Chapter 9

How to define and use interfaces

Objectives

Applied

1. Create an interface that contains abstract methods.
2. Create an interface that contains constants.
3. Create a class that implements one or more interfaces.
4. Create an interface that inherits other interfaces.
5. Use NetBeans to generate the method declarations for the methods defined by an interface.
6. Implement the Cloneable interface for any classes that you've created and then use those classes in an application.
7. 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 one advantage interfaces have over abstract classes.
2. Explain what a tagging interface is.
3. Describe how you can use an interface to specify the type for a parameter.
4. Name two types of methods that you can only add to an interface with Java 8 or later.

A Printable interface that defines an abstract print() method

```
public interface Printable {
    void print();     // auto public and abstract
}
```

A Product class that implements the Printable interface

```
import java.text.NumberFormat;

public class Product implements Printable {
    private String code;
    private String description;
    private double price;

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

    // get and set methods for the fields
```

A Product class that implements the Printable interface (cont.)

```
        // implement the Printable interface
        public void print()
            System.out.println(description);
        }
    }
```

Code that uses the print() method

```
    Printable p = ProductDB.get("java");
    p.print();
```

Resulting output

```
Murach's Java Programming
```

An abstract class compared to an interface

Abstract class	Interface
`Variables`	
`Constants`	
`Static variables`	
`Static constants`	`Static constants`
`Methods`	`Methods (new with Java 8)`
`Static methods`	`Static methods (new with Java 8)`
`Abstract methods`	`Abstract methods`

Advantages of an abstract class

- Can use instance variables and constants as well as static variables and constants.
- Can define regular methods that contain code. Prior to Java 8, an interface couldn't define regular methods.
- Can define static methods. Prior to Java 8, an interface couldn't define static methods.

Advantages of an interface

- A class can only directly inherit one other class, but a class can implement multiple interfaces.

A Printable interface

```
public interface Printable {
    void print();
}
```

A Printable abstract class

```
public abstract class Printable {
    public abstract void print();
}
```

The syntax for declaring an interface

```
public interface InterfaceName {
    type CONSTANT_NAME = value;       // static constant
    // abstract method
    returnType methodName([parameterList]);
}
```

An interface that defines one abstract method

```
public interface Printable {
    void print();
}
```

An interface that defines three abstract methods

```
public interface ProductWriter {
    boolean add(Product p);
    boolean update(Product p);
    boolean delete(Product p);
}
```

An interface that defines three static constants

```
public interface DepartmentConstants {
    int ADMIN = 1;
    int EDITORIAL = 2;
    int MARKETING = 3;
}
```

A tagging interface with no members

```
public interface Serializable {
}
```

The syntax for implementing an interface

```
public class ClassName
    implements Interface1[, Interface2]...{}
```

A class that implements two interfaces

```
public class Employee implements Printable,
        DepartmentConstants {

    private int department;
    private String firstName;
    private String lastName;

    public Employee(int department, String lastName,
            String firstName) {
        this.department = department;
        this.lastName = lastName;
        this.firstName = firstName;
    }
```

A class that implements two interfaces (cont.)

```
        @Override
        public void print() {
            String dept = "Unknown";
            if (department == ADMIN) {
                dept = "Administration";
            } else if (department == EDITORIAL) {
                dept = "Editorial";
            } else if (department == MARKETING) {
                dept = "Marketing";
            }

            System.out.println(
                firstName + " " + lastName +
                " (" + dept + ")");
        }
    }
```

The syntax for inheriting a class and implementing an interface

```
public class SubclassName
    extends SuperclassName
    implements Interface1[, Interface2]...{}
```

A Book class that inherits Product and implements Printable

```
public class Book extends Product implements Printable {

    private String author;

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

A Book class that inherits Product and implements Printable (cont.)

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

    public String getAuthor() {
        return author;
    }

    @Override
    public void print() {    // Printable interface
        System.out.println(super.getDescription() +
        " by " + author);
    }
}
```

A method that accepts a Printable object

```
private static void printMultiple(Printable p, int count)
{
    for (int i = 0; i < count; i++) {
        p.print();
    }
}
```

Code that passes a Product object to the method

```
Product product = new Product(
    "java", "Murach's Java Programming", 57.50);
printMultiple(product, 2);
```

Resulting output

```
Murach's Java Programming
Murach's Java Programming
```

Another way to pass a Product object

```
Printable product = new Product(
    "java", "Murach's Java Programming", 57.50);
printMultiple(product, 2);
```

