“Venus Williams Opens Jamba Juice At Dupont Circle” by www.glynlowe.com https://flic.kr/p/cuMMw3

Java Programming 2 Packages

Mary Ellen Foster

MaryEllen.Foster@glasgow.ac.uk

Packages

Package: groups together related resources (usually classes)

Why put code in a package?

- Makes it obvious that types are related
- Makes it possible to find types for a specific purpose (given good package name)
- Type names won't conflict with names from other packages
- Types within package can have unrestricted access to one another
  - *While restricting access for types outside the package*

Visibility modifiers (revisited)

Modifier	Same class	Same package	Any subclass	Any class
`public`	●	●	●	●
`protected`	●	●	●	
(default)	●	●		
`private`	●			

Creating a package

Choose a name (details on naming scheme next)

Put a `package` statement at the top of every source file for that package

```
  package my.package.name;
```

Ensure that all source files are in a directory corresponding to the package name

If you don't use a `package` statement, then all files are in the default package

Package naming conventions

Rules are the same as java identifiers

- Can't start with a digit; can't contain special characters or hyphens; can't contain a reserved keyword such as `int` or `new`
- Components separated by period "."

Conventions

- All lower case
- Built-in packages start with `java.` or `javax.`
- Companies and organisations usually use their domain name, reversed:
  - *`com.example.mypackage` – a package named `mypackage` from a programmer at* ***example.com***
  - *`uk.ac.glasgow.dcs.jp2` – possible package for code for this class*

Accessing package members

To use a public type from outside the package, do one of:

- Refer to it by its fully qualified name
- Import the member itself
- Import the entire package

Note: `java.lang` is always available by default with no special effort

- `String, Double, Exception,` …

Other packages are imported by default in JShell for convenience

- java.io, java.math, java.net, java.nio.file, java.util, and lots of java.util subpackages
- Just about any packages we are using during this course, actually!

Using fully qualified name

```
java.util.Scanner stdin = new java.util.Scanner(System.in);
```

Works well for infrequent use

Code can easily become repetitive and hard to read

Importing a single member

```
// At top of source file, after package statement
import java.util.Scanner;
// … later on, inside the class …
Scanner stdin = new Scanner(System.in);
```

Works well if you use a few members from a package

Importing a package

```
// At top of source file, after package statement
import java.util.*;
// … later on, inside the class …
Scanner stdin = new Scanner(System.in);
```

Useful if you need lots of members from the same package

Use is controversial:

- http://stackoverflow.com/questions/147454/why-is-using-a-wild-card-with-a-java-import-statement-bad

More on package names

Package names look like they *might* be a hierarchy

```
java
  java.awt
     java.awt.color
     java.awt.font
     …
```

But they are **not**!

`import java.awt.*` does **not** import any classes from `java.awt.color` or anywhere else

File names and directories

Source code for a class should be in a file corresponding to the class name

```
public class CreditCard { … }
```

CreditCard.java

Package determines the directory that the file should be in

```
package uk.ac.glasgow.dcs.jp2;
public class CreditCard { … }
```

…\uk\ac\glasgow\dcs\jp2\CreditCard.java

All paths are relative to the current working directory

Working with packages in Eclipse

"Package explorer" view – useful when you go beyond the default package

Creating a new package:

