Objects

- An object has:
  - state - descriptive characteristics
  - behaviors - what it can do (or what can be done to it)
- The state of a bank account includes its account number and its current balance
- The behaviors associated with a bank account include the ability to make deposits and withdrawals
- Note that the behavior of an object might change its state

Classes

- An object is defined by a class
- A class is the blueprint of an object
- The class uses methods to define the behaviors of the object
- The class that contains the main method of a Java program represents the entire program
- A class represents a concept, and an object represents the embodiment of that concept
- Multiple objects can be created from the same class

Objects and Classes

A class (the concept)  An object (the realization)

- Bank Account
  - John’s Bank Account Balance: $5,257
  - Bill’s Bank Account Balance: $1,245,069
  - Mary’s Bank Account Balance: $16,833

Multiple objects from the same class

Inheritance

- One class can be used to derive another via inheritance
- Classes can be organized into hierarchies
Classes

- A class can contain data declarations and method declarations
  ```
  int size, weight;
  char category;
  ```

Bank Account Example

- `acct1`: `72354 acctNumber 102.56 balance "Ted Murphy"
- `acct2`: `69713 acctNumber 40.00 balance "Jane Smith"

Creating Objects

- A variable holds either a primitive type or a reference to an object
- A class name can be used as a type to declare an object reference variable
  ```
  String title;
  ```
- No object is created with this declaration
- An object reference variable holds the address of an object
- The object itself must be created separately

Creating Objects

- Generally, we use the `new` operator to create an object
  ```
  title = new String ("Java Software Solutions");
  ```
  This calls the String constructor, which is a special method that sets up the object
- Creating an object is called instantiation
- An object is an instance of a particular class

Invoking Methods

- We've seen that once an object has been instantiated, we can use the dot operator to invoke its methods
  ```
  count = title.length();
  ```
- A method may return a value, which can be used in an assignment or expression
- A method invocation can be thought of as asking an object to perform a service

References

- Note that a primitive variable contains the value itself, but an object variable contains the address of the object
- An object reference can be thought of as a pointer to the location of the object
- Rather than dealing with arbitrary addresses, we often depict a reference graphically
  ```
  num1 38
  name1 "Steve Jobs"
  ```
Assignment Revisited

• The act of assignment takes a copy of a value and stores it in a variable
• For primitive types:

  Before: num1 38
  num2 96
  num2 = num1;

  After: num1 38
  num2 38

Reference Assignment

• For object references, assignment copies the address:

  Before:
  name2 = name1;
  "Steve Jobs" "Steve Wozniak"
  name1
  name2

  After:
  name1
  name2

Aliases

• Two or more references that refer to the same object are called aliases of each other
• That creates an interesting situation: one object can be accessed using multiple reference variables
• Aliases can be useful, but should be managed carefully
• Changing an object through one reference changes it for all of its aliases, because there is really only one object

Garbage Collection

• When an object no longer has any valid references to it, it can no longer be accessed by the program
• The object is useless, and therefore is called garbage
• Java performs automatic garbage collection periodically, returning an object’s memory to the system for future use
• In other languages, the programmer is responsible for performing garbage collection

Storing an int

```
9278
9279
9280
9281
9282
9283
9284
9285
9286
```

int (32 bits = 4 bytes)

Reference Variables

```
1023
1024
1025
1026
1027
1028
1029
1030
1031
```

\[ \Rightarrow \]

```
2047
2048
2049
2050
2051
2052
2053
2054
2055
```
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 Class

- `String(String str)`: Constructs a new String object with the same characters as str.
- `char charAt(int index)`: Returns the character at the specified index.
- `int compareTo(String other)`: Returns a negative integer, zero, or a positive integer as this string is less than, equal to, or greater than the specified string.
- `String substring(int startIndex)`: Returns a new string consisting of this string, starting at startIndex.
- `String substring(int startIndex, int endIndex)`: Returns a new string that is a section of this string starting at startIndex and extending through (but not including) endIndex.
- `int length()`: Returns the number of characters in this string.
- `String toLowerCase()`: Returns a new string whose characters are the same as this string, but in lowercase.
- `String toUpperCase()`: Returns a new string whose characters are the same as this string, but in uppercase.
- `String trim()`: Returns a new string in which the whitespace at the beginning, or at the end, or both, has been trimmed.
- `void trimToSize()`: Modifies this string so that its capacity is reduced to the minimum size required.
- `void trimLeading()`: Modifies the beginning of this string so that any leading white space is removed.
- `void trimTrailing()`: Modifies the end of this string so that any trailing white space is removed.
- `String trim()`: Returns a new string version of this string that has all sequence of white space removed.
- `String trimToSize()`: Modifies this string so that its capacity is reduced to the minimum size required.
- `void trimLeading()`: Modifies the beginning of this string so that any leading white space is removed.
- `void trimTrailing()`: Modifies the end of this string so that any trailing white space is removed.
- `String trim()`: Returns a new string in which the white space is removed.

Run examples from the book
Chapter 3
Section 3.3

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

**Packages**

- The classes of the Java standard class library are organized into **packages**
- Some of the packages in the standard class library are:

```
<table>
<thead>
<tr>
<th>Package</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang</td>
<td>General support</td>
</tr>
<tr>
<td>java.applet</td>
<td>Creating applets for the web</td>
</tr>
<tr>
<td>java.awt</td>
<td>Graphics and graphical user interfaces</td>
</tr>
<tr>
<td>java.swing</td>
<td>Additional graphics capabilities</td>
</tr>
<tr>
<td>java.net</td>
<td>Network communication</td>
</tr>
<tr>
<td>java.util</td>
<td>Utilities</td>
</tr>
<tr>
<td>java.xml.parsers</td>
<td>XML document processing</td>
</tr>
</tbody>
</table>
```

**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.*;
  ```

**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\`

How to use it?

Put this line at the top of the file that uses your new package:
```java
import ISU.Cyclone;
```

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();
    }
}
```

HW3 is out

THE END