CONTAINER CLASSES

19TH LECTURE

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

Outline

- Container Classes
- Q&A

Holding Objects

- Container (collection) classes in the java.util library, automatically resizing themselves
  - List
  - Set
    - Holding one object of each value
  - Queue
  - Map
    - Associative array that permits associating objects with others

Type-Safe Containers

```
import java.util.*;
class Apple {
    private static long counter;
    private final long id = counter++;
    public long id() { return id; }
}
class Orange {}
public class ApplesAndOrangesWithoutGenerics {
public static void main(String[] args) {
    ArrayList apples = new ArrayList();
    for(int i = 0; i < 3; i++)
        apples.add(new Apple());
    // Not prevented from adding an Orange to apples:
    apples.add(new Orange());
    for(int i = 0; i < apples.size(); i++)
        ((Apple)apples.get(i)).id();
    // Orange is detected only at run time
}
}
```

ArrayList<Apple>
For compile-time prevention

Type-Safe Containers Cont'd

```
import java.util.*;
public class ApplesAndOrangesWithGenerics {
public static void main(String[] args) {
    ArrayList<Apple> apples = new ArrayList<Apple>();
    for(int i = 0; i < 3; i++)
        apples.add(new Apple());
    // Compile-time error:
    // apples.add(new Orange());
    for(int i = 0; i < apples.size(); i++)
        System.out.println(apples.get(i).id());
    // Using foreach:
    for(Apple c : apples)
        System.out.println(c.id());
}
}
```

```
0
1
2
0
1
2
```

Type-Safe Containers Cont'd

```
import java.util.*;
class GrannySmith extends Apple {}
class Gala extends Apple {}

public class GenericsAndUpcasting {
public static void main(String[] args) {
    ArrayList<Apple> apples = new ArrayList<Apple>();
    apples.add(new GrannySmith());
    apples.add(new Gala());
    for(Apple c : apples)
        System.out.println(c);
}
}
```

```
GrannySmith@7d772e
Gala@11b86e7
```

Basic Concepts of Holding Objects

- Collection: a sequence of elements with rules applied to them
  - List
    - Holding elements in the way that they were inserted
  - Set
    - Without any duplicate elements
  - Queue
    - Producing the elements in the FIFO discipline
- Map: a group of key-value object pairs
  - Permitting looking up an object using another object
    - cf., ArrayList permits looking up an object using a number

```
List<Apple> apples = new ArrayList<Apple>();
List<Apple> apples = new LinkedList<Apple>();
```

Adding Elements

```
import java.util.*;
public class SimpleCollection {
public static void main(String[] args) {
    Collection<Integer> c = new ArrayList<Integer>();
    for(int i = 0; i < 10; i++)
        c.add(i); // Autoboxing
    for(Integer i : c)
    System.out.print(i + ", ");
}
}
```

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
```

```
import java.util.*;
public class AddingGroups {
public static void main(String[] args) {
    Collection<Integer> collection =
        new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 5));
    Integer[] moreInts = { 6, 7, 8, 9, 10 };
    collection.addAll(Arrays.asList(moreInts));
    List<Integer> list = Arrays.asList(16, 17, 18, 19, 20);
    list.set(1, 99); // OK -- modify an element
}
}
```

Printing Containers

```
import java.util.*;
public class PrintingContainers {
static Collection fill(Collection<String> collection) {
    collection.add("rat");
    collection.add("cat");
    collection.add("dog");
    collection.add("dog");
    return collection;
}
static Map fill(Map<String,String> map) {
    map.put("rat", "Fuzzy");
    map.put("cat", "Rags");
    map.put("dog", "Bosco");
    map.put("dog", "Spot");
    return map;
}
```

```
GrannySmith@7d772e
Gala@11b86e7
```

Printing Containers Cont'd

```
public static void main(String[] args) {
    System.out.println(fill(new ArrayList<String>()));
    System.out.println(fill(new LinkedList<String>()));
    System.out.println(fill(new HashSet<String>()));
    System.out.println(fill(new TreeSet<String>()));
    System.out.println(fill(new LinkedHashSet<String>()));
    System.out.println(fill(new HashMap<String,String>()));
    System.out.println(fill(new TreeMap<String,String>()));
    System.out.println(fill(new LinkedHashMap<String,String>()));
}
}
```

```
[rat, cat, dog, dog]
[rat, cat, dog, dog]
[dog, cat, rat]
[cat, dog, rat]
[rat, cat, dog]
{dog=Spot, cat=Rags, rat=Fuzzy}
{cat=Rags, dog=Spot, rat=Fuzzy}
{rat=Fuzzy, cat=Rags, dog=Spot}
```

- Adding a key-value pair to a map: Map.put(key,value)
- Producing the value for a key Map.get(key)

Queue

- **LinkedList** has methods to support queue behavior, implementing the **Queue** interface
  - **offer( )** inserts an element at the tail of the queue if it can, or returns **false**
  - **peek( )** and **element( )** return the head of the queue *without removing it*
    - **peek( )** returns **null** if the queue is empty
    - **element( )** throws **NoSuchElementException**
  - **poll( )** and **remove( )** remove and return the head of the queue
    - **poll( )** returns **null** if the queue is empty
    - **remove( )** throws **NoSuchElementException**

Queue Cont'd

```
import java.util.*;
public class QueueDemo {
public static void printQ(Queue queue) {
    while(queue.peek() != null)
        System.out.print(queue.remove() + " ");
        System.out.println();
}
public static void main(String[] args) {
    Queue<Integer> queue = new LinkedList<Integer>();
    Random rand = new Random(47);
    for(int i = 0; i < 10; i++)
        queue.offer(rand.nextInt(i + 10));
    printQ(queue);
    Queue<Character> qc = new LinkedList<Character>();
    for(char c : "Brontosaurus".toCharArray())
        qc.offer(c);
    printQ(qc);
}
}
```

```
8 1 1 1 5 14 3 1 0 1
B r o n t o s a u r u s
```

Stack

```
// Stack.java
// Making a stack from a LinkedList.
import java.util.LinkedList;
public class Stack<T> {
    private LinkedList<T> storage = new LinkedList<T>();
    public void push(T v) { storage.addFirst(v); }
    public T peek() { return storage.getFirst(); }
    public T pop() { return storage.removeFirst(); }
    public boolean empty() { return storage.isEmpty(); }
    public String toString() { return storage.toString(); }
}
```

```
// StackTest.java
public class StackTest {
public static void main(String[] args) {
    Stack<String> stack = new Stack<String>();
    for(String s : "My dog has fleas".split(" "))
        stack.push(s);
    while(!stack.empty())
        System.out.print(stack.pop() + " ");
}
}
```

fleas has dog My