IKI10400 • Struktur Data & Algoritma: Generics

Fakultas Ilmu Komputer • Universitas Indonesia

Slide acknowledgments:
Suryana Setiawan, Ade Azurat, Denny, Ruli Manurung

Inheritance vs Generics

- Tujuan dari paradigma OOP -> ***code reuse***
- ***Generic mechanism:***
  - Apabila sebuah implementasi bersifat **identik kecuali pada tipe** dari objek, maka sebuah implementasi general dapat digunakan untuk mendeskripsikan fungsi dasar yang dilakukan.
- Contoh:
  - Sebuah method yang dibuat untuk melakukan pengurutan terhadap suatu array yang berisi beberapa items. Mekanisme pengurutan bersifat independen terhadap tipe data, maka sebuah ***generic method*** dapat digunakan.

Motivasi

- **Generics** mengimplementasikan konsep parameterisasi tipe.
  - Kita biasanya menggunakan parameter pada method melalui nilai dari argumennya. Sekarang kita tidak hanya dapat melakukan parameterisasi pada nilai, tapi juga pada tipenya (class).
- Menyediakan suatu *type safe container* yang mencegah terjadinya runtime error pada collections.
- Sehingga dapat digunakan untuk membuat program yang lebih general yang dapat di *re-used* (*reusable and expressiveness*)

Generics

- Mulai digunakan setelah Java 5.0
- Digunakan untuk membatasi tipe elemen dari sebuah *collection*.
- Dapat digunakan pada tipe data, class, dan method yang kita definisikan.
- Contoh generics:
  - **Collections ->** *generic class* yang dapat menerima berbagai tipe parameter.

Generics (lanj.)

- Andaikan kita memiliki class **Employee** yang meng-**extends** class **Person**.
- Apakah **Employee[]** "adalah" **Person[]**? Di Java, iya.
- Sekarang, andai kita memiliki class **Student** yang meng-**extends Person,**

  Person[] arr = new Employee[5];

  arr[0] = new Student(…);

  akan menyebabkan sebuah *runtime* **ArrayStoreException**.
- Java generics didesain untuk mencegah runtime errors demikian, sehingga **List<Employee>** adalah BUKAN **List<Person>**
- Bagaimana kita dapat membuat method yang menerima keduanya?

Wildcards

- Di Java 5, kita menggunakan wildcards untuk merepresentasikan sub/superclasses dari parameter types
- <? extends X> berarti *class* manapun yang "merupakan sebuah" X.
- Method yang akan gagal apabila menerima sebuah List<Employee>

```
void printNames(List<Person> arr)
{
    for(Person p : arr) {
        System.out.println("Name is " + p.getName());
    }
}
```

Wildcards

- Method yang akan sukses apabila menerima sebuah List<Employee>

```
void printNames(List<? extends Person> arr)
{
    for(Person p : arr) {
        System.out.println("Name is " + p.getName());
    }
}
```

Wildcards

- Ada pula <? super X>, yang artinya X atau class yang merupakan superclass dari X
- Contoh penggunaan:
  - lihat: TreeSet (Comparator<? super E> comparator)
    - Comparator<? super Employee> digunakan untuk membandingkan List<Employee>
    - mengapa tidak menggunakan
      - Comparator<Employee> ??
      - Comparator<? extends Employee> ??

Generic classes

- Kita dapat mendefinisikan sendiri sebuah *generic class* di Java.
- Deklarasi dari class terdiri dari *satu atau lebih tipe parameter* setelah nama dari class.
- Tipe generic dapat digunakan dimanapun pada definisi class

```
public class MemoryCell
{
    public Object read()
    {   return storedValue; }

    public void write(Object x)
    {   storedValue = x; }

    private Object storedValue;
}
```

Contoh kelas untuk menyimpan/mengakses object

Generic classes

- Bagaimana implementasi dengan menggunakan generic?

```
public class MemoryCell<AnyType>
{
    public AnyType read()
    {   return storedValue; }

    public void write(AnyType x)
    {   storedValue = x; }

    private AnyType storedValue;
}
```

