

5

Abstract and Nested Classes

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Design general-purpose base classes by using abstract classes
- Construct abstract Java classes and subclasses
- Apply the `final` keyword in Java
- Distinguish between top-level and nested classes



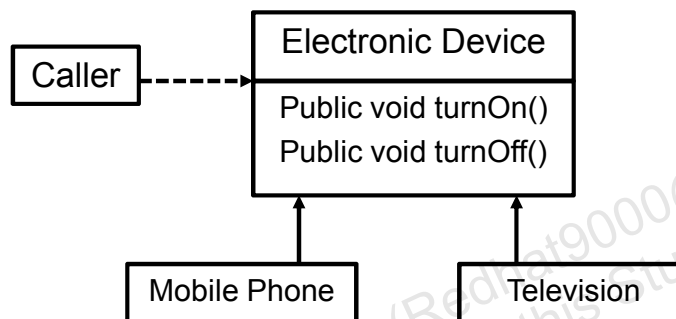
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Modeling Business Problems with Classes

Inheritance (or subclassing) is an essential feature of the Java programming language. Inheritance provides code reuse through:

- **Method inheritance:** Subclasses avoid code duplication by inheriting method implementations.
- **Generalization:** Code that is designed to rely on the most generic type possible is easier to maintain.



**Class
Inheritance
Diagram**

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Class Inheritance

When designing an object-oriented solution, you should attempt to avoid code duplication. One technique to avoid duplication is to create library methods and classes. Libraries function as a central point to contain often reused code. Another technique to avoid code duplication is to use class inheritance. When there is a shared base type identified between two classes, any shared code may be placed in a parent class.

When possible, use object references of the most generic base type possible. In Java, generalization and specialization enable reuse through method inheritance and virtual method invocation (VMI). VMI, sometimes called “late-binding,” enables a caller to dynamically call a method as long as the method has been declared in a generic base type.

Enabling Generalization

Coding to a common base type allows for the introduction of new subclasses with little or no modification of any code that depends on the more generic base type.

```
ElectronicDevice dev = new Television();  
dev.turnOn(); // all ElectronicDevices can be turned on
```

Always use the most generic reference type possible.

The Oracle logo, consisting of the word "ORACLE" in a white, sans-serif font, is positioned on the right side of a solid red horizontal bar.

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Coding for Generalization

Always use the most generic reference type possible. Java IDEs may contain refactoring tools that assist in changing existing references to a more generic base type.

Identifying the Need for Abstract Classes

Subclasses may not need to inherit a method implementation if the method is specialized.

```
public class Television extends ElectronicDevice {  
  
    public void turnOn() {  
        changeChannel(1);  
        initializeScreen();  
    }  
    public void turnOff() {}  
  
    public void changeChannel(int channel) {}  
    public void initializeScreen() {}  
  
}
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Method Implementations

When sibling classes have a common method, it is typically placed in a parent class. Under some circumstances, however, the parent class's implementation will always need to be overridden with a specialized implementation.

In these cases, inclusion of the method in a parent class has both advantages and disadvantages. It allows the use of generic reference types, but developers can easily forget to supply the specialized implementation in the subclasses.

Defining Abstract Classes

A class can be declared as abstract by using the `abstract` class-level modifier.

```
public abstract class ElectronicDevice { }
```

- An abstract class can be subclassed.

```
public class Television extends ElectronicDevice { }
```

- An abstract class **cannot** be instantiated.

```
ElectronicDevice dev = new ElectronicDevice(); // error
```



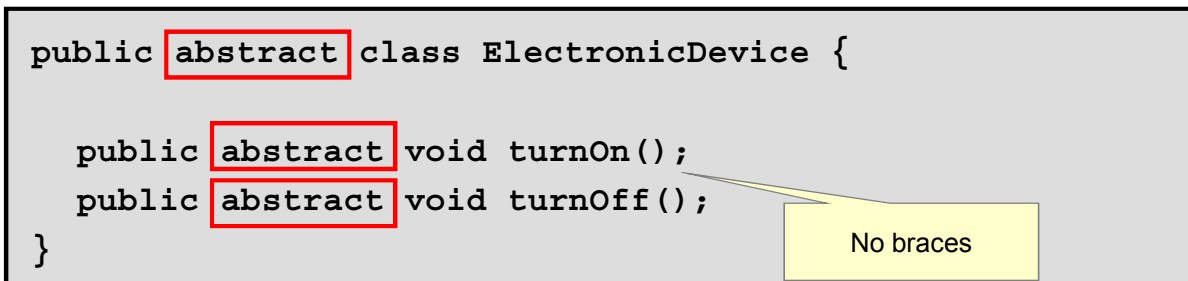
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Declaring a class as abstract prevents any instances of that class from being created. It is a compile-time error to instantiate an abstract class. An abstract class is typically extended by a child class and may be used as a reference type.

Defining Abstract Methods

A method can be declared as abstract by using the `abstract` method-level modifier.

