ECE25100: OBJECT ORIENTED PROGRAMMING

Note 6 _ ArrayList

Instructor: Xiaoli Yang

ArrayList

- Arrays have a fixed size. They cannot grow or shrink in response to an application's changing storage requirements.
- ArrayList:
  - can store arbitrary objects like an array.
  - will grow automatically as space is needed (grows either by a default or user-defined amount).
  - has an efficient method for adding an element, except when it gets full ... in that case, it takes time to "grow" the arraylist before adding the new element.
  - cannot store values of primitive data types unless they are wrapped up in a wrapper class. (version 1.5, automatically)

ArrayList

- Since JAVA version 1.5, it is recommended that we specify the types of objects that will be stored in the ArrayLists. Here is the general format that we will follow to create an ArrayList (or Vector) and assign it to a variable:

```
ArrayList<Type> myList = new ArrayList<Type>();
```

The <Type> indicates the type of Objects that are to be stored in the ArrayList. Here are some examples:

```
ArrayList<String>   names = new ArrayList<String>();
ArrayList<Customer>   customers = new ArrayList<Customer>();
ArrayList<BankAccount>   accounts = new ArrayList<BankAccount>();
ArrayList<Object>   listOfThings = new ArrayList<Object>();
```

ArrayList (Cont.)

- Sometimes it is nice to be able to define an ArrayList with an initial set of objects. Unfortunately, there is no special syntax (as we had with arrays) that allows this.

- So the following code will not work:

```
ArrayList<String>  myList = {"one", "two", "three", "four", "five", "six"};
```

ArrayLists (Cont.)

We can even make ArrayLists of "primitives", by making use of the wrapper classes. In fact, version 1.5 of JAVA automatically wraps and unwraps primitives when adding and extracting them from Collections. So, we can write the following code:

```
ArrayList<Double> rateList = new ArrayList<Double>();
double[]  rates = {23.43, 56.7, 675.56765, 76.7};
for (int i=0; i<rates.length; i++)
    rateList.add(rates[i]);
double sum = 0.0;
for (int i=0; i<rateList.size();i++)
   sum + = rateList.get(i);
System.out.println(sum);
```

ArrayLists (Cont.)

- Consider creating an ArrayList of Customer objects from an existing array by adding them one at a time:

```
Customer[ ]  cust = {new Customer(“Mike"), new Customer("Zack"), new
Customer("Sally"), new Customer("Mel")};
ArrayList<Customer> list = new ArrayList<Customer>();
for (int i=0; i<cust.length; i++) {
    list.add(cust[i]); }
```

- To get the Customers back out later, we can use the

```
for (int i=0; i<list.size();i++) {
   Customer c = list.get(i);
   System.out.println(c.getName());
}
```

- In fact, we do not need the c variable here:

```
for (int i=0; i<list.size();i++) {
   System.out.println(list.get(i).getName());
}
```

ArrayLists (Cont.)

- Another constructor that does take an existing collection and fills up a new ArrayList with the elements

  of that collection:

ArrayList<Type> myList = new ArrayList<Type>(Collection col);

- Here, a Collection may be another ArrayList (or some other kind of Collection ...). However, if we want to make an ArrayList by specifying particular values, we can make an array first and then convert it into an ArrayList:

```
String[ ]  num = {"one", "two", "three", "four", "five "};
ArrayList<String> myList = new ArrayList<String>(java.util.Arrays.asList(num));
```

The Enhanced FOR loop and Iterators

- JAVA version 1.5 has introduced an extension to the FOR loop that makes it easy to traverse through all the elements of an array or ArrayList. Here is the general format:

```
for (<type> <var> : <arraylistName>) {
    // <var> represents each element in the collection;
}
```

You read the code as if it says: "for each <var> in <arraylistName>".

Here is what the values represent:

- <type> represents the type of data stored in the array list.
- <arraylistName> is the actual array list being traversed through.
- <var> is the loop variable representing the actual values of the array list.

The Enhanced FOR loop and Iterators

- We can also use this new FOR loop when traversing through an ArrayList (or any other Collection).
- The example of going through Customers of an ArrayList:

```
for (int i=0; i<list.size();i++) {
   Customer c = list.get(i);
   System.out.println(c.getName());
}
```

Here is the code with the new FOR loop:

```
for (Customer cust: list) {
   System.out.println(cust.getName());
}
```

The Enhanced FOR loop and Iterators

- JAVA does not allow removing certain elements of a Collection as you traverse through it.
  - we want to remove certain elements of a Collection as we traverse through it. We may want to do something like this:

```
for (Person p: listOfPeople) {
   if (p.getAge() < 21)
      people.remove(p);
}
```

  - This code does not compile. JAVA will throw a ConcurrentModificationException, as it prevents us from modifying (e.g., removing elements from) the collection while we are traversing through it.

The Enhanced FOR loop and Iterators

- JAVA does however, provide us with a way of removing elements from the collection which we are traversing, by use of an Iterator. Iterators are objects that:
  - generate elements from a collection one by one.
  - are meant to be used to traversing (i.e., enumerate through) the elements of a collection once.
- Many methods in JAVA return Iterators instead of Collections or ArrayLists. Hence, Iterators are widely used in JAVA.

The Enhanced FOR loop and Iterators

- Iterators basically have 3 available methods:
  - hasNext() ... returns a boolean indicating whether there are any more items left ?
  - next() ... returns the next item
  - remove() ... removes the latest item that was obtained from the last call to next().

