Chapter 7

How to define and use classes

Objectives

Applied

1. Code the instance variables, constructors, and methods of a class that defines an object.
2. Code a class that creates objects from a user-defined class and then uses the methods of the objects to accomplish the required tasks.
3. Code a class that contains static fields and methods, and call these fields and methods from other classes.
4. Given the Java code for an application that uses any of the language elements presented in this chapter, explain what each statement in the application does.

Objectives (cont.)

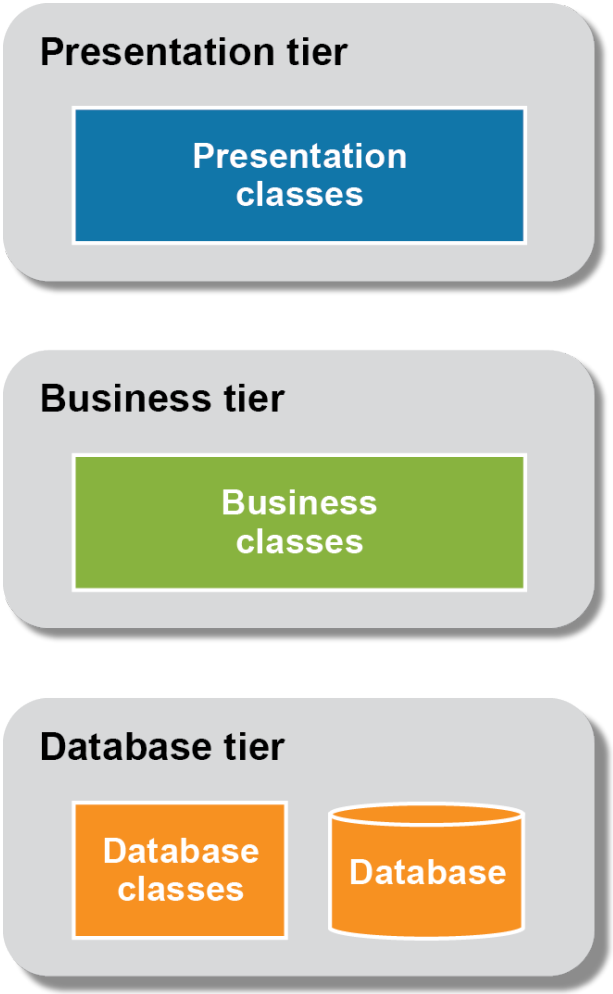
Knowledge

1. Describe the architecture commonly used for object-oriented programs.
2. Describe the concept of encapsulation and explain its importance to object-oriented programming.
3. Differentiate between an object's identity and its state.
4. Describe the basic components of a class.
5. Explain what an instance variable is.
6. Explain when a default constructor is used and what it does.
7. Describe a signature of a constructor or method, and explain what overloading means.
8. List four ways you can use the this keyword within a class definition.

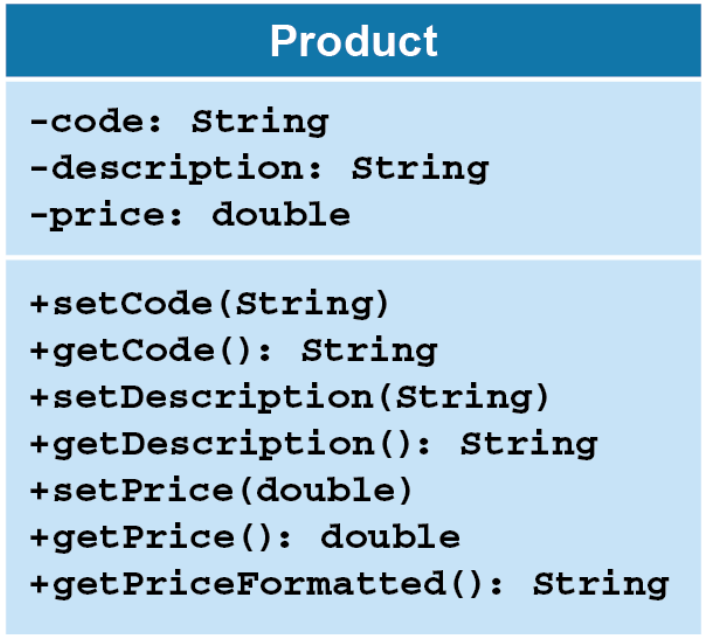
Objectives (cont.)

