More About Objects and Methods

Chapter 6
Writing Methods: Outline

• Case Study: Formatting Output
• Decomposition
• Addressing Compiler Concerns
• Testing Methods
Algorithm to display a double amount as dollars and cents

1. **dollars** = the number of whole dollars in amount.
2. **cents** = the number of cents in amount. Round if there are more than two digits after the decimal point.
3. Display a dollar sign, **dollars**, and a decimal point.
4. Display **cents** as a two-digit integer.
Formatting Output

• View sample code, listing 6.12
  class DollarFormatFirstTry
    ▪ Note code to separate dollars and cents
    ▪ Note if-else statement

• View sample program, listing 6.13
  class DollarFormatFirstTryDriver
    ▪ Note call to the write method
Testing DollarFormatFirstTry.write:
Enter a value of type double:
1.2345
$1.23
Test again?
yes
Enter a value of type double:
1.235
$1.24
Test again?
yes
Enter a value of type double:
9.02
$9.02
Test again?
yes
Enter a value of type double:
-1.20
$-1.0-20
Test again?
no

Oops. There's a problem here.
Formatting Output

• View corrected code, listing 6.14

```java
class DollarFormat
```
  ▪ Note code to handle negative values

• Program in listing 6.13 will now print values correctly
Decomposition

• Recall pseudocode from previous slide
• With this pseudocode we decompose the task into subtasks
  ▪ Then solve each subtask
  ▪ Combine code of subtasks
  ▪ Place in a method
Addressing Compiler Concerns

• Compiler ensures necessary tasks are done
  ▪ Initialize variables
  ▪ Include `return` statement

• Rule of thumb: believe the compiler
  ▪ Change the code as requested by compiler
  ▪ It is most likely correct
Testing Methods

• To test a method use a driver program
  ▪ Example – code in listing 6.13
• Every method in a class should be tested
• Bottom-up testing
  ▪ Test code at end of sequence of method calls first
• Use a stub – simplified version of a method for testing purposes
Overloading: Outline

• Overloading Basics
• Overloading and Automatic Type Conversion
• Overloading and the Return Type
• Programming Example: A Class for Money
Overloading Basics

• When two or more methods have same name within the same class
• Java distinguishes the methods by number and types of parameters
  ▪ If it cannot match a call with a definition, it attempts to do type conversions
• A method's name and number and type of parameters is called the *signature*
Overloading Basics

• View example program, listing 6.15

```java
class Overload {
    // Note overloaded method getAverage
    double getAverage(double a, double b) {
        return (a + b) / 2.0;
    }
    double getAverage(double a, double b, double c) {
        return (a + b + c) / 3.0;
    }
    double getAverage(double a, double b, double c, double d) {
        return (a + b + c + d) / 4.0;
    }
}
```

• Note overloaded method `getAverage`
Overloading and Type Conversion

• Overloading and automatic type conversion can conflict

• Recall definition of Pet class of listing 6.1
  ▪ If we pass an integer to the constructor we get the constructor for age, even if we intended the constructor for weight

• Remember the compiler attempts to overload before it does type conversion

• Use descriptive method names, avoid overloading
Overloading and Return Type

• You must not overload a method where the only difference is the type of value returned.

```java
/**
 * Returns the weight of the pet.
 */
public double getWeight()

/**
 * Returns '+' if overweight, '-' if underweight, and '* when weight is OK.
 */
public char getWeight()
```
Programming Example

• A class for money
• View sample class, listing 6.16

```java
class Money
```

• Note use of
  ▪ Private instance variables
  ▪ Methods to set values
  ▪ Methods for doing arithmetic operations
Programming Example

- View demo program, listing 6.17

```java
class MoneyDemo
```

Enter your current savings:
Enter amount on a line by itself:
$500.99
If you double that, you will have $1001.98, or better yet:
If you triple that original amount, you will have
$1502.97
Remember: A penny saved
is a penny earned.

Sample screen output
Information Hiding Revisited

Privacy Leaks

• Instance variable of a class type contain address where that object is stored

• Assignment of class variables results in two variables pointing to same object
  – Use of method to change *either* variable, changes the actual object itself

• View insecure class, listing 6.18

```java
class petPair
```
Information Hiding Revisited

- View sample program, listing 6.19

```java
class Hacker {
    public static void main(String[] args) {
        Our pair:
        First pet in the pair:
            Name: Faithful Guard Dog
            Age: 5 years
            Weight: 75.0 pounds
        Second pet in the pair:
            Name: Loyal Companion
            Age: 4 years
            Weight: 60.5 pounds
        Our pair now:
        First pet in the pair:
            Name: Dominion Spy
            Age: 1200 years
            Weight: 500.0 pounds
        Second pet in the pair:
            Name: Loyal Companion
            Age: 4 years
            Weight: 60.5 pounds
        The pet wasn’t so private!
        Looks like a security breach.
    }
}
```

Sample screen output:

This program has changed an object named by a private instance variable of the object pair.
Enumeration as a Class

• Consider defining an enumeration for suits of cards

```
enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES}
```

• Compiler creates a class with methods
  
  ▪ `equals`
  ▪ `compareTo`
  ▪ `ordinal`
  ▪ `toString`
  ▪ `valueOf`
Enumeration as a Class

- View enhanced enumeration, listing 6.20
  ```java
  enum Suit
  ```
- Note
  - Instance variables
  - Additional methods
  - Constructor
Packages: Outline

- Packages and Importing
- Package Names and Directories
- Name Clashes
Packages and Importing

• A package is a collection of classes grouped together into a folder
• Name of folder is name of package
• Each class
  ▪ Placed in a separate file
  ▪ Has this line at the beginning of the file
    package Package_Name;
• Classes use packages by use of import statement
Package Names and Directories

• Package name tells compiler path name for directory containing classes of package
• Search for package begins in class path base directory
  ▪ Package name uses dots in place of / or \n• Name of package uses relative path name starting from any directory in class path
Package Names and Directories

• Figure 6.5 A package name

myjavastuff

\myjavastuff\libraries

is a class path base directory
(is on the class path).

general

utilities

Classes in the package

AClass.java

AnotherClass.java

general.utilities

is the package name.
Name Clashes

• Packages help in dealing with name clashes
  ▪ When two classes have same name
• Different programmers may give same name to two classes
  ▪ Ambiguity resolved by using the package name
Summary

• Constructor method creates, initializes object of a class
• Default constructor has no parameters
• Within a constructor use this as name for another constructor in same class
• A static variable shared by all objects of the class
Summary

- Primitive type has wrapper class to allow treatment as an object
- Java performs automatic type cast between primitive type and object of wrapper class as needed
- Divide method tasks into subtasks
- Test all methods individually
Summary

• Methods with same name, different signatures are overloaded methods

• An enumeration is a class – can have instance variables, constructors, methods

• A package of class definitions grouped together in same folder, contain a package statement at beginning of each class