Chapter 4

How to code your own classes and methods

Objectives

Applied

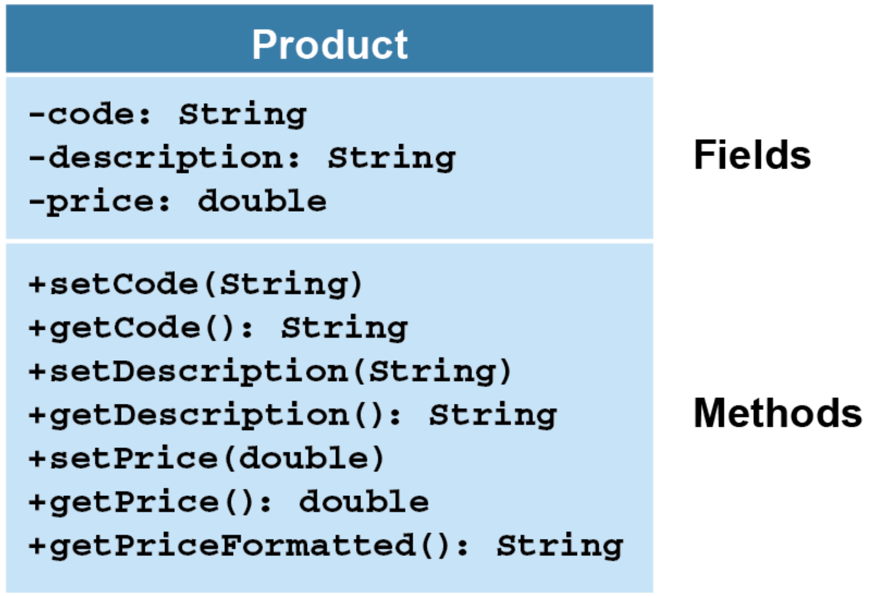
- Use NetBeans to create a new class.
- Code the instance variables, constructors, and methods of a class that defines an object.
- Write code that creates objects from a user-defined class and then uses the methods of the objects to perform tasks.
- Code a class that contains static fields and methods.
- Write code that calls static methods from a user-defined class.
- Write code that overloads a method.

Objectives (cont.)

