Chapter 13

How to work with inner classes, enumerations, and documentation
Objectives

Applied

• Code an inner class.
• Convert an inner class to an anonymous inner class.
• Declare and use an enumeration.
• Add javadoc comments to classes.
• Use NetBeans to generate the API documentation for the classes of a project.
• Use a web browser to view the API documentation for the classes of a NetBeans project.
Objectives (cont.)

Knowledge

• Describe how an inner class relates to its outer class.
• Describe the difference between an inner class and an anonymous inner class.
• Differentiate between a member of an enumeration and a constant.
• Explain why you would want to add javadoc comments to a class.
An introduction to inner classes

```java
public class ClassName {
    // can contain instance variables and methods
    // can contain static variables and methods
    class InnerClassName {
        // can access variables and methods of OuterClass
        // can contain instance variables and methods
        // can't contain static variables or methods
    }
}
```
A GUI that displays a button

![Test Frame](Image)

Test Button
Three Swing classes to display a button

javax.swing.JFrame
javax.swing.JPanel
javax.swing.JButton

An AWT interface and class to handle a button click

java.awt.event.ActionListener
java.awt.event.ActionEvent

The method of the ActionListener interface

void actionPerformed(ActionEvent e);
A class that uses an inner class

package murach.test;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class TestFrame extends JFrame {

    public TestFrame() {
        // code that sets up the frame
        this.setTitle("Test Frame");
        this.setSize(400, 100);
        this.setLocationByPlatform(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    JPanel panel = new JPanel();
    this.add(panel);
A class that uses an inner class (cont.)

// code that creates button and adds listener
JButton button1 = new JButton("Test Button");
ActionListener listener = new ClickListener();
button1.addActionListener(listener);

// code that displays the frame
panel.add(button1);
this.setVisible(true);
}

// the inner class that implements the listener
class ClickListener implements ActionListener {

@Override
public void actionPerformed(ActionEvent e) {
    System.out.println("The button was clicked!");

}
} // end of inner class
}

} // end of the outer class
A class that uses an anonymous class

```java
public class TestFrame extends JFrame {

    public TestFrame() {
        // code that starts setting up the frame

        // code that creates button and adds listener
        JButton button1 = new JButton("Test Button");
        ActionListener listener = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("The button was clicked!");
            }
        };
        button1.addActionListener(listener);

        // code that finishes setting up the frame
    }
}
```
Another way to use an anonymous class

```java
public class TestFrame extends JFrame {

    public TestFrame() {
        // code that starts setting up the frame

        // code that creates button and adds listener
        JButton button1 = new JButton("Test Button");
        button1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button 1 clicked! ");
            }
        });

        // code that finishes setting up the frame
    }
}
```
The syntax for declaring an enumeration

```java
public enum EnumerationName {
    CONSTANT_NAME1, [CONSTANT_NAME2]...
}
```

An enumeration that defines three shipping types

```java
public enum ShippingType {
    UPS_NEXT_DAY, 
    UPS_SECOND_DAY, 
    UPS_GROUND
}
```

A statement that uses an enumeration

```java
ShippingType secondDay = ShippingType.UPS_SECOND_DAY;
```
A method that uses an enumeration as a parameter

```java
public static double getShippingAmount(ShippingType st) {
    double shippingAmount = 2.99;
    if (st == ShippingType.UPS_NEXT_DAY) {
        shippingAmount = 10.99;
    } else if (st == ShippingType.UPS_SECOND_DAY) {
        shippingAmount = 5.99;
    }
    return shippingAmount;
}
```

A statement that calls the method

```java
double shippingAmount =
    getShippingAmount(ShippingType.UPS_SECOND_DAY);  // GOOD

// double shippingAmount2 = getShippingAmount(1);  // BAD
```
Two methods of an enumeration constant

name()
ordinal()
How to add a method to an enumeration

```java
public enum ShippingType {
    UPS_NEXT_DAY,
    UPS_SECOND_DAY,
    UPS_GROUND;

    @Override
    public String toString() {
        String s = "";
        if (ordinal() == 0) {
            s = "UPS Next Day (1 business day)";
        } else if (this.ordinal() == 1) {
            s = "UPS Second Day (2 business days)";
        } else if (this.ordinal() == 2) {
            s = "UPS Ground (5 to 7 business days)";
        }
        return s;
    }
}
```
How to use an enumeration

```java
ShippingType ground = ShippingType.UPS_GROUND;
System.out.println("Shipping type: " +
    ground.toString());
```

Resulting output

```
Shipping type: UPS Ground (5 to 7 business days)
```
How to code a static import statement

    import static murach.business.ShippingType.*;

How to use an enumeration after a static import

    ShippingType ground = UPS_GROUND;
    System.out.println("Shipping type: " + ground.toString());
package murach.business;

import java.text.NumberFormat;

/**
 * The Product class represents a product and is used by
 * the LineItem and ProductDB classes.
 */
public class Product {
    private String code;
    private String description;
    private double price;

    /**
     * Creates a new Product with default values.
     */
    public Product() {
        code = "";
        description = "";
        price = 0;
    }
}
The Product class with javadoc comments (cont.)

/**
   * Sets the product code to the specified String.
   */
public void setCode(String code) {
   this.code = code;
}

/**
   * Returns a String that represents the product code.
   */
public String getCode() {
   return code;
}

...
Common HTML tags for javadoc comments

<code></code>

Common javadoc tags

@author
@version
@param
@return
The Product class with javadoc tags

```java
package murach.business;

import java.text.NumberFormat;

/**
 * The <code>Product</code> class defines
 * a product and is used by the <code>LineItem</code>
 * and <code>ProductDB</code> classes.
 * @author Joel Murach
 * @version 1.0.0
 */

public class Product {
    private String code;
    private String description;
    private double price;
}
```
The Product class with javadoc tags (cont.)

/**
 * Creates a <code>Product</code> with default values
 */
public Product() {
    code = "";
    description = "";
    price = 0;
}

The Product class with javadoc tags (cont.)

/**
 * Sets the product code.
 * @param code A <code>String</code> for the product code.
 */
public void.setCode(String code) {
    this.code = code;
}

/**
 * Gets the product code.
 * @return A <code>String</code> for the product code.
 */
public String.getCode() {
    return code;
}

...
The API documentation that’s generated