Curso de Java orientado a Android – Parte 4

Curso de Java orientado a Android

Introducción

En este último post del curso de Java orientado a Android trabajaremos un poco mas a fondo la programación orientada a objetos y algunas particularidades propias de Java. Este va ser el último post, de esta serie.

- Nested classes
- Beneficios de las inner classes
- Variables de clase (static)
- Funciones (static)
- Enumerated types
- Serialization
- Deserializing

Nested classes

En Java se puede definir una clase dentro de otra clase. Estas

se llaman «nested class»:

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
```

Las nested class pueden ser static:

```
class OuterClass {
    ...
    static class StaticInnerClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

Una nested class es miembra de su outer class. Las nested clases no estáticas tienen acceso a otros miembros de la clase externa, incluso si se declaran como private. Sin embargo, las nested static clases no. De manera similar a las variables y métodos miembros, una clase interna pueden ser declarados privado, público, protegido, o un paquete privado.

Beneficios de utilizar inner class

Las siguientes son algunas de las razones que tientan a un programador para usar clases internas:

- **Mejorar la agrupación lógica de clases** que se utilizan únicamente en un solo lugar. Si una clase B es útil solamente a otra clase A, entonces es lógico que la clase B sea una clase interna de la clase A.
- **Aumentar la encapsulación**. Si la clase B necesita tener acceso a los miembros privados de la clase A, un

programador puede ocultar la clase B dentro A y mantener todos los miembros de la clase A como privados al mismo tiempo que oculta la clase B del resto del entorno.

- **Mejorar la legibilidad del código y facilita el mantenimiento**. La creación de las clases internas dentro de una clase externa proporciona una organización más clara de código.

Variables de clase (static)

Cuando creamos varios objetos de la misma clase, cada objeto (instancia) tiene su propia copia de las variables miembro. A veces, puede ser que deseemos compartir una variable con todos los objetos de la misma clase. Para lograr esto usamos modificador static.

Las variables miembro que tienen el modificador static en su declaración se llaman campos estáticos o variables de clase staticas. Están asociadas con la clase, en lugar de con cualquier objeto. Cada instancia de la clase comparte una variable de clase, que se guarda en una memoria fija. Cualquier objeto puede cambiar el valor de una variable de clase.

Vamos a modificar la clase de coches de la parte 2 de esta serie de posts añadiendo una variable de clase static. La variable numOfSeats puede tener valores diferentes para los distintos objetos del tipo de coche. Sin embargo, podemos añadir una variable de clase llamada numberOfCars que se utilizará para realizar un seguimiento del número de objetos de coches creados.

```
  public class Car extends Vehicle {
    public int numOfSeats;
    //A class variable for the
    //number of Car objects created
    public static int numberOfCars;
```

```
  }
```

Las variables de clase son referenciadas por el propio nombre de la clase:

```
Car.numberOfCars;
```

Funciones de clase (static)

Java también soporta métodos estáticos, estos tienen el modificador static en su firma. Un uso común de los métodos estáticos es tener acceso a los campos estáticos. Por ejemplo, vamos a modificar la clase de coche mediante la adición de un método estático que devuelve la variable statica numOfCars:

```
  public static int getNumberOfCars() {
    return numberOfCars;
  }
```

Enumerated types

Un enumerated type (también llamado enumeration o enum) es un tipo de datos que consiste en un conjunto de constantes llamadas elementos. Un ejemplo común de enumeración es los días de la semana. Dado que son constantes, los nombres de los campos de un enum están en letras mayúsculas.
Para definir un tipo de enum en Java, utilizamos la palabra clave enum. Por ejemplo, el siguiente tipo de enumeración define un conjunto de enumeraciones para las versiones de Android:

```
public enum androidVersionCodes {
    CUPCAKE, DONUT,ECLAIR,FROYO,
    GINGERBREAD,HONEYCOMB,ICE CREAM SANDWICH,
    JELLY BEAN,KITKAT,LOLLIPOP,MARSHMALLOW
}
```

Los enum se deben utilizar siempre que se necesite representar un conjunto fijo de constantes.

Serialization

La serialización es el proceso de convertir un objeto en un formato que puede ser almacenado y luego convertido de nuevo más tarde a un objeto en el mismo o en otro entorno informático.

