Creating Objects & String Class

September 1, 2006

ComS 207: Programming I (in Java)
Iowa State University, FALL 2006
Instructor: Alexander Stoytchev

Quick review of last lecture

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of count

```
count  =  count + 1;
```

Then the result is stored back into count (overwriting the original value)

Increment and Decrement

- The increment and decrement operators use only one operand
- The *increment operator* (++) adds one to its operand
- The *decrement operator* (--) subtracts one from its operand
- The statement

```
count++;
```

is functionally equivalent to

```
count = count + 1;
```


Increment and Decrement

- The increment and decrement operators can be applied in *postfix form*:

```
count++
```

- or *prefix form*:

```
++count
```

- When used as part of a larger expression, the two forms can have different effects
- Because of their subtleties, the increment and decrement operators should be used with care

Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Assignment Operators

- There are many assignment operators in Java, including the following:

Operator	Example	Equivalent To
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Widening Conversions

From	To
byte	short, int, long, float, or double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

FIGURE 2.5 Java widening conversions

Narrowing Conversions

From	To
byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float

FIGURE 2.6 Java narrowing conversions

Conversion Techniques

- 1) Assignment conversion
  - Value of one type is assigned to a variable of another type during which the value is converted to the new type.
- 2) Promotion
  - Occurs automatically when certain operators need to modify their operands.
- 3) Casting (a.k.a. type casting)
  - Specified explicitly by the programmer

Assignment conversion

```
float money;
int dollars;

dollars=5;

money = dollars;  // OK, money is now equal to 5.0

dollars= money;   //Compile error
```


(automatic) promotion

```
float sum, result;
int count;

sum= 12.0;
count=5;

result = sum/count;  // count promoted to float
                     // before the division
```

(automatic) promotion

```
// the number '5' is first promoted to a string and then
// the two strings are concatenated

System.out.printlf("Five is equal to " + 5);
```


Type Casting

```
float money;
int dollars;

dollars=5;

money = dollars;  // OK, money is now equal to 5.0

dollars= (int) money;   //Compile error OK
```


Type Casting + Promotion

```
float result;
int total, count;

total= 12;
count=5;

result =  (float) total / count;        //  result = 2.4
// 1. total is cast to float
// 2. count is promoted to float
// 3. the division is performed
```


Type Casting + Promotion

```
float result;
int total, count;

total= 12;
count=5;

result =  (float) (total / count);   //  result = 2.0
// 1. total and count a divided using integer division
// 2. the intermediary result is cast to a float
// 3. this float value is assigned to result
```


Scanner Class

```
Scanner (InputStream source)
Scanner (File source)
Scanner (String source)
    Constructors: sets up the new scanner to scan values from the specified source.

String next()
    Returns the next input token as a character string.

String nextLine()
    Returns all input remaining on the current line as a character string.

boolean nextBoolean()
byte nextByte()
double nextDouble()
float nextFloat()
int nextInt()
long nextLong()
short nextShort()
    Returns the next input token as the indicated type. Throws
    InputMismatchException if the next token is inconsistent with the type.

boolean hasNext()
    Returns true if the scanner has another token in its input.

Scanner useDelimiter (String pattern)
Scanner useDelimiter (Pattern pattern)
    Sets the scanner's delimiting pattern.

Pattern delimiter()
    Returns the pattern the scanner is currently using to match delimiters.

String findInLine (String pattern)
String findInLine (Pattern pattern)
    Attempts to find the next occurrence of the specified pattern, ignoring delimiters.
```

FIGURE 2.7 Some methods of the Scanner class

Reading Input

- The following line creates a Scanner object that reads from the keyboard:

```
Scanner scan = new Scanner (System.in);
```

- The new operator creates the Scanner object
- Once created, the Scanner object can be used to invoke various input methods, such as:

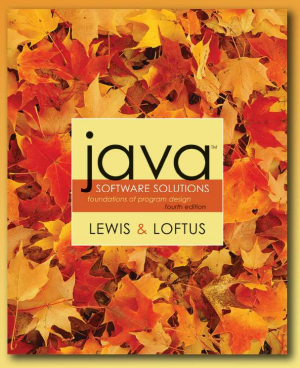
```
answer = scan.nextLine();
```

- In order to use the Scanner object you must put this line at the top of your Java program

