Packages and import

James Brucker

Packages

- Java uses packages to organize classes.
- Packages reduce size of *name space* and avoid *name conflicts (two classes with same name)*

Example: there are 2 `Date` classes.

`java.util.Date` "`Date`" class in `java.util`

`java.sql.Date` "`Date`" class in `java.sql`

Core Packages

`java.lang`	Java language core classes `Object, String, System, Integer, Double, Math` You never need to "import" classes in java.lang. Its automatic.
`java.io`	Classes for input and output `InputStream, BufferedReader, File`
`java.util`	Collections, utilities, old Date/time `Calendar, Date, Scanner, List, ArrayList, Set`
`java.time`	LocalDate, LocalTime, DateTime, Duration

Useful Packages

`java.net`	Network access `URL, URI, Socket`
`javafx`	Java FX graphics framework `Button, Scene, Animation, event handlers`
`javax.swing`	Older Swing graphics framework `JButton, JFrame, etc.`

Importing classes

Write "import" statements at top of file,
after the "package" statement (if you have one).

```
package coinpurse;
import java.util.Scanner;
import java.util.List;
/**
 * User interface for coin purse.
 */
public class ConsoleDialog {
      Scanner console = new Scanner( System.in );
      ...
```

imports come after package statement and before class Javadoc comment.

What is "import"?

`import` tells the compiler *where* to find classes.

It doesn't actually "import" any code!

```
package guessinggame;
import java.util.Random;
/**
 * User interface for guessing game.
 */
public class GameDialog {
      private Random rand = new Random( );
      ...
```

tell the compiler where to find the Random class

Why **import**?

The reason for "import" to to resolve ambiguity.

Many classes can have the *same name*.

Java API has 2 classes named "`Date`".

5 "`Element`" classes and interfaces.

3 "`Timer`" classes.

If your program uses a `Date`, you need import to specify which `Date` you want:

```
import java.util.Date;
class Appointment {
     private Date startDate;
```

Import Everything

You can import everything from a package. Use *

```
package graphics;
import java.util.*;  // Date, List, Scanner, ...
import java.io.InputStream;

class Person {
      private static Scanner console = ...;
      private Date birthday;
      private List<Person> friends;
      ...
```

Ambiguity in Import

If a class matches more than one wildcard "*", Java requires you to resolve the ambiguity using an import without the wildcard.

Example: There are 2 Date classes: `java.util.Date` and `java.sql.Date`. These imports are *ambiguous:*

```
import java.util.*;
import java.sql.*;
/** a class using a Date */
class Ambiguous {
     private Date today;
```

which **`Date`** class should Java use?

How to Resolve Ambiguity?

There is a `java.util.Date` and `java.sql.Date`

```
import java.util.*;
import java.sql.*;
class Ambiguous {
      Date today = new Date( );
```

Resolving Ambiguity

There are two ways to resolve ambiguity.

1. import a specific class (no wildcard)
2. use the fully qualified name in Java code

```
import java.util.*;
import java.sql.*;
import java.util.Date; // Solution #1
class Ambiguous {
     private Date today = new Date( );
      // Solution #2
     private java.sql.Date mdate
               = new java.sql.Date( );
```

import and namespace

A name space means the collection of all names or words that are defined at some point in your code.

The Java compiler uses a namespace to compile code.

```
import java.util.Scanner;
class Person {
     private String name;
     public void setName(String aname) {
```

Name space includes:
Scanner, Person, setName, aname, name
+ everything in java.lang

```
     }
```

`import` and namespace

"`import`" simply adds more names to the compiler's namespace.

It does not have any effect on the size of compiled code.

```
import java.util.*;
class Person {
    private String name;
```

import static

"`import static`" is used to add static members of a class to the namespace.

It is a ***convenience*** so the programmer does not need to type the class name.

```
import static java.lang.Math.abs;
class MyClass {
     private double mean;
     public double deviation(double x) {
          return abs( x - mean );
     }
```

Same as **Math.abs**( )

`import static` for `System`

"`import static Math.abs`" is <u>not</u> useful: it makes the meaning of "abs" <u>less</u> clear.

`import static` is <u>more</u> useful for reducing lots of redundant text that makes code harder to read.

```
import static java.lang.System.out;
class MyClass {
   public static void main(String[] args) {
      out.print("I hate typing ");
      out.println("System.out so much");
   }
```

`import static` with wildcard *

"`import static`" can use wildcard to mean "import all static members".

Example: JOptionPane has a lot of static constants for dialog options.

```
import static javax.swing.JOptionPane.*;
class MyClass {
   public String getReply(String prompt) {
      showInputDialog(null, prompt,
            "input", QUESTION_MESSAGE);
   }
```

Why use `package`?

- Oracle recommends you <u>always</u> use a package for your code.

Why?

1. Default package cannot be imported. Therefore...
2. classes in the default package cannot be "seen" by classes in other packages.

Package Names use Domain Name

Convention: use domain name in reverse order for base package name.

- `http://junit.org` is the home for the JUnit unit testing framework.

  The package name for JUnit is:

  `org.junit`

- `http://commons.apache.org` provides reusable software for Java. It contains many subprojects.

  The base package name for Apache Commons is:

  `org.apache.commons`