Lecture 18

Even more object-oriented thinking
Primitive data types

- Not objects in Java for performance reasons
- But lots of methods expect an object (and not a primitive data type)
- One can “wrap” up these primitive data types into an object and use them like other objects
Primitive data type wrappers

- Wrappers included in Java.lang:
  - Boolean
  - Character
  - Double
  - Float
  - Byte
  - Short
  - Integer
  - Long

Mostly just the same as the primitive data type, except for Integer (int) and Character (char)
Primitive data type wrappers

• Numeric wrapper classes have <class>Value() methods, like `doubleValue()`, `floatValue()`, `intValue()` that you can use to convert the object into different primitive types.
Primitive data type wrappers

- Can construct using either a value, or a string of a value:
  - `new Integer("17")`
  - `new Double(7.8)`
- No no-arg constructor and are immutable
Primitive data type wrappers

- Each have constants for the MAX_VALUE and MIN_VALUE
- compareTo() method as well for comparing numbers
Primitive data type wrappers

- The wrapper classes also have static methods that are super helpful
  - `valueOf(String s)`
  - parsing methods
• Converting a primitive value to an object is called *boxing*, the reverse is called unboxing

• Java will do this *automagically*, called *autoboxing and autounboxing* *(Not convinced that’s really a word)*
Primitive data type autoboxing

Integer intObject = new Integer (2);   Equivalent   Integer intObject = 2;

(a)                                      (b)  

autoboxing
BigInteger and BigDecimal

• any guesses on what these classes are?
BigInteger and BigDecimal

• *Super* big integers, or *super* precise decimals

• add, subtract, multiply, divide and remainder
import java.math.*;
public class BigNumbers {

    public static void main(String[] args) {
        System.out.println(factorial(50));
    }

    public static BigInteger factorial(int n){
        BigInteger result = BigInteger.ONE;
        for (int i = 1; i <= n; i++){
            result = result.multiply(new BigInteger(i+""));
        }
        return result;
    }
}
String class

- 13 constructors and 40+ methods!
- Can construct with a string literal, for example “hello”
- Can also construct with a char array

```java
cchar[] charArray = {'h', 'e', 'l', 'l', 'o'};
```
String class

- Remember that Strings are immutable and are really only holding a reference to a string. So when you assign a different value to a string, it's just pointing to a different object in memory.
String class

String s = "Java";
s = "HTML";

After executing String s = "Java";

Contents cannot be changed

After executing s = "HTML";

This string object is now unreferenced
Interned string

Remember how we don’t use `==` for strings? THIS is why:

```java
String s1 = "Welcome to Java";
String s2 = new String("Welcome to Java");
String s3 = "Welcome to Java";

System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```

display

s1 == s2 is false
s1 == s3 is true
Splitting and replacing strings

• Methods to replace and split strings
public class StringWork {

    public static void main(String[] args) {

        String s1 = "Hello, my most favorite class!";

        // s1 = s1.replace("most", "least"); // Just kidding!
        // s1 = s1.replace("favorite", "excellent");
        s1 = s1.replace('H', 'h');

        System.out.println(s1);

        String[] stringArray = s1.split(" ");
        System.out.println(stringArray[0]);
    }
}
Regular expressions (or regex) can be used for searching, replacing and splitting strings

You define patterns that you can use to find, replace or split stuff in strings

the matches() method lets you use these patterns for searching

These can get very complex. We’ll stick to simple cases outlined in the book for now
boolean itMatches = false;
itMatches = "440-02-4534".matches("\d{3}-\d{2}-\d{4}");
System.out.println(itMatches);
Converting between strings and arrays

• strings and arrays can be converted into each other

• String.toCharArray()

• String.getChars() method to copy parts of a string into an existing array

• to convert an array of chars to a string, use the constructor or valueOf() method
StringBuilder, StringBuffer

- Can be used (for the most part) where strings are used.
- More flexible than strings
- add, insert and append, delete
- StringBuffer for synchronization
StringBuilder, StringBuffer

**StringBuilder**

- **append(data: char[]):** StringBuilder
- **append(data: char[], offset: int, len: int):** StringBuilder
- **append(v: aPrimitiveType):** StringBuilder
- **append(s: String):** StringBuilder
- **delete(startIndex: int, endIndex: int):** StringBuilder
- **deleteCharAt(index: int):** StringBuilder
- **insert(index: int, data: char[], offset: int, len: int):** StringBuilder
- **insert(offset: int, data: char[]):** StringBuilder
- **insert(offset: int, b: aPrimitiveType):** StringBuilder
- **insert(offset: int, s: String):** StringBuilder
- **replace(startIndex: int, endIndex: int, s: String):** StringBuilder
- **reverse():** StringBuilder
- **setCharAt(index: int, ch: char):** void

Adds a char array into this string builder.

Appends a subarray in data into this string builder.

Appends a primitive type value as a string to this builder.

Appends a string to this string builder.

Deletes characters from startIndex to endIndex-1.

Deletes a character at the specified index.

Inserts a subarray of the data in the array into the builder at the specified index.

Inserts data into this builder at the position offset.

Inserts a value converted to a string into this builder.

Inserts a string into this builder at the position offset.

Replaces the characters in this builder from startIndex to endIndex-1 with the specified string.

Reverses the characters in the builder.

Sets a new character at the specified index in this builder.
StringBuilder, StringBuffer

• The contents of the actual string are changed, instead of creating another string in memory.

• Especially useful for inserting, deleting, reversing

• Mostly used for making programs more efficient