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

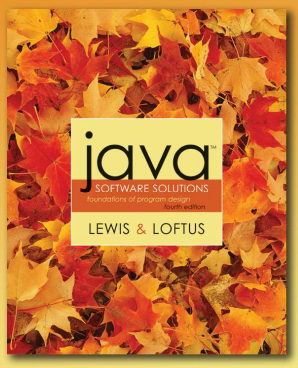
Chapter 3

Using Classes and Objects



Section 1.6

Sections 3.1 & 3.2



Problem Solving

- The purpose of writing a program is to solve a problem
- Solving a problem consists of multiple activities:
  - Understand the problem
  - Design a solution
  - Consider alternatives and refine the solution
  - Implement the solution
  - Test the solution
- These activities are not purely linear – they overlap and interact

Problem Solving

- The key to designing a solution is breaking it down into manageable pieces
- When writing software, we design separate pieces that are responsible for certain parts of the solution
- An *object-oriented approach* lends itself to this kind of solution decomposition
- We will dissect our solutions into pieces called objects and classes

Object-Oriented Programming

- Java is an object-oriented programming language
- As the term implies, an object is a fundamental entity in a Java program
- Objects can be used effectively to represent real-world entities
- For instance, an object might represent a particular employee in a company
- Each employee object handles the processing and data management related to that employee

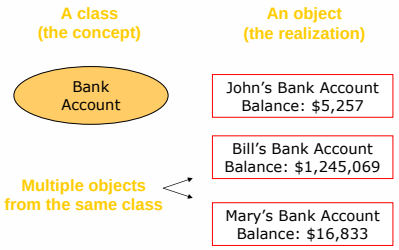
Objects

- 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

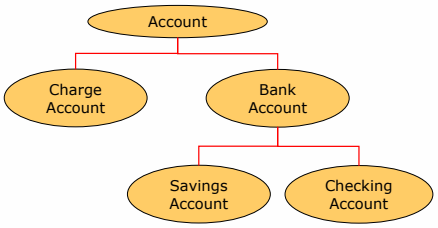
- An object is defined by a *class*
- A class is the blueprint of an object
- The class uses methods to define the behaviors of the object
- The class that contains the main method of a Java program represents the entire program
- A class represents a concept, and an object represents the embodiment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

Objects and Classes



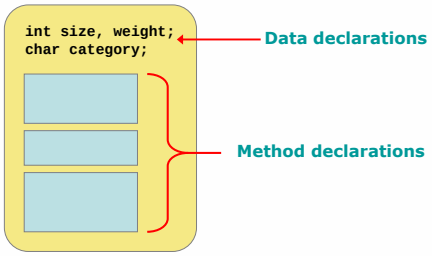
Inheritance

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

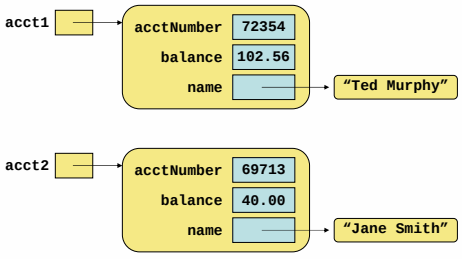


Classes

- 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

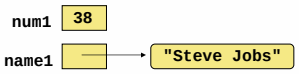
- We've seen that once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
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

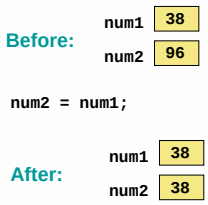
References

- 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



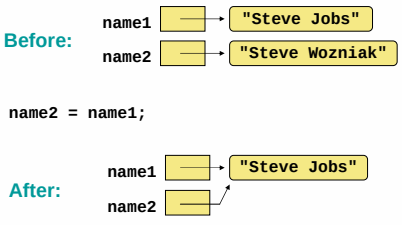
Assignment Revisited

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

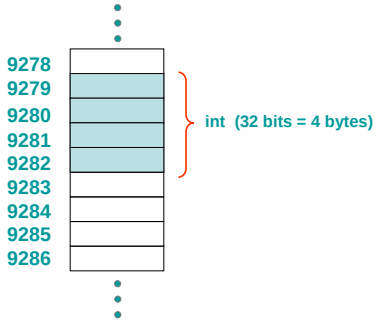


Reference Assignment

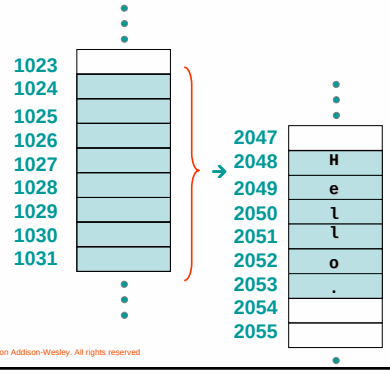
- For object references, assignment copies the address:



Storing an int



Reference Variables



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

Bank Example Code

THE END