Java

Package, Interface & Exception

Package

Package

- Java package provides a mechanism for partitioning the class name space into more manageable chunks
  - Both **naming** and **visibility** control mechanism
- Define classes inside a package that are not accessible by code outside that package
- Define class members that are exposed only to other members of the same package
- This allows classes to have intimate knowledge of each other
  - Not expose that knowledge to the rest of the world

Declaring Package

- ***package pkg***
  - Here, pkg is the name of the package
- ***package MyPackage***
  - creates a package called MyPackage
- The package statement defines a name space in which classes are stored
- If you omit the package statement, the class names are put into the **default package**, which has no name

Declaring Package

- Java uses file system directories to store packages
  - the .class files for any classes that are part of MyPackage must be stored in a directory called MyPackage
- More than one file can include the same package statement
- The package statement simply specifies to which package the classes defined in a file belong
- To create hierarchy of packages, separate each package name from the one above it by use of a (.)

Package Example

```
1    package MyPackage;
2
3    class Balance {
4        String name;
5        double bal;
6
7        Balance(String n, double b) {
8            name = n;
9            bal = b;
10       }
11       void show() {
12           if(bal < 0)
13               System.out.print("--> ");
14           System.out.println(name + ": $" + bal);
15       }
16   }
17
18   public class AccountBalance {
19       public static void main(String[] args) {
20           Balance current[] = new Balance[3];
21           current[0] = new Balance("K. J. Fielding", 123.23);
22           current[1] = new Balance("Will Tell", 157.02);
23           current[2] = new Balance("Tom Jackson", -12.33);
24           for(Balance b : current) {
25               b.show();
26           }
27       }
28   }
```

javac -d . AccountBalance.java

java MyPackage.AccountBalance

Package Syntax

- The general form of a multilevel package statement
  - *package pkg1[.pkg2[.pkg3]]*
  - *package java.awt.image*
- In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions
- The general form of the import statement
  - *import pkg1 [.pkg2].(classname | *)*
  - *import java.util.Scanner*

Access Protection

- Packages act as containers for classes and other subordinate packages

- Classes act as containers for data and code

- The class is Java's smallest unit of abstraction

- Four categories of visibility for class members
  - Subclasses in the same package
  - Non-subclasses in the same package
  - Subclasses in different package
  - Classes that are neither in the same package nor subclasses

Access Protection

- The three access modifiers provide a variety of ways to produce the many levels of access required
  - private, public, and protected
- The following applies only to members of classes

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Access Protection

- Anything declared **public** can be accessed from anywhere

- Anything declared **private** cannot be seen outside of its class

- When a member does not have an explicit access specification, it is visible to subclasses as well as to other classes in the same package (**default access**)

- If you want to allow an element to be seen outside your current package, but only to classes that subclass the class directly, then declare that element **protected**

10

Access Protection

- A non-nested class has only two possible access levels

  - default and public

- When a class is declared as public, it is accessible by any other code

- If a class has default access, then it can only be accessed by other code within its same package

- When a class is public, it must be the only public class declared in the file, and the file must have the same name as the class

Interface

Interface

- We can call it a pure abstract class having no concrete methods
  - All methods declared in an interface are implicitly **public** and **abstract**
  - All variables declared in an interface are implicitly **public**, **static** and **final**
- *An interface can't have instance variables, so can't maintain state information unlike class*
- A class can only extend from a **single class**, but a class can implement **multiple interfaces**

Implementing Interface

- When you implement an interface method, it must be declared as public

- By implementing an interface, a class signs a contract with the compiler that it will definitely provide implementation of all the methods

- If it fails to do so, the class will be considered as abstract

- Then it must be declared as abstract and no object of that class can be created

Simple Interface

```java
 3    interface Callback
 4    {
 5        void callback(int param);
 6    }

 8    class Client implements Callback
 9    {
10        public void callback(int p)
11        {
12            System.out.println("callback called with " + p);
13        }
14    }

16    public class InterfaceTest {
17        public static void main(String[] args) {
18            // Can't instantiate an interface directly
19            //Callback c1 = new Callback();
20            //c1.callback(21);
21            Client c2 = new Client();
22            c2.callback(42);
23            // Accessing implementations through Interface reference
24            Callback c3 = new Client();
25            c3.callback(84);
26        }
27    }
```

Applying Interfaces

```java
3    interface MyInterface {
4        void print(String msg);
5    }
6
7    class MyClass1 implements MyInterface {
8        public void print(String msg) {
9            System.out.println(msg + ":" + msg.length());
10       }
11   }
12
13   class MyClass2 implements MyInterface {
14       public void print(String msg) {
15           System.out.println(msg.length() + ":" + msg);
16       }
17   }
18
19   public class InterfaceApplyTest {
20       public static void main(String[] args) {
21           MyClass1 mc1 = new MyClass1();
22           MyClass2 mc2 = new MyClass2();
23           MyInterface mi; // create an interface reference variable
24           mi = mc1;
25           mi.print("Hello World");
26           mi = mc2;
27           mi.print("Hello World");
28       }
29   }
```