The Enhanced FOR loop and Iterators

- Users make successive calls to next() to obtain the elements from the collection.

```
Iterator people = listOfPeople.iterator();
while (people.hasNext()) {
    Person p = (Person)people.next();
    if (p.getAge() < 21)
        people.remove();
}
```

- Notice that the remove() method is sent to the iterator, NOT to the underlying collection. Also, the only item that can be removed is the last one that was returned from a call to next().

The Car and Autoshow Case Study

Example description:

Consider an Autoshow that contains various Cars from various companies. Customers go to the Autoshow with dreams of someday owning cars such as these or to investigate the cars/trucks/minivans that are accessible in their price range. We will pick four characteristics of the cars that are interesting. For each car, we will keep the name, make, color and topSpeed.

The Car and Autoshow Case Study

1. To define the Car class and create its get and set methods, its constructor and a toString() method.
2. An Autoshow should maintain an ArrayList of Car objects. Again, we'll make appropriate get, constructor and toString() methods.
3. Now in order to add cars to the auto show, we need an addCar() method. We'll also add a method that will return the number of cars in the auto show.
4. Find the "fastest car" with a method topSpeed().
5. Make a method carsWithMake(String m)that returns all the "cars with a given make".

The Car and Autoshow Case Study

6. Make a method `differentMakes()` that returns all of the "different makes of cars" at the auto show. We want only a list of the different makes, so no duplicates are allowed.
7. Write a method `printByMake()` that "prints out all the cars in alphabetical order of their make". Cars with the same make will appear in arbitrary order amongst themselves.
8. Write a method `printBySpeed()` that "prints out all the cars in order of their top speed" (fastest cars first). Cars with the same speed will appear in arbitrary order among themselves.
9. Find the "most common color of all cars" with a method `mostCommonColor()` at the auto show.

Strings

- Strings are one of the most commonly used objects in all programming languages since strings represent text, and people communicate using words.
- We have seen that making a literal String is quite easy ... we just use double quotes. Strings are actually objects, not primitives as you might have initially expected.
- We have also seen that an object can be joined or concatenated to a String by using the + operator:
  - ownerName + "'s Bank Account with $" + balance;

Strings (Cont'd)

- We will now take a more in-depth look at the String objects and what we can do with them.
- Strings are:
  - objects
  - fixed size and CANNOT be modified once created !
  - a sequence of char primitives which are accessible by index (where index starts at 0, not 1).

Strings

- Creating a String:
  - We can create a String as follows:

```
String name = “Computer Engineering";
```

  - The string is automatically created and initialized without needing to use new as we would with other objects. To make a null or empty string, we do the following:

```
String name;          //an uninitialized null String
String name2 = "";    //an initialized String that is empty
```

- Strings can also be created using a constructor:

```
new String();              //makes an empty String
new String(String value);  //makes copy of another string
new String(char[] value);
new String(char[] value, int offset, int count);
new String(StringBuffer buffer);
```

String Builders

- Strings cannot be changed. Instead, when we want to manipulate them, we always get back a new String. The StringBuilder class can be thought of as a String that can be modified:
  - StringBuilder objects are modifiable Strings.
  - StringBuilder objects are generally slower.
  - If the StringBuilder is never to be changed, use a String instead.
  - StringBuilder methods cannot be used with Strings.
- When making some kind of text editor application in which text is being edited by the user, it is nice to store the text in a StringBuilder as it allows for easy appending, insertion and removal.

String Builders (Cont'd)

- Here are two constructors for the StringBuilder class:

```
new StringBuilder();
new StringBuilder(String aString);
```

- The first creates a StringBuilder with no characters to begin with and the second creates one with the characters equal to the ones in the given String.
- More methods are available.

String Tokenizer

- The java.util.StringTokenizer class allows us to break a string into individual substrings based on some separation criteria.
- For example, we can extract the words from a sentence one at a time. The term delimeter is used to indicate the character(s) that separate the "words" (also known as tokens).
- So it allows you to extract various tokens (i.e., pieces) of a String one token at a time:
  - words from a sentence, one by one
  - fields from a database, separated by commas or other chars

String Tokenizer (Cont'd)

- A tokenizer is created for a particular String which is specified in the constructor:

```
new StringTokenizer(String aString);
```

- Tokens are separated by characters which are called delimiters. The default delimiter is the space character.
- You may specify other delimiters in the constructor:

```
new StringTokenizer(String aString, String delimiters);
 (e.g., delimiters may be ",:;|_()" etc...)
```

- The countTokens() method returns the number of tokens in the String.

```
aTokenizer.countTokens();
```

String Tokenizer

- Tokens are repeatedly extracted using the nextToken() method which returns a String representing the token:

```
aTokenizer.nextToken();
```

- Each token may be extracted only once.
- A final method called hasMoreTokens() returns whether or not there are any more tokens remaining:

```
String sentence = "Banks, Rob, 34, Indiana, 12.67";
StringTokenizer words = new StringTokenizer(sentence,", ");
while(words.hasMoreTokens()) {
    String aWord = words.nextToken();
    System.out.println(aWord);
}
```

- Notice that both the comma and space characters are counted as delimiters. That means that either commas or spaces or any combinations of consecutive commas and spaces count as a single token separator (or delimiter).
- Example: StringEditorTester.java, ScannerTokenizingTest.java

References

- Carleton University COMP 1005 and 1006:
  http://www.scs.carleton.ca/~lanthier/teaching/COMP1405/Notes/
  http://www.scs.carleton.ca/~lanthier/teaching/COMP1406/Notes/
- Armstrong Atlantic State University
  http://www.cs.armstrong.edu/liang/intro10e/
- Oracle Java tutorial:
  http://docs.oracle.com/javase/tutorial/
- MIT opencourseware
  https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/