9. Explain the difference between how primitive types and reference types are passed to a method.
10. Explain how static fields and static methods differ from instance variables and regular methods.
11. Explain what a static initialization block is and when it's executed.

The architecture of a three-tiered application



A class diagram for the Product class



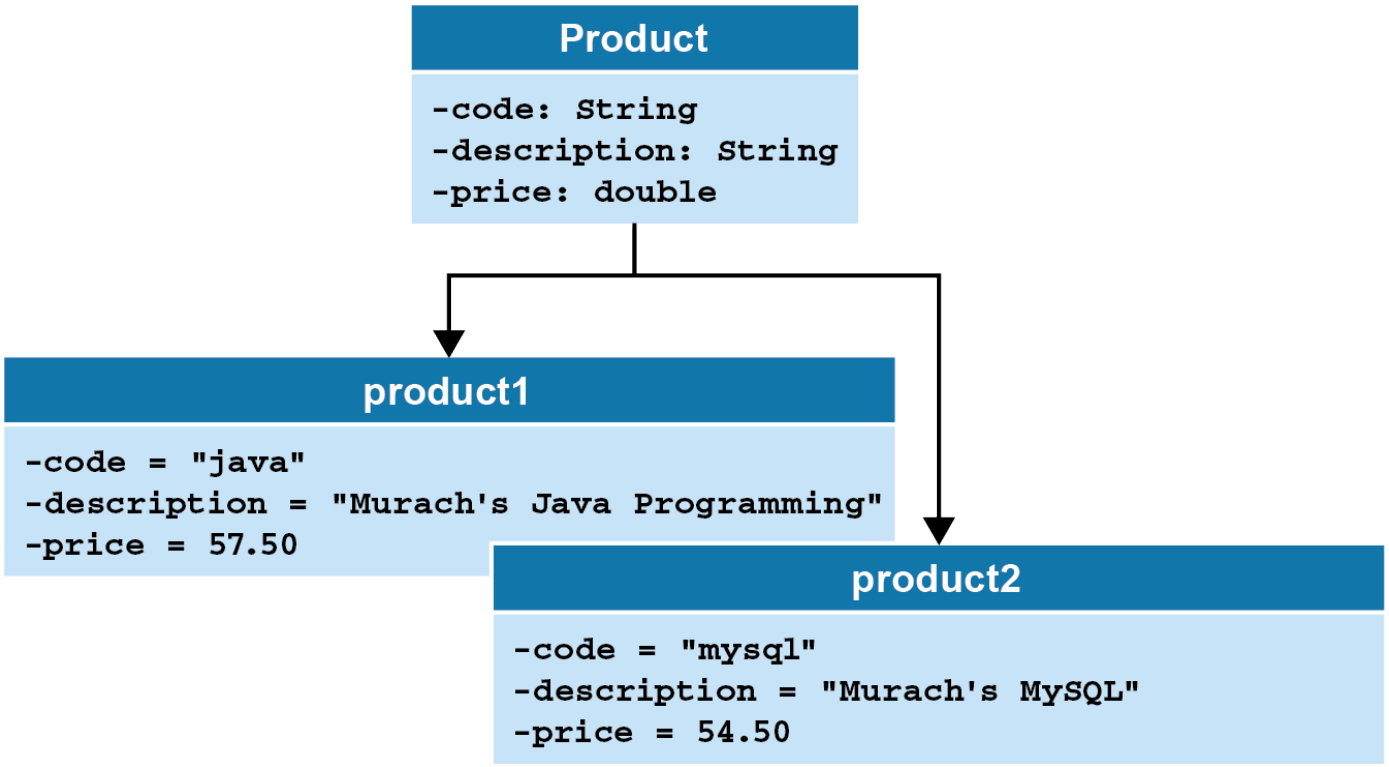
Fields

Methods

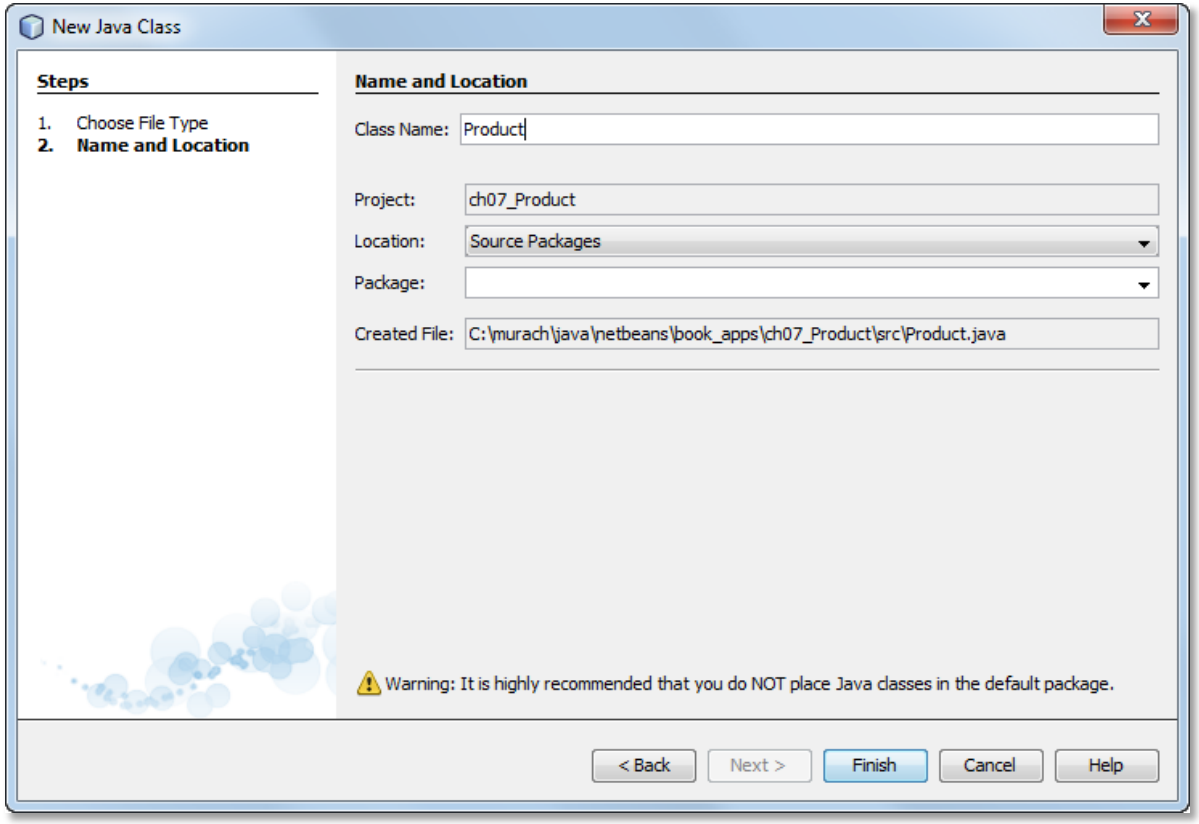
UML diagramming notes

- *UML* (*Unified Modeling Language*) is the industry standard used to describe the classes and objects of an object-oriented application.
- The minus sign (**-**) in a UML *class diagram* marks the fields and methods that can't be accessed by other classes, while the plus sign (**+**) marks the fields and methods that can be accessed by other classes.
- For each field, the name is given, followed by a colon, followed by the data type.
- For each method, the name is given, followed by a set of parentheses. If a method requires parameters, the data type of each parameter is listed in the parentheses. Otherwise, the parentheses are left empty, and the data type of the value that's going to be returned is given after the colon.

