Advanced OOP

Bob Singh

Packages

User-defined Packages(Pkg1 in Program1.java, Program2.java)

- First declare the name of the package using the keyword **`package`**, followed by the package name.
- Packages can be named using the standard Java naming rules.
- By convention, however, packages begin with lowercase letters, in order to make it easy to distinguish package names from class names. Every package name must be unique.
- The statement, declaring the package should be the first statement in a Java source file.

User-defined Packages

```
package MyPackage; //  package declaration
public class MyClass // class definition
{
      //(body of class)
}
```

- In this example, the package name is **`MyPackage`**. The class **`MyClass`** is now considered a part of **`MyPackage`**.

User-defined Packages

- In the above example, the class would now be saved as a file called **`MyClass.java`** and located in a directory named **`MyPackage`**.
- When the source file is compiled, Java will create a **`.class`** file and store it in the same directory.
- The **`.class`** files must be located in the directory that has the same name as the package and this directory should be a sub directory of the directory, where classes that will import the package are located.

User-defined Packages

- Steps for creating user-defined packages:
  - Declare the package at the beginning of a file using the notation: `package packagename;`
  - Define the class that is to be put in the package and declare it `public`.
  - Create a subdirectory under the directory, where the main source files are stored.
  - Store the listing as the `classname.java` file in the subdirectory created.
  - Compile the file. This creates `.class` file in the subdirectory.

Scope Management

Variable Scope with all Modifiers (Program3.java)

	scope	public	protected	default	private
1	**from within the object**	yes	yes	yes	yes
2	**from a class within the same package**	yes	yes	yes	no
3	**from a class outside of a package**	yes	no	**no**	no
4	**from a subclass located in the same package**	yes	yes	yes	no
5	**from a subclass located outside of the package.**	yes	yes	**no**	no
	For information hiding all variables should be declared private from here on forward.				
	final - final properties cannot be changed				
	final - final methods cannot be overridden.				
	static - properties/methods are global to objects of the same objecttypes.				

Abstract Methods and Classes

Abstract Methods and Classes

- Represents an Abstract idea.
- What is Abstract idea?
  - An idea that physically cannot exist.
  - Something that is difficult to understand.
  - Insufficiently factual
  - Expressing a quality apart from the object
    - For e.g. poem is concrete and poetry is abstract.
  - Dealing with a subject of an abstract nature (i.e.
    - i.e. of a theoretical nature, is impersonal or detached.
  - Has an intrinsic form, with very little pictorial representation or narrative content.

Abstract Methods and Classes

- So then, what methods should be abstract?
  - Those method whose implementation details are unknown. i.e. you just know its what its name will be and some arguments.
- Since abstract means an idea that physically cannot exist
  - We can deduce that abstract classes cannot be instantiated.

Abstract Methods and Classes

- In Java, we can also define methods that must be redefined in a subclass, thus making overriding compulsory.
- This is done, by using the modifier keyword **`abstract`** in the method definition.

```
abstract class Shape {
      ......
     abstract void draw( );
      ....
}
```

Abstract Methods and Classes

- When a class contains one or more abstract methods, it should be declared as **`abstract`**.
- Cannot use abstract classes to instantiate objects directly.
- For example,

```
Shape s = new Shape(); //illegal
//declaration, Shape is an
//abstract class
```

Abstract Methods and Classes

- The abstract methods of an abstract class must be defined in its subclass.
- We cannot declare abstract constructors or abstract static methods.
- You may define some methods in an abstract class. They can only be used in an instance of a child class.

Abstract Methods and Classes

```
abstract class Shape
{
     int x, y;
     . . .
     void moveTo(int newX, int newY)
     {
          . . .
     }
     abstract void draw();
}

class Circle extends Shape {
                void draw() { . . . }
    }

class Rectangle extends Shape
        { void draw() { . . . }
}
```

Abstract Methods and Classes (Program4.java)

- What is the point of declaring a class that has no Abstract Method in it?
  - **It cannot be instantiated.**
- But any class that has an abstract method in it or that does not provide an implementation for any abstract methods declared in its superclasses *must* be declared as an abstract class.

Interfaces

Interface – Webster's meaning

- **1** - a surface forming a common boundary of two bodies, spaces, or phases <an oil-water *interface*>
  2 - the place at which independent and often unrelated systems meet and act on or communicate with each other
- <the man-machine *interface*> the means by which interaction or communication is achieved at an interface

Interfaces

- An interface is a class
- It has collection of method definitions (without implementations) and constant values.
- Defining an interface is similar to creating a new class.

Advantages of Interfaces

- Useful for capturing similarities between unrelated classes, without forcing a class relationship
- Declares methods, without implementation that one or more classes are expected to implement

Interface Definition

- An interface definition has two components: the interface declaration and the interface body.

```
interfaceDeclaration
{
    interfaceBody
}
```

- The definition of an interface is a definition of a new reference data type.

