seed
beyond the obvious

Core Java

Java Miscellaneous

Objectives

- Identify the need for packages & state how to use them
- How classpath works
- Import and static import
- Describe the classes : String & StringBuffer/StringBuilder
- State what are Wrapper classes

seed
beyond the obvious

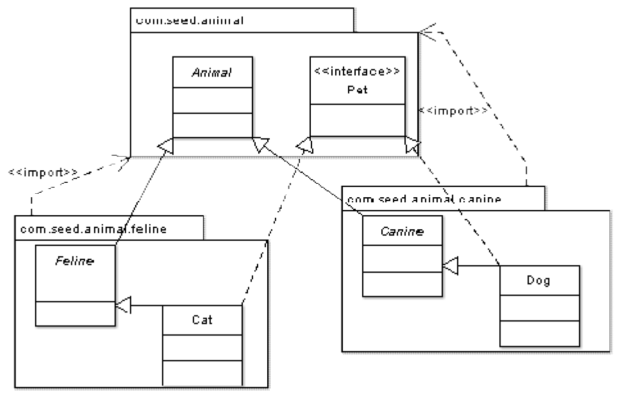
Packages

- Packages are a named collection of classes grouped in a directory.
- Packages are a way of grouping related classes & interfaces
- A package can contain any number of classes that are related in purpose, in scope or by inheritance
- Convenient for organizing your work & separating your work from code libraries provided by others
- Reduce problems with naming conflicts

Steps for Creating a Package

- Use keyword **package** at the beginning of the file.
- Create a subdirectory of that package name.
- Compile the file and keep the .class files in the directory.
- Set the classpath from the root up to the subdirectory created above.
- Use the import keyword whenever the class in the particular package has to be used.

Animal hierarchy packages



Example:- Pet.java, Animal.java,
Canine.java, Dog.java
Feline.java, Cat.java

Using Packages

- To use a public class of a package, simply use the full package name

  E.g. `java.util.Date = new java.util.Date();`
- import statement: allows to import all the public classes in a package

  E.g. `import java.awt.*;`
- If the required class is in java.lang package, it can be used directly

Java Source File Structure

```
// PART 1: (OPTIONAL)
// Package name
package com.company.project.fragilePackage;
```

```
// PART 2: (ZERO OR MORE)
// Packages used
import java.util.*;
import java.io.*;
```

```
// PART 3: (ZERO OR MORE)
// Definitions of classes and interfaces (in any order)
public class NewApp { }
class C1 { }
interface I1 { }
// ...
class Cn { }
interface Im { }
// end of file
```

- Part I – Package Statement (Optional)
- Part II – Import Statements (Zero or More)
- Part III – Definitions of classes or interfaces (Zero or More)

seed
beyond the obvious

Characteristics of Packages

- Import statement imports all the public classes within the package; it does not import sub packages.

- Wild card characters cannot be used.

  E.g. import java.awt.B*; // is illegal

seed
beyond the obvious

Package Scope Access

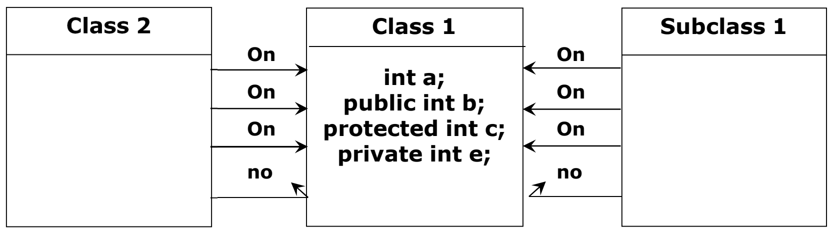
- Default : features of a class having default scope can be accessed by all classes in the same package.
- Protected :

  An entity declared as protected can be accessed

  1. within the same package

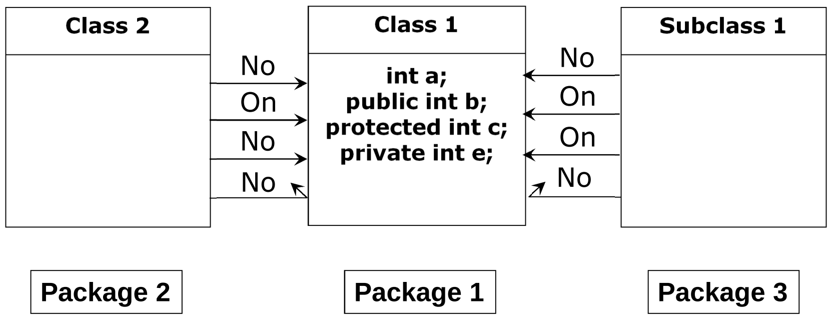
  2. within subclasses of the class in which it is declared

Accessibility From within Same Package



Package 1

Accessibility From different Packages



Import keyword

- To use a class in the API
  - we have to know which package the class is in
  - we have to know the full name of the class we want to use in our code
  - Put an **import** statement at the top of source code file. For example

```
import java.util.Date
public class MyClass {…}
```

OR

  - Type the fully qualified name of the class in your code. For example

```
public class MyClass{
   public void processDate() {
   Date d1 = new java.util.Date();
   }
}
```

Example :- GregCalDemo.java

String Class

- Java library contains a predefined class called String.
- The String type is not a primitive type
- But it is so important, that in certain areas Java treats it like one.

  E.g The ability to declare String literals instead of using new to instantiate a copy of the class

  E.g. `String s = “Hello”;`

String Class

- String class represents an immutable string
  - i.e. Once an instance is created, the string it contains cannot be changed
- To change the string referenced by a string variable, you throw away the reference to the old string & replace it with a reference of the new one

Example:- SubStringDemo.java

Some Methods of String Class

- boolean equals(Object o)
- int length()
- boolean endsWith(String suffix)
- char charAt(int index)
- String replace(char oldchar, char newchar)
- String subString(int begin, int end)
- String toLowercase() / toUpperCase()

Example:- StrTokDemo3.java

StringBuffer Class

- It would be inefficient to use string concatenation at the time of processing the strings, every time we append characters to a string, the string object needs to find new memory to hold the larger string: this is time consuming, results in performance hit
- Solution is to use StringBuffer
- StringBuffer class allows to create mutable strings
- It preallocates memory of a given length
- The buffer grows automatically as characters are added
- E.g. StringBuffer sb = new StringBuffer();

Example:- StringBufferDemo.java

Wrapper Classes

- Java provides 8 primitive data types. But sometimes there's a need to convert a primitive type to an object
- All java primitive types have class counterparts
- These are called object wrappers or wrapper classes

Wrapper Classes

- Need :
    - To provide a home for methods & variables related to the type
    - Create objects to hold values for generically written classes that know how to handle only object references

Wrapper Classes

- Wrapper classes are final
- They furnish methods that provide basic capabilities such as class conversions, value testing etc
- Constructors of wrapped classes allow objects to be created & converted from primitive values & strings

  E.g.
  ```
  int intVal = 10;
      Integer intObj  = new Integer(intVal);
  ```

seed
beyond the obvious