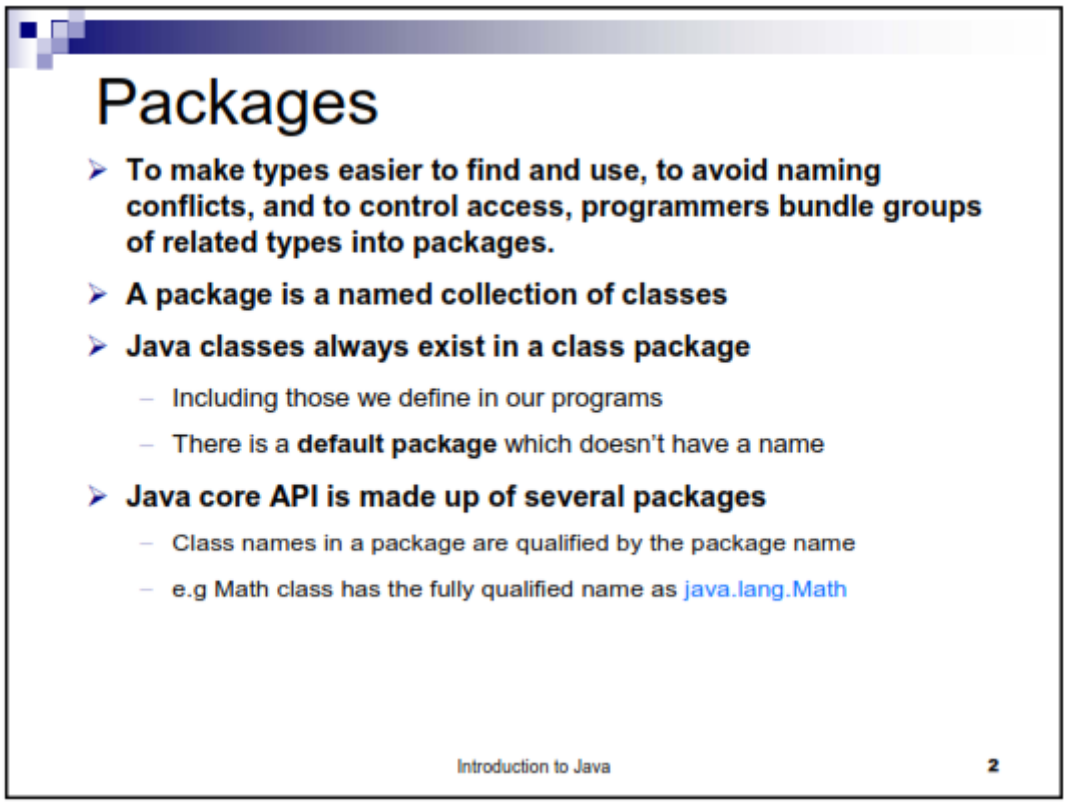
Packages and Access Modifiers



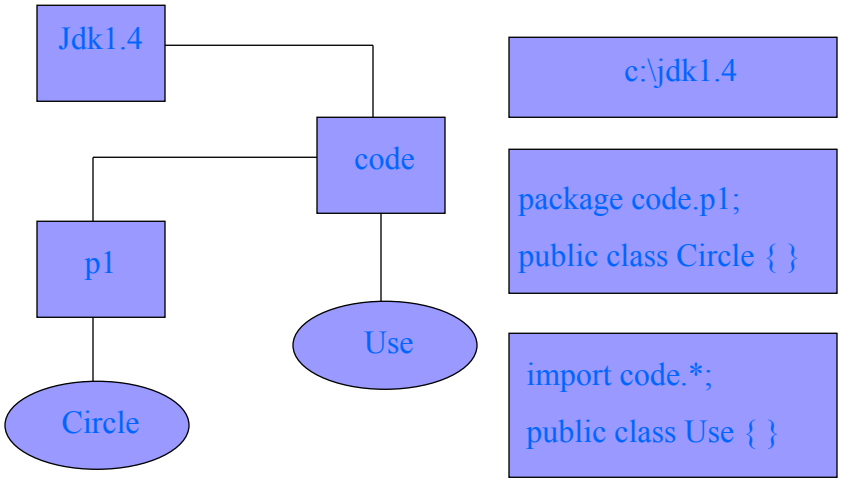
Uses of packages

- **To create different class name spaces**
  - Names used for classes in one package will not interfere with the names of classes in another package
- **Co-package classes enjoy special access to each other members**
  - You don't have to import those classes

Creating packages

- **Add a package statement as the first statement in your source file containing the class definition**
  - Only comments and blank lines are allowed to precede the package statement
- **A package statement consist of the keyword package followed by the package name terminated by a semicolon.**
- **You can specify a package name as a sequence of names separated by periods.**
- **Packages are intimately related to the directory structure in which they are stored**
  - Class files must be in a directory named by the package