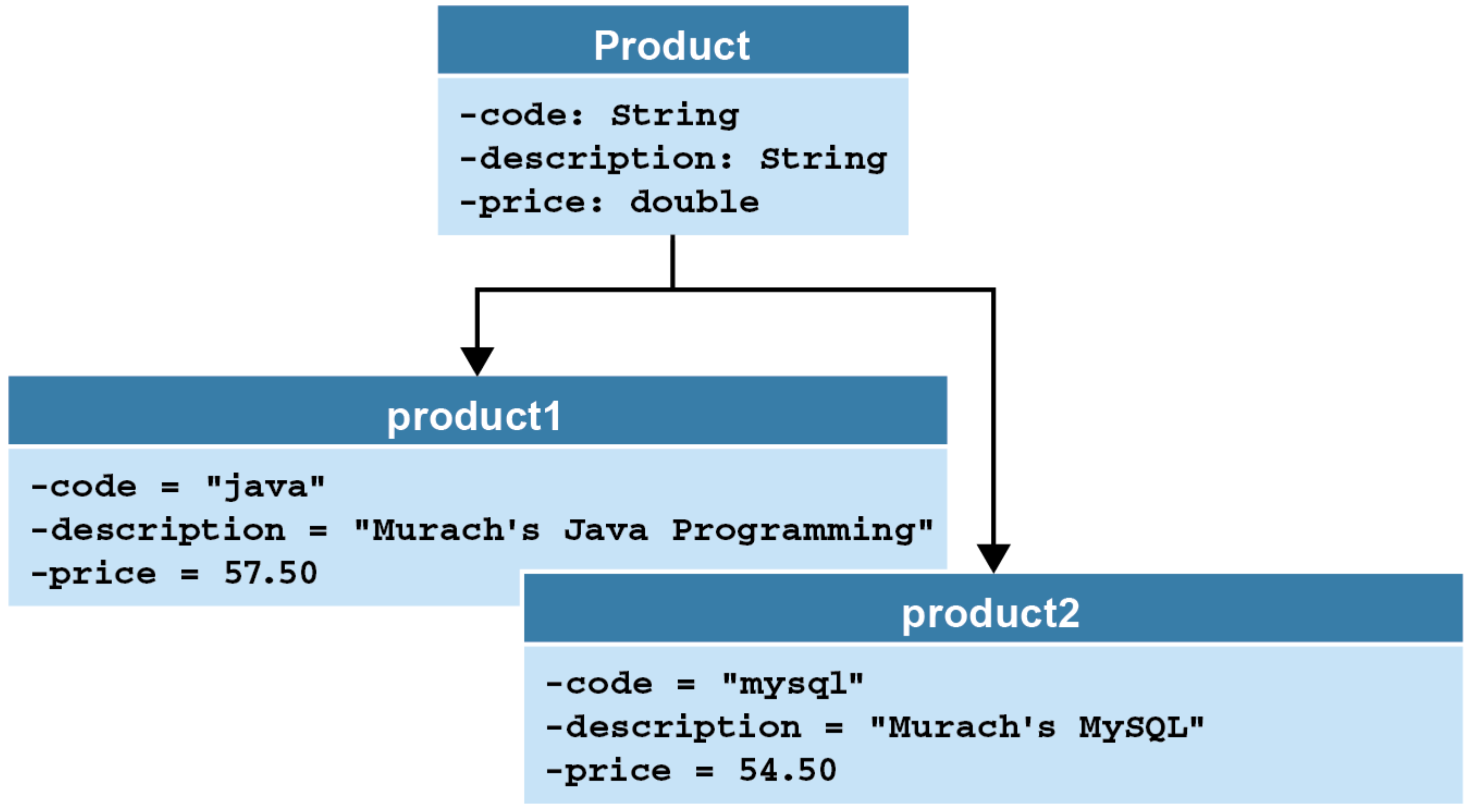
Knowledge

- Describe the concept of encapsulation and explain its importance to object-oriented programming.
- Differentiate between an object’s identity and its state.
- Explain what a default constructor is and when the Java compiler automatically creates one.
- Explain what an access modifier is and how it affects the methods of a class.
- Differentiate between a static method and a regular method.
- Differentiate between primitive types and reference types.
- List four ways you can use the this keyword within a class.