The relationship between a class and its objects



The dialog box for creating a new Java class



The code that's generated for the Product class

```
public class Product {
}
```

The Product class

```
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();
        return currency.format(price);
    }
}
```

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 {

    // common to code instance variables here
    private String code;
    private String description;
    private double price;

    // the constructors 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 formattedPrice; }

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

The syntax for coding constructors

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

A constructor that assigns default values

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

A custom 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 this constructor

```
public Product(String c, String d, double p) {
    code = c;
    description = d;
    price = p;
}
```

A constructor with one parameter

```
public Product(String code) {
    this.code = code;
    Product p = ProductDB.getProduct(code);
    description = p.getDescription();
    price = p.getPrice();
}
```

The syntax for coding a method

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

A method that doesn’t accept parameters or return data

```
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 custom get method

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

A set method

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

Another way to code a set method

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

How to create an object in two statements

Syntax

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

Example with 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();
```

One literal argument

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

One variable argument

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

Three arguments

```
Product product = new Product(code, description, price);
```

How to call a method

Syntax

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

Passes no arguments and returns nothing

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

Passes one argument and returns nothing

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

Passes no arguments and returns a double value

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

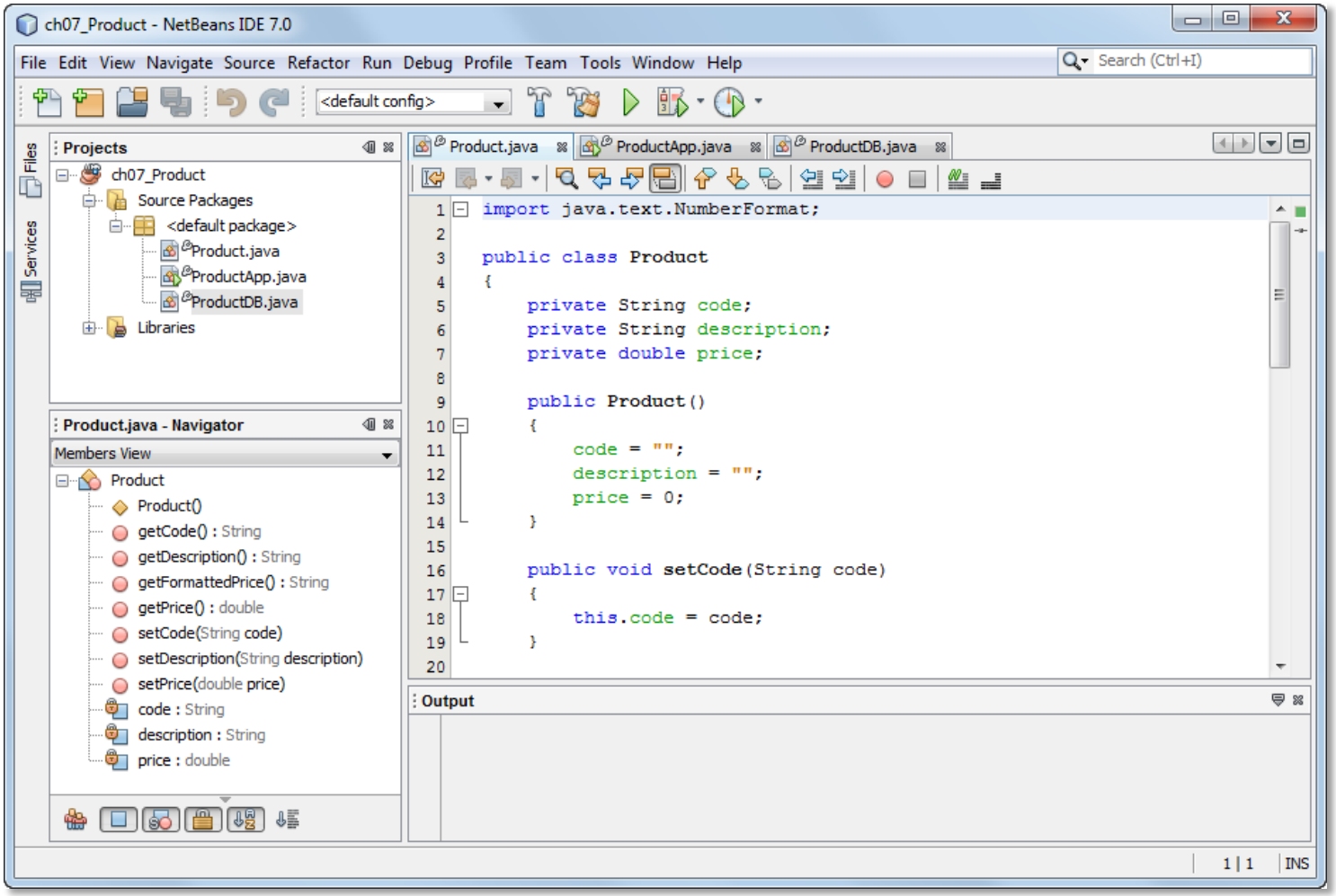
Passes an argument and returns a String object

```
String formattedPrice =
    product.getPriceFormatted(includeDollarSign);
```