Generic interfaces

- Interface juga dapat dideklarasikan sebagai generic.
- Dikutip dari `java.util`:

```
public interface Iterator<E>
{
        E next();
        boolean hasNext();
}
```

Generics methods (and type bounds)

- Kita dapat pula mendefinisikan sebuah *generic methods*. Tipe parameter diberikan sebelum nama parameter.

```
boolean <T extends Person> findNamed(List<T> arr, T x)
{
        for (Person p : arr)
                if(x.equals(p))
                        return true;
        return false;
}
```

Kita dapat melakukan:

```
List<Student> class = ….
Student s = new Student(…)
findNamed(class, s);
```

Selain itu:

```
List<Employee> office = …
Employee e = new Employee(…)
findNamed(office, e);
```

Perhatikan bahwa kita tidak dapat mengirimkan sebuah List dari **`Fruits`** atau **`Animals`** -> kita telah mendefinisikan sebuah **type bound**.

Collections interface

- **`Interface Collection` adalah interface utama yang menetapkan operasi-operasi dasar, antara lain:**

```
int size();
boolean isEmpty();
boolean contains(Object element);
boolean add(E element);
boolean remove(Object element);
Iterator<E> iterator();
.....
```

- ***E*** **menyatakan tipe parameter.**

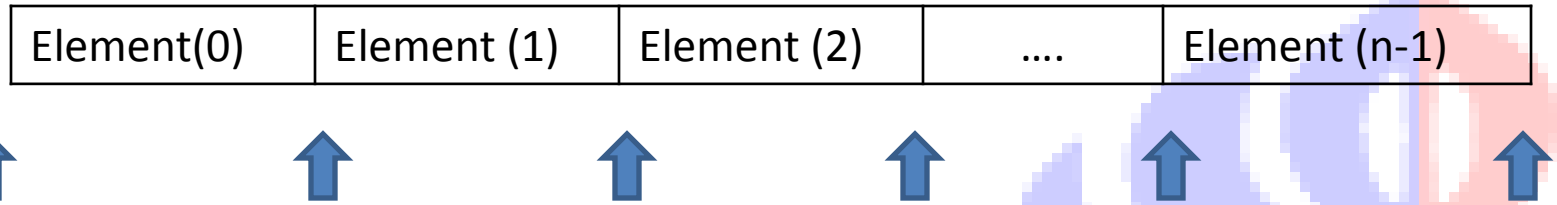
Iterator

- Setiap **collection** telah mendefinisikan implementasi dari masing-masing interface **Iterator**-nya pada package **java.util.**
- **Iterator** interface pada **Collections API** hanya terdiri dari tiga buah method.

```
boolean hasNext();
AnyType hasNext();
void remove();
```

