ICS 45J Extending Interfaces and Collections

Collections provide mechanisms for inserting and retrieving various types of data.
- We've talked about ArrayLists and LinkedLists in this class so far.

In Java, you can have interfaces extending other existing Interfaces
- Usually done when you want to make an Interface with more specific functionality
- Polymorphism allows the correct method to be called on a specific object

Example: Assume we have one interface extending another Interface

```
public interface InterfaceA {
      public abstract void method1();
      public abstract void method2();
}

public interface InterfaceB extends InterfaceA {
      public abstract void method3();
}

public class ClassA implements InterfaceA {
      public void method1() { System.out.println("ClassA.method1"); }

      public void method2() { System.out.println("ClassA.method2"); }
}

public class ClassB implements InterfaceB {
      public void method1() { System.out.println("ClassB.method1"); }

      public void method2() { System.out.println("ClassB.method2"); }

      public void method3() { System.out.println("ClassB.method3"); }
}

ClassA a = new ClassA();
ClassB b = new ClassB();

ArrayList<InterfaceA> listA = new ArrayList<InterfaceA>();
ArrayList<InterfaceB> listB = new ArrayList<InterfaceB>();

listB.add(a); // Compilation error!
listB.add(b);

listA.add(a);
listA.add(b);

for (InterfaceA item : listA) {
      item.method1();
      item.method2();
}
```

```
Output: (polymorphism)
ClassA.method1
ClassA.method2
ClassB.method1
ClassB.method2
```

```
for (InterfaceB item : listB) {
      item.method1();
      item.method2();
      item.method3();
}
```

```
Output:
ClassB.method1
ClassB.method2
ClassB.method3
```

Collection

- An unordered group of objects allowing duplicate entries
- Common methods:
    - int .size()
    - boolean .add(item)
    - boolean .remove(item)
    - boolean .isEmpty()
    - // and many more!

List

- An unordered group of objects allowing duplicate entries that are indexed.
- Common methods:
    - Object .get(index)
    - int .indexOf(item)
    - …

Set

- A collection containing no duplicate elements
- Does not contain any new methods from Collection, but ensures that no pair of objects are equal to each other
- For every object in the set, .equals(item) should return false

Queue

- A collection designed to hold elements for ordered processing
- Boolean .add(item)
- Object .remove() // only removes from the head of the queue
- Object .peek() // returns head, but doesn’t remove it. Returns null if empty
- Object .element() // returns head of list, throws exception if empty

Map

- Technically doesn't extend the Collection interface, but it's considered a Collection of values.
- A collection of items stored as (key, value) pairs.
- Keys are unique and can only map to one Object (but the Object can contain multiple values)
- Keys are represented as any object (normally strings or ints)
- Values can be represented as any Object
- Common methods:
    - boolean .containsKey(key)
    - boolean containsValue(value)
    - Object .get(key)
    - Object .put(key, value) // returns previous value for key or null
    - boolean .remove(key, value) // returns true if the key mapped to the value and was removed
    - Object .remove(key) // returns previous value for key or null

- There are a lot of interfaces defining the functionality for specific types of Collections

Java Collection Objects implementing Interfaces

- ArrayList implements List
- Stack implements List
- LinkedList implements List and Deque (which implements Queue)
- HashSet implements Set

- HashMap implements Map

Examples (of sub classes implementing Collections)

```
//Stack
Stack<String> s = new Stack<String>();
s.add("S1");
s.add("S2");
System.out.println(s.peek()); // returns "S2"
System.out.println(s.pop()); // returns "S2"
System.out.println(s.peek()); // returns "S1"
System.out.println(s.pop());
System.out.println(s.pop()); // throws java.util.EmptyStackException

// HashSet implements Set (no guaranteed order)
HashSet<String> s = new HashSet<String>();
s.add("S1");
s.add("S2");
s.add("S2"); // returns false. HashSet didn't add item
System.out.println(s.size());
System.out.println(s.remove("S1")); // returns true and removes
System.out.println(s.remove("S1")); // returns false
```

```
// HashMap
HashMap<Integer, String> s = new HashMap<Integer,String>();
System.out.println(s.put(0, "Richert")); // null
System.out.println(s.put(1, "Wang")); // null
System.out.println(s.put(0, "RichARD")); // Richert

System.out.println(s.containsKey(1)); // true
System.out.println(s.containsKey(10)); // false
System.out.println(s.containsValue("Richert")); // false
System.out.println(s.containsValue("RichARD")); // true

// Get Value for specific key
System.out.println(s.get(1)); // Wang

// Traverse Keys
for (Integer i : s.keySet()) {
      System.out.println(i);
}

// Traverse values
for (String i : s.values()) {
      System.out.println(i);
}

System.out.println(s.remove(0)); // RichARD
System.out.println(s.containsValue("RichARD")); // false
System.out.println(s.remove(1, "fjskj")); // false
System.out.println(s.remove(1, "Wang")); // true
System.out.println(s.size()); // 0
```