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;
}

void setSuit(int suit) {...}

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

Problems:
- no type checking
- readability
Better way: Objects as constants

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

    private Suit() {}
}
```

- no new Suits can be created
- cannot modify Suit objects
- must use `==`
**Enum (enumeration) declaration**

can be any access modifier

```
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();`

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

1. `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!
Enum odds and ends

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.
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

- AbstractCollection<E>
- AbstractList<E>
- ArrayList<E>
- Collection<E>
- Iterable<E>
- List<E>

Format of ArrayList object
Important interfaces, some methods in them

**Collection<E>**
- `add(E);`
- `contains(Object);`
- `isEmpty();`
- `remove(Object);`
- `size();`
- ...

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

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

**Set<E>**

No new methods in Set<E>, just changes specifications
Important classes and interfaces

- Collection<E>
  - Set<E>
    - HashSet<E>
  - List<E>
    - LinkedList<E>
    - ArrayList<E>
  - Map<K,V>
    - HashMap<K,V>
Queues? Stacks?

- Collection\(<E>\)
  - Queue\(<E>\)
    - Deque\(<E>\)
      - LinkedList\(<E>\)
      - ArrayDeque\(<E>\)

Deque:
Double-Ended Queue
Iterating over a HashSet or ArrayList

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:
```
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:

```java
lst.toArray(new Integer[lst.size()]);
```
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.

![Ransom Note Image]
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. (ie 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