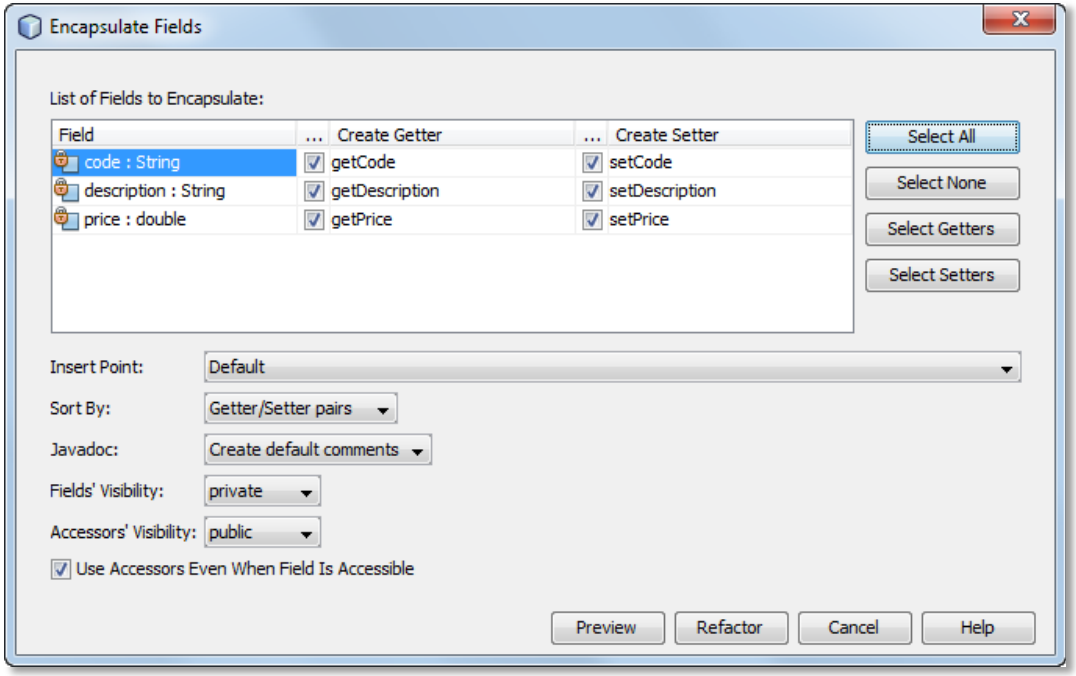
A method call within an expression

```
String message = "Code: " + product.getCode() + "\n\n" +
    "Press Enter to continue or enter 'x' to exit:";
```

The NetBeans window for the Product application



Dialog box for generating get and set methods



How to declare static fields

```
private static int numberOfObjects = 0;
private static double majorityPercent = .51;
public static final int DAYS_IN_JANUARY = 31;
public static final float EARTH_MASS_IN_KG = 5.972e24F;
```