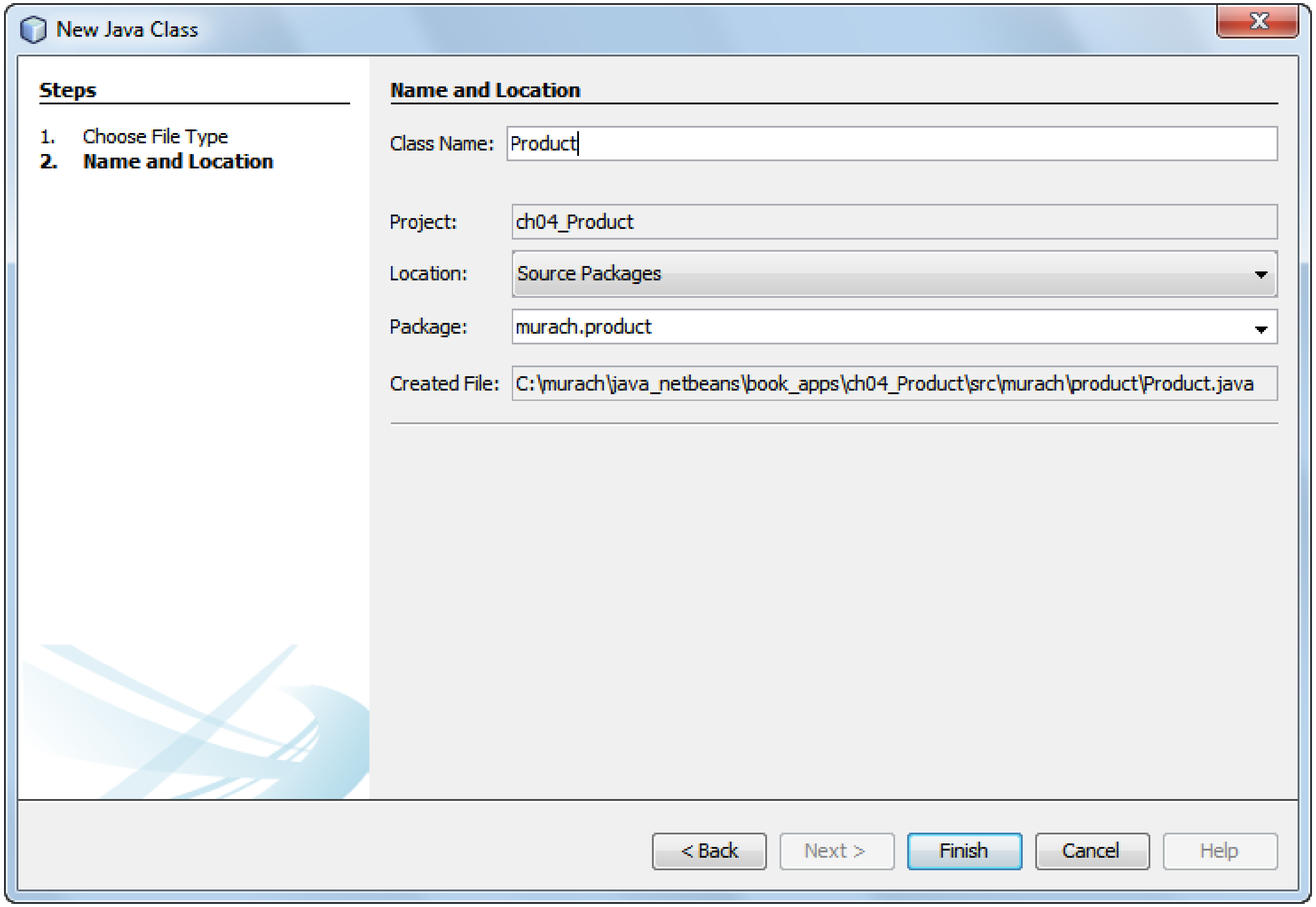
A class diagram for the Product class



The relationship between a class and its objects



The dialog box for creating a new Java class


New Java Class

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: Product

Project: ch04_Product

Location: Source Packages

Package: murach.product

Created File: C:\murach\java_netbeans\book_apps\ch04_Product\src\murach\product\Product.java

The code that’s generated for the Product class

```
package murach.product;

public class Product {

}
```

The Product class

```
package murach.product;

import java.text.NumberFormat;

public class Product {

    // the instance variables
    private String code;
    private String description;
    private double price;

    // the constructor
    public Product() {
        code = "";
        description = "";
        price = 0;
    }
```

The Product class (cont.)

```
// the set and get methods for the code variable
public void setCode(String code) {
    this.code = code;
}

public String getCode() {
    return code;
}

// the set and get methods for the description variable
public void setDescription(String description) {
    this.description = description;
}

public String getDescription() {
    return description;
}
```

The Product class (cont.)

```
    // the set and get methods for the price variable
    public void setPrice(double price) {
        this.price = price;
    }

    public double getPrice() {
        return price;
    }

    // a custom get method for the price variable
    public String getPriceFormatted() {
        NumberFormat currency =
            NumberFormat.getCurrencyInstance();
        String priceFormatted = currency.format(price);
        return priceFormatted;
    }
}
```

The syntax for declaring instance variables

```
public|private primitiveType|ClassName variableName;
```

Examples

```
private double price;
private int quantity;
private String code;
private Product product;
```

Where you can declare instance variables

```
public class Product {
    // typical to code instance variables here
    private String code;
    private String description;
    private double price;

    //the constructor and methods of the class
    public Product(){}
    public void setCode(String code){}
    public String getCode(){ return code; }
    public void setDescription(String description){}
    public String getDescription(){ return description; }
    public void setPrice(double price){}
    public double getPrice(){ return price; }
    public String getPriceFormatted(){ return priceFormatted; }

    // possible to code instance variables here
    private int test;
}
```

