

4. Java OOP

5. Abstract Classes. Interfaces

Abstract Classes

- An *abstract class* is a class that is declared abstract
- Abstract classes cannot be instantiated, but they can be subclassed

Abstract Methods

- An *abstract method* is a method that is declared without an implementation:

abstract void moveTo(double deltaX, double deltaY);

- If a class includes abstract methods, the class itself *must* be declared abstract

An Abstract Classes II

- A subclass of an abstract class usually provides implementations for all of the abstract methods in its parent class
- If it does not, the subclass must also be declared abstract
- Abstract classes can contain fields and implemented methods (partial implementation)
- An abstract class may have static fields and static methods. You can use these static members with a class reference as you would with any other class

Example: Abstract DepoBase Class

- Modify 442InterestSum project using DepoBase as an abstract class.

Type	Start	Long	Sum	Rate
Simple	08.09.2013	20	1000.00	15.0
Simple	08.09.2013	20	2500.00	18.0
Barrier	08.09.2013	40	15000.00	11.5
Barrier	08.09.2013	80	5000.00	14.0
MonthCap	08.09.2013	180	2000	16.5
MonthCap	08.09.2013	91	40000	12.1

Sum = 1763.41

Example: Abstract DepoBase Class

- See 451AbstractDepo project for the full text

How to Create and Use Library

- To create depo library:
 - Right click on the app package -> Export -> Java -> JAR file -> Next
 - JAR file = depo.jar
 - Finish
- How you can use depo.jar library:
 - Create new project 451aAbstractDepo
 - Right click on the project name -> Build Path -> Configure Build Path
 - Go to Library tab -> Add External JARs -> find and click on depo.jar -> Open -> Ok

Interfaces

- An *interface* is a reference type, similar to a class, that can contain *only* constants, method signatures, and nested types
- There are no method bodies
- Interfaces cannot be instantiated—they can only be *implemented* by classes or *extended* by other interfaces

Defining an Interface

- An interface declaration consists of:
 - modifiers
 - the keyword interface
 - the interface name
 - a comma-separated list of parent interfaces (if any)
 - the interface body
- An interface can extend any number of interfaces

Interface Definition Example

```
public interface GroupedInterface extends
    Interface1, Interface2, Interface3 {
    // constant declarations
    double E = 2.718282;
    // method signatures
    void doSomething (int i, double x);
    int doSomethingElse(String s);
}
```

The Interface Body

- The interface body contains method declarations for all the methods included in the interface
- A method declaration within an interface is followed by a semicolon, but no braces
- All methods declared in an interface are implicitly public
- An interface can contain constant declarations in addition to method declarations
- All constant values defined in an interface are implicitly public, static, and final

Use an Interface

- To use an interface, you write a class that *implements* the interface
- When an instantiable class implements an interface, it provides a method body for each of the methods declared in the interface

```
public class OperateBMW760i implements  
    GroupedInterface {
```

```
    . . . .
```

```
}
```

Interfaces and Multiple Inheritance

- In Java, a class can inherit from only one class but it can implement more than one interface
- This means that if a variable is declared to be the type of an interface, its value can reference any object that is instantiated from any class that implements the interface

Using an Interface as a Type

- You can use interface names anywhere you can use any other data type name
- If you define a reference variable whose type is an interface, any object you assign to it *must* be an instance of a class that implements the interface

Exercise: InterestInterface

- Modify 442InterestSum project using interface

Exercise: InterestInterface

- See 452InterfaceDepo project for the full text

Cloning (1 of 2)

- Use clone() method to get independent object instead of object's assignment
- clone() method can make only a field-by-field copy
- Cloning is correct if a class contains only primitive fields or references to immutable objects
- “Deep” cloning is necessary otherwise

Cloning (2 of 2)

- The clone method is a *protected* method of Object, which means that your code cannot simply call it
- To make clone method accessible a class must:
 1. Implement the Cloneable interface
 2. Redefine the clone method with the public access modifier.

Deep Cloning

- To make a deep copy, you have clone the mutable instance fields in the redefined clone method

Example: DepoBase Cloning

```
public abstract class DepoBase implements  
Cloneable{  
    . . . . .  
    public DepoBase clone() throws  
CloneNotSupportedException{  
        DepoBase cln = (DepoBase)super.clone();  
        cln.startDate = (Date)startDate.clone();  
        return cln;  
    }  
    . . . . .  
}
```

Interfaces in Java SE 8

- The interface body can contain:
 - **abstract methods** (followed by a semicolon, but no braces – it does not contain an implementation)
 - **default methods** (are defined with the default modifier)
 - **static methods** (with the static keyword)
 - **constant declarations**

Interface Default Methods

- You specify that a method definition in an interface is a default method with the **default** keyword at the **beginning** of the method signature
- Default method defines a *default implementation*
- Default methods enable you to add new functionality to the interfaces of your libraries and ensure binary compatibility with code written for older versions of those interfaces
- Any class that implements the interface with default method will have this method **already defined**

Exercise: Default Method

- Modify 452InterfaceDepo project:
 - Remove abstract DepoBase class
 - Add calculateInterest method as default method of the InterestInterface

Exercise: Default Method

- See `453DefaultMethod` project for the full text

Functional Interfaces

- A functional interface is any interface that contains only one abstract method
- A functional interface may contain one or more default methods or static methods
- The abstract method of a functional interface can be implemented with help of lambda expression

Lambda Expression

- A lambda expression looks a lot like a method declaration
- You can consider lambda expressions as anonymous methods—methods without a name

Example of Lambda Expression I

```
public class Calculator {  
    interface IntegerMath {  
        int operation(int a, int b);  
    }  
    public int operateBinary(int a, int b,  
IntegerMath op) {  
        return op.operation(a, b);  
    }  
}
```

Example of Lambda Expression II

```
public static void main(String... args) {  
    Calculator myApp = new Calculator();  
    IntegerMath addition = (a, b) -> a + b;  
    IntegerMath subtraction = (a, b) -> a - b;  
    System.out.println("40 + 2 = " +  
        myApp.operateBinary(40, 2, addition));  
    System.out.println("20 - 10 = " +  
        myApp.operateBinary(20, 10, subtraction));  
}  
}
```

Example of Lambda Expression

- See `454LambdaCalulator` for the full text

Lambda Expression Syntax I

- A lambda expression consists of the following:
 - a comma-separated list of formal parameters enclosed in parentheses
 - the arrow token, ->
 - a body, which consists of a single expression or a statement block

Lambda Expression Syntax II

- You can omit the data type of the parameters in a lambda expression
- You can omit the parentheses if there is only one parameter
- If you specify a single expression, then the Java runtime evaluates the expression and then returns its value
- Alternatively, you must enclose statements in braces `{ }`

Manuals

- <http://docs.oracle.com/javase/tutorial/java/andI/index.html>