CMSC 131: Chapter 24 (Supplement)
Miscellany

Miscellany

Today we discuss a number of unrelated but useful topics that have just not fit into earlier lectures. These include:

StringBuffer: A handy class for dealing with strings whose contents can vary dynamically.

Java's Stack Data Structure: One of the simplest data structures in Java's class library.

Java's Method Call Stack: How does Java keep track of local variables and parameter when methods are called?

StringBuffer

The problem with Strings:

- Strings are **immutable** objects. This means that once constructed it is not possible to change its contents.

- **Example**: Form a string by repeated concatenation:

```
char[ ] c = { 'H', 'e', 'l', 'l', 'o' };
String s = "";
for ( int i = 0; i < c.length; i++ )
    s += c[i];
```

  This is quite **inefficient** because it produces one entirely new String object for each new assignment to s.

Q: What is a more efficient way to do this?

Ans: Store s as an **expandable** array of chars. (Double its size whenever we run out of space.) Cast the final array to a String.

StringBuffer: Java offers just such a structure for you.

StringBuffer

StringBuffer: A "**mutable**" representation of a string, which provides efficient methods for modifying the contents.

Some StringBuffer Methods:

StringBuffer( ) – Constructs an **empty** string buffer.

append(…) - Converts the argument into a string, and appends it to the end of the string buffer. (Possible arguments include boolean, char, double, float, int, long, Object, and char[ ].)

charAt(int index) - Returns the character at the **specified index**.

length( ) – Returns the **number of characters** in the buffer.

toString( ) - Returns a **String representation** of the buffer.

Example:

```
StringBuffer b = new StringBuffer( );
b.append( 99.5 );
b.append( '%' );
b.append( " pure" );
System.out.println( b );
```

StringBuffer: Example

Example: A method **getWords** which is given a string and strips off spaces and punctuation, converts words to lower-case, and outputs a string with the results.

Java Class Library Utilities:

String split: Splits a string about a given regular expression pattern.
- **[abc]** matches characters 'a', 'b', 'c'.
- **[abc]*** matches **0 or more repetitions** of these characters.
- **[abc]+** matches **1 or more repetitions** of these characters.

To remove punctuation we split about 1 or more occurrences of space, comma, period, question mark: `split( "[ ,.?]+" )`

String toLowerCase: Returns an equivalent **lower-case** string.

String valueOf: Produces a String representation of an object.

StringBuffer: Example

```
public static String getWords( String s ) {
    String[ ] words = s.split( "[ ,.?]+");
    StringBuffer buffer = new StringBuffer( );
    for ( int i = 0; i < words.length; i++ ) {
        buffer.append( words[i].toLowerCase( ) );
        if ( i < words.length-1 ) buffer.append( " " );
    }
    return String.valueOf( buffer );
}

public static void getWordsTest( ) {
       String s1 = "Do you wake up in the morning feeling sleepy and grumpy?";
       System.out.println ( "[" + getWords( s1 ) + "]" );
       String s2 = "Then you must be Snow White.";
       System.out.println ( "[" + getWords( s2 ) + "]" );
}
```

Stacks

Stack: A stack is an **abstract data structure** for storing a collection of items. Items can be **inserted** into the stack and **removed** from the stack, but the rule is the most recent item inserted is the first item to be removed. (**Last in, first out**)

Intuition: Think of it like a stack of plates in a restaurant. Items:

- can be inserted (or **pushed**) onto the top of the stack.
- can be removed (or **popped**) off of the top of the stack.
- insertions/removals from other positions are **not allowed**.

Stack Operations

Stack Operations: An abstract (mathematical) stack supports:

push(x): **inserts** item x at the top of the stack
pop( ): **removes** the item at the top of the stack (if one exists) and returns its value.
top( ): **returns the value** of the item at the top of the stack, without removing it.
empty( ): **returns true** if the stack is **empty**

Java's Stack Class

Java's Stack class: (in **java.util**) Java provides a Stack, with the following corresponding operations.

Stack( ): creates an empty stack
push(Object x): pushes x on the stack
pop( ): pops the stack and returns its value. (Exception if empty)
peek( ): returns (without removal) the top value of the stack. (Exception if empty)
empty( ): returns true if the Stack is empty.

ArrayList

The Problem with Arrays:

Resizing: Arrays are not suitable for situations where the size of the array changes frequently.
Appending to an Array: if we reach the maximum capacity of an array and we need to add an element, we have to create a new array, copy over elements, and add the desire element.

ArrayList:

- A class in the Java class library that implements a **resizable array**.
- It is part of the **java.util** package, and therefore an appropriate **import** statement is required.
- An ArrayList holds **generic Object references**. Each element is:
  - a reference to any **class object** or **array**
  - any **primitive type**, by first putting it in a **wrapper**, such as Integer.
- When an element is removed from the array, you must **explicitly cast** it back to its **original** type.

ArrayList Methods

Some of ArrayList methods:

ArrayList( ): Initializes an array list of size 0.
add(Object obj): Adds the object to the end of the array. (Automatically expands the array if needed.)
add(int i, Object obj): Adds the object at position i (and shifts remaining objects down to make room).

The following return an object of type **Object** or **Object[ ]**. You must cast to the original type:
- **remove(int i)**: Removes the element at index i. (Shifts the remaining elements to close the gap.)
- **get(int i)**: Returns a reference to the element at index i.
- **toArray( )**: Returns a (standard) array with all the elements.

clear( ): removes all the elements from ArrayList.
size( ): returns the number of elements in ArrayList.

ArrayList Example

Here is an example using an **ArrayList of Strings**:

```
ArrayList a = new ArrayList( );
a.add( new String( "Bob" ) );      // [Bob]
a.add( new String( "Carol" ) );    // [Bob, Carol]
a.add( 1, new String( "Ted" ) );   // [Bob, Ted, Carol]
System.out.println( a.size( ) );   // prints: 3
String x = a.get( 2 );             // illegal: cannot convert Object to String
String y = (String) a.get( 2 );    // okay: returns "Carol"
a.clear( );                        // clear it out
System.out.println( a.size( ) );   // prints: 0
```