Paquetes

Nesta unidade verase como agrupar clases e interfaces en paquetes, como usar clases que se atopen en determinados paquetes e como organizar o sistema de ficheiros para que o compilador poda atopar os nosos ficheiros de código fonte.

Sumario

Creación de paquetes

Un paquete é un grupo de **tipos** relacionados que proporciona protección de acceso e un espazo de nomes. Os tipos refírense a clases, interfaces, enumeracións e anotacións.

En Java os tipos organízanse en paquetes. Así, no paquete java.lang atopamos clases fundamentais da linguaxe; no paquete java.io temos clases para escritura e lectura, etc. As nosas clases tamén se poden agrupar en paquetes.

Supoñamos que temos un grupo de clases para representar obxectos gráficos como círculos, rectángulos, liñas e puntos. Temos adeamis un interface, Arrastrable, que implementan as clases que poden arrastrarse co rato.

```
// no ficheiro Arrastrable.java
public interface Arrastrable  {
        . . .
}

// no ficheiro Grafico.java
public abstract class Grafico {
        . . .
}

// no ficheiro Circulo.java
public class Circulo extends Grafico implements Arrastrable {
        . . .
}

// no ficheiro Rectangulo.java
public class Rectangulo extends Grafico implements Arrastrable {
        . . .
}

// no ficheiro Punto.java
public class Punto extends Grafico implements Arrastrable {
        . . .
}

// no ficheiro Linea.java
public class Linea extends Grafico implements Arrastrable {
        . . .
}
```

Deberíamos agrupar estas clases e o interface nun paquete polos seguintes motivos::

- Outros programadores, e incluso nós mesmos, poden determinar facilmente que estes tipos están relacionados.
- Outros programadores, e incluso nós mesmos, saben onde atopar tipos que proporcionen funcións relacionadas cos gráficos.

- Os nomes destes tipos non entrarán en conflito cos nomes dos tipos noutros paquetes porque o paquete crea un novo espazo de nomes.
- Pódese permitir que os tipos dentro do paquete teñan acceso sen restrinxir entre eles, aínda que os tipos de fóra do paquete non podan.

Para crear un paquete hai que escribir no principo de cada ficheiro fonte que queramos que pertenza a ese paquete o seguinte:

```
package nome_do_paquete;
```

A sentenza `package` (por exemplo, `package graficos;`) debe ser a primeira liña no ficheiro fonte. Só pode haber unha sentenza deste tipo en cada ficheiro fonte e aplícase a todos os tipos no ficheiro. No exemplo anterior teríamos:

```
// no ficheiro Arrastrable.java
package graficos;
public interface Arrastrable {
        . . .
}

// no ficheiro Grafico.java
package graficos;
public abstract class Grafico {
        . . .
}

// no ficheiro Circulo.java
package graficos;
public class Circulo extends Grafico implements Arrastrable {
        . . .
}

// no ficheiro Rectangulo.java
package graficos;
public class Rectangulo extends Grafico implements Arrastrable {
        . . .
}

// no ficheiro Punto.java
package graficos;
public class Punto extends Grafico implements Arrastrable {
        . . .
}

// no ficheiro Linea.java
package graficos;
public class Linea extends Grafico implements Arrastrable {
        . . .
}
```

Se estamos nas primeiras fases de desenvolvemento da aplicación ou esta é moi pequena non é necesario empregar paquetes. No caso contrario é moi recomendable polos motivos xa comentados anteriormente.

Nomes de paquetes

Dada a cantidade de programadores existentes por todo o planeta é bastante fácil que dous deles coincidan á hora de poñer nome a un tipo (clase, interface, etc.). Isto non é problema porque o compilador distíngueos a partir do nome completo. Así, no exemplo anterior, a clase `Rectangulo` para o compilador é `graficos.Rectangulo;`.

O problema aparece cando dous programadores coinciden tamén no nome do paquete que están a usar. Para estes casos existe unha norma para evitar as duplicidades.

Normas

- Os nomes dos paquetes escríbense en minúsculas para evitar confundilos coas clases e interfaces.
- As compañías deben usar o seu nome de dominio de Internet ao revés para nomear os paquetes. Por exemplo: net.iessanclemente.mestre para un paquete creado na máquina mestre por un programador no iessanclemente.net. Os posibles confitos dentro da propia institución deben resolverse internamente.
- Os paquetes da linguaxe de programación Java comezan por `java.` or `javax`.
- Se o dominio de Internet non é un nome válido (por exemplo, porque empece por un carácter non válido ou conteña unha palabra reservada

como `int`) suxírese o seguinte:

```
lipart-open.org pasa a ser org.clipart_open
free.fonts.int pasa a ser int_.fonts.free
poetry.7days.com pasa a ser com._7days.poetry
```

Uso de paquetes

Aos tipos que forman parte dun paquete chámaselles **membros do paquete**. Para poder usar un membro **público** dun paquete dende fóra dese paquete hai que facer algunha das seguintes cousas:

- Referirse ao membro polo seu nome completo
- Importar o membro dende o paquete
- Import o paquete enteiro

Uso mediante o nome completo

Pódese usar o nome do membro dun paquete sen máis sempre e cando o código que se estea a escribir estea no mesmo paquete que o membro ou se se importou o membro explicitamente.

