Curso de Java orientado a Android – Parte 3

Curso de Java orientado a Android

Introducción

Después de cubrir los conceptos básicos de los principios de la programación orientada a objetos en el post anterior, ahora atenderemos a otros conceptos que nos encontraremos a la hora de programar en Java. Exactamente como tratar la gestión de errores y colecciones de información.

- Exceptions
- Java Collections
    - Interfaces
    - Implementations
        - ArrayList
        - HashSet
        - HashMap

Exceptions

La gestión de errores es una parte esencial para asegurar un código robusto. Java utiliza excepciones para manejar errores. Cuando se produce un error, el entorno de ejecución de Java maneja un objeto de error que se crea por el método en el que se produce el error. Este objeto se llama excepción, y contiene información básica sobre el error (como el tipo de error, la ubicación, la pila de los métodos que conducen al error … etc.).
La lista de los métodos que conducen al error se le llama call stack (pila de llamadas). Al manejar el error, el sistema busca a través de esa pila un controlador de errores en el código; es decir, un controlador de excepciones. Todos los objetos de excepción son los hijos de la clase padre Exception.

Las Excepciones en Java se pueden clasificar en dos tipos:

Categoría	**Descripción**
Excepción controlada	Estos son errores dentro de código de la aplicación. Un programador que tiene la intención de crear un código bien escrito y robusto debe poder recuperar el programa de estos errores. Por ejemplo, leyendo un archivo en disco, un programador debe tener en cuenta que el fichero quizás no exista. En este caso, el programador debe esperar una excepción del tipo java.io.FileNotFoundException. Debera, enconces, capturar para notificar al usuario de lo que ha ocurrido de una forma correcta y controlada sin detener la ejecución del programa.

Excepción no controlada	Estos vienen en dos tipos: los propios errores y tiempo de ejecuciónexcepciones. Se agrupan en una categoría porque ambosno es posible anticipar o recuperarse de un programador.Los errores son externos a las aplicaciones. Por ejemplo, supongamosque una aplicación abre correctamente un archivo para la entrada, pero es no puede leer el archivo debido a un hardware o del sistemaFuncionamiento defectuoso. La lectura infructuosa arrojarájava.io.IOError, y tiene sentido para que el programaimprimir un seguimiento de pila y salir. Los errores son esas excepciones de tipoClase de fallo y sus subclases.Excepciones de tiempo de ejecución generalmente indican errores de programación,como errores lógicos. Son o una clase de tipo RuntimeException ysus subclases.

La gestión de errores (excepciones) en Java se realiza a través de los bloques try-catch-finally . Mientras que la bloque finally es opcional, los bloques try y catch son obligatorios para cumplir con el tratamiento de errores.
Veamos el siguiente código:

```
 public class ExceptionTest {
        public static void main(String[] args) {
                        System.out.println("Hello World
adictosalainformatica!");
               String nullString = null;
               System.out.println("Entered try statement");
                                String partialString =
nullString.substring(1);
                // Execution will break before reaching this
line
                 System.out.println("Partial string is: " +
partialString);
        }
 }
```

El resultado de ejecutar el código será un error del tipo NullPointerException, en concreto en la línea 6, donde estamos tratando de leer de un objeto de cadena que es nulo (no inicializado) Para manejar adecuadamente este error, debemos modificar el código anterior de la siguiente manera:

```
public class ExceptionTest {
        public static void main(String[] args) {
                        System.out.println("Hello World
adictosalainformatica!");
                String nullString = null;
                try {
                          System.out.println("Entered try
statement");
                                  String partialString =
nullString.substring(1);
                          // Execution will continue in the
exception block
                        System.out.println("Partial string is:
"+partialString);
                } catch (Exception e) {
                          System.out.println("Error occured:
"+e.getMessage());
                        e.printStackTrace();
                }
        }
}
```

En vez de romper la ejecución de código y detener el programa, este código se encargará de manejar la excepción NullPointerException mediante la impresión de los detalles del error y continuando la ejecución más allá de la bloque catch. El bloque finally se puede utilizar después del bloque de excepción. Este bloque de código se ejecutará siempre, tanto si se lanza una excepción como no.

```
try {
        System.out.println("Entered try statement");
        String partialString = nullString.substring(1);
        // Execution will break before reaching this line
           System.out.println("Partial string is: " +
partialString);
} catch (Exception e) {
        System.out.println("Error occured: "+e.getMessage());
        e.printStackTrace();
} finally {
         System.out.println("This line of code will always
```

```
run!");
}
```

En muchos casos utilizamos el bloque finally cuando hay algunos recursos que necesitan ser liberados. Es posible que después de una excepción no lo esto no ocurra porque el código que debería hacerlo no es ejecutado. Por ejemplo, un programa que lee un fichero, este debe cerrar el archivo después de terminar la lectura y/o escritura. Si una excepción es lanzada, la línea de código que cierra el archivo puede ser omitida. El bloque finally sería el mejor lugar donde cerrar el fichero.

Java Collections

Java proporciona un conjunto de clases e interfaces para ayudar a los desarrolladores manejar colecciones de objetos. Esta colección de clases són similiares a un array, excepto por su tamaño (puede crecer de forma dinámica durante tiempo de ejecución). En esta sección se ofrecerá una visión general de algunas de las clases de Java collection más populares.

Interfaces

Las Java Collections se encuentran principalmente en el paquete java.util. Proporciona dos interfaces principales: Collection y Map. Estas dos forman el núcleo del Java Collection framework. Otras interfaces heredan de estas dos. Por ejemplo, las interfaces List y Set heredan de la interfaz Collection. Todas estas interfaces son genéricas; es decir, el tipo de objeto contenido en la colección debe ser especificado por el programador. Hay una diferencia fundamental entre las subclases de la interfaz Collection y de la interfaz Map:

- Collection contiene un grupo de objetos que pueden ser manipulados y traspasados. Los elementos pueden ser

duplicados o únicos, dependiendo del tipo de sub-clase. Por ejemplo, Set sólo contiene objetos únicos.

