CMSC 131: Chapter 6 (Supplement)
Introduction to Objects

Objects
Object: is the fundamental entity around which Java programs are designed.

State:
Behavior:
Examples:
Bank account:
  State: account number, owner’s name and address, current account balance.
  Behaviors: credit (increase balance), debit (decrease balance), change address, add interest, print monthly statement, ...
Voice-mail system:
  State: caller’s greeting, list of messages, number of the current message.
  Behaviors: rewind messages, play current message, erase current message, change caller’s greeting, ...

Object-oriented programming:

Classes
Class: a definition or “blueprint” for an object.

Instance Data:

Methods:

Example: Bank account
Instance data:
  String accountNumber;
  String ownerName;
  String ownerAddress;
  double currentBalance;

Methods:
  debit: decrease the current balance by a given amount
  credit: increase the current balance by a given amount
  addInterest: compute interest and add it to the balance
  ...

Encapsulation: A class is a way of combining all the data and functions that are involved with a bank account in one place.
Creating Objects

Before discussing how to define objects, we discuss how to create and use them. In Java a variable can hold either:

Primitive type value:

Reference to an object:

Variable declaration:

Primitive types:

Class object:

Examples:

```java
int x = 52;
String s = new String( "Schultzie" );
String t = "von Wienerschnitzel";
```

Under the Hood: What does "new" do?

What happens when "new" is called:

```java
String s = new String( "Schultzie" );
```

- Space for a new string object is allocated in a special area of memory called the heap.
- This storage is initialized with the characters comprising the name.
- The address in memory where the object has been created is returned by new.
- This address, also known as a reference, is stored in the variable s.

The Java Class Library

Java's class library: One of Java's nicest features. The library is organized into packages.

- **java.lang**: General support (String, Integer, Float, Math, ...).
- **java.util**: General utilities (random numbers, currency, calendar, useful data structures).
- **java.text**: Contains utilities for formatting text (dates, currency, etc)
- **java.awt**: Abstract Window Toolkit, basic support for graphics and graphical user interfaces (GUI's).
- **javax.swing**: Extension of java.awt with more advanced GUI elements. (e.g., JOptionPane).
- **java.io**: A variety of functions for input/output and formatting.
- **java.applet**: Used for writing Java applets (programs that run in a Web browser).

See also Appendix M in L&L and Sun's Java API Specs (a link can be found in the Resources class web page).
Accessing the Class Library

Fully qualified name:
To access a class from the library you generally need to specify the package and the class.

Example:
The JOptionPane class is part of javax.swing.

```java
javax.swing.JOptionPane.showMessageDialog( null, "This is really inconvenient! ");
String input = javax.swing.JOptionPane.showInputDialog( "I agree! ");
```

The Import Statement

Java allows you to "import" the entire class description into your program.

```java
import java.util.Random; // import class Random from java.util
...
double r = Random.nextDouble(); // no need for "java.util" anymore
```

Importing All the Classes:
Rather than import each class individually, you can use `*` to make all the classes accessible from a package.

```java
import java.util.*; // import all classes from java.util
import javax.swing.*; // import all classes from javax.swing
```

java.lang: This package is so widely used it is automatically imported into all programs.

The cmsc131 Picture Library

In addition to the Java class library, you can define your own libraries. We have defined one such library for image manipulation, called cmsc131PictureLib.

Importing the library:
```
import cmsc131PictureLib.*;
```

PictureLib's classes:

Image: This is an object that contains a single image.
To create a new image, give it the web address or path name of an image file (e.g. a jpeg file).
```
Image myImage = new Image( "C:\My Pictures\mom.jpg" );
```
This creates a new Image object, but does not display it.
The cmsc131 Picture Library

PictureLib's classes: (cont)

PictureUtil: This class contains some utility functions for displaying images. The following command displays an image on your screen.

    PictureUtil.show( myImage );  // display an image

The following command clears all the images from your screen.

    PictureUtil.clearScreen();  // clear the screen of images

(The images will also go away when you execute System.exit(0))

The cmsc131 Picture Library

PictureLib's classes: (cont) There are a number of image manipulation classes. Each class takes one or more images and produces a new image.