Interface Definition

- Interface names can be used anywhere in the program, like using any other type name.
- Interface doesn't have a constructor.
- Cannot instantiate objects of the interface type.

Interface Declaration

- Contains the Java keyword **`interface`** and the name of the interface.
- An interface declaration can have two other components:
  - The public access specifier and
  - A list of interfaces
- An interface can **`extend`** other interfaces.

Interface Declaration

- Interfaces provide a way of multiple inheritance in Java.
  - Any class can only extend one other class, but any number of interfaces can be implemented.
- The **`public`** access specifier indicates that the interface can be used by any class in any package.
- If not public, the interface will only be accessible to classes that are defined in the same package.

Interface Declaration

- An interface cannot extend classes.
- The list of superinterfaces is a comma-separated list of all of the interfaces extended by the new interface.
- An interface inherits all constants and methods from its superinterface unless:
  - the interface hides a constant with another of the same name, or
  - redeclares a method with a new method declaration.

Interface Body

- The body of the interface contains method declarations.
- The method declaration does not have a method body. All methods declared in an interface are **`public`** and **`abstract`**.
- The body of the interface may also define constants. Constant values defined in an interface are **`public`**, **`static`**, and **`final`**.

Implementing an Interface

- An interface can be implemented by defining a class that **`implements`** the interface by name.
- Keyword **`implements`** is used by a class implementing the interface.
- When a class **`implements`** an interface, it must provide a full definition for all the methods declared in the interface and the methods declared in the superinterfaces of that interface.

Implementing an Interface (Program5.java)

- A class can implement more than one interface by including several interface names in a comma-separated list of interface names.
- In that case, the class must provide a full definition for all the methods declared in all of the interfaces listed, as well as all of the superinterfaces of those interfaces.

Inner Classes

Inner Classes

- Classes that belong to a package are called top-level classes. When Java was introduced, they were the only classes supported by the language.

- Beginning with Java 1.1, the concept of an inner class was introduced.

  – An inner class is a class that is defined inside another class, locally within a block of statements, or (anonymously) within an expression.

  For example,

```
class Outer
{
        class Inner
        {

        }
}
```

  – Each inner class has its own class file and these class files must be included along with any top-level classes.

Why do you need an inner class?

- An inner class can access all the private resources methods, properties of the top-level class.
- Inner classes can be hidden from other classes in the same package. You don't have to worry about name conflicts between it and other classes.
- Inner classes are convenient, when you are writing event-driven programs.
- An inner class is a short class file, that exists only for a limited purpose.

Types of inner classes

- Type 1 [Defining an inner class inside a class] (Program6.java)
  - Standard type, where we can specify a class inside another class.
- Type 2 [Defining an inner class inside a method] (Program7.java)
  - This type specifies the definition of an inner class inside a method. It is not available outside the method.

```
class Outer
{
        public void SomeMethod()
        {
                class Inner
                {
                 }
        }
 }
```

Types of inner classes

- Type 3 [An inner class that lives within the body of a top level class (not within a method or another inner class) can be declared static] (Program8.java)

```
class Outer
{
      static class Inner
      {
            ...
      }
}
```

  - A static inner class such as this acts just like a new top level class called Outer.Inner; we can use it without regard to any enclosing instances. A static inner class could be private, protected, default, or publicly visible.

```
Outer.Inner example = new Outer.Inner();
```

Types of inner classes

- Type 4 [Anonymous Inner Classes] (Program9.java)
  - Anonymous inner classes are an extension of the syntax of the new operation. When you create an anonymous inner class, you combine the class's declaration with the allocation of an instance of that class. After the new operator, you specify either the name of a class or an interface, followed by a class body.

```
Enumeration getEnumeration() {
          return new Enumeration() {
          int element = 0;
          boolean hasMoreElements() {
              return  element < employees.length ;
          }
          Object nextElement() {
              if ( hasMoreElements() )
                  return employees[ element++ ];
              else
                  throw NoSuchElementException();
          }
      };
  }
```

Scope of an inner class

- Rules governing the scope of an inner class match those governing variables.
- An inner class's name is not visible outside its scope, except in a fully qualified name.
- The code for an inner class can use simple names from enclosing scopes, including class and member variables of enclosing classes, as well as local variables of enclosing blocks.
- You can also define a top-level class as a static member of another top-level class. By nesting, you can organize classes.

Generics

Template – What does Webster say?

- : a shape or pattern that is cut out of a hard material (such as metal or plastic) and used to make the same shape and pattern in other pieces of material
- *computers* : a computer document that has the basic format of something (such as a business letter, chart, graph, etc.) and that can be used many different times
- : something that is used as an example of how to do, make, or achieve something

Generics

- Basic Definition – Gen<T> (Program10.java)
- Gen<T, V> (Program11.java)
- Gen(T extends V) (Program12.java)
- Gen<?> in method definition (Program13.java)