Accessing packages



Importing classes

- **There are two ways to use classes in other packages provided they are defined public**
  - Use fully qualified class name
  - Import the class. This allows you to use just the class name
- **You can import any or all of the classes in a package to your code by using the import statement**
  - Keyword import is followed by name of class/classes you want to import terminated by a semicolon.
  - e.g. import java.io.*;

Automatic imports

- **Two packages are automatically imported in your source code. You don't need to write explicit import statements for them**
- **java. lang.***
- **Classes in your current default package**

Accessing packages

- **Put the source code for a class, interface, enumeration, or annotation type in a text file whose name is the simple name of the type and whose extension is .java**

```
// in the Rectangle.java file
package graphics;

public class Rectangle() { . . . }
```

- Then, put the source file in a directory whose name reflects the name of the package to which the type belongs:
  .....\graphics\Rectangle.java

class name graphics.Rectangle

pathname to file graphics\Rectangle.java

Accessing packages

- When you compile a source file, the compiler creates a different output file for each type defined in it. The base name of the output file is the name of the type, and its extension is .class.

```
// in the Rectangle.java file
package com.example.graphics;
public class Rectangle{ . . . }
class Helper{ . . . }
```

then the compiled files will be located at:

<path to the parent directory of the output files>\com\example\graphics\Rectangle.class
<path to the parent directory of the output files>\com\example\graphics\Helper.class

Managing Packages

- Like the .java source files, the compiled .class files should be in a series of directories that reflect the package name. However, the path to the .class files does not have to be the same as the path to the .java source files. You can arrange your source and class directories separately, as:
- <path_one>\sources\com\example\graphics\Rectangle.java
  <path_two>\classes\com\example\graphics\Rectangle.class
- By doing this, you can give the classes directory to other programmers without revealing your sources
- The full path to the classes directory, <path_two>\classes, is called the *class path*, and is set with the CLASSPATH system variable

Access Modifiers

- **Modifiers are java keywords that give the compiler information about the nature of code, data or classes**
- **Modifiers controls access to features**
- **Features are**
  - The class itself
  - Its variables
  - Its methods (including constructors)
- **Access Modifiers are**
  - public
  - protected
  - private

Access Modifiers

- **A feature may have at most one access modifier**
- **Each access modifier controls the access for only that particular feature**
  - What happens in C++?
- **Java provides a default access which is used when no access modifier is present. Any class, field, method or constructor that has no declared access modifier is accessible only by classes in the same package**
- **The only variables that may be controlled by access modifiers are class level variables**
  - Local variables may not have access modifiers

Access Modifiers

- **Public Access Modifiers are most generous**
  - These Access Modifiers are available to the class itself and to any other class without any restriction
  - Can be applied to class, variable or method
- **Protected Access Modifiers can be applied to variables, methods and inner classes only**
  - Non-inner classes cannot be made protected
  - These are available to all classes in the same package AND
  - To all subclasses of the class that owns that protected feature
  - This access is provided even to subclasses that reside in a different package from the class that owns the protected feature

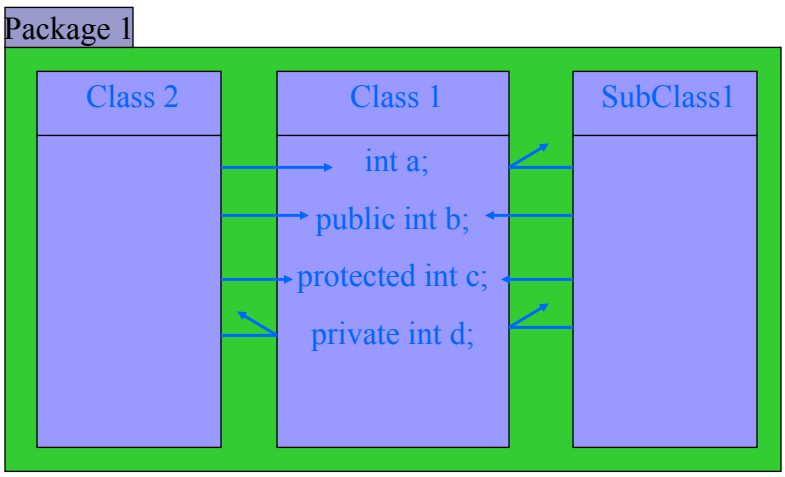
Access Modifiers

- **Private Access Modifiers are of the most restricted kind.**
  - Only variables and methods can be made private
  - A private feature can only be accessed by an instance of the class
  - An instance of a sub class of the class in question cannot access private features

