GERERICS & MORE

20TH LECTURE – SUPPLEMENT

엄현상(Eom, Hyeonsang)
School of Computer Science and Engineering
Seoul National University

Outline

- Generic Interface
- Generic Method
- Anonymous Inner Classes
  - Inner classes
- Adapting to an Interface
- Q&A

Generic Interface

```
interface Generator<T> { T next(); }
public class Fibonacci implements Generator<Integer> {
private int count = 0;
public Integer next() { return fib(count++); }
private int fib(int n) {
    if(n < 2) return 1;
    return fib(n-2) + fib(n-1);
}
public static void main(String[] args) {
    Fibonacci gen = new Fibonacci();
    for(int i = 0; i < 18; i++)
        System.out.print(gen.next() + " ");
}
}
```

generator class 정의 후에
fib method parameter 등등 main에서 gen.next()
통해서 call 하고 collection에 추가하는 행위 일어남.

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584
```

Generic Method

```
import java.util.*;
interface Generator<T> { T next(); }
public class Generators {
public static <T> Collection<T>
    fill(Collection<T> coll, Generator<T> gen, int n) {
    for(int i = 0; i < n; i++)
        coll.add(gen.next());
    return coll;
}
public static void main(String[] args) {
Collection<Integer> fnumbers = fill(
    new ArrayList<Integer>(), new Fibonacci(), 12);
    for(int i : fnumbers)
        System.out.print(i + ", ");
}
}
```

```
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
```

Anonymous Inner Classes

```
import java.util.*;
interface Generator<T> { T next(); }
class Customer {
private static long counter = 1;
private final long id = counter++;
private Customer() {}
public String toString() { return "Customer " + id; }
// A method to produce Generator objects:
public static Generator<Customer> generator() {
    return new Generator<Customer>() {
        public Customer next() { return new Customer(); }
        };
}
}
class Teller {
private static long counter = 1;
private final long id = counter++;
private Teller() {}
public String toString() { return "Teller " + id; }
```

Anonymous Inner Classes Cont'd

```
// A single Generator object:
public static Generator<Teller> generator =
    new Generator<Teller>() {
        public Teller next() { return new Teller(); }
    };
}
public class BankTeller {
public static void serve(Teller t, Customer c) {
    System.out.println(t + " serves " + c);
}
public static void main(String[] args) {
Random rand = new Random(47);
Queue<Customer> line = new LinkedList<Customer>();
Generators.fill(line, Customer.generator(), 15);
List<Teller> tellers = new ArrayList<Teller>();
Generators.fill(tellers, Teller.generator, 4);
for(Customer c : line)
    serve(tellers.get(rand.nextInt(tellers.size())), c);
}
}
```

annonymous inner class 정의
1)
2) method 내에 또다른 정의가 있는 경우

Anonymous Inner Classes Cont'd

```
Teller 3 serves Customer 1
Teller 2 serves Customer 2
Teller 3 serves Customer 3
Teller 1 serves Customer 4
Teller 1 serves Customer 5
Teller 3 serves Customer 6
Teller 1 serves Customer 7
Teller 2 serves Customer 8
Teller 3 serves Customer 9
Teller 3 serves Customer 10
Teller 2 serves Customer 11
Teller 4 serves Customer 12
Teller 2 serves Customer 13
Teller 1 serves Customer 14
Teller 1 serves Customer 15
```

Inner Classes

```
public class Parcel1 {
class Contents {
    private int i = 11;
    public int value() { return i; }
}
class Destination {
    private String label;
    Destination(String whereTo) {
        label = whereTo;
    }
    String readLabel() { return label; }
}
public void ship(String dest) {
    Contents c = new Contents();
    Destination d = new Destination(dest);
    System.out.println(d.readLabel());
}
public static void main(String[] args) {
    Parcel1 p = new Parcel1();
    p.ship("Tasmania");
}
}
```

inner class 와 anonymous inner class 비교!

Tasmania

Using Inheritance to Generate the Class

```
import java.util.*;
public class IterableFibonacci
    extends Fibonacci implements Iterable<Integer> {
Private int n;
public IterableFibonacci(int count) { n = count; }
public Iterator<Integer> iterator() {
    return new Iterator<Integer>() {
        public boolean hasNext() { return n > 0; }
        public Integer next() {
            n--;
            return IterableFibonacci.this.next();
        }
        public void remove() { // Not implemented
            throw new UnsupportedOperationException();
        }
    };
}
public static void main(String[] args) {
    for(int i : new IterableFibonacci(18))
        System.out.print(i + " ");
}
}
```

이름이 없는 —— 사용해서 iterator 정의
-> annonymous 사용한다는 것

iterator iterable
등등의 사용 예를 알자,...

default 값은 interface 내에 정의
되어 있고, 이것을 재정의 하거나
(override) 혹은 그냥 그대로 쓰거나....??

run time exception 이다.

텍스트

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584
```

Adapting to an Interface

```
import java.nio.*;
import java.util.*;
public class RandomWords implements Readable {
private static Random rand = new Random(47);
private static final char[] capitals =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ".toCharArray();
private static final char[] lowers =
    "abcdefghijklmnopqrstuvwxyz".toCharArray();
private static final char[] vowels = "aeiou".toCharArray();
private int count;
public RandomWords(int count) { this.count = count; }
public int read(CharBuffer cb) {
    if(count-- == 0)
        return -1; // Indicates end of input
    cb.append(capitals[rand.nextInt(capitals.length)]);
    for(int i = 0; i < 4; i++) {
        cb.append(vowels[rand.nextInt(vowels.length)]);
        cb.append(lowers[rand.nextInt(lowers.length)]);
    }
    cb.append(" ");
    return 10; // Number of characters appended
}
```

0- length ()-1

Adapting to an Interface Cont'd

```
public static void main(String[] args) {
Scanner s = new Scanner(new RandomWords(10));
while(s.hasNext())
    System.out.println(s.next());
}
}
```

Readable이라곤 했지만, read 꼭 필요한것도 알겠지만,
어디서 불려지는 거지?

```
Yazeruyac
Fowenucor
Goeazimom
Raeuuacio
Nuoadesiw
Hageaikux
Ruqicibui
Numasetih
Kuuuuozog
Waqizeyoy
```

argument 따라 여러개
여기서는 readable interface를 사용한다