A class with a static constant and a static method

```
public class FinancialCalculations {

    public static final int MONTHS_IN_YEAR = 12;

    public static double calculateFutureValue(
            double monthlyPayment,
            double yearlyInterestRate, int years) {
        int months = years * MONTHS_IN_YEAR;
        double monthlyInterestRate =
            yearlyInterestRate/MONTHS_IN_YEAR/100;
        double futureValue = 0;
        for (int i = 1; i <= months; i++)
            futureValue =
                (futureValue + monthlyPayment) *
                (1 + monthlyInterestRate);
        return futureValue;
    }
}
```

The Product class with a static variable and a static method

```
public class Product {

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

    // declare a static variable
    private static int objectCount = 0;

    public Product() {
        code = "";
        description = "";
        price = 0;
        objectCount++;     // update the static variable
    }
```

The Product class with a static variable and a static method (cont.)

```
    // get the static variable
     public static int getObjectCount() {
         return objectCount;
     }
    ...
}
```

The syntax for calling a static field or method

```
className.FINAL_FIELD_NAME
className.fieldName
className.methodName(argumentList)
```

How to call static fields

From the Java API

```
Math.PI
```

From a user-defined class

```
FinancialCalculations.MONTHS_IN_YEAR
Product.objectCount   // if objectCount is declared
                      // as public
```

How to call static methods

From the Java API

```
NumberFormat currency =
    NumberFormat.getCurrencyInstance();
int quantity = Integer.parseInt(inputQuantity);
double rSquared = Math.pow(r, 2);
```

From a user-defined class

```
double futureValue =
    FinancialCalculations.calculateFutureValue(
        monthlyPayment, yearlyInterestRate, years);
int productCount = Product.getObjectCount();
```

Call a static field and a static method

```
// pi times r squared
double area = Math.PI * Math.pow(r, 2);
```

The syntax for coding a static initialization block

```
public class className {
    // any field declarations
    static {
        // any initialization statements
        // for static fields
    }
    // the rest of the code for the class
}
```

A class with a static initialization block

```
public class ProductDB {

    // a static variable
    private static Connection connection;

    // the static initialization block
    static {
        try {
            String url =
                "jdbc:mysql://localhost:3306/MurachDB";
            String user = "root";
            String password = "sesame";
            connection =
                DriverManager.getConnection(
                    url, user, password);
        }
        catch (Exception e) {
            System.err.println(
                "Error connecting to database.");
        }
    }
```

A class with a static initialization block (cont.)

```
    // static methods that use the Connection object
    public static Product get(String code){...}
    public static boolean add(Product product){...}
    public static boolean update(Product product){...}
    public static boolean delete(String code){...}
}
```

The ProductDB class

```
public class ProductDB  {

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

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

The ProductDB class (cont.)

```
        else if (productCode.equalsIgnoreCase("mysql")) {
            p.setDescription("Murach's MySQL");
            p.setPrice(54.50);
        }
        else {
            p.setDescription("Unknown");
        }
        return p;
    }
}
```

The console for the Product Viewer application

```
Welcome to the Product Viewer

Enter product code: java

SELECTED PRODUCT
Description: Murach's Java Programming
Price:       $57.50

Continue? (y/n):
```

The ProductApp class

```
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();

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

The ProductApp class (cont.)

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

            // display the output
            System.out.println();
            System.out.println("SELECTED PRODUCT");
            System.out.println("Description: " +
                product.getDescription());
            System.out.println("Price:       " +
                product.getPriceFormatted());
            System.out.println();

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

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 method that accepts one argument

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

An overloaded method that provides a default value

```
public void printToConsole() {
    printToConsole("|");  // calls the first method above
}
```

An overloaded method with two arguments

```
public void printToConsole(String sep, boolean
printLineAfter) {
    printToConsole(sep);  // calls the first method above
    if (printLineAfter)
        System.out.println();
}
```

Code that calls the printToConsole() methods

```
Product p = ProductDB.getProduct("java");
p.printToConsole();
p.printToConsole("    ", true);
p.printToConsole("    ");
```

The console

```
java|Murach's Java Programming 2|57.5
java    Murach's Java Programming  57.5

java    Murach's Java Programming  57.5
```

How to refer to instance variables of the current object

Syntax

```
this.variableName
```

A constructor that refers to three instance variables

```
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);
```

