LECTURE 18

Prof. Anita Agrawal,
BITS-Pilani, K.K.Birla Goa Campus

Package???

- Uptill now, in all the examples, a unique name was given to each class to avoid collision, as all the names were taken from the same name space.
- Java provides a mechanism for partitioning the class name space into more manageable chunks.

  ..... “**package**”…….
- Additionally, you can define classes inside a package that are not accessible by code outside that package.
- You can also define class members that are exposed only to other members of the same package.

Package

- Package is namespace that organizes a set of related classes and interfaces.
- A mechanism to encapsulate a group of classes, sub classes and interfaces

in short…

A naming and a visibility control mechanism

Benefits of Package

- Preventing naming conflicts:
  - For example there can be two classes with name city in two packages: car.honda.sedan.city and car.honda.auto.city
- Encapsulation
- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier

Defining a package

- To create a package,
  - Include the ‘package’ command as the first name in a Java source file.

    syntax: package package_name;
  - Any classes declared within that file will belong to the specified package
- If you omit the **package** statement,
  - The classes are put into the default package, which has no name

Most of the time it is best to define a package for your code.

Package

- Java uses file system directories to store packages
- The **.class** files for the classes belonging to a package must be stored in a directory whose name is same as the package
- The directory name must match the package name exactly
- More than one file can include the same **package**
- The **package** statement simply specifies to which package the classes defined in a file belong.

Package Hierarchy

- You can create a hierarchy of packages
- By simply separating each package name from the one above it by use of a period
  - package *pkg1*[.*pkg2*[.*pkg3*]];
- Need to be stored in pkg1\pkg2\pkg3 in a Windows environment

Compilation and Execution

- The easiest way: Simply create the package directories below your current development directory.
- Put the **.class** files into the appropriate directories (packages)
- Then execute the programs from the development directory

Access Protection

- Classes and packages: Means of encapsulating, Acts as containers
- Classes act as containers for data and code
- Packages act as containers for classes and othersubordinate packages
- Four categories of visibility for class members:
- Subclasses in the same package
- Non-subclasses in the same package
- Classes that are neither in the same package nor subclasses
- Subclasses in different packages

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. This is the default access
- If you want to allow an element to be seen outside your current package, but only to classes that subclass your class directly
  - Declare that element **protected**

Class Member Access

	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

Import statement

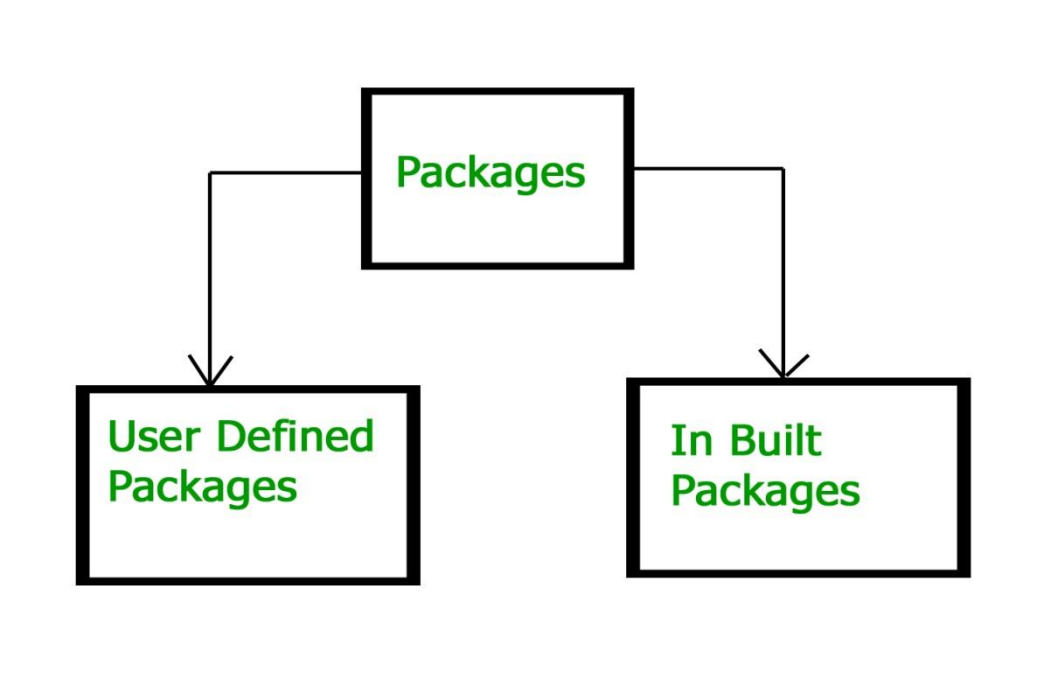
- Packages are a good mechanism for Compartmentalizing diverse classes
- All of the built-in Java classes are stored in packages
- All of the standard classes are stored in some named package
- Classes within packages must be fully qualified withtheir package name
- Tedious to type in the long dot-separated package path name for every class you want to use **import** statement
- Bring certain classes, or entire packages, into visibility