- La interfaz Map, sin embargo, mapea los valores a los ids (keys) y no puede contener keys duplicadas y a cada una sólo se le puede asignar un valor

Implementations

Las implementaciones son los objetos de datos que se utilizan para almacenar colecciones, que implementan las interfaces de la sección anterior. A continuación se explican las siguientes implementaciones:

ArrayList
Un ArrayList es una implementación de array redimensionable de la interfaz List. Implementa todas las operaciones opcionales de lista y permite todo tipo de elementos, incluyendo null. También proporciona métodos para manipular el tamaño del array que se utiliza internamente para almacenar la lista.

```
import java.util.*;

class TestArrayList {
        public static void main(String args[]) {
                // Creating an array list
                ArrayList androids = new ArrayList();
                // Adding elements
                androids.add("Cupcake");
                androids.add("Donut");
                androids.add("Eclair");
                androids.add("Froyo");
                androids.add("Gingerbread");
                androids.add("Honeycomb");
                androids.add("Ice Cream Sandwich");
                androids.add("Jelly Bean");
                  System.out.println("Size of ArrayList: " +
androids.size());
                // Display the contents of the array list
                  System.out.println("The ArrayList has the
following elements: "
```

```
                + androids);
                // Remove elements from the array list
                    System.out.println("Deleting second
element...");
                androids.remove(3);
                 System.out.println("Size after deletions: " +
androids.size());
                 System.out.println("Contents after deletions:
" + androids);
        }
}
```

El resultado será:

```
Size of ArrayList: 8
The ArrayList has the following elements: [Cupcake, Donut,
Eclair, Froyo,
Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean]
Deleting second element...
Size after deletions: 7
Contents after deletions: [Cupcake, Donut, Eclair,
Gingerbread, Honeycomb,
Ice Cream Sandwich, Jelly Bean]
```

HashSet

Esta clase implementa la interfaz Set y permite el elementos null. Esta colección no permite duplicados. Crea una colección que utiliza una tabla hash para el almacenamiento. Una tabla almacena información de hash mediante el uso de un mecanismo llamado hashing donde se utiliza el valor almacenado para determinar una clave única, que se utiliza como el índice en el que los datos se almacenan. La ventaja del hashing es que permite tiempos de ejecución rápidos para operaciones básicas, como add() y remove().

```
class TestHashSet {
        public static void main(String args[]) {
                // Creating a HashSet
                HashSet androids = new HashSet();
                // Adding elements
```

```
                androids.add("Cupcake");
                androids.add("Cupcake");
                androids.add("Eclair");
                androids.add("Eclair");
                androids.add("Gingerbread");
                androids.add("Honeycomb");
                androids.add("Ice Cream Sandwich");
                androids.add("Jelly Bean");
                  System.out.println("The contents of the
HashSet: " + androids);
        }
}
```

El resultado será:

The contents of the HashSet: [Eclair, Cupcake, Honeycomb, Ice Cream Sandwich, Jelly Bean, Gingerbread]

Como podemos comprobar solo hay un elemento “Cupcake” y uno “Éclair” aunque en el código los hayamos añadido dos veces.

HashMap

Esta es una hashtable basada en la implementación de la interface Map. Los elementos introducidos no tendrán un orden específico y permite elementos null.

El siguiente programa muestra un ejemplo de HashMap. Se asignan nombres para contabilizar saldos.

```
import java.util.*;
class TestHashMap {
        public static void main(String args[]) {
                // Creating a HashMap
                   HashMap<String,Double> androids = new
HashMap<String,Double>();
                // Adding elements
```

```
                androids.put("Cupcake", new Double(1.5) );
                androids.put("Donut",new Double(1.6));
                androids.put("Eclair", new Double(2.1));
                androids.put("Froyo", new Double(2.2));
                androids.put("Gingerbread", new Double(2.3));
                androids.put("Honeycomb", new Double(3.1));
                   androids.put("Ice Cream Sandwich", new
Double(4.0));
                androids.put("Jelly Bean", new Double(4.1));
                // Get a set of the entries
                   Set<Map.Entry<String, Double>> set =
androids.entrySet();
                // Get an iterator
                  Iterator<Map.Entry<String, Double>> i =
set.iterator();
                // Display elements
                while (i.hasNext()) {
                          Map.Entry<String, Double> me =
(Map.Entry<String,Double>)
                       i.next();
                       System.out.print(me.getKey() + ": ");
                       System.out.println(me.getValue());
                }
                System.out.println();
                // Increase version number of Eclair
                Double version = androids.get("Eclair");
                 androids.put("Eclair", new Double(version +
0.1));
                  System.out.println("New version number of
Eclair: "
                + androids.get("Eclair"));
        }
}
```

El resultado será:

```
Eclair: 2.1
Cupcake: 1.5
Honeycomb: 3.1
Froyo: 2.2
Donut: 1.6
Ice Cream Sandwich: 4.0
```

Jelly Bean: 4.1
Gingerbread: 2.3
New version number of Eclair: 2.2

Observaciones

En este tercer post hemos mostrado el manejo de excepciones y la utilización de Collections. Se ha mostrado una parte específica y concreta, sería recomendable profundizar un poco más en estos conceptos dado que, aquí solo se muestran conceptos básicos.

- Oracle Exception tutorial
- Oracle Collections tutorial