A constructor that calls another constructor of the current object

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

How to call a method of the current object

Syntax

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

A method that calls another method of the current object

```
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)
```

A method that passes the current object to another method

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

The console for the Line Item application

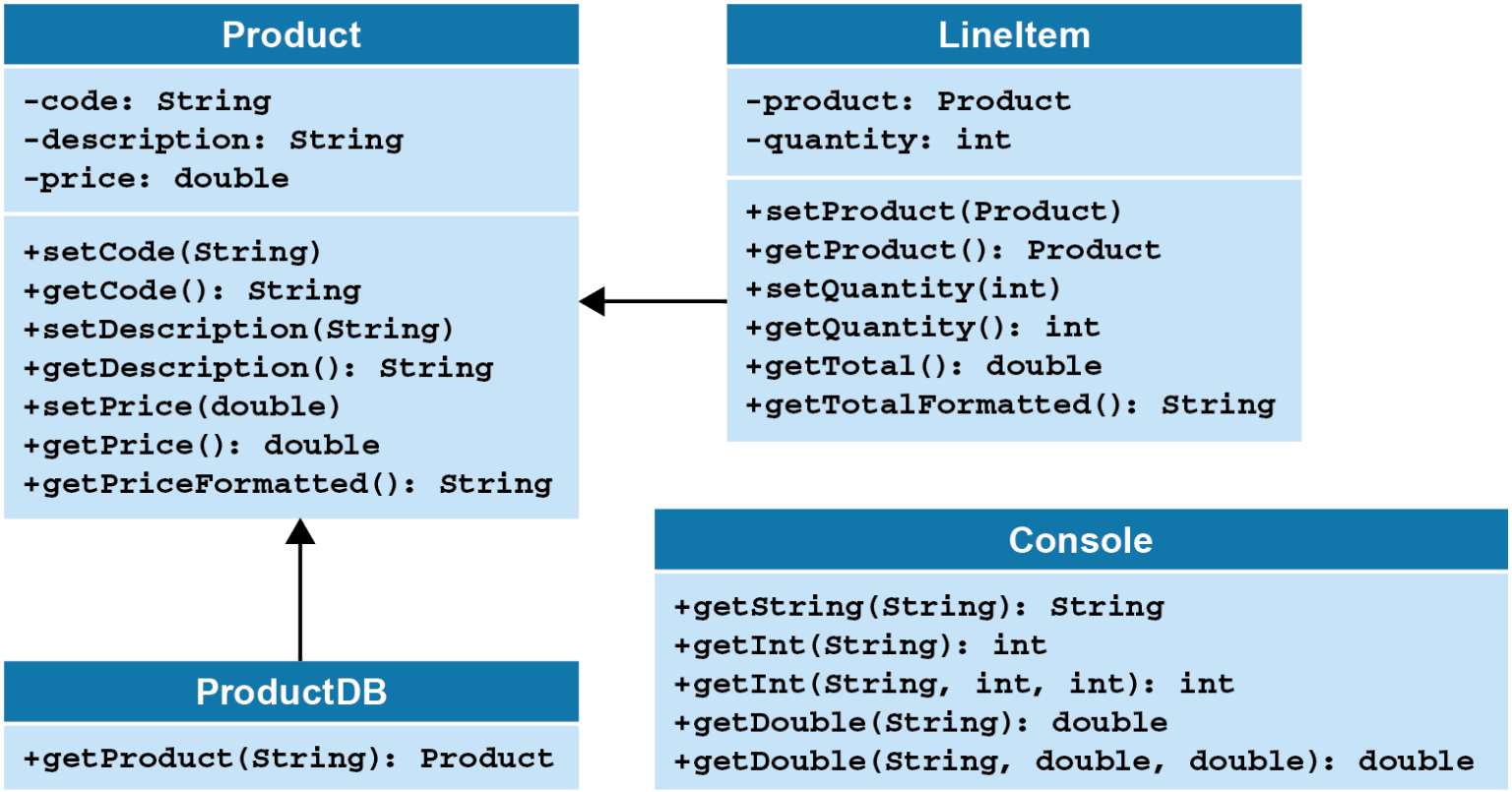
```
Welcome to the Line Item Calculator

Enter product code: java
Enter quantity:     2

LINE ITEM
Code:        java
Description: Murach's Java Programming
Price:       $57.50
Quantity:    2
Total:       $115.00

Continue? (y/n):
```

