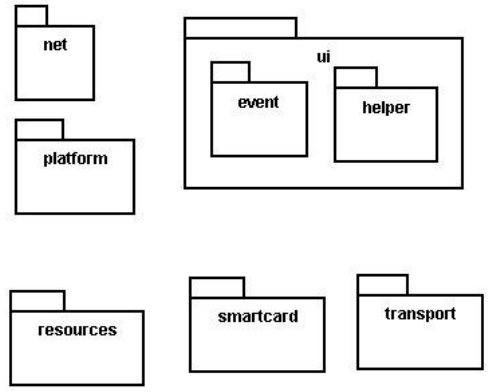
Java packages

- **package**: A collection of related classes.
  - Can also "contain" sub-packages.
  - *Sub-packages* can have similar names, but are not actually contained inside.
    - `java.awt` does not contain `java.awt.event`



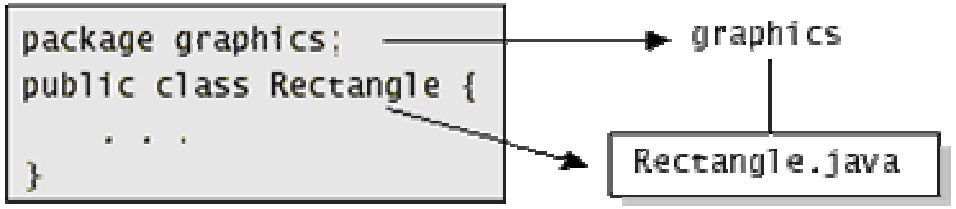
- Uses of Java packages:
  - group related classes together
  - as a *namespace* to avoid name collisions
  - provide a layer of access / protection
  - keep pieces of a project down to a manageable size

Packages and directories

- package ←→ directory (folder)
- class ←→ file

- A class named `D` in package `a.b.c` should reside in this file:

`a/b/c/D.class`



  - (relative to the root of your project)

- The "root" directory of the package hierarchy is determined by your *class path* or the directory from which `java` was run.

Classpath

- **class path**: The location(s) in which Java looks for class files.

- Can include:
  - the current "working directory" from which you ran javac / java
  - other folders
  - JAR archives
  - URLs
  - ...

- Can set class path manually when running java at command line:
  - `java -cp /home/stepp/libs:/foo/bar/jbl MyClass`

A package declaration

```
package name;

public class name { ...
```

Example:

```
package pacman.model;

public class Ghost extends Sprite {
    ...
}
```

- File `Sprite.java` should go in folder `pacman/model`.

Importing a package

```
import packageName.*;          // all classes
```

Example:

```
package pacman.gui;
import pacman.model.*;

public class PacManGui {
    ...
    Ghost blinky = new Ghost();
}
```

- `PacManGui` must import the model package in order to use it.

Importing a class

```
import packageName.className;   // one class
```

Example:

```
package pacman.gui;
import pacman.model.Sprite;

public class PacManGui {
    Ghost blinky = new Ghost();
}
```

- Importing single classes has high precedence:
  - if you import `.*`, a same-named class in the current dir will override
  - if you import `.className`, it will not

Static import

```
import static packageName.className.*;
```

Example:

```
import static java.lang.Math.*;

...
double angle = sin(PI / 2) + ln(E * E);
```

- Static import allows you to refer to the members of another class without writing that class's name.
- Should be used rarely and only with classes whose contents are entirely static "utility" code.

Referring to packages

packageName . className

Example:

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

- You can use a type from any package without importing it if you write its full name.
- Sometimes this is useful to disambiguate similar names.
  - Example: `java.awt.List` and `java.util.List`
  - Or, explicitly import one of the classes.

The default package

- Compilation units (files) that do not declare a package are put into a default, unnamed, package.

- Classes in the default package:
  - Cannot be imported
  - Cannot be used by classes in other packages

- Many editors discourage the use of the default package.

- Package java.lang is implicitly imported in all programs by default.
  - `import java.lang.*;`

Package access

- Java provides the following access modifiers:
  - `public` : Visible to all other classes.
  - `private` : Visible only to the current class (and any nested types).
  - `protected` : Visible to the current class, any of its subclasses, and any other types within the same package.
  - default (package): Visible to the current class and any other types within the same package.

- To give a member default scope, do not write a modifier:

```
package pacman.model;
public class Sprite {
    int points;       // visible to pacman.model.*
    String name;      // visible to pacman.model.*
```

Package exercise

- Add packages to the Rock-Paper-Scissors game.
  - Create a package for core "model" data.
  - Create a package for graphical "view" classes.

  - Any general utility code can go into a default package or into another named utility (util) package.
  - Add appropriate package and import statements so that the types can use each other properly.