More About Objects and Methods

Chapter 6
Objectives

• Define and use constructors
• Write and use static variables and methods
• Use methods from class Math
• Use predefined wrapper classes
• Use stubs, drivers to test classes and programs
Objectives

• Write and use **overloaded** methods
• Define and use enumeration methods
• Define and use packages and **import** statements
Defining Constructors

• A special method called when instance of an object created with `new`
  • Create objects
  • Initialize values of instance variables

• Can have parameters
  • To specify initial values if desired

• May have multiple definitions
  • Each with different numbers or types of parameters
Defining Constructors

• Example class to represent pets

• Figure 6.1 Class Diagram for Class Pet

```
+ writeOutput(): void
+ setPet(String newName, int newAge, double newWeight): void
+ setName(String newName): void
+ setAge(int newAge): void
+ setWeight(double newWeight): void
+ getName(): String
+ getAge(): int
+ getWeight(): double
```
Defining Constructors

• **class Pet**

• Note different constructors
  • Default
  • With 3 parameters
  • With String parameter
  • With double parameter

• **class PetDemo**
Defining Constructors

My records on your pet are inaccurate. Here is what they currently say:
Name: Jane Doe
Age: 0
Weight: 0.0 pounds
Please enter the correct pet name:
Moon Child
Please enter the correct pet age:
5
Please enter the correct pet weight:
24.5
My updated records now say:
Name: Moon Child
Age: 5
Weight: 24.5 pounds
Calling Methods from Other Constructors

- Constructor can call other class methods

```java
public Pet(String initialName, int initialAge, double initialWeight)
{
    setPet(initialName, initialAge, initialWeight);
}
```

- View sample code, listing 6.3

```java
class Pet2
{
    // Note method setPet
    // Keeps from repeating code
}
```

JAVA: An Introduction to Problem Solving & Programming, 7th Ed. By Walter Savitch
Calling Constructor from Other Constructors

• From listing 6.3 we have the initial constructor and method set
• In the other constructors use the this reference to call initial constructor
• Use `this(<other constructor’s params>)` instead of `ClassName(<other constructor’s params>)`

• `class Pet3`
  • Note calls to initial constructor
Defining Constructors

• Constructor without parameters is the default constructor
  • Java will define this automatically if the class designer does not define any constructors
  • If you do define a constructor, Java will not automatically define a default constructor

• Usually default constructors not included in class diagram
Defining Constructors

- Figure 6.2 A constructor returning a reference

```java
Pet fish;
Assigns a memory location to fish
```

```java
fish = new Pet();
Assigns a chunk of memory for an object of the class Pet—that is, memory for a name, an age, and a weight—and places the address of this memory chunk in the memory location assigned to fish
```

```
<table>
<thead>
<tr>
<th>fish</th>
<th>Memory location assigned to fish</th>
</tr>
</thead>
<tbody>
<tr>
<td>fish</td>
<td>5432</td>
</tr>
</tbody>
</table>

```

```
The chunk of memory assigned to fish.name, fish.age, and fish.weight might have the address 5432.
```

```
<table>
<thead>
<tr>
<th>5432</th>
<th>Wanda</th>
</tr>
</thead>
<tbody>
<tr>
<td>5432</td>
<td>2</td>
</tr>
<tr>
<td>5432</td>
<td>0.25</td>
</tr>
</tbody>
</table>
```
Static Variables

• Static variables are shared by all objects of a class
  • Variables declared `static final` are considered constants – value cannot be changed

• Variables declared `static` (without `final`) can be changed
  • Only one instance of the variable exists
  • It can be accessed by all instances of the class

• Static variables also called `class variables`
  • Contrast with `instance/member variables`

• Do not confuse class variables with variables of a class type

• Both static variables and instance variables are sometimes called `fields` or `data members`
Static Methods

• Some methods may have no relation to any type of object
• Example
  • Compute max of two integers
  • Convert character from upper to lower case
• Static method declared in a class
  • Can be invoked without using an object
  • Instead use the class name (unnecessary if used in same class defined)
• Make a method static if the logic
  • Does not use instance variables
  • Or, a more specific version of the above, it just takes parameters and just returns a result
    • Example 1: a method that takes two parameters and returns the distance between them
    • Example 2: a utility method such as print() or println() such as used in class to avoid typing “System.out” every time for print-heavy programs
Static Methods