    BlackAndWhite: Produces an image that is the black and white image equivalent of the original.
      BlackAndWhite d = new BlackAndWhite( myImage );

    SelectComponents: Produces an image by filtering the red, green, and blue components of the original.
      SelectComponents p = new SelectComponents( myImage, true, false, true );

    CombineTopBottom: Combines two images, one atop the other.
      CombineTopBottom ctb = new CombineTopBottom( topImage, bottomImage );

    CombineLeftRight: Combines images, one to the left of the other.
      CombineLeftRight clr = new CombineLeftRight( leftImage, rightImage );

After creating any of these images, you can call PictureUtil.show to display them.
cmisc131PictureLib Example

/* A simple demonstration of the cmisc 131 picture library */
import javax.swing.*;
import cmisc131PictureLib.*;

public class PictureDemo {
    public static void main(String[] args) {
        String imageName = JOptionPane.showInputDialog("Enter image to process");
        Image myImage = new Image(imageName); // create new Image object
        PictureUtil.show(myImage); // display the image

        int answer = JOptionPane.showConfirmDialog(null,
        "Do you want to generate a black and white image?";
        if (answer == JOptionPane.YES_OPTION) {
            BlackAndWhite blackAndWhiteImage = new BlackAndWhite(myImage);
            PictureUtil.show(blackAndWhiteImage); // display the new image
        }
        JOptionPane.showMessageDialog(null, "Click "OK" when done");
        System.exit(0); // terminate
    }
}

cmisc131PictureLib Example
**JOptionPane - Revisited**

`JOptionPane`: provides a few other features that we have not discussed yet.

```java
JOptionPane.showMessageDialog( null, // (always null for us)
    "Your prompt here", // prompt for the user
    "Window title here", // title for the window
    JOptionPane.YES_NO_OPTION); // what options to give the user
```

**Example:**

```java
JOptionPane.showMessageDialog( null,
    "Are you sure you want to quit?\nI mean really sure?", // prompt for the user
    "Confirm Quit", // title for the window
    JOptionPane.YES_NO_OPTION); // what options to give the user
```

---

**The String Class - Revisited**

**Let:**

```java
String s = new String( "Schultzie" );
String t = new String( "sCHultzIE" );
```

**Length:**

```java
int i = s.length(); // returns 9
```

**Comparisons:**

```java
boolean a = s.equals( t ); // false: case sensitive
boolean b = s.equalsIgnoreCase( t ); // true: ignores case
int j = s.compareTo( t ); // ??: (avoid) depends on case ordering
int k = s.compareToIgnore( t ); // 0: (okay)
```

**Access character at a given index:** (index runs from 0 to length-1)

```java
char c = s.charAt( 0 ); // returns 'S'
char d = s.charAt( 2 ); // returns 'h'
```

**Change Case:**

```java
String x = t.toLowerCase(); // returns "schultzie"
String y = t.toUpperCase(); // returns "SCHULTZIE"
```

(See Page 89 in L&L for more examples.)
The Math Class

**Math:** is a class that offers many common math functions.

**Absolute value: (Can be used with int, long, float, and double)**

```java
double x = Math.abs(-14.3);  // returns +14.3
```

**Square Root:**

```java
double y = Math.sqrt(2.0);  // returns √2 = 1.4142...
```

**Trig functions: (provides all the trigonometric functions)**

```java
double pi = Math.PI;          // returns π = 3.14159...
double c = Math.cos(pi);     // returns cosine(π) = -1.0
double s = Math.sin(pi);     // returns sine(π) = 0.0
...
```

**Powers:**

```java
double z = Math.pow(2.0, 3.0);  // returns 2³ = 8.0
```

**Random Numbers:**

```java
double r = Math.random();    // random double from 0.0 to 1.0
```

The NumberFormat Class

**NumberFormat:** This class provides capabilities for formatting text in useful ways, e.g. currency and percentages.

**Import:**

```java
import java.text.*;
```

**Currency Conversion:** First generate a currency object:

```java
NumberFormat money = NumberFormat.getCurrencyInstance();
```

