Packages

By Rahul Barve

Objectives

- Understanding Packages
- Need for Packages
- Access Modifiers Revisited
- Exploring `java.lang`

Package

Package

- Package is a collection of classes and interfaces.
- Used to keep class library isolated from other libraries.
- Can be used to reduce naming conflicts about classes and interfaces.

Creating a Package

Creating a Package

- Packages are created using `package` statement.
- If used, it must be the first statement in the Java source file.

Creating a Package

- Syntax:

```
package <package-name>;
   //class definition
```

- E.g.

```
package test;
   public class Test {
   ….
   }
```

Accessing Classes

- If two or multiple classes are belonging to same package, one class can directly access other classes irrespective of whether they are declared as `public` or not.

Accessing Classes

- E.g.

```
package business;
public class Address {….}

package business;
public class Customer {
   Address commAddress;
   …
}
```

Accessing Classes

- If classes belong to different packages, then one class has to import classes from other packages provided they are declared as `public`.
- This is done using the `import` statement.

Accessing Classes

- E.g.

```
package residence;
public class Address {…}

package college;
import residence.Address;
public class Student {
    Address residentialAddress;
    …
}
```

Sub Packages

Sub Packages

- A package within another package is called as a sub package.
- To import classes from a sub package, the name of the super package is mandatory.
- E.g.

```
import p1.p2.*;
```

Default Package

Default Package

- Whenever a class is declared without `package` statement, then that class is said to be a part of a `default package`.

Default Package

- Such classes cannot be imported by classes coming from other packages; and hence the use of default package is discouraged.

Access Modifiers Revisited

By Rahul Barve

Access Modifiers Revisited

- Java provides 4 access modifiers: `private`, `public`, `protected` and `default`.
- Except `private`, remaining behave same unless different packages are used.

Access Modifiers Revisited

- If different packages are used then:
  - `public` makes the member accessible from anywhere.
  - `protected` makes the member accessible throughout the entire package as well as outside the package if the class is a subclass.
  - `default` makes the member accessible throughout the entire package but not outside the package. Hence it is also known as package level access modifier.

Exploring `java.lang`

By Rahul Barve

Exploring `java.lang`

- Java provides a predefined class library and all classes belong to some specific packages as it’s a standard practice.
- One of the predefined packages of Java library is `java.lang`.

Exploring `java.lang`

- It is the package that is imported by default and hence classes belonging to `java.lang` can be used directly even without any `import` statement.

Exploring `java.lang`

- The 2 basic classes used so far:
  - `String`
  - `System`

Exploring `java.lang`

- Other Classes:
  - `Object`
  - `StringBuilder`
  - Wrapper Classes

`Object` Class

- It is the topmost class in a Java class hierarchy.
- All classes either predefined or user defined are implicitly inherited from `Object`.

`Object` Class

- A variable of type `Object` can be used to refer to objects of any type.
- E.g.

```
Object emp = new Employee();
Object cst = new Customer();
```

Methods of Object Class

- `toString()`
- `finalize()`
- `equals(Object)`
- `clone()`
- `hashCode()`
- `getClass()`
- `wait()`
- `notify()`
- `notifyAll()`

`toString()`

- Returns a `String` that represents a value of an object on which it is invoked.
- Has to be overridden in the class.
- An object can be converted into a String with explicit or implicit call.

`finalize()`

- Can be overridden by a class; executes implicitly when GC is about to run.
- Generally overridden to perform clean-up operations.

equals(Object)

- `public boolean equals(Object)`
- Used to test whether one object is equal to another or not.

Wrapper Classes

Wrapper Classes

- Java provides 8 primitive data types.
- Sometimes, there’s a need to convert primitives into objects.
- All Java primitive types have class counterparts, called as wrappers or wrapper classes.

Wrapper Classes

- `Byte`
- `Short`
- `Integer`
- `Long`
- `Float`
- `Double`
- `Character`
- `Boolean`

Using Wrapper Classes

- Pre JDK 1.5

```
int val1 = 100;
Integer iVal1 = new Integer(val1);
int val2 = iVal1.intValue();
```

Using Wrapper Classes

- Since JDK 1.5

```
 int val1 = 100;
Integer iVal1 = val1 //Autoboxing
int val2 = iVal1; //Unboxing
```

Using Wrapper Classes

- Wrapper classes can also be used to convert a qualified String into the appropriate primitive type.
- E.g.

```
int x = Integer.parseInt("1234");
  double y =
 Double.parseDouble("3445.56");
```

`String` Class

By Rahul Barve

`String` Class

- Java library provides a predefined class `String`.
- Java Strings are First Class objects.
- Represents an immutable string.
- Degrades the performance.

StringBuilder Class

By Rahul Barve

`StringBuilder` Class

- Java library provides an alternative called as `StringBuilder`.
- Represents mutable strings.

`StringBuilder` Class

- Used to improve the performance.
- E.g.

```
StringBuilder buf;
buf = new StringBuilder(“Hello”);
```

Lets Summarize

- What are Packages
- Benefits of Packages
- Access Modifiers Revisited
- Classes from `java.lang`