The syntax for coding constructors

```
public|private ClassName([parameterList]) {
        // the statements of the constructor
 }
```

A default (zero-parameter) constructor

```
public Product() {
    code = "";
    description = "";
    price = 0.0;
}
```

A constructor with three parameters

```
public Product(String code,
        String description, double price) {
    this.code = code;
    this.description = description;
    this.price = price;
}
```

Another way to code the same constructor

```
public Product(String productCode,
        String productDescription, double productPrice) {
    code = productCode;
    description = productDescription;
    price = productPrice;
}
```

The syntax for coding a method

```
public|private returnType methodName([parameterList]) {
   // the statements of the method
}
```

No parameters or return value

```
public void printToConsole() {
    System.out.println(
        code + "|" + description +  "|" + price);
}
```

A get method that returns a string

```
public String getCode() {
   return code;
}
```

A get method that returns a double value

```
public double getPrice() {
   return price;
}
```

A get method that returns a formatted string

```
public String getPriceFormatted() {
    NumberFormat currency =
        NumberFormat.getCurrencyInstance();
    String priceFormatted = currency.format(price);
    return priceFormatted;
}
```

A set method

```
public void setCode(String code) {
    this.code = code;
}
```

Another way to code the same set method

```
public void setCode(String productCode) {
    code = productCode;
}
```

How to create an object in two statements

Syntax

```
ClassName variableName;
variableName = new ClassName(argumentList);
```

No arguments

```
Product product;
product = new Product();
```

How to create an object in one statement

Syntax

```
ClassName variableName = new ClassName(argumentList);
```

No arguments

```
Product product = new Product();
```

Three arguments

```
Product product = new Product("java", "Murach's Java
Programming", 57.50);
```

The syntax for calling a method

```
objectName.methodName(argumentList)
```

A statement that sends no arguments and returns nothing

```
product.printToConsole();
```

A statement that sends one argument and returns nothing

```
product.setCode(productCode);
```

A statement that sends no arguments and returns a double value

```
double price = product.getPrice();
```

A statement that sends no arguments and returns a String object

```
String priceFormatted = product.getPriceFormatted();
```

A statement that calls a method within an expression

```
String message = "Code: " + product.getCode() + "\n";
```

The ProductDB class

```
package murach.product;

public class ProductDB {

   // static method
   public static Product getProduct(String productCode) {
       // create the Product object
       Product product = new Product();
```

The ProductDB class (cont.)

```
        // fill the Product object with data
        product.setCode(productCode);
        if (productCode.equalsIgnoreCase("java")) {
            product.setDescription(
                "Murach's Java Programming");
            product.setPrice(57.50);
        } else if (productCode.equalsIgnoreCase("jsp")) {
            product.setDescription(
                "Murach's Java Servlets and JSP");
            product.setPrice(57.50);
        } else if (productCode.equalsIgnoreCase("mysql")) {
            product.setDescription(
                "Murach's MySQL");
            product.setPrice(54.50);
        } else {
            product.setDescription("Unknown");
        }
        return product;
    }
}
```

How to code static methods and fields

```
package murach.product;

import java.util.Scanner;

public class Console {

    private static Scanner sc = new Scanner(System.in);
    public static String message;

    public static String getString(String prompt) {
        System.out.print(prompt);
        String s = sc.nextLine();
        return s;
    }
}
```

How to call static methods

Syntax

```
ClassName.methodName(argumentList)
```

A static method of the Console class

```
String productCode = Console.getString(
    "Enter the product code: ");
```

How to call static fields

Syntax

```
ClassName.fieldName
```

A static field of the Console class

```
Console.message = "This is a test.";    // set the field
String message = Console.message;       // get the field
```

The console

```
Welcome to the Product Viewer

Enter product code: java

PRODUCT
Code:         java
Description:  Murach's Beginning Java
Price:        $49.50

Continue? (y/n): n

Bye!
```