- **import** statements occur
  - immediately following the **package** statement (if it exists)
  - And before any class definitions
- This is the general form of the **import** statement:
  - import *pkg1* [.*pkg2*].(*classname* | *);

Import and public

- When a package is imported
  - Items within the package declared as **public** will be available to non-subclasses in the importing code

Types of Packages



Built-in Packages

- These packages consist of a large number of classes which are a part of Java **API**.
- Some of the commonly used built-in packages are:
  - ✓ **java.lang:** Contains language support classes(e.g classes which define primitive data types, math operations). This package is automatically imported.
  - ✓ **java.io:** Contains classes for supporting input / output operations.
  - ✓ **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
  - ✓ **java.applet:** Contains classes for creating Applets.
  - ✓ **java.awt:** Contain classes for implementing the components for graphical user interfaces (like button , menus etc).
  - ✓ **java.net:** Contain classes for supporting networking operations.

Java Lang package

- **Boolean**: The Boolean class wraps a value of the primitive type boolean in an object.
- **Byte**: The Byte class wraps a value of primitive type byte in an object.
- **Double**: The Double class wraps a value of the primitive type double in an object
- **Float:** The Float class wraps a value of primitive type float in an object.
- **Integer** :The Integer class wraps a value of the primitive type int in an object.
- **Long**: The Long class wraps a value of the primitive type long in an object.

User-defined Packages

- These are the packages defined by the user.
- Using static import feature:
  - Static import is a feature introduced in **Java** programming language (versions 5 and above) that allows members (fields and methods) defined in a class as public **static** to be used in Java code without specifying the class in which the field is defined.

REMAINING TOPICS FROM WRAPPERS

Deprecated Constructors

- Till now, we have used various constructors for creating object of type wrappers
- Character(char ch)
- Integer(int num)
- Integer(String str)
- These constructors are now **deprecated**
- Discouraged from using these constructors
- Better alternative exists

Utility methods

- ValueOf() method
- Parse method

Utility method ‘valueOf()

- Three types:
  - Wrapper valueOf(String s):
    - Every wrapper class except Character class contains a static valueOf() method to create Wrapper class object for given String.
  - Wrapper valueOf(String s, int radix)
    - Every Integer Wrapper class (Byte, Short, Integer, Long) contains the following method to create a Wrapper object for the given String with specified radix. The range of the radix is 2 to 36.
  - Wrapper valueOf(primitive p) :
    - Every Wrapper class including Character class contains the following method to create a Wrapper object for the given primitive type.

valueOf() for string

```
class integ {
    public static void main(String[] args)
    {
        Integer I = Integer.valueOf(“25");
        System.out.println(I);
        Double D = Double.valueOf(“25.0");
        System.out.println(D);
        Boolean B = Boolean.valueOf("true");
        System.out.println(B);
}
}
```

- Integer I1 = Integer.valueOf(“one"); Will give an error.

Output:25

25.0

true

Wrapper valueOf(String s, int radix)

```
class radi {
    public static void main(String[] args)
    {
                Integer I = Integer.valueOf("1110", 2); //1110 is expressed in binary,
                                                        convert to decimal
        System.out.println(I);
        Integer I1 = Integer.valueOf("1110", 3);  //1110  is expressed in base 3.
                                                  convert it into decimal
        System.out.println(I1);
    }
}
```

Output:

14

39

Wrapper valueOf(primitive p)

```
class cha {
    public static void main(String[] args)
    {
        Integer I = Integer.valueOf(25);
        Double D = Double.valueOf(25.0);
        Character C = Character.valueOf('a');
        System.out.println(I);
        System.out.println(D);
        System.out.println(C);
    }
}
```

Utility method of wrapper class: Parse

parseXxx() method to convert String to primitive.

Two types:

primitive parseXxx(String s) : Every Wrapper class except character class contains the following parseXxx() method to find primitive for the given String object.

parseXxx(String s, int radix) : Every Integer type Wrapper class (Byte, Short, Integer, Long) contains the following parseXxx() method to convert specified radix String to primitive.

Converting string to object via parse…

Example : parse