Recitation 5

Enums and The Java Collections classes

How do we represent . . .

- Suits - Clubs, Spades, Diamonds, Hearts
- Directions - North, South, East, West
- Days of week - Monday, Tuesday . . .
- Planets - Mercury, Venus, Earth . . .

Other small sets of values that do not change

Using constants

```java
public class Suit {
    public static final int CLUBS = 0;
    public static final int SPADES = 1;
    public static final int DIAMONDS = 2;
    public static final int HEARTS = 3;
}
```

Problems:
- no type checking
- readability

```java
void setSuit(int suit) {...}

int getSuit() {...}
```

Better way: Objects as constants

```java
public class Suit {
    public static Suit CLUBS = new Suit();
    public static Suit SPADES = new Suit();
    public static Suit DIAMONDS = new Suit();
    public static Suit HEARTS = new Suit();
}
```

- cannot modify Suit objects
- no new Suits can be created

```java
Suit v; ...
if (v == Suit.CLUBS) {...}
```

Cannot use ==

Enum (enumeration) declaration

- can be any access modifier

```java
public enum Suit {CLUBS, SPADES, DIAMONDS, HEARTS};
```

new keyword

name of enum

static final variables

of enum Suit

About enums

1. Can contain methods, fields, constructors
   - `Suit.HEARTS.getColor();`

2. Suit’s constructor is private!
   - Cannot instantiate except for initial constants

3. Suit.values() returns a Suit[] of constants in the enum
Demo: Enums in action

Look at `enum` Suit.

Create a class PlayingCard and a class Deck.

What would be the fields for a PlayingCard object?

Enum odds and ends

1. Suit is a subclass of `java.lang.Enum`
2. `ordinal()` returns position in list (i.e. the order it was declared)
   a. `Suit.CLUBS.ordinal()` == 0
3. Enums automatically implement `Comparable`
   a. `Suit.CLUBS.compareTo(Suit.HEARTS)` uses the ordinals for Clubs and Hearts
4. `toString()` of `Suit.CLUBS` is "CLUBS"
   a. you can override this!

5. `switch` statement

   ```java
   Suit s = Suit.CLUBS;
   switch(s) {
      case CLUBS:
      case SPADES:
         color = "black"; break;
      case DIAMONDS:
      case HEARTS:
         color = "red"; break;
   }
   ```

   `s == Suit.CLUBS` is true

   `switch` statements are fall through!

   `break` keyword is necessary.

Collections and Maps

The Collections classes and interfaces that come with Java provide implementations of

- bags (a.k.a. multiset – sets with repeated values)
- sets (and sorted sets)
- lists
- stacks
- queues
- maps (and sorted maps) [like dictionaries]

You will see in later assignments how easy it is to use these

ArrayList as example of structure

Class ArrayList implements a list in an array that can grow and shrink. Example of code:

```java
ArrayList<Integer> t = new ArrayList<Integer>();
t.add(5);
t.add(7);
System.out.println(t.get(0)); // prints 5
t.add(0, 2); // insert 2 at index 0, shifting other // values up. Can be costly.
System.out.println(t); // prints [2, 5, 7]
```

Power of inheritance and interfaces

```
Object
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Collection&lt;E&gt;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>AbstractCollection&lt;E&gt;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>List&lt;E&gt;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>AbstractList&lt;E&gt;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ArrayList&lt;E&gt;</td>
</tr>
</tbody>
</table>
```

Format of ArrayList object
Important interfaces, some methods in them

- **Collection<E>**
  - `add(E)`
  - `contains(Object)`
  - `isEmpty()`
  - `remove(Object)`
  - `size()`
  - No new methods in Set<E>, just changes specifications

- **List<E>**
  - `get(int)`
  - `indexOf(int)`
  - `add(int,E)`

- **Map<K,V>**
  - `put(K,V)`
  - `get(Object)`

Important classes and interfaces

- **Collections and Map**

- **HashSet<E>**
- **ArrayList<E>**

Queues? Stacks?

- **Collection<E>**
- **Queue<E>**
- **Deque<E>**
- **LinkedList<E>**
- **ArrayDeque<E>**

Iterating over a HashSet or ArrayList

```java
HashSet<E> s = new HashSet<E>();

... store values in the set ...

for (E e : s) {
    System.out.println(e);
}
```

Body of loop is executed once with `e` being each element of the set. Don't know order in which set elements are processed.

Collections problems

1. Remove duplicates from an array
2. Find all negative numbers in array
3. Create ransom note
4. Implement a Stack with a max API
5. Braces parsing

Collections problems

**Complete**

```java
Integer[] removeDuplicates(int[])
```

Remove all duplicates from an array of integers.

Very useful HashSet method:

```java
hs.toArray(new Integer[hs.size()]);
```
Collections problems

Find Negative Numbers
Find all negative numbers in array and return an array with those integers

Very useful ArrayList method:
`lst.toArray(new Integer[lst.size()]);`

Collections problems

Create Ransom Note
Given a note (String) that you would like to create and a magazine (String), return whether you can create your note from the magazine letters.

Implement a Stack<E> with a max() function in O(1) time
No matter how full the stack is, the max function should be in constant time. (i.e. you should not iterate through the Linked List to find the maximum element)

Collections problems

Braces parsing in O(n) time
Return whether a String has the right format of square brackets and parenthesis.

- e.g.
  - "array[4] = ((( new Integer(3) )))" <- is true
  - "[]" <- is false
  - "[]" <- is false
  - "[ ]" <- is false