The ProductApp class

```
package murach.product;

import java.util.Scanner;

public class ProductApp {

    public static void main(String args[]) {
        // display a welcome message
        System.out.println("Welcome to the Product Viewer");
        System.out.println();

        // create 1 or more line items
        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            // get input from user
            System.out.print("Enter product code:  ");
            String productCode = sc.nextLine();
```

The ProductApp class (cont.)

```
            // get the Product object
            Product product = ProductDB.getProduct(productCode);

            // display the output
            String message = "\nPRODUCT\n" +
                "Code:          " + product.getCode() + "\n" +
                "Description: " + product.getDescription() + "\n" +
                "Price:         " + product.getPriceFormatted() + "\n";
            System.out.println(message);

            // see if the user wants to continue
            System.out.print("Continue? (y/n): ");
            choice = sc.nextLine();
            System.out.println();
        }
        System.out.println("Bye!");
    }
}
```

How assignment statements work

For primitive types

```
double p1 = 54.50;
double p2 = p1;         // p1 and p2 store copies of 54.50
p2 = 57.50;             // only changes p2
```

For reference types

```
Product p1 = new Product("mysql", "Murach's MySQL", 54.50);
Product p2 = p1;             // p1 and p2 refer to the same object
p2.setPrice(57.50);          // changes p1 and p2
```

How parameters work

For primitive types

```
public static double increasePrice(double price) {
    // the price parameter is a copy of the double value
    price = price * 1.1;
                 // does not change price in calling code
    return price;
                 // returns changed price to calling code
}
```

For reference types

```
public static void increasePrice(Product product) {
    // the product parameter refers to the Product object
    double price = product.getPrice() * 1.1;
    product.setPrice(price);
               // changes price in calling code
}
```

How method calls work

For primitive types

```
double price = 54.50;
price = increasePrice(price);  // assignment necessary
```

For reference types

```
Product product = new Product();
product.setPrice(54.50);
increasePrice(product);       // assignment not necessary
```

A signature that has one parameter

```
public void printToConsole(String separator) {
    System.out.println(code + separator + description +
       separator + price);
}
```

A signature that doesn't have any parameters

```
public void printToConsole() {
    printToConsole("|");
                  // call the method in the first example
}
```

A signature that has two parameters

```
public void printToConsole(String separator,
                           String label) {
    System.out.print(label);    // print label to console
    printToConsole(separator);  // call first example
}
```

Code that calls the printToConsole method

```
Product p = new Product(
     "java", "Murach's Java Programming", 57.50);

p.printToConsole();          // use the default separator
p.printToConsole("/");       // use a non-default separator
p.printToConsole("|", "Product: ");  // include a label
```

The console

```
java|Murach's Java Programming|57.5
java/Murach's Java Programming/57.5
Product: java|Murach's Java Programming|57.5
```

How to refer to instance variables of the current object

Syntax

```
this.variableName
```

Example

```
public Product(String code, String description,
               double price) {
    this.code = code;
    this.description = description;
    this.price = price;
}
```

How to call a constructor of the current object

Syntax

```
this(argumentList);
```

Example

```
public Product() {
    this("", "", 0.0);
}
```

How to call a method of the current object

Syntax

```
this.methodName(argumentList)
```

Example

```
public String getPriceFormatted() {
    NumberFormat currency =
        NumberFormat.getCurrencyInstance();
    String priceFormatted =
        currency.format(this.getPrice());
    return priceFormatted;
}
```

How to pass the current object to a method

Syntax

```
methodName(this)
```

Example

```
public void printCurrentObject() {
    System.out.println(this);
}
```

The Product class with overloading

```
package murach.product;

import java.text.NumberFormat;

public class Product {

    private String code;
    private String description;
    private double price;

    public Product() {
        this("", "", 0);
    }

    public Product(String code, String description,
                   double price) {
        this.code = code;
        this.description = description;
        this.price = price;
    }
```

The Product class with overloading (cont.)

```
public void printToConsole() {
        printToConsole("|");
    }

    public void printToConsole(String separator) {
        System.out.println(
            code + separator + description + separator + price);
    }

    public void printToConsole(String separator, String label) {
        System.out.print(label);
        printToConsole(separator);
    }
}
```