculnya bug.
- Ada bug yang lebih mudah dideteksi yaitu compile-time bug (dibandingkan run-time bug)



Pengantar

- Dengan menggunakan konsep Generik akan menambah stabilitas kode dengan membuat bug terdeteksi saat kompilasi

Contoh kasus: Simple Box class

```
public class Box {

    private Object object;

    public void add(Object object) {
        this.object = object;
    }

    public Object get() {
        return object;
    }
}
```

```
public class BoxDemo1 {

    public static void main(String[] args) {

        // ONLY place Integer objects into this box!
        Box integerBox = new Box();

        integerBox.add(new Integer(10));
        Integer someInteger = (Integer)integerBox.get();
        System.out.println(someInteger);
    }
}
```

Contoh kasus: Simple Box class

```
public class BoxDemo2 {

    public static void main(String[] args) {

        // ONLY place Integer objects into this box!
        Box integerBox = new Box();

        // Imagine this is one part of a large application
        // modified by one programmer.
        integerBox.add("10"); // note how the type is now String

        // ... and this is another, perhaps written
        // by a different programmer
        Integer someInteger = (Integer)integerBox.get();
        System.out.println(someInteger);
    }
}
```

```
Exception in thread "main"
     java.lang.ClassCastException:
        java.lang.String cannot be cast to java.lang.Integer
        at BoxDemo2.main(BoxDemo2.java:6)
```

Generic Type

- Generic type declaration

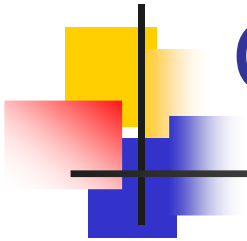
```
/**
 * Generic version of the Box class.
 */
public class Box<T> {

    private T t; // T stands for "Type"

    public void add(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }
}
```

Generic Type

- Generic type invocation

```
public class BoxDemo3 {

    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.add(new Integer(10));
        Integer someInteger = integerBox.get(); // no cast!
        System.out.println(someInteger);
    }
}
```

```
BoxDemo3.java:5: add(java.lang.Integer) in Box<java.lang.Integer>
cannot be applied to (java.lang.String)
    integerBox.add("10");
               ^
1 error
```

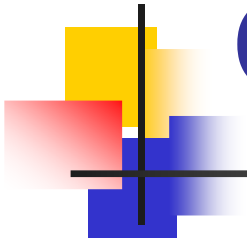


Generic Methods and Constructors

- Type parameter dapat juga digunakan pada method dan konstruktor menjadi *generic methods* dan *generic constructors*

Contoh Generic Methods

```
public class Box<T> {

    private T t;

    public void add(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }

    public <U> void inspect(U u){
        System.out.println("T: " + t.getClass().getName());
        System.out.println("U: " + u.getClass().getName());
    }

    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.add(new Integer(10));
        integerBox.inspect("some text");
    }
}
```



Contoh Type Inference

```
public static <U> void fillBoxes(U u, List<Box<U>> boxes) {
    for (Box<U> box : boxes) {
        box.add(u);
    }
}


Crayon red = ...;
List<Box<Crayon>> crayonBoxes = ...;

Box.<Crayon>fillBoxes(red, crayonBoxes);

Box.fillBoxes(red, crayonBoxes); // compiler infers that U is Crayon
```

Bounded Type Parameters

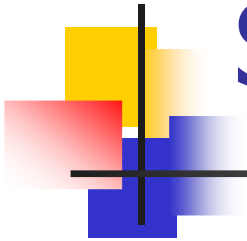
- Ada kebutuhan untuk membatasi tipe-tipe apa saja yang diberbolehkan untuk masuk sebagai parameter
- Sebagai contoh mengharapkan hanya tipe angka (turunan Number) yang boleh

Contoh Bounded Type Parameters

```
public <U extends Number> void inspect(U u){
    System.out.println("T: " + t.getClass().getName());
    System.out.println("U: " + u.getClass().getName());
}

public static void main(String[] args) {
    Box<Integer> integerBox = new Box<Integer>();
    integerBox.add(new Integer(10));
    integerBox.inspect("some text"); // error: this is still String!
}
```

```
Box.java:21: <U>inspect(U) in Box<java.lang.Integer> cannot
  be applied to (java.lang.String)
                        integerBox.inspect("10");
                                   ^
1 error
```

```
<U extends Number & MyInterface>
```



Subtyping

- Dimungkinkan meng-assign sebuah objek bertipe tertentu ke reference bertipe lain yang kompatibel

```
Object someObject = new Object();
Integer someInteger = new Integer(10);
someObject = someInteger; // OK
```

- Pada OOP disebut relasi “is a”. Integer adalah “is a kind of” Object

Subtyping

- Kode berikut juga valid

```
public void someMethod(Number n){
    // method body omitted
}

someMethod(new Integer(10)); // OK
someMethod(new Double(10.1)); // OK
```

- Hal yang sama juga bisa dilakukan pada tipe generik

```
Box<Number> box = new Box<Number>();
box.add(new Integer(10)); // OK
box.add(new Double(10.1)); // OK
```

Subtyping

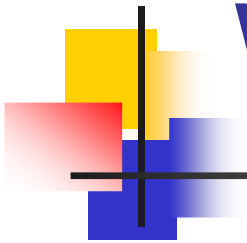
- Perhatikan kode berikut

```
public void boxTest(Box<Number> n){
    // method body omitted
}
```

- Tipe parameter apa saja yang bisa digunakan? Apakah Box<Integer> atau Box<Double> boleh?

Subtyping

```
// A cage is a collection of things, with bars to keep them in.
interface Cage<E> extends Collection<E>;

interface Lion extends Animal {}
Lion king = ...;

Animal a = king;

Cage<Lion> lionCage = ...;
lionCage.add(king);

interface Butterfly extends Animal {}
Butterfly monarch = ...;
Cage<Butterfly> butterflyCage = ...;
butterflyCage.add(monarch);

Cage<Animal> animalCage = ...;

animalCage.add(king);
animalCage.add(monarch);

animalCage = lionCage;  // compile-time error
animalCage = butterflyCage; // compile-time error
```



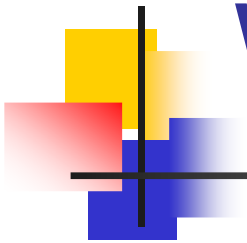
Wildcard

- Dimungkinkan kode berikut

```
Cage<? extends Animal> someCage = ...;
```

- Disebut *bounded* wilcard yang dalam kasus ini *upper bound*

```
someCage = lionCage; // OK
someCage = butterflyCage; // OK

someCage.add(king);     // compiler-time error
someCage.add(monarch);  // compiler-time error
```



Wildcard

- Kode berikut bisa digunakan

```
void feedAnimals(Cage<? extends Animal> someCage) {
    for (Animal a : someCage)
        a.feedMe();
}

feedAnimals(lionCage);
feedAnimals(butterflyCage);

feedAnimals(animalCage);
```

Type Erasure

- Bila tipe generik digunakan maka kompilator akan membuang semua informasi yang berhubungan dengan tipe parameter dalam kelas atau method

```
public class MyClass<E> {
    public static void myMethod(Object item) {
        if (item instanceof E) {  //Compiler error
            ...
        }
        E item2 = new E();   //Compiler error
        E[] iArray = new E[10]; //Compiler error
        E obj = (E)new Object(); //Unchecked cast warning
    }
}
```