

Lecture 33: Iterators and String Buffer

Last time:

1. Midterm 2 Review 4/18

Today

1. Iterators
2. String Buffer
3. Introducing inheritance ...



Iterators

- Special objects for iterating through elements in collection
- Created using `iterator()` method of collection

```
ArrayList<String> a = ...;  
Iterator<String> i = a.iterator();
```

- Iterator objects implement `Iterator<T>` interface
 - Methods of interest
 - `<T> next()` Return “current” element, advance
 - `boolean hasNext()` Is there a next element?
 - `void remove()` Remove current element
 - Be sure to import `java.util.Iterator!`
- In array paradigm:
 - `next()` equivalent to `a[i++]`;
 - `hasNext()` equivalent to `i < a.length`

Example

```
public static String concat (ArrayList<String> a) {  
    Iterator<String> i = a.iterator();  
    String conc = "";  
    while (i.hasNext()) {  
        conc += i.next();  
    }  
    return conc;  
}
```

- Concatenates elements in an ArrayList
- Note use of iterator!

Iterator Behavior

- When iterator is created, its “marker” starts at initial element of collection
- An iterator only moves forward
 - No backing up
 - No resetting
- To cycle through data again, create another iterator
- See `IteratorExample.java`

Iterators and Mutability

- If you are iterating through a collection and someone adds or removes an element, your iterator is no longer valid!!!
- **Usually** a `ConcurrentModification` exception is thrown
- Calling the iterator's own `remove()` method does not ruin the iterator
- More than one iterator possible for the same collection simultaneously
 - Do not use `remove` in this case!
 - Why? Because of `ConcurrentModification` problem with other iterator

Mutable Strings

- Strings are **immutable**
 - Once a String object is created, it cannot be altered
 - For String objects, reference = shallow = deep copying (why?)
- Sometime mutable strings would be handy
 - Sometimes a small change needs to be made to a string (e.g. misspelled name)
 - Don't want to create a whole new `String` object in this case
- `StringBuffer`: Java's class for mutable Strings

StringBuffer Basics

- See documentation at:
<http://java.sun.com/j2se/1.5.0/docs/api/java/lang/StringBuffer.html>
- Main methods
 - `append`: add characters to end
 - `insert`: add characters in middle
 - `delete`: remove characters
- Note
 - `append`, `insert` return object of type `StringBuffer`
 - This is alias to object that the methods belong to!
- See `StringBufferExample.java`

Inheritance

- A crucial feature of object-oriented programming languages
 - One class (**derived class, subclass**) is constructed ...
 - ... by importing (**inheriting**) information ...
 - ... from another (**base class, superclass, parent class**) ...
 - ... and adding new information / redefining existing
- Example
 - Base class: Clock
 - setTime
 - getTime
 - tick
 - Derived class: Alarm Clock
 - Same methods as Clock plus a few additional ones: setAlarm, ring

Can We Avoid Code Copying?

- Clock “is a” Alarm Clock
- Operations on Clock (e.g. setTime) should be inherited by Alarm Clock
- Alarm Clock should only have to add information specific to alarm clocks
 - setAlarm
 - ring
- **Inheritance** provides just this capability

Clock, Alarm Clock Example

- **Clock Class**

```
class Clock {
    private int hours, minutes, seconds;
    public String getTime() {    // returns "HH:MM:SS"
        ...
    }
    public void setTime(int, int, int) {
        ...
    }
    public void tick() {    // adds one second
    }
}
```

- **Alarm Clock Class**

```
class AlarmClock extends Clock {
    private int alarmHour, alarmMin, alarmSec;
    public void setAlarm(int, int, int) {
        ...
    }
    public ring() {
        ...
    }
}
```