Java proporciona serialización automática, para tulizarla el objeto debe implementar la interfaz java.io.Serializable . Java entonces maneja serialización internamente.

La siguiente es una clase Java que almazena una versión de android (nombre y codigo de versión). Es serializable, y tiene dos variables miembro: androidVersionName y androidVersionCode.

```
import java.io.Serializable;

public class AndroidVersions implements Serializable {
    private Double androidVersionCode;
    private String androidVersionName;

    public Double getAndroidVersionCode () {
        return androidVersionCode;
    }

          public void setAndroidVersionCode(Double
androidVersionCode) {
        this.androidVersionCode = androidVersionCode;
    }

    public String getAndroidVersionNmae() {
        return getAndroidVersionNmae;
    }

          public void setAndroidVersionName(String
getAndroidVersionNmae) {
        this.getAndroidVersionNmae = getAndroidVersionNmae;
    }
```

```
}
```

Ahora que tenemos un objeto serializable, podemos hacer una prueba del proceso de serialización escribiendo un objeto en un fichero. El siguiente código escribe un objeto AndroidVersions en un fichero llamado android.ser:

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

public class SerializeAndroidVersions {

    public static void main(String[] args) {

        AndroidVersions andVersions = new AndroidVersions();
        andVersions.setName("Jelly Bean");
        andVersions.setAndroidVersionCode(4.1);

        try
        {
                        FileOutputStream fileOut = new
FileOutputStream("/home/user_name/android.ser");
                         ObjectOutputStream out = new
ObjectOutputStream(fileOut);
            out.writeObject(e);
            System.out.println("Serialized...");
            out.close();
            fileOut.close();
        }
        catch (IOException i) {
            i.printStackTrace();
        }
    }
}
```

Como resultado tendremosun fichero android.ser en nuestra carpeta de usuario.

Deserializing

Ahora podemos crear un objeto a partir del fichero guardado en disco. Lo único que debemos hace es acceder al fichero y convertirlo en un objeto.

El siguiente ejemplo muestra como realizar este proceso:

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.ArrayList;

public class DeserializeAndroidVersions {
  @SuppressWarnings(
    "unchecked"
  )

  public static void main(String[] args) {
    AndroidVersions androidVersions = new AndroidVersions();
    try{
                    FileInputStream   fileIn   =   new
FileInputStream("/home/_user_name/android.ser");
      ObjectInputStream in = new ObjectInputStream(fileIn);
      androidVersions = (AndroidVersions) in.readObject();
      in.close();
      fileIn.close();
    }catch (IOException i) {
      return;
    }catch ClassNotFoundException c) {
      System.out.println("AndroidVersions class not found.");
      c.printStackTrace();
      return;
    }

    if(androidVersion instanceof AndroidVersions) {
      System.out.println("-----------------------");
         System.out.println("Deserialized AndroidVersions
object...");
                     System.out.println("Name:   "   +
androidVersions.getAndroidVersionNmae());
             System.out.println("Code  version:  "  +
androidVersions.getAndroidVersionCode());
```

```
      System.out.println("-----------------------");
    }
  }
}
```

Ejecutando el código anterior obtendremos el siguiente resultado:

```
-----------------------
Deserialized AndroidVersions object...
Name: Jelly Bean
Code version: 4.1
-----------------------
```

Llegados a este punto ahora entendemos la lógica de la serialización. Esta bien a nivel conceptual, de aprendizaje, pese a ello este procedimiento no es el usual en Android. Podemos utilizar la serialización, pero en vez de ello utilizaremos Parcelable. En este artículo se explica el porqué debemos utilizar parcelabes.

Observaciones

Con este post damos por terminada la introducción a la programación en Java para Android. La idea era realizar un curso de Android una vez finalizada esta serie, pero no va a ser así. Des de que empezó la serie hasta ahora viendo como se esta desarrollando el entorno Android he optado por no hacerlo. Me centraré ejemplos concretos y en librerías. Exsisten muschas y ayudan muchissímo en el dia a dia de la programación en Android. Tareas que a priori pueden parecer muy complicadas o introducir excesivo código nos las facilitan. No por eso voy a hacer el salto sin más, partiendo de lo aprendido anteriormente, a continuación dejo tres muy buenos recursos para aprender Android:

- Android Official Training (Inglés)
- Vogella tutorials (Inglés)

- Sgoliver curso Android (Castellano)