The class diagram for the Line Item application



The LineItemApp class

```
public class LineItemApp {

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

        // create 1 or more line items
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            // get the input from the user
            String productCode = Console.getString(
                "Enter product code: ");
            int quantity = Console.getInt(
                "Enter quantity:     ", 0, 1000);
```

The LineItemApp class (cont.)

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

// create the LineItem object
LineItem lineItem =
    new LineItem(product, quantity);

// display the output
System.out.println();
System.out.println("LINE ITEM");
System.out.println("Code:        " +
    product.getCode());
System.out.println("Description: " +
    product.getDescription());
System.out.println("Price:       " +
    product.getPriceFormatted());
System.out.println("Quantity:    " +
    lineItem.getQuantity());
System.out.println("Total:       " +
    lineItem.getTotalFormatted() + "\n");
```

The LineItemApp class (cont.)

```
            // see if the user wants to continue
            choice =
                Console.getString("Continue? (y/n): ");
            System.out.println();
        }
    }
}
```

The Console class

```
import java.util.Scanner;

public class Console {

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

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

The Console class (cont.)

```
public static int getInt(String prompt) {
    int i = 0;
    boolean isValid = false;
    while (!isValid) {
        System.out.print(prompt);
        if (sc.hasNextInt()) {
            i = sc.nextInt();
            isValid = true;
        } else {
            System.out.println(
                "Error! Invalid integer. Try again.");
        }
        sc.nextLine();
    }
    return i;
}
```

The Console class (cont.)

```
public static int getInt(String prompt, int min,
        int max) {
    int i = 0;
    boolean isValid = false;
    while (isValid == false) {
        i = getInt(sc, prompt);
        if (i <= min) {
            System.out.println(
                "Error! Number must be greater than " +
                min + ".");
        } else if (i >= max) {
            System.out.println(
                "Error! Number must be less than " +
                max + ".");
        } else {
            isValid = true;
        }
    }
    return i;
}
```

The Console class (cont.)

```
public static double getDouble(String prompt) {
    double d = 0;
    boolean isValid = false;
    while (!isValid) {
        System.out.print(prompt);
        if (sc.hasNextDouble()) {
            d = sc.nextDouble();
            isValid = true;
        } else {
            System.out.println(
                "Error! Invalid number. Try again.");
        }
        sc.nextLine();
    }
    return d;
}
```

The Console class (cont.)

```
    public static double getDouble(String prompt,
            double min, double max) {
        double d = 0;
        boolean isValid = false;
        while (!isValid) {
            d = getDouble(sc, prompt);
            if (d <= min) {
                System.out.println(
                    "Error! Number must be greater than " +
                    min + ".");
            } else if (d >= max) {
                System.out.println(
                    "Error! Number must be less than " +
                    max + ".");
            } else {
                isValid = true;
            }
        }
        return d;
    }
}
```

The LineItem class

```
import java.text.NumberFormat;

public class LineItem {

    private Product product;
    private int quantity;

    public LineItem() {
        this.product = null;
        this.quantity = 0;
    }

    public LineItem(Product product, int quantity) {
        this.product = product;
        this.quantity = quantity;
    }
```

The LineItem class (cont.)

```
public void setProduct(Product product) {
    this.product = product;
}

public Product getProduct() {
    return product;
}

public void setQuantity(int quantity) {
    this.quantity = quantity;
}

public int getQuantity() {
    return quantity;
}
```

The LineItem class (cont.)

```
    public double getTotal() {
        double total = product.getPrice() * quantity;
        return total;
    }

    public String getTotalFormatted() {
        NumberFormat currency =
            NumberFormat.getCurrencyInstance();
        return currency.format(this.getTotal());
    }
}
```