Then to generate a string:

```java
double amount = 123.6;
System.out.println(money.format(amount));  // output: "$123.60"
```

**Percentage Conversion:** First generate a percent object:

```java
NumberFormat percent = NumberFormat.getPercentInstance();
```

Then to generate a string:

```java
double taxRate = 0.0793;
System.out.println(percent.format(taxRate));  // output: "8%"
```
Example of Currency Conversion

/* This is a simple cashier checkout program */
import java.text.*;
import javax.swing.*;

public class Cashier {
    public static void main( String[] args ) {
        double priceOfMilk = 3.50; // item prices
        double priceOfSugar = 1.25;
        double total = 0.0; // initialize total price
        String item;
        do {
            // read items until 'quit'
            item = JOptionPane.showInputDialog( "Enter Item (or 'quit') ");
            if ( item.equals( "milk" ) )
                total += priceOfMilk;
            else if ( item.equals( "sugar" ) )
                total += priceOfSugar;
        } while ( !item.equals( "quit" ) ); // convert to currency
        NumberFormat money = NumberFormat.getCurrencyInstance( );
        JOptionPane.showMessageDialog( null, "Amount Due: " + money.format( total ) );
        System.exit( 0 );
    }
}

Example of Currency Conversion

Input
Enter Items (or 'quit')

OK Cancel

total = 0 initially

Input
Enter Items (or 'quit')
[OK Cancel]

Input
Enter Items (or 'quit')
sugar

Input
Enter Items (or 'quit')
milk

Input
Enter Items (or 'quit')

Input
Enter Items (or 'quit')
quuit

Message
Amount Due: $4.75

OK Cancel

Terminate
Under the Hood: Objects vs. Primitive Types

Technical Question: Why does Java treat primitive types and objects differently?
All primitive types use a predictable amount of storage (e.g. all int variables use 4 bytes).
But...objects of the same type may involve vastly different amounts of storage. Suppose we change a string's value:

```java
String s = "Schultzie";
s = "this might be a ridiculously long string...";
```

If s contained the actual characters, we would need to allocate additional memory for the new string. This would disrupt all the variables around s. By storing a reference in s, we can simply change the reference.

Under the Hood: Assignment and Aliasing

Just for Strings:
```java
String t = "von Wienerschnitzel";
```
is convenient shorthand for what Java really does:
```java
String t = new String("von Wienerschnitzel" );
```

Aliasing: When one object reference is assigned to another, the memory address is copied, but not the object itself.
```java
String u = t;  // copies the address (reference)
String w = new String(t );  // creates a new string initialized to t
```
The variable u is effectively an alias for t, that is, a different name for the same object.

Under the Hood: '==' and 'equals' Revisited

Start with the initializations:
```java
String t = new String("von Wienerschnitzel" );
String u = t;  // copies the reference (address)
String w = new String(t );  // creates a new string initialized to t
```
Consider now the difference between == and equals().

`==`: Tests whether the references (addresses) are equal.
`equals()`: Tests whether the contents are equal.
```java
if ( u == t ) ...  // true: u and t refer to the same object
if ( w == t ) ...  // false: w and t refer to different objects
if ( u.equals(w) ) // true: u's and w's contents are equal
```
The “null” Reference

Must every reference refer to an instance of some object?
No: it is possible to have a reference refer to nothing. The special reference null does this.
(Let u and t be as before.)

```java
    u = null;       // now u does not refer to any object
    int i = t.length(); // okay: returns 19
    int j = u.length(); // Error! null pointer exception
    System.out.println(u); // Error? prints the word "null"
```

Note: There is difference between a null reference and an empty string
(sometimes called a null string).

Under the Hood: Garbage Collection

Garbage: When we change an object reference, we may produce a chunk of memory that
cannot be accessed, called garbage.

```java
    int x = 52;
    String s = new String("Schultzie");
    String t = new String("von Wienersnitzel");
    s = new String("Another string");
```

Garbage Collection: Garbage tends
to accumulate. When you start running out of memory, Java automatically does garbage
collection, to recover this space, so you don't run out of memory.