• View **sample class**, listing 6.5
  ```java
class DimensionConverter
```

• View **demonstration program**, listing 6.6
  ```java
class DimensionConverterDemo
```

Sample screen output

Enter a measurement in inches: 18
18.0 inches = 1.5 feet.
Enter a measurement in feet: 1.5
1.5 feet = 18.0 inches.
Tasks of \texttt{main} in Subtasks

- Program may have
  - Complicated logic
  - Repetitive code

- Create static methods to accomplish subtasks

- Must be static or will require an instance of the class to call – often does not make sense to create an instance of a class just to call a utility method or subtask
Adding Method **main** to a Class

- Method main used so far in its own class within a separate file
- Often useful to include method main within class definition
  - To create objects in other classes
  - To be run as a program
- Note **example code**, listing 6.11
  - a redefined **class Species**
    - When used as ordinary class, method **main** ignored
The **Math** Class

- Provides many standard mathematical methods
  - Automatically provided, no import needed
- Example methods, figure 6.3a

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Argument Type</th>
<th>Return Type</th>
<th>Example</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>pow</td>
<td>Power</td>
<td>double</td>
<td>double</td>
<td>Math.pow(2.0, 3.0)</td>
<td>8.0</td>
</tr>
<tr>
<td>abs</td>
<td>Absolute value</td>
<td>int, long, float, or double</td>
<td>Same as the type of the argument</td>
<td>Math.abs(-7)</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Math.abs(7)</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Math.abs(-3.5)</td>
<td>3.5</td>
</tr>
<tr>
<td>max</td>
<td>Maximum</td>
<td>int, long, float, or double</td>
<td>Same as the type of the arguments</td>
<td>Math.max(5, 6)</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Math.max(5.5, 5.3)</td>
<td>5.5</td>
</tr>
</tbody>
</table>
The **Math** Class

- Example methods, figure 6.3b

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Argument Type</th>
<th>Return Type</th>
<th>Example</th>
<th>Value Returned</th>
</tr>
</thead>
</table>
| min  | Minimum     | int, long, float, or double | Same as the type of the arguments | Math.min(5, 6)  
Math.min(5.5, 5.3) | 5  
5.3 |
| round | Rounding | float or double | int or long, respectively | Math.round(6.2)  
Math.round(6.8) | 6  
7 |
| ceil | Ceiling | double | double | Math.ceil(3.2)  
Math.ceil(3.9) | 4.0  
4.0 |
| floor | Floor | double | double | Math.floor(3.2)  
Math.floor(3.9) | 3.0  
3.0 |
| sqrt | Square root | double | double | sqrt(4.0) | 2.0 |
Random Numbers

- `Math.random()` returns a random double that is greater than or equal to zero and less than 1
- Java also has a `Random` class to generate random numbers
- Can scale using addition and multiplication; the following simulates rolling a six sided die
  ```java
  int die = (int) (6.0 * Math.random()) + 1;
  ```
Wrapper Classes

• Recall that arguments of primitive type treated differently from those of a class type
  • May need to treat primitive value as an object occasionally
• Java provides \textit{wrapper classes} for each primitive type
  • Methods provided to act on values (generally useful)
Wrapper Classes

• Allow programmer to have an object that corresponds to value of primitive type
• Contain useful predefined constants and methods
  • Double.MAX_VALUE
• Wrapper classes have no default constructor
  • Programmer must specify an initializing value when creating new object
• Wrapper classes have no `set` methods – they are immutable, like Strings
## Wrapper Classes

- **Figure 6.4a Static methods in class** `Character`

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Argument Type</th>
<th>Return Type</th>
<th>Examples</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toUpperCase</code></td>
<td>Convert to uppercase</td>
<td><code>char</code></td>
<td><code>char</code></td>
<td><code>Character.toUpperCase('a')</code> <code>Character.toUpperCase('A')</code></td>
<td>'A' 'A'</td>
</tr>
<tr>
<td><code>toLowerCase</code></td>
<td>Convert to lowercase</td>
<td><code>char</code></td>
<td><code>char</code></td>
<td><code>Character.toLowerCase('a')</code> <code>Character.toLowerCase('A')</code></td>
<td>'a' 'a'</td>
</tr>
<tr>
<td><code>isUpperCase</code></td>
<td>Test for uppercase</td>
<td><code>char</code></td>
<td><code>boolean</code></td>
<td><code>Character.isUpperCase('A')</code> <code>Character.isUpperCase('a')</code></td>
<td><code>true false</code></td>
</tr>
</tbody>
</table>
Wrapper Classes

