Packages & Random and Math Classes

September 5, 2007

ComS 207: Programming I (in Java)
Iowa State University, FALL 2007
Instructor: Alexander Stoytchev

HW3 is out

Demo of WebCT

Quick review of last lecture

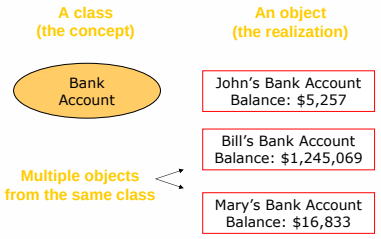
Objects

- 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

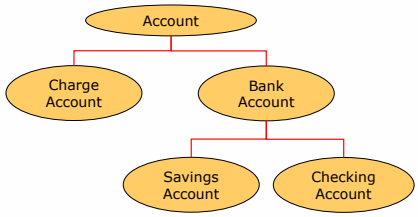
- An object is defined by a *class*
- A class is the blueprint of an object
- The class uses methods to define the behaviors of the object
- The class that contains the main method of a Java program represents the entire program
- A class represents a concept, and an object represents the embodiment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

Objects and Classes



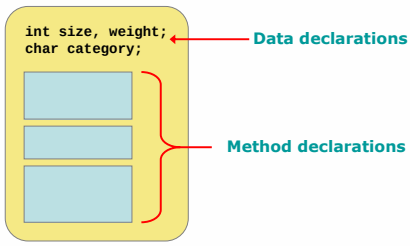
Inheritance

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

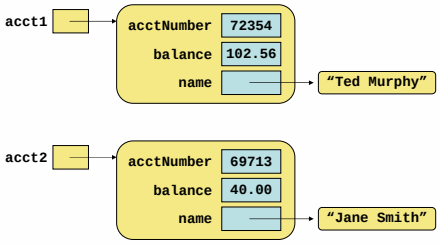


Classes

- A class can contain data declarations and method declarations



Bank Account Example



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

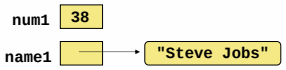
- We've seen that once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
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

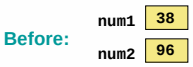
References

- 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

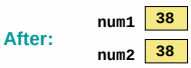


Assignment Revisited

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

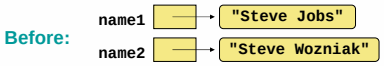



```
num2 = num1;
```

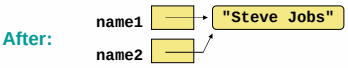


Reference Assignment

- For object references, assignment copies the address:




```
name2 = name1;
```



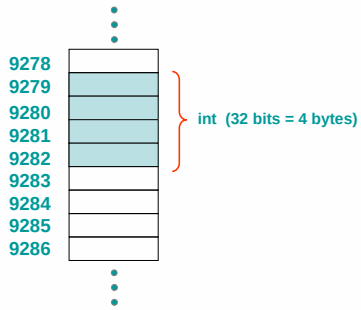
Aliases

- 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

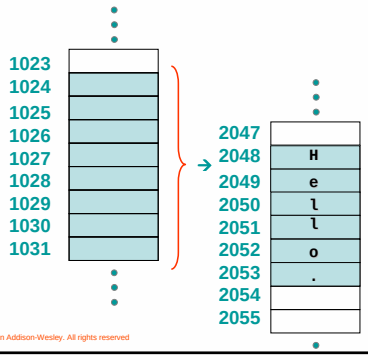
Garbage Collection

- 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



Reference Variables



Other Material from Sec 3.2

The String Class

- Because strings are so common, we don't have to use the new operator to create a `String` object

```
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)`
Constructor: creates a new string object with the same characters as `str`.

`char charAt (int index)`
Returns the character at the specified `index`.

`int compareTo (String str)`
Returns an integer indicating if this string is lexically before (a negative return value), equal to (a zero return value), or lexically after (a positive return value), the string `str`.

`String concat (String str)`
Returns a new string consisting of this string concatenated with `str`.

`boolean equals (String str)`
Returns true if this string contains the same characters as `str` (including case) and false otherwise.

`boolean equalsIgnoreCase (String str)`
Returns true if this string contains the same characters as `str` (without regard to case) and false otherwise.

`int length ()`
Returns the number of characters in this string.

`String replace (char oldChar, char newChar)`
Returns a new string that is identical with this string except that every occurrence of `oldChar` is replaced by `newChar`.

`String substring (int offset, int endIndex)`
Returns a new string that is a subset of this string starting at index `offset` and extending through `endIndex-1`.

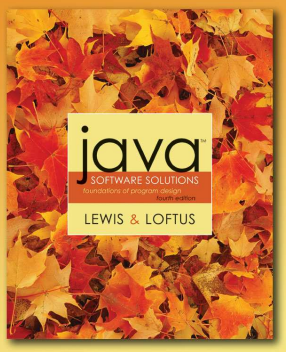
`String toLowerCase ()`
Returns a new string identical to this string except all uppercase letters are converted to their lowercase equivalent.

`String toUpperCase ()`
Returns a new string identical to this string except all lowercase letters are converted to their uppercase equivalent.

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:

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities
java.net	Network communication
java.util	Utilities
javax.xml.parsers	XML document processing

The import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Scanner
```

- Or you can *import* the class, and then use just the class name

```
import java.util.Scanner;
```

- To import all classes in a particular package, you can use the * wildcard character

```
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:

```
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:

```
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:

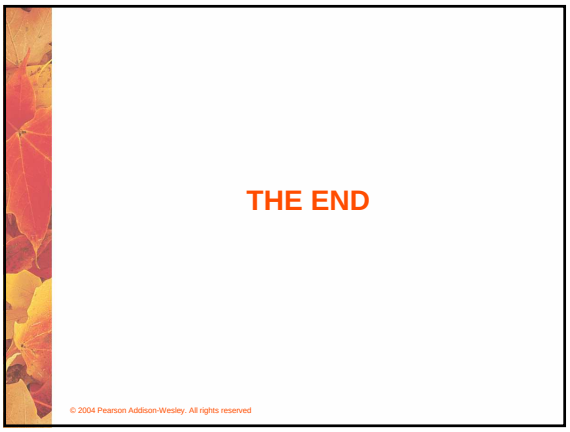
```
import ISU.Cyclone;
```


Cyclone.java

```
package ISU;

public class Cyclone
{
    private String msg;

    public Cyclone (String message)
    {
        msg=message;
    }

    public void printMessage ()
    {
        System.out.println(msg);
    }
}
```


TestCyclone.java

```
import ISU.Cyclone;

public class TestCyclone
{
   public static void main(String[] args)
   {
        Cyclone cy= new Cyclone("Go Cyclones!");

        cy.printMessage();
   }
}
```


THE END
© 2004 Pearson Addison-Wesley. All rights reserved