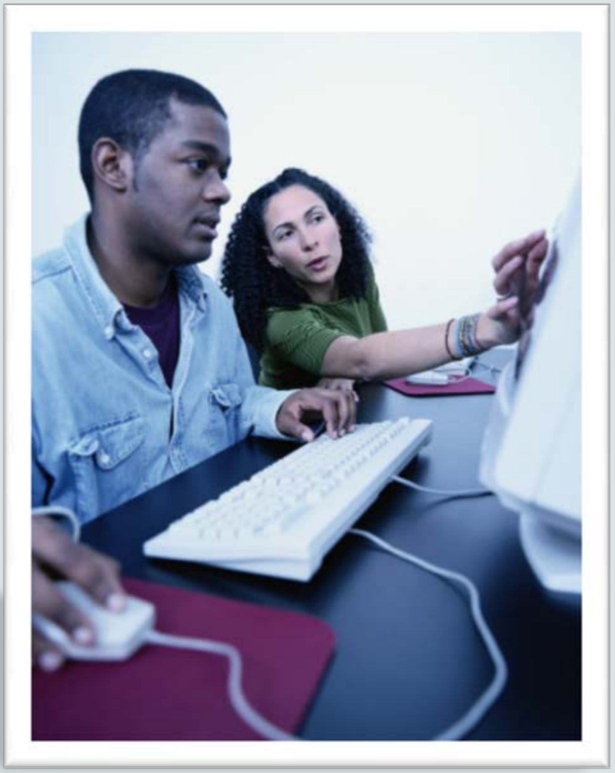
ORACLE®
ACADEMY

Java Programming

2-1
Java Class Design - Interfaces



Overview

This lesson covers the following topics:

- Model business problems using Java classes
- Make classes immutable
- Use Interfaces

Classes

- A Java class is a template/blueprint that defines the features of an object.
- A class can be thought of as a category used to define groups of things.
- Classes:
  - Class variables
  - Define and implement methods.
  - Implement methods from implemented interfaces.

Objects

- An object is an instance of a class.
- A program may have many objects.
- An object stores data in the class variables to give it state.
- This state will differentiate it from other objects of the same class.

What Classes Can and Cannot Do

- Classes can be instantiated by:
  – A public or protected constructor.
  – A public or protected static method or nested class.
- Classes cannot:
  – Override inherited methods when the method is final.

When Classes Can be Subclassed or Made Immutable

- A class can be subclassed when:
  - The class is not declared final.
  - The methods are public or protected.
- Strategy for making a class immutable:
  - Make it final.
  - Limit instantiation to the class constructors.
  - Eliminate any methods that change instance variables.
  - Make all fields final and private

Immutable Using Final

- Declaring a class as final means that it cannot be extended.
- Example: You may have a class that has a method to allow users to login by using some secure call.
- You would not want someone to later extend it and remove the security.

```
public final class ImmutableClass {
  public static boolean logOn(String username, String password) {
      //call to public boolean someSecureAuthentication(username,password);
         return someSecureAuthentication(username,password);
         }
}
```

Immutable by Limiting Instantiation to the Class Constructor

- By removing any method that changes instance variables and limiting their setting to the constructor, the class variables will be made immutable.
- Example: When we create an instance of the ImmutableClass, the immutableInt variable cannot be changed.

```
public final class ImmutableClass {
  private final int immutableInt;
    public ImmutableClass (int mutableIntIn) {
        immutableInt = mutableIntIn;
    }
    private int getImutableInt() {
    return immutableInt;
  }
}
```

Interface

- An interface is a Java construct that helps define the roles that an object can assume.
- It is implemented by a class or extended by another interface.
- An interface looks like a class with abstract methods (no implementation), but we cannot create an instance of it.
- Interfaces define collections of related methods without implementations.
- All methods in a Java interface are abstract.
  (Default methods are possible)

Why Use Interfaces

- When implementing a class from an interface we force it to implement all of the abstract methods.
- The interface forces separation of what a class can do, to how it actually does it.
- So a programmer can change how something is done at any point, without changing the function of the class.
- This facilitates the idea of polymorphism as the methods described in the interface will be implemented by all classes that implement the interface.

What An Interface Can Do

- An interface:
  - Can declare public constants.
  - Define methods without implementation.
  - Can only refer to its constants and defined methods.
  - Can be used with the instanceof operator.