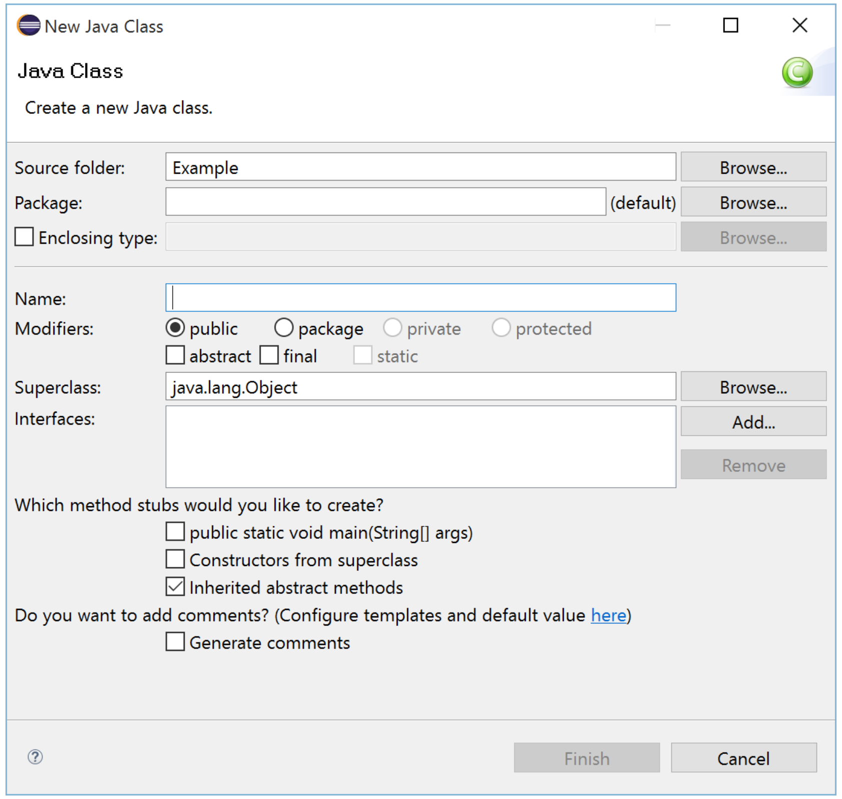
- Right-click project -> New -> Package

Creating a class in a package

- Right-click project -> New -> Class
- Specify package – it will be created if it does not already exist

Moving class to a new package

- Right-click class -> Refactor -> Move



New Java Class

Java Class

Create a new Java class.

Source folder:	Example	Browse...
Package:	(default)	Browse...
☐ Enclosing type:		Browse...
Name:		
Modifiers:	● public ○ package ○ private ○ protected	
	☐ abstract ☐ final ☐ static	
Superclass:	java.lang.Object	Browse...
Interfaces:		Add... / Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods

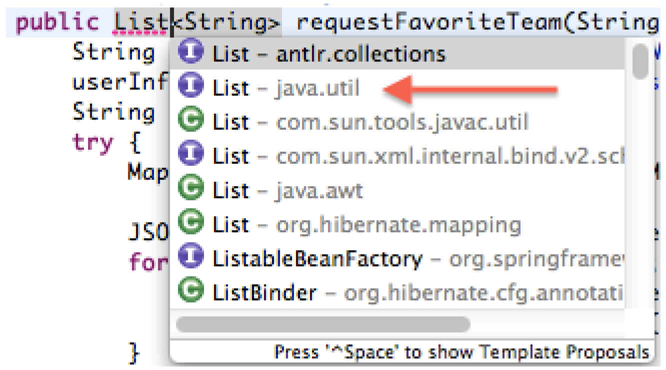
Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Managing imports in Eclipse

Essential keyboard shortcuts:

- **Ctrl-<space>** on (partial) class name*
  - *Pops up class-name autocompletion*
  - *Once you choose a class, it automatically adds the necessary `import` statement*
- **Ctrl-Shift-O** (letter “o”, not number “zero”)
  - *Organises imports*
  - *Removes unused ones*
  - *Sorts the rest nicely*



```
public List<String> requestFavoriteTeam(String
    String
    userInf
    String
    try {
        Map

        JSO
        for

        }
```

List – antlr.collections
List – java.util
List – com.sun.tools.javac.util
List – com.sun.xml.internal.bind.v2.scl
List – java.awt
List – org.hibernate.mapping
ListableBeanFactory – org.springframe
ListBinder – org.hibernate.cfg.annotati
Press '^Space' to show Template Proposals

Image from *[https://mihail.stoynov.com/2011/08/24/longstanding-eclipse- issues-fix-them-finally-please/](https://mihail.stoynov.com/2011/08/24/longstanding-eclipse-issues-fix-them-finally-please/)*

* Ctrl-<space> also works to autocomplete many other things – fields, method names, variables, exceptions, even whole methods (try “mai Ctrl-<space>”). Try it out and see!