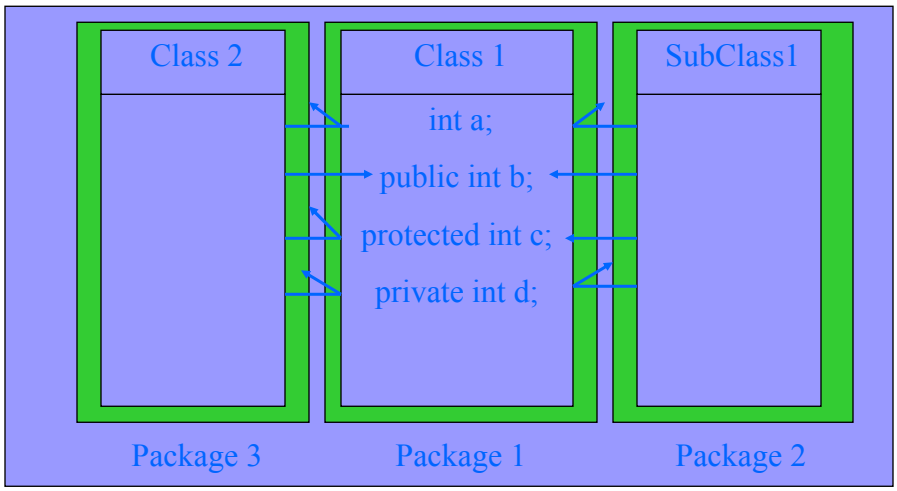
Access Modifiers

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Access Modifiers



Access Modifiers



Access Modifiers

Access Modifiers

- **There can be only one public class per compilation unit (file). It can have as many supporting classes as you want.**
- **The name of the public class must exactly match the name of the file including capitalization**
- **It is possible to have a file with no public class at all. In this case, you can name the file whatever you like**
  - What happens in this case when you compile the file?

Other Modifiers

- **Modifiers are java keywords that give the compiler information about the nature of code, data or classes**
- **Other Modifiers are**
  - final
  - static
  - abstract
  - native
  - transient
  - synchronized
  - volatile

Other Modifiers

- **Java does not care about order of appearance of modifiers**
  - public static is same as static public (in main method( ))
- **Not every modifier can be applied to every kind of feature**
- **Some combinations of modifiers are not allowed**

Final Modifier

- **Final modifier can be applied to classes, methods and variables**
- **Final classes cannot be sub classed**
  - Java.lang.Math class is final, you cannot make a subclass of Math class
- **Final variable may not be modified once it has been assigned a value**
  - Same as const in C++
  - Java.lang.Math has final variable of type double called PI
  - If the final variable is a reference to an object , it is the reference that must stay the same, not the object
- **A final method may not be overridden**

Static Modifier

- **Static modifier can be applied to variables, methods and blocks**
- **Static features belong to a class rather than associated with an individual instance of the class**
- **A static variable is also called class variable and can be referenced in two ways**
  - Via the class name
  - Via a reference to any instance of the class
- **A static method is also called a class method and it can only access static features of the class**
- **Static initialization block is executed at class load time in the order of appearance**

Abstract Modifier

- **Abstract modifier can be applied to classes and methods only**
- **An abstract method has no body**
  - abstract void sound( );
- **Any class which contains an abstract method must be declared as abstract class**
- **An abstract class cannot be instantiated**
  - Abstract classes defer the implementation to subclasses
  - The subclass must provide the implementation of the abstract method or declare itself to be abstract in which case the implementation is deferred once again

Abstract Modifier

- **A class must be declared abstract if any of the following condition is true**
  - The class has one or more abstract methods
  - The class inherits one or more abstract methods for which it does not provide implementation
  - The class declares that it implements an interface but does not provide implementation for every method of that interface

Native Modifier

- **Native modifier can be applied to methods only**
- **A Native method indicates that the body of the method is to be found elsewhere**
  - native void sound( );
  - Unlike abstract in which the body is found in the subclass, the body of native methods lies outside JVM in a library
- **Native code is written in a non-java language typically C/C++ and is compiled for a single target machine**
- **The standard API for implementing native methods in C is called JNI (Java native Interface)**

Synchronized Modifier

- **Synchronized modifier is used to control access to critical code in multi threaded programs**
- **It can only be applied to methods and local blocks within the methods**

Modifiers

Modifier	Class	Variable	Method	Block
Public	yes	yes	yes	no
Protected	no	yes	yes	no
Default	yes	yes	yes	yes
Private	no	yes	yes	no
Static	yes (inner)	yes	yes	yes
Final	yes	yes	yes	no
Abstract	yes	no	yes	no
Native	no	no	yes	no
Synchronized	no	no	yes	yes(inside method)