```
public abstract class ElectronicDevice {  
  
    public abstract void turnOn();  
    public abstract void turnOff();  
}
```



An abstract method:

- Cannot have a method body
- Must be declared in an abstract class
- Is overridden in subclasses

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Inheriting Abstract Methods

When a child class inherits an abstract method, it is inheriting a method signature but no implementation. For this reason, no braces are allowed when defining an abstract method. An abstract method is a way to guarantee that any child class will contain a method with a matching signature.

Note: An abstract method can take arguments and return values. For example:

```
abstract double calculateArea(double dim1, double dim2);
```

Validating Abstract Classes

The following additional rules apply when you use abstract classes and methods:

- An abstract class may have any number of abstract and nonabstract methods.
- When inheriting from an abstract class, you must do either of the following:
 - Declare the child class as abstract.
 - Override all abstract methods inherited from the parent class. Failure to do so will result in a compile-time error.

```
error: Television is not abstract and does not override  
abstract method turnOn() in ElectronicDevice
```

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Making Use of Abstract Classes

While it is possible to avoid implementing an abstract method by declaring child classes as abstract, this only serves to delay the inevitable. Applications require nonabstract methods to create objects. Use abstract methods to outline functionality required in child classes.

Final Methods

A method can be declared `final`. Final methods may not be overridden.

```
public class MethodParentClass {  
    public final void printMessage() {  
        System.out.println("This is a final method");  
    }  
}
```

```
public class MethodChildClass extends MethodParentClass {  
    // compile-time error  
    public void printMessage() {  
        System.out.println("Cannot override method");  
    }  
}
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Performance Myths

There is little to no performance benefit when you declare a method as `final`. Methods should be declared as `final` only to disable method overriding.

Final Classes

A class can be declared `final`. Final classes may not be extended.

```
public final class FinalParentClass { }
```

```
// compile-time error  
public class ChildClass extends FinalParentClass { }
```

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Final Variables

The `final` modifier can be applied to variables.

Final variables may not change their values after they are initialized.

Final variables can be:

- Class fields
 - Final fields with compile-time constant expressions are constant variables.
 - Static can be combined with final to create an always-available, never-changing variable.
- Method parameters
- Local variables

Note: Final references must always reference the same object, but the contents of that object may be modified.

The Oracle logo, consisting of the word "ORACLE" in a white, sans-serif font, is positioned on the right side of a solid red horizontal bar.

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Benefits and Drawbacks of Final Variables

Bug Prevention

Final variables can never have their values modified after they are initialized. This behavior functions as a bug-prevention mechanism.

Thread Safety

The immutable nature of final variables eliminates any of the concerns that come with concurrent access by multiple threads.

Final Reference to Objects

A `final` object reference only prevents a reference from pointing to another object. If you are designing immutable objects, you must prevent the object's fields from being modified. Final references also prevent you from assigning a value of `null` to the reference. Maintaining an object's references prevents that object from being available for garbage collection.

Declaring Final Variables

```
public class VariableExampleClass {
    private final int field;
    public static final int JAVA_CONSTANT = 10;

    public VariableExampleClass() {
        field = 100;
    }

    public void changeValues(final int param) {
        param = 1; // compile-time error
        final int localVar;
        localVar = 42;
        localVar = 43; // compile-time error
    }
}
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Final Fields

Initializing

Final fields (instance variables) must be either of the following:

- Assigned a value when declared
- Assigned a value in every constructor

Static and Final Combined

A field that is both static and final is considered a constant. By convention, constant fields use identifiers consisting of only uppercase letters and underscores.

Nested Classes

A nested class is a class declared within the body of another class. Nested classes:

- Have multiple categories
 - **Inner classes**
 - Member classes
 - Local classes
 - Anonymous classes
 - **Static nested classes**
- Are commonly used in applications with GUI elements
- Can limit utilization of a "helper class" to the enclosing top-level class

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

An **inner** nested class is considered part of the outer class and inherits access to all the private members of the outer class.

A **static** nested class is not an inner class, but its declaration appears similar with an additional `static` modifier on the nested class. Static nested classes can be instantiated before the enclosing outer class and, therefore, are denied access to all nonstatic members of the enclosing class.

Note: Anonymous classes are covered in detail in the lesson titled "Interfaces and Lambda Expressions."

Reasons to Use Nested Classes

The following information is obtained from

<http://download.oracle.com/javase/tutorial/java/javaOO/nested.html>.

- **Logical Grouping of Classes**
 - If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.
- **Increased Encapsulation**
 - Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.
- **More Readable, Maintainable Code**
 - Nesting small classes within top-level classes places the code closer to where it is used.

Example: Member Class