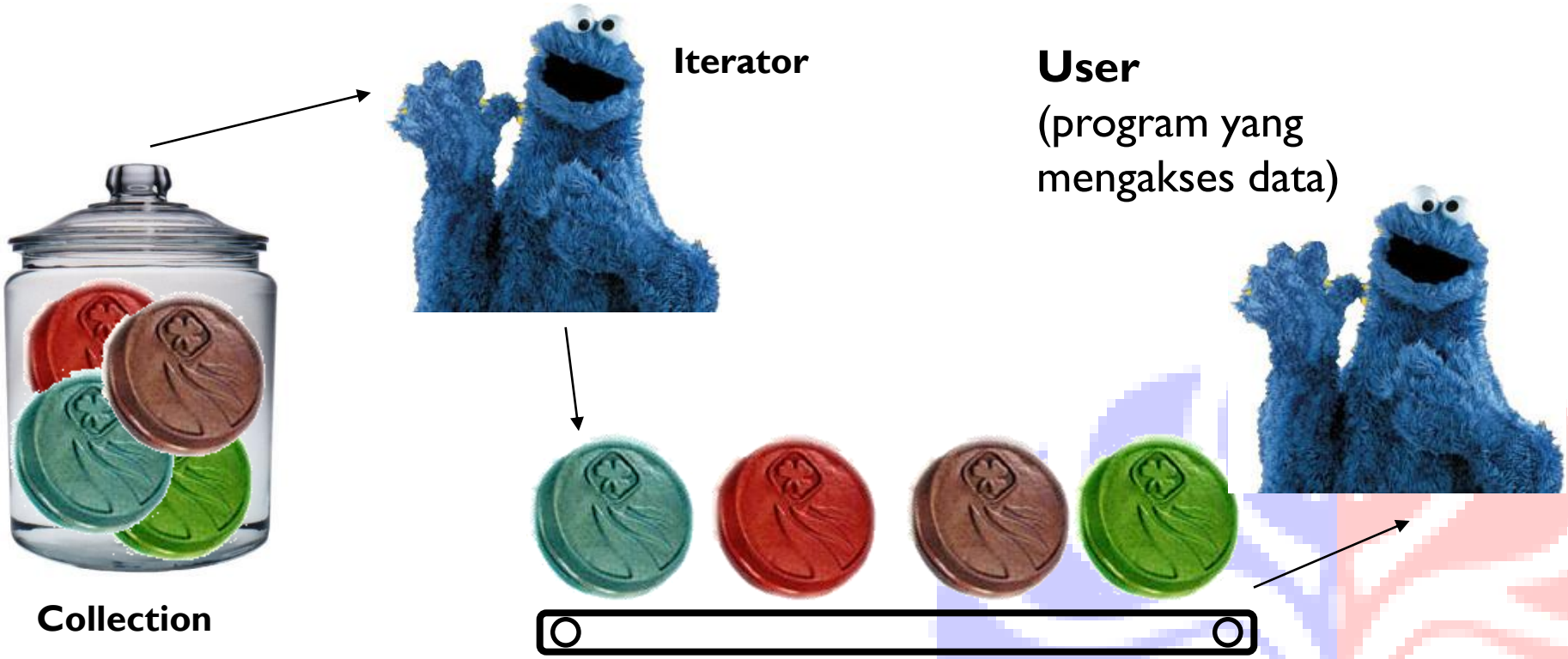
ListIterator

- Merupakan iterator untuk *list,* dengan super interface adalah Iterator.
- Posisi *cursor* akan selalu berada diantara dua elemen.



- Memiliki akses ke elemen sesudah (*next*) dan sebelum (*previous*)

Ilustrasi: Iterator



Contoh penggunaan Iterator: List

```
ArrayList<String> lst = new ArrayList<String>();
lst.add(new String("Hello"));
lst.add(new String("World"));

ListIterator<String> itr = lst.listIterator();

while (itr.hasNext()) {
    System.out.println(itr.next());
}
```

Apa yang dilakukan oleh program tersebut?

Latihan

- Apakah output dari program ini?

```
ArrayList<Integer> list1 = new ArrayList<Integer>();
list1.add(4);
list1.add(new Integer(5));
Object list = list1;
ArrayList<String> list2 = new ArrayList<String>();
String s = list2.get(0);
```

Latihan

- Apakah output dari `java StackTester this is a funny little test`?

```
import java.util.Stack;
public class StackTester
{
    public static void main(String[] args)
    {
        Stack<String> myS = new Stack<String>();
        for(int ii=0; ii<args.length; ii++) {
            myS.push(args[ii]);
        }
        while(!myS.isEmpty()) {
            String myString = myS.pop();
            System.out.println(myString);
        }
    }
}
```

Latihan

- Apakah output dari `java QueueTester this is a funny little test`?

```
import java.util.LinkedList;
import java.util.Queue;
public class QueueTester
{
    public static void main(String[] args)
    {
        Queue<String> myQ = new LinkedList<String>();
        for(int ii=0; ii<args.length; ii++) {
            myQ.offer(args[ii]);
        }
        while(!myQ.isEmpty()) {
            String myString = myQ.remove();
            System.out.println(myString);
        }
    }
}
```

Latihan

- Buatlah sebuah implementasi method `printReversed(Collection c)` yang menggunakan **Collections API** untuk mengeluarkan elemen dari suatu **Collection** dalam urutan terbalik. Jangan menggunakan **ListIterator**!

  Hint: Anda harus menentukan jenis Collection apa yang cocok digunakan untuk problem ini.