16

Variables in Interfaces

```java
3    import java.util.Random;
4
5    interface SharedConstants {
6        int NO = 0;
7        int YES = 1;
8        int LATER = 3;
9        int SOON = 4;
10       int NEVER = 5;
11   }
12
13   class Question implements SharedConstants {
14       Random rand = new Random();
15       int ask() {
16           int prob = (int) (100 * rand.nextDouble());
17           if (prob < 30) return NO;
18           else if (prob < 60) return YES;
19           else if (prob < 75) return LATER;
20           else if (prob < 98) return SOON;
21           else return NEVER;
22       }
23   }
24
25   public class InterfaceVariableTest {
26       public static void main(String[] args) {
27           Question q = new Question();
28           System.out.println(q.ask());
29       }
30   }
```

17

Extending Interfaces

```java
 3   interface I1 {
 4       void f1();
 5       void f2();
 6   }

 8   interface I2 extends I1 {
 9       void f3();
10   }

12   class MyClass implements I2 {
13       public void f1() { System.out.println("Implement f1"); }
16       public void f2() { System.out.println("Implement f2"); }
19       public void f3() { System.out.println("Implement f3"); }
22   }

24   public class InterfaceExtendsTest {
25       public static void main(String[] args) {
26           MyClass m = new MyClass();
27           m.f1();
28           m.f2();
29           m.f3();
30       }
31   }
```

18

Default Interface Methods

- Prior to JDK 8, an interface could not define any implementation whatsoever

- The release of JDK 8 has changed this by adding a new capability to interface called the *default method*

  - A default method lets you define a default implementation for an interface method

  - Its primary motivation was to provide a means by which interfaces could be expanded without breaking existing code

Default Interface Methods

```java
interface MyIF {
    // This is a "normal" interface method declaration.
    int getNumber();
    // This is a default method. Notice that it provides
    // a default implementation.
    default String getString() {
        return "Default String";
    }
}

class MyIFImp implements MyIF {
    // Only getNumber() defined by MyIF needs to be implemented.
    // getString() can be allowed to default.
    public int getNumber() {
        return 100;
    }
}

public class InterfaceDefaultMethodTest {
    public static void main(String[] args) {
        MyIFImp m = new MyIFImp();
        System.out.println(m.getNumber());
        System.out.println(m.getString());
    }
}
```

Multiple Inheritance Issues

```java
interface Alpha {
    default void reset() {
        System.out.println("Alpha's reset");
    }
}

interface Beta {
    default void reset() {
        System.out.println("Beta's reset");
    }
}

class TestClass implements Alpha, Beta {
    public void reset() {
        System.out.println("TestClass's reset");
    }
}
```

```java
interface Alpha {
    default void reset() {
        System.out.println("Alpha's reset");
    }
}

interface Beta extends Alpha {
    default void reset() {
        System.out.println("Beta's reset");
        // Alpha.super.reset();
    }
}

class TestClass implements Beta {

}
```

Static Methods in Interface

```java
3     interface MyIFStatic {
4
5         int getNumber();
6
7         default String getString() {
8             return "Default String";
9         }
10
11        // This is a static interface method.
12 @      static int getDefaultNumber() {
13            return 0;
14        }
15    }
16
17    public class InterfaceStaticMethodTest {
18        public static void main(String[] args) {
19            System.out.println(MyIFStatic.getDefaultNumber());
20        }
21    }
```

Exception

Exception Handling

- Uncaught exceptions
- Caught exceptions
- try
- catch
- finally
- throw
- throws
- Creating custom exceptions

Uncaught Exceptions

```java
3   public class TestException1
4   {
5       public static void main(String args[]) {
6           int a = 10, b = 0;
7           int c = a/b; // ArithmeticException: / by zero
8           System.out.println(a);
9           System.out.println(b);
10          System.out.println(c);
11          String s = null;
12          System.out.println(s.length()); // NullPointerException
13      }
14  }
```

Caught Exceptions

```java
public class TestException2
{
    public static void main(String args[])
    {
        int a = 10, b = 0, c = 0;
        try {
            c = a/b;
        } catch(Exception e) {
            System.out.println (e);
        } finally {
            // finally block will always execute
            System.out.println ("In Finally");
        }
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

Caught Exceptions

```
 3    public class TestException5
 4    {
 5        public static void main(String args[])
 6        {
 7            int a = 10, b = 0, c = 0;
 8            try {
 9                c = a / b;          catch(ArithmeticException | NullPointerException e)
10            } catch(ArithmeticException e1) {
11                System.out.println(e1);
12            } catch(NullPointerException e2) {
13                System.out.println(e2);
14            } finally {
15                // finally block will always execute
16                System.out.println ("In Finally");
17            }
18            System.out.println(a);
19            System.out.println(b);
20            System.out.println(c);
21        }
22    }
```

Throws

```java
public class TestException3
{
    public static void f() throws Exception {
        int a = 10;
        int b = 0;
        int c = a/b;
    }

    public static void main(String args[])
    {
        try {
            f();
        } catch(Exception e) {
            System.out.println (e);
            e.printStackTrace();
        }
        System.out.println("Hello World");
    }
}
```

Creating Custom Exceptions

```java
3   class MyException extends Exception {
4       private int detail;
5
6       MyException(int a) {
7           detail = a;
8       }
9
10      public String toString() {
11          return "My Exception :  "  + detail;
12      }
13  }
14
15  public class TestException4 {
16      static void compute(int a) throws MyException {
17          if(a > 10) {
18              throw new MyException(a);
19          }
20          System.out.println(a);
21      }
22
23      public static void main(String args[]) {
24          try {
25              compute(10);
26              compute(20);
27          } catch(MyException e) {
28              System.out.println(e);
29          }
30      }
31  }
```

29