Chapter 8

How to work with inheritance

Objectives

Applied

1. Write the code for a class that inherits another class and overrides one or more of its methods.
2. In a user-defined class, write code that overrides the toString() method of the Object class to control the String object that's returned by this method.
3. In a user-defined class, write code that overrides the equals() method of the Object class to control how the user-defined class tests for equality.
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.)

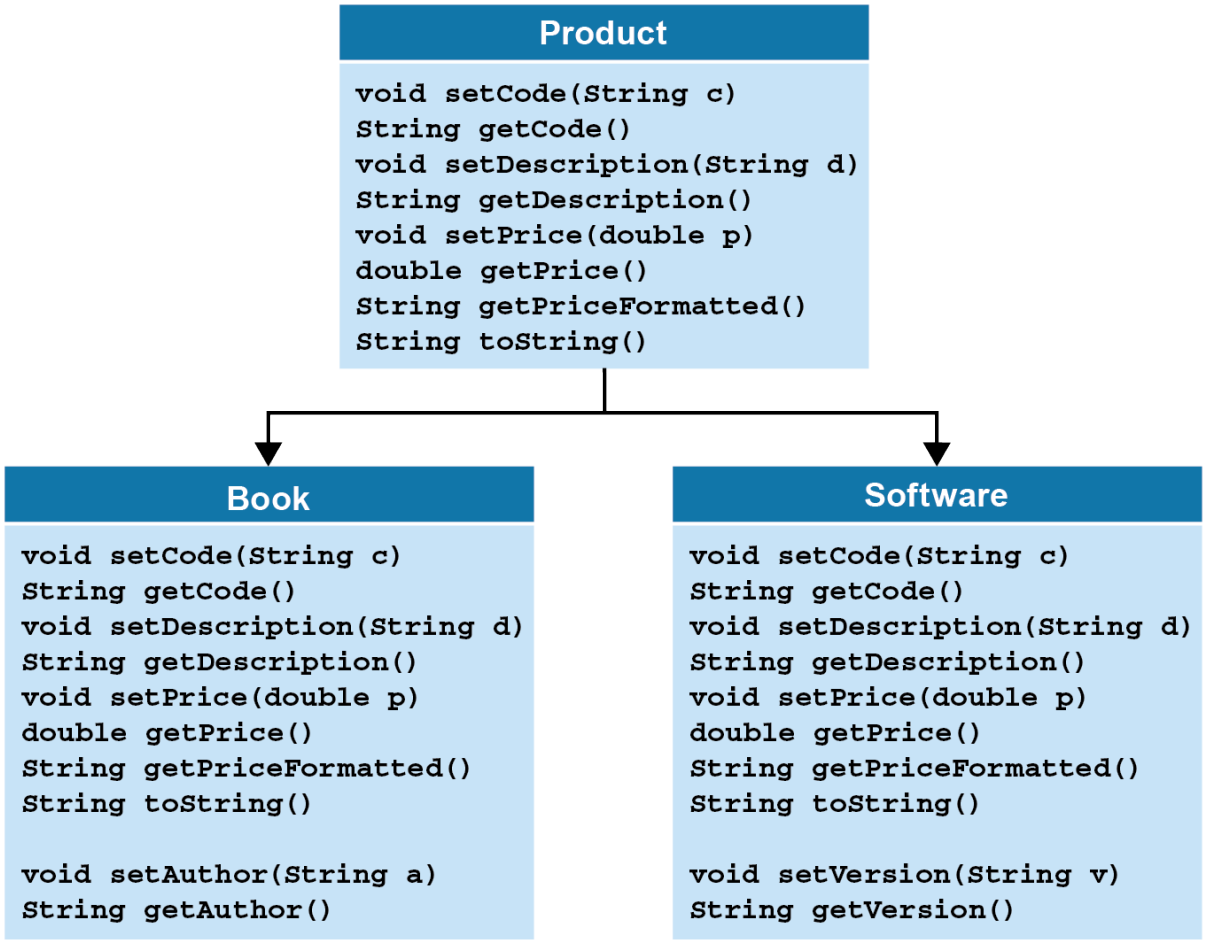
Knowledge

1. Explain what it means for a subclass to extend a superclass.
2. Explain why the toString() method and the equals() method are available to all objects and when you might override these methods.
3. Describe the access modifiers that you can use for the members of a class.
4. Describe what the @Override annotation does.
5. Explain what polymorphism is and how it works.
6. Name the operator that you can use to check whether an object is an instance of a specified class.
7. Explain when it's necessary to use explicit casting when working with objects created from derived classes.