Con todo, se se quere usar un membro dende outro paquete e o paquete non se importou hai que usar o nome completo do membro para referirse a el. Por exemplo, para a clase Rectangulo anterior:

```
graficos.Rectangulo
```

Así, para usar a clase Rectangulo poderiamos escribir o seguinte:

```
graficos.Rectangulo meuRect = new graficos.Rectangulo();
```

Esta forma de acceder aos membro dun paquete non se usa normalmente porque hai que coñecer o nome completo do paquete ao que pertence o membro.

Importación dun membro do paquete

Para importar o membro dun paquete escríbese a palabra import ao principio do ficheiro de código fonte, antes da definición de calquera tipo (clase, interface, etc.) pero despois da palabra package:

```
import graphics.Rectangle;
```

Para referirnos á clase Rectangulo xa podemos usar o seu nome directamente:

```
Rectangulo meuRectangulo = new Rectangulo();
```

Isto é útil se só queremos importar un membro dun paquete pero se usamos moitos tipos dun paquete habería que importar o paquete enteiro.

Importación do paquete completo

Para importar o paquete completo úsase o carácter comodín *:

```
import graficos.*;
```

Así, podemos referirnos a calquera clase do paquete graficos do seguinte xeito:

```
Circulo meuCirculo = new Circulo();
    Rectangulo meuRectangulo = new Rectangulo();
```

O asterisco só se pode usar para especificar todas as clases dentro dun paquete pero non para un conxunto concreto. Por exemplo, o seguinte código dará un erro de de compilación

```
import graphics.A*;     //Non importará todas clases que emprecen por A
```

O compilador de Java importa automaticamente o paquete `java.lang`.

Xerarquía aparente de paquetes

A estruturación de paquetes non é necesariamente xerárquica. Por exemplo, a API de Java inclúe o paquete `java.awt`, o paquete `java.awt.color`, o paquete `java.awt.font` e moitos outros que empezan por java.awt. Con todo, o paquete `java.awt.color`, `java.awt.font` e outros que empezan por `java.awt.xxxx` **non están incluídos no paquete `java.awt`**. O prefixo java.awt (Java Abstract Window Toolkit) úsase para referirse a un número de paquetes que están relacionados, sen máis.

Se importamos `java.awt.*` importaremos todos os tipos dese paquete, pero non importaremos o paquete `java.awt.color`, `java.awt.font` e outros que empezan por `java.awt.xxxx`. Se queremos usar outros tipos en `java.awt.color` hai que importar o paquete explicitamente:

```
import java.awt.*;
import java.awt.color.*;
```

Nomes ambiguos

Pode darse o caso de que dous tipos teñan o mesmo nome e pertenzan a paquetes distintos. Se importamos os dous paquetes e queremos usar un tipo, por exemplo, unha clase, que estea nos dous hai que referirse a ela empregando o nome completo. Por exemplo, temos unha clase `Rectangle` que pertence ao paquete `graficos` e existe outra clase Rectangle no paquete java.awt. Para referirnos a esa clase usaremos o nome completo:

```
graphics.Rectangle rect;
```

Importación de constantes

O paquete `java.lang.Math` define a constante PI e varios métodos de clase. Por exemplo:

```
public static final double PI 3.141592653589793
public static double cos(double a)
```

Normalmente, para usar estes métodos e constantes hai que escribir o nome da clase como segue:

```
double r = Math.cos(Math.PI * theta);
```

Pódese usar a sentenza **import static** para importar este tipo de métodos e constantes e evitar así ter que escribir sempre o prefixo Math:

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

Ou tamén:

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

E para usalos faríamos o seguinte:

```
double r = cos(PI * theta);
```

Ficheiros fonte e .class

Xa vimos que as compañías, por convención, usan o seu nome de dominio ao revés para establecer os nomes dos paquetes. Por exemplo, se temos unha empresa con dominio exemplo.com e un paquete graficos que contén o ficheiro Rectangulo.java teremos o nome de paquete com.exemplo.graficos. O ficheiro fonte estará no directorio seguinte:

```
....\com\example\graphics\Rectangle.java
```

En Linux o carácter separador é /.

Cando compilamos un ficheiro fonte o compilador crea un ficheiro por cada tipo (clase, interface, etc.) definido nel. O nome base do ficheiro de saída é o nome do tipo e a súa extensión é.class. Por exemplo, se o ficheiro fonte é o seguinte:

```
// Ficheiro Rectangulo.java
    package com.exemplo.graficos;
    public class Rectangulo{
          . . .
    }
```

```
class UnhaClase{
      . . .
}
```

Os ficheiros compilados estarán nas seguintes rutas

```
<ruta ao directorio pai dos ficheiros de saída>\com\exemplo\graficos\Rectangulo.class
<ruta ao directorio pai dos ficheiros de saída>\com\exemplo\graficos\UnhaClase.class
```

Os ficheiros fonte e os binarios normalmente están almacenados en directorios distintos:

```
\fontes\com\exemplo\graficos\Rectangulo.java
\clases\com\exemplo\graficos\Rectangulo.class
```

A ruta completa ao directorio onde están as clases está na variable do sistema **CLASSPATH**. No caso anterior, o CLASSPATH sería \clases se o paquete é com.exemplo.graficos. O compilador e a JVM buscarán os ficheiros .class no directorio:

```
\classes\com\exemple\graficos
```

A variable CLASSPATH pode incluír varias rutas separadas por puntos e coma (Windows) ou por comas (Linux). Por defecto, o compilador e a JVM buscan no directorio actual e no ficheiro JAR de Java polas clases, así que non é preciso incluír estes directorios.