```
public class BankEMICalculator {
    private String CustomerName;
    private String AccountNo;
    private double loanAmount;
    private double monthlypayment;
    private EMICalculatorHelper helper = new EMICalculatorHelper();

    /*Setters ad Getters*/

    private class EMICalculatorHelper {
        int loanTerm = 60;
        double interestRate = 0.9;
        double interestpermonth=interestRate/loanTerm;

        protected double calcMonthlyPayment(double loanAmount)
        {
            double EMI= (loanAmount * interestpermonth) / ((1.0) - ((1.0) /
            Math.pow(1.0 + interestpermonth, loanTerm)));
            return (Math.round(EMI));
        }
    }
}
```

Inner class,
EMICalculatorHelper

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The example in the slide demonstrates an inner class, EMICalculatorHelper, which is defined in the BankEMICalculator class.

Enumerations

Java includes a typesafe enum to the language.

Enumerations (enums):

- Are created by using a variation of a Java class
- Provide a compile-time range check

```
public enum PowerState {  
    OFF,  
    ON,  
    SUSPEND;  
}
```

These are references to the only three `PowerState` objects that can exist.

An enum can be used in the following way:

```
Computer comp = new Computer();  
comp.setState(PowerState.SUSPEND);
```

This method takes a `PowerState` reference.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

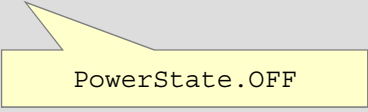
Compile-Time Range Checking

In the example in the slide, the compiler performs a compile-time check to ensure that only valid `PowerState` instances are passed to the `setState` method. No range checking overhead is incurred at runtime.

Enum Usage

Enums can be used as the expression in a switch statement.

```
public void setState(PowerState state) {  
    switch(state) {  
        case OFF:  
            //...  
    }  
}
```



PowerState.OFF

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Complex Enums

Enums can have fields, methods, and private constructors.

```
public enum PowerState {  
    OFF("The power is off"),  
    ON("The usage power is high"),  
    SUSPEND("The power usage is low");  
  
    private String description;  
    private PowerState(String d) {  
        description = d;  
    }  
    public String getDescription() {  
        return description;  
    }  
}
```

Call a `PowerState` constructor to initialize the public static final `OFF` reference.

The constructor may not be public or protected.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Enum Constructors

You may not instantiate an enum instance with `new`.

Complex Enums

- Here is the complex enum in action.

```
public class ComplexEnumsMain {  
  
    public static void main(String[] args) {  
        Computer comp = new Computer();  
        comp.setState(PowerState.SUSPEND);  
        System.out.println("Current state: " +  
            comp.getState());  
        System.out.println("Description: " +  
            comp.getState().getDescription());  
    }  
}
```

- Output

```
Current state: SUSPEND  
Description: The power usage is low
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Enum Usage

When sent to be printed, the default behavior for an enumeration is to print the current value. Typically, an additional call to the enumeration's methods is necessary to get information stored in the enumeration.

Summary

In this lesson, you should have learned how to:

- Design general-purpose base classes by using abstract classes
- Construct abstract Java classes and subclasses
- Apply the `final` keyword in Java
- Distinguish between top-level and nested classes



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 5-1 Overview: Applying the Abstract Keyword

This practice covers the following topics:

- Identifying potential problems that can be solved using abstract classes
- Refactoring an existing Java application to use abstract classes and methods



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 5-2 Overview: Using Inner Class As a Helper Class

This practice covers using an inner class as a helper class to perform some calculations in an Employee class.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 5-3 Overview: Using Java Enumerations

This practice covers taking an existing application and refactoring the code to use an enum.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Quiz

Which two of the following should an abstract method not have to compile successfully?

- a. A return value
- b. A method implementation
- c. Method parameters
- d. `private` access

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Quiz

Which of the following nested class types are inner classes?

- a. Anonymous
- b. Local
- c. Static
- d. Member

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Quiz

A final field (instance variable) can be assigned a value either when declared or in all constructors.

- a. True
- b. False

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Andres Fernando Prada Suarez (Redhat9000@gmail.com) has a
non-transferable license to use this Student Guide.