Quick review of last lecture

The String Class

- Because strings are so common, we don’t have to use the new operator to create a string object
  
  ```java
  title = "Java Software Solutions";
  ```
  
  - This is special syntax that works only for strings
  - Each string literal (enclosed in double quotes) represents a string object

String Methods

- Once a string object has been created, neither its value nor its length can be changed
- Thus we say that an object of the string class is immutable
- However, several methods of the string class return new string objects that are modified versions of the original
- See the list of string methods on page 119 and in Appendix M

String Indexes

- It is occasionally helpful to refer to a particular character within a string
- This can be done by specifying the character’s numeric index
- The indexes begin at zero in each string
- In the string “Hello”, the character ‘H’ is at index 0 and the ‘o’ is at index 4
- See StringMutation.java (page 120)
String Mutations Example

`mutation1 = phrase.concat(", except from vending machines.");`

`mutation2 = mutation1.toUpperCase();`

`mutation3 = mutation2.replace('E', 'X');`

`mutation4 = mutation3.substring(3, 30);`

Class Libraries

- A class library is a collection of classes that we can use when developing programs
- The Java standard class library is part of any Java development environment
- Its classes are not part of the Java language per se, but we rely on them heavily
- Various classes we've already used (System, Scanner, String) are part of the Java standard class library
- Other class libraries can be obtained through third party vendors, or you can create them yourself

The import Declaration

- When you want to use a class from a package, you could use its fully qualified name
  ```java`
  java.util.Scanner
  ```
- Or you can import the class, and then use just the class name
  ```java`
  import java.util.Scanner;
  ```
- To import all classes in a particular package, you can use the `*` wildcard character
  ```java`
  import java.util.*;
  ```
The import Declaration

- All classes of the `java.lang` package are imported automatically into all programs
- It's as if all programs contain the following line:
  
```java
import java.lang.*;
```
- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs
- The `Scanner` class, on the other hand, is part of the `java.util` package, and therefore must be imported

Where are the packages located?

- C:\Program Files\Java\jdk1.5.0\src.zip
- The zip file contains all libraries that ship with the Java language.

Can you add new packages?

Create a directory `c:\some_path\ISU`

In that directory save the file `Cyclone.java`

At the top of `Cyclone.java` put:

```java
package ISU;
```

Compile 'Cyclone.java' but don’t run it.

Set your CLASSPATH to `c:\some_path\`

Cyclone.java

```java
package ISU;
public class Cyclone {
    private String msg;
    public Cyclone (String message)
    {
        msg=message;
    }
    public void printMessage ()
    {
        System.out.println(msg);
    }
}
```

TestCyclone.java

```java
import ISU.Cyclone;
public class TestCyclone {
    public static void main(String[] args)
    {
        Cyclone cy=new Cyclone("Go Cyclones!");
        cy.printMessage();
    }
}
```
The Random Class

- The Random class is part of the java.util package
- It provides methods that generate pseudorandom numbers
- A Random object performs complicated calculations based on a seed value to produce a stream of seemingly random values

Random Numbers Example

- See RandomNumbers.java (page 126)

The Math Class

- The Math class is part of the java.lang package
- The Math class contains methods that perform various mathematical functions
- These include:
  - absolute value
  - square root
  - exponentiation
  - trigonometric functions

Math Class

- The methods of the Math class are static methods (also called class methods)
- Static methods can be invoked through the class name – no object of the Math class is needed
  ```java
  value = Math.cos(90) + Math.sqrt(delta);
  ```
- See Quadratic.java (page 129)
- We discuss static methods further in Chapter 6
Run examples from the book

Chapter 3
Sections 3.6 - 3.8

Formatting Output

- Two new classes
  - DecimalFormat
  - NumberFormat

NumberFormat Example

double dollars=5.994;
NumberFormat fmt = NumberFormat.getCurrencyInstance();
System.out.println ( "Price = " + fmt.format(dollars) );

RESULT:
Price = $5.99

Methods in NumberFormat Class

String format (double number)
returns a string containing the specified number formatted according to
this object's pattern.

static NumberFormat getCurrencyInstance();
returns a NumberFormat object that represents a currency format for the
current locale.

static NumberFormat getPercentInstance();
returns a NumberFormat object that represents a percentage format for the
current locale.

Methods in DecimalFormat Class

DecimalFormat (String pattern)
Applies the specified pattern to this DecimalFormat object.

applyPattern (String pattern)
Applies the specified pattern to this DecimalFormat object.

String format (double number)
Returns a string containing the specified number formatted according to the
current pattern.
### DecimalFormat Example

```java
double miles = 0.5395;
DecimalFormat fmt = new DecimalFormat("0.###");
System.out.println("Miles = \{\}m", fmt.format(miles));
```

RESULT:

```
Miles = 0.540
```

### The printf Method

- Provided as a courtesy to C programmers
- `System.out.printf("ID: %5d\tName: %s", id, name);`

### The printf convention

- `%d` print an int argument in decimal
- `%ld` print a long int argument in decimal
- `%c` print a character
- `%s` print a string
- `%f` print a float or double argument
- `%e` same as `%f`, but use exponential notation
- `%g` use `%e` or `%f`, whichever is better
- `%x` print an int argument in hexadec (base 16)
- `%o` print an int argument in octadec (base 8)
- `%s` print a single `%`

### Wrapper Classes

- **The java.lang package contains wrapper classes** that correspond to each primitive type:

<table>
<thead>
<tr>
<th>Primitive Type</th>
<th>Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>void</td>
<td>Void</td>
</tr>
</tbody>
</table>

### Integer Class

```
Integer (int value)

Constructor: creates a new Integer object storing the specified value.

byte byteValue();
short shortValue();
int intValue();
long longValue();
float floatValue();
double doubleValue();
String toString();

static int parseInt(String str); Returns the int corresponding to the value stored in the specified string.
static String string(int num); static String toString(int num); Returns a string representation of the specified integer value in the corresponding base.
```

### Wrapper Classes

- The following declaration creates an **Integer object** which represents the integer 40 as an object

  ```java
  Integer age = new Integer(40);
  ```
- An **object of a wrapper class** can be used in any situation where a primitive value will not suffice
- For example, some objects serve as containers of other objects
- **Primitive values could not be stored in such containers, but wrapper objects could be**
Wrapper Classes

- Wrapper classes also contain static methods that help manage the associated type.
- For example, the `Integer` class contains a method to convert an integer stored in a String to an `int` value:
  ```java
  num = Integer.parseInt(str);
  ```
- The wrapper classes often contain useful constants as well.
- For example, the `Integer` class contains `MIN_VALUE` and `MAX_VALUE` which hold the smallest and largest `int` values.

Autoboxing

- **Autoboxing** is the automatic conversion of a primitive value to a corresponding wrapper object:
  ```java
  Integer obj;
  int num = 42;
  obj = num;
  ```
- The assignment creates the appropriate `Integer` object.
- The reverse conversion (called `unboxing`) also occurs automatically as needed.

Autoboxing Examples

```java
Integer obj1;
int num1 = 69;
obj1 = num1; // automatically creates an //integer object

Integer obj2 = new Integer(69);
int num2;
num2 = obj2; // automatically extracts //the int value
```

THE END