Objectives (cont.)

8. Explain how abstract classes and methods work.
9. Explain how final classes, methods, and parameters work.

Business classes for a Product Manager application



The Object class

```
java.lang.Object
```

Methods of the class

```
toString()
equals(Object)
getClass()
clone()
hashCode()
```

A typical value returned by a Product object's…

toString() method

```
murach.business.Product@15db9742
```

hashCode() method

```
366712642
```

Access modifiers

Keyword	Available…
`private`	within the current class.
`public`	to classes in all packages.
`protected`	to classes in the same package and to subclasses.
no keyword coded	to classes in the same package.

An annotation for overriding a method

```
@Override
// method declaration goes here
```

The code for the Product superclass

```
import java.text.NumberFormat;

public class Product {

    private String code;
    private String description;
    private double price;
    protected static int count = 0;

    public Product() {
    }
```

The code for the Product superclass (cont.)

```
    // get and set accessors for the code, description,
    // and price instance variables

    @Override
    public String toString() {
        return description;
    }

    // create public access for the count variable
    public static int getCount()  {
        return count;
    }
}
```

The syntax for creating subclasses

To declare a subclass

```
public class SubclassName extends SuperClassName{}
```

To call a superclass constructor

```
super(argumentList)
```

To call a superclass method

```
super.methodName(argumentList)
```

The code for a Book subclass

```
public class Book extends Product {

    private String author;

    public Book() {
        super();  // constructor of Product superclass
        author = "";
        count++;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getAuthor() {
        return author;
    }
```

The code for a Book subclass

```
    @Override
    public String toString() {
        return super.toString() +  // Product superclass
            " by " + author;
    }
}
```

The toString() method in the Product superclass

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

The toString() method in the Book class

```
public String toString() {
    return super.toString() + " by " + author;
}
```

The toString() method in the Software class

```
public String toString() {
    return super.toString() + " " + version;
}
```

Code that uses the overridden methods

```
Book b = new Book();
b.setCode("java");
b.setDescription("Murach's Java Programming");
b.setPrice(57.50);
b.setAuthor("Joel Murach");

Software s = new Software();
s.setCode("netbeans");
s.setDescription("NetBeans");
s.setPrice(0.00);
s.setVersion("8.2");

Product p;
p = b;
System.out.println(p.toString());
                // calls toString from the Book class
p = s;
System.out.println(p.toString());
                // calls toString from the Software class
```

The console for the Product application

```
Welcome to the Product Viewer

Enter product code: java

Description: Murach's Java Programming by Joel
Murach
Price:       $57.50

Product count: 1

Continue? (y/n): y

Enter product code: netbeans

Description: NetBeans 8.2
Price:       $0.00

Product count: 2

Continue? (y/n): y
```

The console for the Product application (cont.)

```
Enter product code: xxxx

No product matches this product code.

Continue? (y/n):
```

The code for the Product class

```
import java.text.NumberFormat;

public class Product  {

    private String code;
    private String description;
    private double price;
    protected static int count = 0;

    public Product() {}

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

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

The code for the Product class (cont.)

```
public void setDescription(String description) {
    this.description = description;
}

public String getDescription()  {
    return description;
}

public void setPrice(double price) {
    this.price = price;
}

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

The code for the Product class (cont.)

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

    @Override
    public String toString() {
        return description;
    }

    public static int getCount() {
        return count;
    }
}
```

The code for the Book class

```
public class Book extends Product  {

    private String author;

    public Book() {
        super();
        author = "";
        count++;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getAuthor() {
        return author;
    }

    @Override
    public String toString() {
        return super.toString() + " by " + author;
    }
}
```

The code for the Software class

```
public class Software extends Product {

    private String version;

    public Software()  {
        super();
        version = "";
        count++;
    }

    public void setVersion(String version) {
        this.version = version;
    }

    public String getVersion() {
        return version;
    }

    @Override
    public String toString() {
        return super.toString() + " " + version;
    }
}
```

The code for the ProductDB class

```
public class ProductDB {

    public static Product getProduct(String productCode) {
        // In a more realistic application, this code would
        // get the data for the product from a file or
        // database.For now, this code just uses if/else
        // statements to return the correct product data.

