Objects, Classes, and Packages

• “Static” Classes
• Introduction to Classes
• Object Variables and Object References
• Instantiating Objects
• Using Methods in Objects
• Reading for this Lecture: L&L, 3.1 - 3.3
• Familiarize yourself with Sun Website as a reference for the Java Class Library
“Static” Classes

• A class that has static members only – both attributes and methods is a “static” class
• We do not instantiate any objects of that class
• We use the class name to access its members
• Examples:
  – Math class
    Math.sqrt(doubleValue) or Math.PI
  – QuadraticSolver class in Project 1
    QuadraticSolver.getSolution()
## "Static" Classes

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Math</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of its Variables</td>
<td>+ PI</td>
</tr>
<tr>
<td></td>
<td>(Other constants would be here)</td>
</tr>
<tr>
<td>List of its Methods</td>
<td>+ sqrt(value : double) : double</td>
</tr>
<tr>
<td></td>
<td>(Other methods would be here)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Name</th>
<th>QuadraticSolver</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of its Variables</td>
<td>Typically, there are no variables</td>
</tr>
<tr>
<td></td>
<td>There may be some class constants</td>
</tr>
<tr>
<td>List of its Methods</td>
<td>+ getEquation(a : int, b : int, c : int) : String</td>
</tr>
<tr>
<td></td>
<td>+ getSolution(a : int, b : int, c : int) : String</td>
</tr>
</tbody>
</table>
“Static” classes

• In such a class, we define attributes and methods including the reserved word static

• Examples:
  – In the Math class source code
    ```java
    public static final double PI = 3.14…..;
    public static double sqrt(double input) {  }
    ```
  – In the QuadraticSolver source code
    ```java
    public static String getEquation(int a, int b, int c) {  }
    public static String getSolution(int a, int b, int c) {  }
    ```
"Static" Classes

• Although this is a valid way to break a Java program into smaller pieces, it is not the true intent of Object-Oriented Programming (OOP)
• It is more like procedural programming (e.g. C)
• In true OOP, we:
  – Use classes to encapsulate data and use methods to define the valid operations on that data
  – Instantiate objects from the classes and access methods using object names – not class names
Introduction to Classes

• A class defines the attributes and behavior of a specific type of object
  – Attributes are variables declared in the class
  – Behaviors are methods defined in the class
• Normally, we access an object by calling a method defined by its class
• We may sometimes access an attribute defined by its class, but this is discouraged
“Classifying” into Classes

• To understand the context of the word “class” in Java, think about the word “classify”
• Classes “classify” different “objects” based on the similarities in attributes and behaviors
• The desks, chairs, and tables in this room can be classified as “Furniture” class objects
• There’s a sense of common attributes and behaviors that all “Furniture” objects share
Introduction to Classes

• A class has a name that we can use as if it were a data type when declaring a variable

• When we declare a variable with the name of a class as its type, we are creating a reference variable (It can contain a reference to an object)

• We access an object’s methods / attributes using the reference variable name and the . notation, e.g.

```java
ClassName objectName;  //reference variable
objectName.methodName() // Note the ( )
objectName.variableName // Note no ( )
```
Example of a Class Definition

- We can draw a diagram of a class to outline its important features before writing code – its name, attributes, and behaviors.

<table>
<thead>
<tr>
<th>Class Name</th>
<th>BankAccount</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of its Variables</td>
<td>- balance</td>
</tr>
<tr>
<td></td>
<td>. . .</td>
</tr>
<tr>
<td>List of its Methods</td>
<td>+ BankAccount (initial : double)</td>
</tr>
<tr>
<td></td>
<td>Note: Constructor</td>
</tr>
<tr>
<td></td>
<td>+ getBalance() : double</td>
</tr>
<tr>
<td></td>
<td>+ deposit(amount : double) : boolean</td>
</tr>
<tr>
<td></td>
<td>+ withdraw(amount : double) : boolean</td>
</tr>
</tbody>
</table>
Example of a Class Definition

```java
public class BankAccount {
    // an attribute or variable
    private double balance;

    // the constructor method
    public BankAccount(double initial) {
        balance = initial;
    }
}
```
Example of a Class Definition

