Programación Orientada a Objetos
Sesión 7: Excepciones, Miembros y Métodos Estáticos y Paquetes de Clases
Curso 2017-18

Félix Gómez Mármol, Santiago Paredes Moreno y Gregorio Martínez Pérez
{felixgm, chapu, gregorio}@um.es

Índice

I Teoría **1**

1. Concepto de excepción **1**

2. Lanzar excepciones **1**

3. Capturar excepciones **2**

4. Crear excepciones **3**

5. Miembros y métodos estáticos **4**

6. El método estático `main()` **5**

7. Los paquetes de clases y su uso **5**

8. Creando paquetes de clases **5**

II Prácticas **6**

9. Ejercicios sobre excepciones, miembros y métodos estáticos y paquetes de clases **6**

Parte I
Teoría

1. Concepto de excepción

Al invocar un método estamos delegando la realización de una tarea en el objeto receptor del mismo. Generalmente, el resultado depende del estado del objeto y de la información que le suministramos a través de los argumentos usados en la llamada. Pero si los parámetros del método contienen datos inapropiados, el objeto receptor podría detectar el problema pero, probablemente, no podrá corregirlo.

Por el contrario, quien invocó al método puede que sí sea capaz de hacerlo. Por ejemplo, pensemos en un objeto al que se le pide que haga una división de dos números que, tras ser introducidos por el usuario, le son pasados como argumentos. El método puede detectar que el denominador es 0 y, por lo tanto, que no se puede hacer la división, pero no puede hacer nada al respecto. Sin embargo, desde el código en el que se produjo la invocación, sí que podríamos volver a pedir el denominador al usuario.

Las **excepciones** son un mecanismo por el cual un método puede informar a su llamador (al método que lo invoca) de la aparición de un problema que no puede tratar para que éste se encargue de hacerlo. En Java existe una jerarquía de clases, encabezada por la clase `Throwable`, que permiten especificar diferentes causas de error. `Throwable` tiene dos subclases directas, `Exception` y `Error`. La primera se usa para informar de problemas para los cuales puede haber solución, mientras que la segunda se usa para los problemas muy graves. Generalmente trabajaremos con objetos de la clase `Exception` o sus clases derivadas.

`Exception` tiene distintas subclases cada una de las cuales se refiere a un tipo de problema. Además, todas estas clases tienen dos constructores: uno sin parámetros y otro con un parámetro de tipo `String` que sirve para almacenar un mensaje de error específico en el objeto que representa el error.

2. Lanzar excepciones

Para informar al método llamador sobre el problema que se ha detectado creamos un objeto de la subclase de `Throwable` que mejor lo represente y lo *lanzamos* con la palabra reservada `throw`. El listado 1 muestra la clase `Circulo` cuyo método `setRadio()` está preparado para detectar valores cero o negativos del radio que consideramos carecen de sentido. En cada caso genera una excepción distinta personalizada con un mensaje de texto que es lanzada con `throw`.

En la definición de un método que puede lanzar excepciones debemos especificar cuáles son. Para hacer esto incluiremos justo antes del código del método la palabra reservada `throws` seguida de la lista de subclases de `Throwable` que pueden ser lanzadas separadas por comas.

```
1  public class Circulo extends Figura {
2       public void setRadio(double radio) throws IllegalArgumentException {
3         if (radio < 0) {
4             throw new IllegalArgumentException("Error: Valor negativo del radio");
5         }
6         if (radio == 0) {
7             throw new IllegalArgumentException("Error: Valor cero del radio");
8         }
9
10        this.radio = radio;
11    }
12 }
```

Listado 1: Ejemplo de lanzamiento de excepciones en Java

3. Capturar excepciones

La ejecución de `throw`, es decir, el lanzamiento de una excepción, termina con la ejecución del método donde se produce. Si el método llamador no es capaz de **manejar o capturar la excepción**, es decir, no está preparado para corregir el problema, también terminará y así sucesivamente hasta terminar el programa completo. Java incluye una estructura para especificar la forma en que se pueden manejar las excepciones denominada bloque `try-catch`.

