Chapter 12

How to work with interfaces
Objectives

Applied

• Create an interface that contains abstract methods.
• Create an interface that contains constants.
• Create a class that implements one or more interfaces.
• Create an interface that inherits other interfaces.

Knowledge

• Describe one advantage interfaces have over abstract classes.
• Explain what a tagging interface is.
• Describe how you can use an interface to specify the type for a parameter.
• 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

```
package murach.business;

public interface Printable {
    void print();     // auto public and abstract
}
```
A Product class that implements the Printable interface

```java
package murach.business;

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.)

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

Code that uses the print method

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

Resulting output

```plaintext
Murach's Java Programming
```
An abstract class compared to an interface

<table>
<thead>
<tr>
<th>Abstract class</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>Static constants</td>
</tr>
<tr>
<td>Constants</td>
<td></td>
</tr>
<tr>
<td>Static variables</td>
<td></td>
</tr>
<tr>
<td>Static constants</td>
<td></td>
</tr>
<tr>
<td>Methods</td>
<td>Methods (new with Java 8)</td>
</tr>
<tr>
<td>Static methods</td>
<td>Static methods (new with Java 8)</td>
</tr>
<tr>
<td>Abstract methods</td>
<td>Abstract methods</td>
</tr>
</tbody>
</table>
Advantages of an abstract class

• Can use instance variables and constants as well as static variables.

• An abstract class can define regular methods that contain code. Prior to Java 8, an interface can’t.

• An abstract class can define static methods. Prior to Java 8, an interface can’t.

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

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

An interface that defines one abstract method

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

An interface that defines three abstract methods

```java
public interface ProductWriter {
    boolean add(Product p);
    boolean update(Product p);
    boolean delete(Product p);
}
```
An interface that defines three static constants

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

A tagging interface with no members

```java
public interface Serializable {
}
```
The syntax for implementing an interface

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

A class that implements two interfaces

```java
package murach.business;

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.)

```java
@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

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

A Book class that inherits Product and implements Printable

```java
package murach.business;

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.)

```java
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

```java
private static void printMultiple(Printable p,
                                   int count) {
    for (int i = 0; i < count; i++) {
        p.print();
    }
}
```
Code that passes a Product object

```
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

```java
public interface ProductReader {
    Product getProduct(String code);
    String getProducts();
}
```

A ProductWriter interface

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

A ProductConstants interface

```java
public interface ProductConstants {
    int CODE_SIZE = 10;
    int DESCRIPTION_SIZE = 34;
    int PRICE_SIZE = 10;
}
```
The syntax for declaring an interface that inherits other interfaces

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

A ProductDAO interface that inherits three interfaces

```java
public interface ProductDAO extends ProductReader,
    ProductWriter, ProductConstants {
    // all methods and constants are inherited
}
```
The syntax for declaring a default method (Java 8 and later)

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

An interface that defines a default method

```java
package murach.business;

public interface Printable {
    default void print() {
        System.out.println(toString());
    }
}
```
A class that uses the default method

```java
package murach.business;

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

```java
package murach.business;

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 implements a static method

```java
package murach.business;

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

Code that calls a static method from an interface

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

Welcome to the Product Viewer

Enter product code: java

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

Continue? (y/n): n
The `ProductReader` interface

```java
package murach.db;

import murach.business.Product;

public interface ProductReader {
    Product getProduct(String code);
    String getProducts();
}
```
The ProductDB class

```java
package murach.db;

import murach.business.Product;

public class ProductDB implements ProductReader {

    @Override
    public Product getProduct(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");
        }
        return product;
    }
}
```
The ProductDB class (cont.)

```java
    product.setPrice(57.50);
    } else if (productCode.equalsIgnoreCase("mysql")) {
        product.setDescription("Murach's MySQL");
        product.setPrice(54.50);
    } else {
        product.setDescription("Unknown");
    }
    return product;

@Override
public String getProducts() {
    return "This method hasn't been implemented yet.";
}
```
The ProductApp class

package murach.ui;

import java.util.Scanner;
import murach.business.Product;
import murach.db.ProductDB;
import murach.db.ProductReader;

public class ProductApp {

    public static void main(String args[]) {

        // display a welcome message
        System.out.println("Welcome to the Product Viewer");
        System.out.println();
The ProductApp class (cont.)

// 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();

    // Use ProductReader to get Product
    ProductReader reader = new ProductDB();
    Product product =
        reader.getProduct(productCode);
The ProductApp class (cont.)

    // 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!");
}