// other behaviors or normal methods

public double getBalance() {
    return balance;
}

public boolean deposit(double amount) {
    balance += amount;
    return true;
}

// additional behaviors or methods

} // end of class definition
Creating Objects

• To declare a variable as a *reference* to a BankAccount object, we use the class name as the type name

  ```java
  BankAccount myAccount;
  ```

• This declaration does not create an object

• It only creates a reference variable that can hold a reference to a BankAccount object
Example of a Class Definition

• Declaring a BankAccount object:

```java
BankAccount myAccount =
    new BankAccount(100.00); //constructor
```

• Accessing other BankAccount methods:

```java
boolean status = myAccount.deposit(50.00);
double myMoney = myAccount.getBalance();
```

• Why can’t we just do this?

```java
myAccount.balance += 50.00;
```
Prototype for a Class Definition

• We use the Java reserved word *private* to prevent access to a variable or method from code that is written outside the class

• We use the Java reserved word *public* to allow access to a variable or method from code that is written outside the class

• Normally, we declare variables to be *private*

• Normally, we declare methods to be *public*

• We will see some valid exceptions later
Creating Objects

• We use the `new` operator to create an object

```java
BankAccount myAccount = new BankAccount(100.00);
```

• Creating an object is called `instantiation`

• An object is an `instance` of a particular class

• `myAccount` is assigned a reference to an object of type `BankAccount` that encapsulates the balance
Invoking Methods

• Once an object has been instantiated, we can use the *dot operator* to invoke or “call” any of the object’s methods

  ```java
  double myMoney = myAccount.getBalance();
  ```

• A method invocation can be thought of as:
  – Asking an object to perform a service  OR
  – Doing something to the state of the object
References

- A primitive variable contains the value itself, but a reference variable contains an object reference.

- An object reference can be thought of as a pointer to the location of the object in memory.

- Rather than dealing with arbitrary address values, we often depict a reference graphically.

```java
int num1 = 38;
BankAccount myAccount = new BankAccount();
myAccount.setBalance(100.00);
```

"Reference" (or Pointer)
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 reference:

**Before:**

```
myAccount $100.00
yourAccount $50.00
```

```java
if (myAccount == yourAccount)  // note use of ==
System.out.println("The Same");  // no
yourAccount = myAccount;
```

**After:**

```
myAccount $100.00
yourAccount $50.00
```

Garbage: See later slide

```java
if (myAccount == yourAccount)  
    System.out.println("The Same");  // yes
```
Aliases

• Two or more references that refer to the same object are called *aliases* of each other

• One object can be accessed using more than one reference variable

• Changing an object via one reference variable changes it for all of its aliases, because there is really only one object

• Aliases can be useful, but should be managed carefully (Do you want me to be able to withdraw money from your account? I doubt it!)
Garbage Collection

• When there are no longer any variables containing a reference to an object (e.g. the $50.00 on the earlier slide), the program can no longer access it

• The object is useless and is considered garbage

• Periodically, Java performs automatic garbage collection and returns an object's memory to the system for future use

• In other languages such as C/C++, the programmer must write explicit code to do the garbage collection
Garbage Collection

- Setting reference variable’s value to null, makes the object garbage (unavailable):

Before:    myAccount   $100.00

myAccount  = null;

After:     myAccount   null

No object

Garbage now $100.00
Garbage Collection

• If a reference variable’s value is equal to null, any reference to an attribute or method of that object will cause your program to fail.

```java
myAccount = new BankAccount(100.00);

System.out.println(myAccount.balance());  // OK

myAccount = null;   // $100 BankAccount => garbage

System.out.println(myAccount.balance());  // Fails
```