El listado 2 muestra cómo se escribe un bloque `try-catch`. El bloque encabezado por la palabra reservada `try` incluye el código que puede generar excepciones. A continuación, por cada excepción que esperemos capturar, añadimos una entrada `catch` con el bloque de código que deba ejecutarse al capturar dicha excepción. Por último, si deseamos que se ejecute cierto código independientemente de si se produce o no la excepción, lo añadiremos en un bloque final encabezado por la palabra reservada `finally`.

```
1  public class Main {
2     public static void main(String[] args) {
3           Circulo circulo = new Circulo("Blanco", 100, 100, 5.0);
4           try {
5               circulo.setRadio(-5);
6           } catch (IllegalArgumentException iae) {
7               System.out.println(iae.getMessage());
8           } finally {
9               System.out.println("Se ha ejecutado setRadio()");
10          }
11    }
12 }
```

Listado 2: Captura de una excepción en Java

En el ejemplo, al ejecutarse el método `setRadio(-5)`, se detecta un radio con valor negativo y se lanza la excepción. La ejecución del método se detiene y se vuelve al llamador que es el método `main()`. Como el método se ejecutó dentro de un bloque `try-catch`, Java compara en orden las clases usadas en los distintos bloques `catch` con el tipo de la excepción lanzada y cuando alguna concuerda ejecuta el código asociado.

En este caso, el primer bloque es el que coincide, ya que la excepción lanzada es de la clase `IllegalArgumentException`. La excepción particular que se ha lanzado se queda contenida en el objeto `iae`, que tiene este nombre por la concatenación de la primera letra de las palabras que dan nombre a la excepción capturada que es de tipo `IllegalArgumentException`, esto es, `iae`. Sobre este objeto se aplica el método `getMessage()` que permite mostrar el mensaje que se definía por parte del programador como parte del listado 1.

A continuación, dado que existe un bloque `finally`, se ejecuta su código asociado mostrándose el mensaje `Se ha ejecutado setRadio()`. Este mensaje también se habría mostrado si la actualización del radio se hubiera realizado correctamente.

4. Crear excepciones

Las distintas subclases de `Exception` no cubren todos los posibles problemas que nos podemos encontrar en un programa, con lo que Java nos permite crear nuestras propias excepciones escribiendo clases derivadas o hijas de `Exception`. El listado 3 muestra el modo más simple de crear nuestra propia excepción, aunque también es posible dotarla de otros constructores y sobrescribir los métodos de la clase padre.

```
1  public class SetRadioException extends Exception {
2      public SetRadioException() {
3          super();
4      }
5
6      public SetRadioException(String cadena) {
7          super(cadena);
8      }
9  }
```

Listado 3: Creación de excepciones personalizadas Java

Ésta tiene definidos dos constructores, uno sin parámetros y otro con un parámetro de tipo `String` por lo que al heredar podemos decidir qué hacer.

Sobre esta base que se plantea aquí tenemos la posibilidad de ir etiquetando los errores particulares que van ocurriendo y hacer uso de estas etiquetas definidas por el programador, tanto en el `try catch` como en las cláusulas `throw` y `throws`.

En el listado 3 nos encontramos con la definición de la excepción `SetRadioException` que hace una referencia explícita a los errores asociados con el cambio del atributo radio dentro de una clase. En ella observamos como se extiende o hereda de la clase `Exception` con lo que los métodos de la misma, como `getMessage()` de los que haremos uso normalmente, ya están disponibles y, para un uso normal, no será necesario sobrescribirlos.

5. Miembros y métodos estáticos

Tal y como ya sabemos los miembros son los atributos o propiedades que definen a los objetos de una clase. En la definición de una clase se especifican como una lista de variables. Todos los objetos de una misma clase comparten estas definiciones de atributos, es decir, en cada objeto existe una copia propia del mismo conjunto de variables, por lo que los valores pueden ser diferentes en diferentes objetos. A estas variables miembro les llamamos **miembros de instancia** porque cada instancia (objeto) de la clase tiene los suyos propios.

En contraposición, existen los denominados **miembros de clase** o **miembros estáticos**. La definición de éstos también es común a todos los objetos de la clase a la que pertenecen, pero de ellos existe una única copia compartida por todos los objetos de esa clase. Para hacer que una variable miembro sea de clase basta con anteponer a su declaración la palabra reservada `static`.

La inicialización de los miembros de clase se hace en su declaración, dentro de la definición de la clase. Cuando el miembro estático es un objeto es posible usar `new` en su inicialización. De este modo, la variable tendrá un valor antes de que se cree el primer objeto de esa clase. Por otro lado, podemos utilizar las variables de clase en el código de cualquier método, pero debemos tener en cuenta que cualquier cambio que se haga afectará globalmente a todos los objetos de la clase por ser variables compartidas.