Code that passes an Employee object

```
Employee employee = new Employee(
        DepartmentConstants.EDITORIAL, "Murach", "Joel");
printMultiple(employee, 1);
```

Resulting output

```
Joel Murach (Editorial)
```

A ProductReader interface

```
public interface ProductReader {
    Product get(String code);
    String getAll();
}
```

A ProductWriter interface

```
public interface ProductWriter {
    boolean add(Product p);
    boolean update(Product p);
    boolean delete(Product p);
}
```

A ProductConstants interface

```
public interface ProductConstants {
    int CODE_SIZE = 5;
    int DESCRIPTION_SIZE = 34;
    int PRICE_SIZE = 10;
}
```

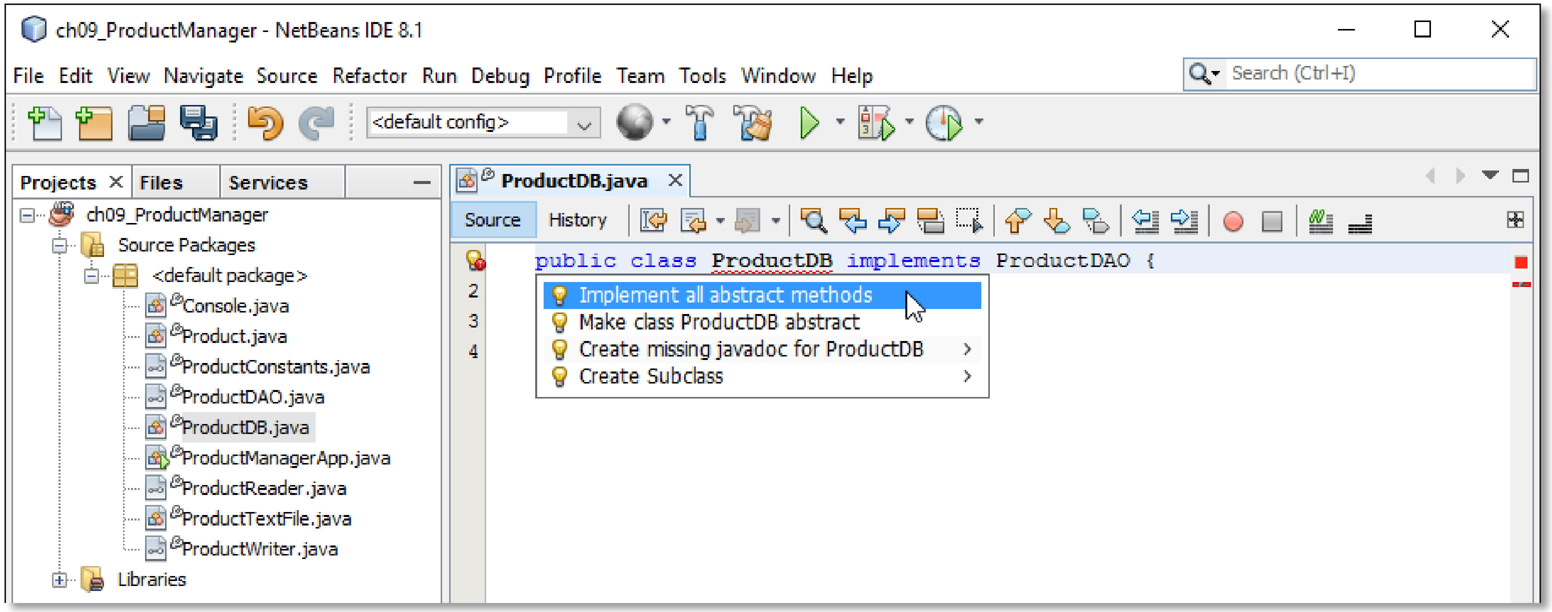
The syntax for declaring an interface that inherits other interfaces

```
public interface InterfaceName
    extends InterfaceName1[, InterfaceName2]... {
    // the constants and methods of the interface
}
```

A ProductDAO interface that inherits three interfaces

```
public interface ProductDAO extends ProductReader,
        ProductWriter, ProductConstants {
    // all methods and constants are inherited
}
```

A class that implements the ProductDAO interface



```
public class ProductDB implements ProductDAO {
```

Implement all abstract methods
Make class ProductDB abstract
Create missing javadoc for ProductDB
Create Subclass