- Figure 6.4b Static methods in class **Character**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Argument Type</th>
<th>Return Type</th>
<th>Examples</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>isLowerCase</td>
<td>Test for lowercase</td>
<td>char</td>
<td>boolean</td>
<td>Character.isLowerCase('A')</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Character.isLowerCase('a')</td>
<td>true</td>
</tr>
<tr>
<td>isLetter</td>
<td>Test for a letter</td>
<td>char</td>
<td>boolean</td>
<td>Character.isLetter('A')</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Character.isLetter('%')</td>
<td>false</td>
</tr>
<tr>
<td>isDigit</td>
<td>Test for a digit</td>
<td>char</td>
<td>boolean</td>
<td>Character.isDigit('5')</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Character.isDigit('A')</td>
<td>false</td>
</tr>
<tr>
<td>isWhitespace</td>
<td>Test for whitespace</td>
<td>char</td>
<td>boolean</td>
<td>Character.isWhitespace(' ')</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Character.isWhitespace('A')</td>
<td>false</td>
</tr>
</tbody>
</table>

Whitespace characters are those that print as white space, such as the blank, the tab character (\'\t\'), and the line-break character (\'\n\').
Writing Methods: Outline

- Case Study: Formatting Output
- Decomposition
- Addressing Compiler Concerns
- Testing Methods
Formatting Output

Algorithm to display a double amount as dollars and cents (corrected)

0. get total number of cents (rounded)

1. \texttt{dollars} = the number of whole dollars in total cents amount.

2. \texttt{cents} = the number of remaining cents in amount.
   Round if there are more than two digits after the decimal point.

3. Display a dollar sign, \texttt{dollars}, and a decimal point.

4. Display \texttt{cents} as a two-digit integer.
Formatting Output

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.
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 (or use unit tests...)
• Every method in a class should be tested – not just that a method was called but that all code paths tested too... still not perfect

• Bottom-up testing
  • Test lowest levels first
  • Failed tests at lower levels generally suggest to work on the lower level before the higher level

• Can do Top-Down also by using a stub – simplified version of a method for testing purposes – for the lower levels. Examples:
  • Stub for Database/Network: may just return answers without querying database or making internet connection
  • Stub for video camera input: may just read a given video
Overloading: Outline

• Overloading Basics
• Overloading and Automatic Type Conversion
• Overloading and the Return Type
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
  • return type is not considered
• A method's name and number and type of parameters is called the signature
Overloading Basics

• View example program, listing 6.15
  class Overload
• Note overloaded method `getAverage`

average1 = 45.0
average2 = 2.0
average3 = b
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 -- ambiguous
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
  class petPair
Information Hiding Revisited

- View sample program, listing 6.19

```java
class Hacker {
    public static void main(String[] args) {
        String firstPetName = "Faithful Guard Dog";
        int firstPetAge = 5;
        double firstPetWeight = 75.0;
        String secondPetName = "Loyal Companion";
        int secondPetAge = 4;
        double secondPetWeight = 60.5;
        System.out.println("Our pair:");
        System.out.println("First pet in the pair:");
        System.out.println("Name: "+ firstPetName);
        System.out.println("Age: "+ firstPetAge + " years");
        System.out.println("Weight: "+ firstPetWeight + " pounds");
        System.out.println("Second pet in the pair:");
        System.out.println("Name: "+ secondPetName);
        System.out.println("Age: "+ secondPetAge + " years");
        System.out.println("Weight: "+ secondPetWeight + " pounds");
        System.out.println("Our pair now:");
        System.out.println("First pet in the pair:");
        System.out.println("Name: " + firstPetName);
        System.out.println("Age: " + firstPetAge + " years");
        System.out.println("Weight: " + firstPetWeight + " pounds");
        System.out.println("Second pet in the pair:");
        System.out.println("Name: " + secondPetName);
        System.out.println("Age: " + secondPetAge + " years");
        System.out.println("Weight: " + secondPetWeight + " pounds");
        System.out.println("The pet wasn’t so private!");
        System.out.println("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
  
  ```java
  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
Figure 6.5 A 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