What An Interface Can Do

- While a class can only inherit a single superclass, a class can implement more than one interface.

```
public class className implements interfaceName{
...class implementation...}
```

Implements is a keyword in Java that is used when a class inherits an interface.

- Example of SavingsAccount implementing two interfaces – Account and Transactionlog

```
public class SavingsAccount implements Account,Transactionlog{
...class implementation...}
```

Interface Method

- An interface method:
  - Each method is public even when you forget to declare it as public.
  - Is implicitly abstract but you can also use the abstract keyword.

Declaring an Interface

- To declare a class as an interface you must replace the keyword class with the keyword interface.
- This will declare your interface and force all methods to be abstract and make the default access modifier public.

```
public interface InterfaceBankAccount
{
   public final String bank= "JavaBank";
   public void deposit(int amt);
   public void withdraw(int amt);
   public int getbalance();
}
```

Replace class with interface.

Bank Example

```
public interface InterfaceBankAccount
{
  public final String bank= "JavaBank";
  public void deposit(int amt);
  public void withdraw(int amt);
  public int getbalance();
}
```

Classes that extend InterfaceBankAccount will have to provide working methods for methods defined here.

- Class implementing the BankAccount interface:

```
public class Account implements InterfaceBankAccount{
  private String bankname;
  public Account() {
    this.bankname = InterfaceBankAccount.bank; }
  public void deposit(int amt) { /* deposit code */ }
  public void withdraw(int amt) {/* withdraw code */ }
  public int getbalance() { /* getBalance code */ }
```

Classes implementing an interface can access the interface constants.

Bank Example Explained

- The keyword final means that the variable bank is a constant in the interface because you can only define constants and method stubs here.

```
this.bankname = InterfaceBankAccount.bank
```

- The assignment of the constant from an interface uses the same syntax that you use when assigning a static variable to another variable.

Bank Example Explained 2

- The interface defined 3 methods – deposit, withdraw and getbalance.
- These must be implemented in our class.

```
public class Account implements InterfaceBankAccount{
  private String bankname;
  int balance;
  public Account() {
      this.bankname = InterfaceBankAccount.bank; }
  public void deposit(int amt) {
      balance = balance + amt;
  }
  public void withdraw(int amt) {
      balance = balance – amt
  }
  public int getbalance() {
      return balance
  }
}
```

Why use interfaces with Bank Example?

- You may be wondering why you would want to create a class that has no implementation.
- One reason could be to force all classes that implement the interface to have specific qualities.
- In our bank example we would know that all classes that implement the interface InterfaceBankAccount must have methods for deposit, withdraw and getbalance.
- Classes can only have one superclass, but can implement multiple interfaces.

Store Example

- A store owner wants to create a website that displays all items in the store.
- We know:
  - Each item has a name.
  - Each item has a price.
  - Each item is organized by department.
- It would be in the store owner's best interest to create an interface for what defines an item.
- This will serve as the blueprints for all items in the store, requiring all items to at least have the defined qualities above.

Adding a New Item to Store Example

- The owner adds a new item to his store named cookie:
  - Each costs 1 US dollar.
  - Cookies can be found in the Bakery department.
  - Each cookie is identified by a type.
- The owner may create a Cookie class that implements the Item interface such as shown on the next slide, adding methods or fields that are specific to cookies.

Item Interface

- Possible Item interface

```
public interface Item {
  public String getItemName();
  public int getPrice();
  public void setPrice(int price);
  public String getDepartment();
}
```

- We now force any class or interface that implements Item interface to implement the methods defined above.

Create Cookie Class

- The owner may create a Cookie class that implements the Item interface, such as shown below, adding a method or two specific to cookie items.

```
public class Cookie implements Item{
  public String cookieType;
  private int price;
  public Cookie(String type){
    cookieType = type;
    price = 1;
  }
  public String getItemName() { return "Cookie";}
  public int getPrice() {return price;}
  public void setPrice(int price){this.price = price;}
  public String getDepartment() {return "Bakery";}
  public String getType() {return cookieType;}
}
```

Terminology

Key terms used in this lesson included:

- Immutable
- Interface

Summary

In this lesson, you should have learned how to:

- Model business problems using Java classes
- Make classes immutable
- Use Interfaces

ORACLE® ACADEMY