Además de miembros estáticos, también es posible declarar **métodos estáticos o de clase**. Para indicar que un método es estático basta con anteponer `static` a su declaración. Por definición, un método estático debe realizar funciones generales de la clase no dependientes de una instancia concreta de la misma u objeto. De hecho, los métodos estáticos tienen como limitaciones principales que, por un lado, no se puede acceder a los atributos de las clases, pues dichos atributos son propios de cada objeto y, por otro lado, no puede invocar a los métodos de las clases, pues dichos métodos son propios de cada objeto.

Los métodos estáticos se invocan utilizando el operador punto aplicado al nombre de la clase. En este sentido, si seguimos la convención de usar minúscula para la primera letra de los nombres de variables y métodos, y mayúscula para el nombre de las clases será sencillo identificar una invocación a un método estático.

El listado 4 muestra un ejemplo de uso de miembros y métodos estáticos. Continuamos con la definición de la clase `Circulo` de la sesión anterior (que heredaba de la clase abstracta `Figura`) y donde definíamos el miembro de instancia `radio`, del que cada instancia u objeto de la clase tendrá el suyo propio, y ahora definimos el miembro de clase o estático `numeroCirculos` que es compartido por todos los objetos de la clase `Circulo`. Este atributo servirá para contabilizar el número de círculos que se hayan creado en un determinado momento. A la variable estática `numeroCirculos` se le da un valor inicial de 0 y se incrementa en uno cada vez que se crea un objeto de la clase, es decir, en el código del constructor de la clase `Circulo`. Finalmente, para poder conocer el número de objetos de la clase `Circulo` que se han creado se ofrece el método estático `getNumeroCirculos()` que devuelve el valor de dicha variable estática.

```
1  public class Circulo extends Figura {
2      private double radio;
3      private static int numeroCirculos = 0;

5      public Circulo(String color, int posicionX, int posicionY, double radio) {
6          super(color, posicionX, posicionY);
7          this.radio = radio;
8          Circulo.numeroCirculos++;
9      }

11     public static int getNumeroCirculos() {
12         return Circulo.numeroCirculos;
13     }

15       // Resto de métodos de la clase ...
16 }
```

Listado 4: Ejemplo de definición de miembros y métodos estáticos

6. El método estático `main()`

Las clases definen los miembros y métodos que tendrán los objetos que se van a utilizar durante la ejecución de un programa. Además, en Programación Orientada a Objetos, el propio programa es una clase a la que llamamos **clase principal**. De este modo, el código de un programa Orientado a Objetos consiste en un conjunto de clases, de las cuales, una será la principal. La clase principal se encargará de crear y utilizar objetos del resto de clases para resolver el problema para el que se diseñó el programa.

En Java, la clase principal debe contener un método denominado `main` que no devuelva nada. Además, este método debe ser público, estático o de clase (concepto que hemos explicado en el apartado anterior) y tener como parámetro un array de objetos de la clase `String`. El Sistema Operativo (Windows, Linux, Mac, etc.) es el encargado de ejecutar el método `main` de la clase principal, pasándole los argumentos que sean necesarios. El método `main()` para Java tiene que ser necesariamente estático porque debe existir un método que se pueda ejecutarse en primer lugar sin que existan objetos creados con anterioridad.

El listado 5 muestra un ejemplo de clase principal en Java, como las que ya hemos utilizado de manera amplia en la asignatura, pero ahora entendiendo bien qué significa la definición de la clase principal y del método estático o de clase `main ()`.

```
1  class Main {
2     public static void main(String[] args) {
3        // Código del método main() de la clase Main...
4     }
5  }
```

Listado 5: Ejemplo de clase principal en un programa Orientado a Objetos en Java

7. Los paquetes de clases y su uso

Para conseguir una mayor claridad y organización del código cada clase debe escribirse en un fichero distinto cuyo nombre coincidirá con el de la clase que contiene. Además, Java permite agrupar clases relacionadas entre sí mediante lo que se denominan **paquetes de clases**. Los paquetes se pueden organizar de forma jerárquica al igual que las carpetas de nuestro ordenador.