        Product p = null;

        if (productCode.equalsIgnoreCase("java")
                || productCode.equalsIgnoreCase("jsp")
                || productCode.equalsIgnoreCase("mysql")) {
            Book b = new Book();
            if (productCode.equalsIgnoreCase("java")) {
                b.setCode(productCode);
                b.setDescription(
                    "Murach's Java Programming");
                b.setPrice(57.50);
                b.setAuthor("Joel Murach");
```

The code for the ProductDB class (cont.)

```
            } else if (productCode.equalsIgnoreCase("jsp")) {
                b.setCode(productCode);
                b.setDescription(
                    "Murach's Java Servlets and JSP");
                b.setPrice(57.50);
                b.setAuthor("Mike Urban");
            } else if (productCode.equalsIgnoreCase("mysql")) {
                b.setCode(productCode);
                b.setDescription("Murach's MySQL");
                b.setPrice(54.50);
                b.setAuthor("Joel Murach");
            }
            p = b; // set Product object equal to the Book object
        } else if (productCode.equalsIgnoreCase("netbeans")) {
            Software s = new Software();
            s.setCode("netbeans");
            s.setDescription("NetBeans");
            s.setPrice(0.00);
            s.setVersion("8.2");
            p = s; // set Product object equal to the Software
        }
        return p;
    }
}
```

The code for 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();

        // perform 1 or more selections
        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            System.out.print("Enter product code: ");
            String productCode = sc.nextLine();

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

The code for the ProductApp class (cont.)

```
            // display the output
            System.out.println();
            if (p != null) {
                System.out.println("Description: " +
                    p.toString());
                System.out.println("Price:       " +
                    p.getPriceFormatted());
            } else {
                System.out.println(
                    "No product matches this product code.");
            }

            System.out.println();
            System.out.println("Product count: " +
                Product.getCount() + "\n");

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

Code that casts Product and Book objects

```
Book b = new Book();
b.setCode("java");
b.setDescription("Murach's Beginning Java");
b.setAuthor("Andrea Steelman");
b.setPrice(49.50);

Product p = b;      // cast Book object to Product object
p.setDescription("Test"); // OK
//p.setAuthor("Test");    // not OK

Book b2 = (Book) p; // cast Product object to Book object
b2.setAuthor("Test");     // OK

Product p2 = new Product();
Book b3 = (Book) p2;  // throws a ClassCastException
                      // because p2 is a Product object
```

Code that checks an object's type

```
Product p = new Book();  // create a Book object
if (p instanceof Book) {
    System.out.println("This is a Book object");
}
```

The console

```
This is a Book object
```

How the equals() method of the Object class works

Both variables refer to the same object

```
Product product1 = new Product();
Product product2 = product1;
if (product1.equals(product2))    // returns true
```

Both variables refer to different objects that store the same data

```
Product product1 = new Product();
Product product2 = new Product();
if (product1.equals(product2))    // returns false
```

The equals() method of the Product class

```
@Override
public boolean equals(Object object) {
    if (object instanceof Product)  {
        Product product2 = (Product) object;
        if (code.equals(product2.getCode()) &&
            description.equals(
                product2.getDescription()) &&
            price == product2.getPrice()) {
                return true;
        }
    }
    return false;
}
```

The equals() method of the LineItem class

```
@Override
public boolean equals(Object object) {
    if (object instanceof LineItem) {
        LineItem li = (LineItem) object;
        if (product.equals(li.getProduct()) &&
            quantity == li.getQuantity()) {
                return true;
        }
    }
    return false;
}
```

An abstract Product class

```
public abstract class Product {
    private String code;
    private String description;
    private double price;

    // regular constructors and methods
    // for instance variables

    @Override
    public String toString() {
        return description;
    }

    // an abstract method
    public abstract String getDisplayText();
}
```

A class that inherits the abstract Product class

```
public class Book extends Product {
    private String author;

    // regular constructor and methods for the Book class

    @Override{    // implement the abstract method
    public String getDisplayText()
        return super.toString() + " by " + author;
    }
}
```

A final class

```
public final class Book extends Product {
    // all methods in the class are automatically final
}
```

A final method

```
public final String getVersion() {
    return version;
}
```

A final parameter

```
public void setVersion(final String version)  {

    // version = "new value"; // not allowed
    this.version = version;
}
```