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
Constructors: Outline

• Defining Constructors
• Calling Methods from Constructors
• Calling a Constructor from Other Constructors
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`

```java
// Example class to represent pets
public class Pet {
    String name;
    int age;
    double weight;

    public void writeOutput() {
        // Method to output pet information
    }

    public void setPet(String newName, int newAge, double newWeight) {
        // Method to set pet information
    }

    public void setName(String newName) {
        // Method to set name
    }

    public void setAge(int newAge) {
        // Method to set age
    }

    public void setWeight(double newWeight) {
        // Method to set weight
    }

    public String getName() {
        // Method to get name
    }

    public int getAge() {
        // Method to get age
    }

    public double getWeight() {
        // Method to get weight
    }
}
```
Defining Constructors

• Note sample code, listing 6.1
  class Pet

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

• Note listing 6.2 class PetDemo
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
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

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

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

fish
Memory location assigned to fish

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

fish
5432
Wanda
2
0.25
```
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
```
  ```java
  ▪ Note method `set`
  ▪ Keeps from repeating code
  ```
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.
- View revised class, listing 6.4

```java
class Pet3
```
- Note calls to initial constructor.
Static Variables & Methods: Outline

• Static Variables
• Static Methods
• Dividing the Task of a main Method into Subtasks
• Adding a main Method to a class
• The Math Class
• Wrapper Classes
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

- Static variables also called *class variables*
  - Contrast with *instance 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
Static Methods

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

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

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.
Mixing Static and Nonstatic Methods

• View sample class, listing 6.7
class SavingsAccount
• View demo program, listing 6.8
class SavingsAccountDemo

Sample screen output

I deposited $10.75.
You deposited $75.
You deposited $55.
You withdrew $15.75.
You received interest.
Your savings is $115.3925
My savings is $10.75
We opened 2 savings accounts today.
Tasks of **main** in Subtasks

• Program may have
  ▪ Complicated logic
  ▪ Repetitive code

• Create static methods to accomplish subtasks

• Consider example code, listing 6.9
  a **main** method with repetitive code

• Note alternative code, listing 6.10
  uses helping methods
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), Math.abs(7), Math.abs(-3.5)</td>
<td>7, 7, 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), Math.max(5.5, 5.3)</td>
<td>6, 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>
<tbody>
<tr>
<td>min</td>
<td>Minimum</td>
<td>int, long, float, or double</td>
<td>Same as the type of the arguments</td>
<td>Math.min(5, 6), Math.min(5.5, 5.3)</td>
<td>5, 5.3</td>
</tr>
<tr>
<td>round</td>
<td>Rounding</td>
<td>float or double</td>
<td>int or long, respectively</td>
<td>Math.round(6.2), Math.round(6.8)</td>
<td>6, 7</td>
</tr>
<tr>
<td>ceil</td>
<td>Ceiling</td>
<td>double</td>
<td>double</td>
<td>Math.ceil(3.2), Math.ceil(3.9)</td>
<td>4.0, 4.0</td>
</tr>
<tr>
<td>floor</td>
<td>Floor</td>
<td>double</td>
<td>double</td>
<td>Math.floor(3.2), Math.floor(3.9)</td>
<td>3.0, 3.0</td>
</tr>
<tr>
<td>sqrt</td>
<td>Square root</td>
<td>double</td>
<td>double</td>
<td>sqrt(4.0)</td>
<td>2.0</td>
</tr>
</tbody>
</table>
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

• Java provides *wrapper classes* for each primitive type
  ▪ Methods provided to act on values
Wrapper Classes

• Allow programmer to have an object that corresponds to value of primitive type
• Contain useful predefined constants and methods
• Wrapper classes have no default constructor
  ▪ Programmer must specify an initializing value when creating a new object
• Wrapper classes have no set methods
### 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>toUpperCase</td>
<td>Convert to uppercase</td>
<td>char</td>
<td>char</td>
<td>Character.toUpperCase('a') Character.toUpperCase('A')</td>
<td>'A' 'A'</td>
</tr>
<tr>
<td>toLowerCase</td>
<td>Convert to lowercase</td>
<td>char</td>
<td>char</td>
<td>Character.toLowerCase('a') Character.toLowerCase('A')</td>
<td>'a' 'a'</td>
</tr>
<tr>
<td>isUpperCase</td>
<td>Test for uppercase</td>
<td>char</td>
<td>boolean</td>
<td>Character.isUpperCase('A') Character.isUpperCase('a')</td>
<td>true false</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 true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Character.isLowerCase('a')</td>
<td></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 false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Character.isLetter('%')</td>
<td></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 false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Character.isDigit('A')</td>
<td></td>
</tr>
<tr>
<td>isWhitespace</td>
<td>Test for whitespace</td>
<td>char</td>
<td>boolean</td>
<td>Character.isWhitespace(' ')</td>
<td>true false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Character.isWhitespace('A')</td>
<td></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').