El lenguaje Java incluye una amplia biblioteca de clases que podemos utilizar en nuestros programas. Dichas clases están agrupadas en paquetes por temas. Para poder utilizarlas en nuestros programas basta con **importarlas**. Para ello escribiremos al principio del fichero que contenga nuestra clase la palabra reservada `import` seguida del nombre del paquete y clase que queremos utilizar, y un punto y coma. Se pueden importar tantas clases como queramos y se puede usar el carácter `*` para indicar que queremos importar todas las clases de un mismo paquete. El listado 6 muestra distintas formas de importar clases.

```
1  import P.C;         // Esta línea importa la clase C del paquete P
2  import P1.P2.C2;    // Esta línea importa la clase C2 del paquete P2 que esta dentro del paquete P1
3  import P3.*;        // Esta línea importa todas las clases del paquete P3
```

Listado 6: Distintas formas de importar Clases en Java

8. Creando paquetes de clases

Además de usar otros paquetes de clases, también podemos crear los nuestros propios. Esta división del código en unidades facilita la reutilización. De este modo, cuando escribimos clases para representar objetos de un tema determinado podemos guardarlas todas juntas en un paquete y cada vez que necesitemos manejar objetos de ese tipo podremos utilizarlo sin volver a escribir dicho código.

Para crear un paquete, guardaremos todas sus clases en una carpeta que llamaremos con el nombre del paquete. Como primera línea de cada fichero de clase escribiremos la palabra reservada `package` seguida del nombre del paquete y un punto y coma. Finalmente, para que una clase de un paquete pueda

ser utilizada desde fuera de éste se debe indicar anteponiendo el modificador de visibilidad `public` a la definición de la clase.

Las clases de un mismo paquete pueden ser utilizadas directamente pero, como hemos indicado con anterioridad, para utilizar clases de paquetes distintos es necesario indicarlo expresamente con la palabra `import`.

Parte II
Prácticas

9. Ejercicios sobre excepciones, miembros y métodos estáticos y paquetes de clases

En esta sesión de prácticas, para la que debes haber leído y entendido en detalle la parte teórica de esta sesión, vamos a trabajar sobre los conceptos de excepciones, miembros y métodos estáticos y paquetes de clases, entre otros.

Ejercicio 1. *Busca el listado de subclases de la clase* `Exception` *en Java y comprueba aquellas subclases que pueden ser útiles para lo que llevamos estudiado en la asignatura.*

Ejercicio 2. *Crea un nuevo proyecto para esta sesión 7 y copia todas las clases del proyecto de la sesión 6. Modifica las clases* `Hora` *y* `Fecha` *de manera que cada vez que se haga un método* `set()` *se pueda lanzar una excepción en caso de que el valor introducido no sea correcto como ocurre, a modo de ejemplo, cuando se indica que el día de un objeto de la clase* `Fecha` *tiene valor* 32.

Ejercicio 3. *Crea en el método estático* `main()` *dos objetos de la clase* `Hora` *y dos objetos de la clase* `Fecha` *de manera que cada uno de ellos devuelva una excepción distinta.*

Ejercicio 4. *Crea una clase para las excepciones asociadas a la clase* `Hora` *y otra para las excepciones asociadas a la clase* `Fecha`*. Modifica los dos apartados anteriores para que se lancen estas excepciones como objetos de las dos clases hijas de* `Exception` *que acabas de crear.*

Ejercicio 5. *Añade a la clase* `Usuario` *un miembro estático que permita contar cuántos objetos se han creado hasta el momento de esta clase. De igual manera, define un método estático que devuelva el valor de este contador.*

Ejercicio 6. *En el método estático* `main()` *que tienes de la sesión anterior crea tres usuarios distintos y muestra por pantalla el valor del atributo estático contador tanto antes de crearlos como después.*

Ejercicio 7. *Averigua mediante qué método estático se pueden generar valores aleatorios en Java. Adapta la invocación a dicho método para que utilizándolo las veces que sea necesario seamos capaces de generar el valor del atributo* `numeroDeTelefonoMovil` *de cada uno de estos tres objetos de la clase* `Usuario`*. Para resolver este ejercicio se recomienda el uso de otros métodos estáticos que permitan operar sobre valores decimales como los de redondeo, por ejemplo.*

Ejercicio 8. *Para incluir en el ejercicio anterior la invocación al método estático que permite generar los valores aleatorios, ¿es necesario usar la palabra reservada* `import`*? ¿Por qué?*