The code that’s generated by NetBeans

```
public class ProductDB implements ProductDAO {

    @Override
    public Product getProduct(String code) {
        throw new UnsupportedOperationException(
            "Not supported yet.");
    }

    @Override
    public String getProductsString() {
        throw new UnsupportedOperationException(
            "Not supported yet.");
    }

    @Override
    public boolean addProduct(Product p) {
        throw new UnsupportedOperationException(
            "Not supported yet.");
    }
```

The code that’s generated by NetBeans (cont.)

```
    @Override
    public boolean updateProduct(Product p) {
        throw new UnsupportedOperationException(
            "Not supported yet.");
    }

    @Override
    public boolean deleteProduct(Product p) {
        throw new UnsupportedOperationException(
            "Not supported yet.");
    }
}
```

The syntax for declaring a default method (Java 8 and later)

```
default returnType methodName([parameterList]);
```

An interface that defines a default method

```
public interface Printable {
    default void print() {
        System.out.println(toString());
    }
}
```

A class that uses the default method

```
public class Product implements Printable {
    // This class doesn't override the print method.
    // It uses the print method defined by the interface.
}
```

A class that overrides the default method

```
public class Product implements Printable {
    @Override
    public void print() {
        System.out.println(getDescription() + "|" +
            getPriceFormatted());
    }
}
```

The syntax for declaring a static method (Java 8 and later)

```
static returnType methodName([parameterList]);
```

An interface that defines a static method

```
public interface Printer {
    static void print(Printable p) {
        p.print();
    }
}
```

Code that calls a static method from an interface

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

Resulting output

```
Murach's Java Programming
```

The console for the Product Viewer application

```
Welcome to the Product Viewer

Enter product code: java

PRODUCT
Code:        java
Description: Murach's Java Programming
Price:       $57.50

Continue? (y/n):
```

The ProductReader interface

```
public interface ProductReader {
    Product get(String code);
    String getAll();
}
```

The ProductDB class

```
public class ProductDB implements ProductReader {

    public ProductDB() {}

    @Override
    public Product get(String productCode) {
        Product product = new Product();
        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);
```

The ProductDB class (cont.)

```
        } else {
            product.setDescription("Unknown");
        }
        return product;
    }

    @Override
    public String getAll() {
        throw new UnsupportedOperationException(
            "This method hasn't been implemented yet.");
    }
}
```

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 input from user
            System.out.print("Enter product code: ");
            String productCode = sc.nextLine();
```

The ProductApp class (cont.)

```
// Use a ProductReader object to get
// the Product object
ProductReader reader = new ProductDB();
Product product = reader.get(productCode);

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

The ProductApp class (cont.)

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

A Product class that implements the Cloneable interface

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

    // the code for the constructor and methods

    @Override
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

Code that uses the clone() method of the Product class

```
try {
    // create a new product
    Product p1 = new Product();
    p1.setCode("java");
    p1.setDescription("Murach's Java Programming");
    p1.setPrice(54.50);

    // clone the product
    Product p2 = (Product) p1.clone();

    // change a value in the cloned product
    p2.setPrice(57.50);

    // print the results
    System.out.println(p1);
    System.out.println(p2);
} catch (CloneNotSupportedException ex) {
    System.out.println(ex);
}
```

The result of cloning a Product object

```
Code:        java
Description: Murach's Java Programming
Price:       $54.50

Code:        java
Description: Murach's Java Programming
Price:       $57.50
```

A LineItem class that implements the Cloneable interface

```
public class LineItem implements Cloneable {
    private Product product;
    private int quantity;
    private double total;

    // the code for the constructors and methods

    @Override
    public Object clone() throws CloneNotSupportedException {
        LineItem li = (LineItem) super.clone();
        Product p = (Product) product.clone();
        li.setProduct(p);
        return li;
    }
}
```

Code that uses the clone() method of the LineItem class

```
Product p1 = new Product();
p1.setCode("java");
p1.setDescription("Murach's Java Programming");
p1.setPrice(54.50);

LineItem li1 = new LineItem(p1, 3);

// clone the line item
LineItem li2 = (LineItem) li1.clone();

// change values in the cloned LineItem
// and its Product object
li2.setQuantity(2);
li2.getProduct().setPrice(57.50);

// print the results
System.out.println(li1);
System.out.println(li2);
```

The result of cloning a LineItem object

```
Code: java
Description: Murach's Java Programming
Price: $54.50
Quantity: 3
Total: $163.50

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