FICHEROS

Acceso a Datos

10 DE NOVIEMBRE DE 2024

ERIK AMO TOQUERO

Contenido

Introducción

Java permite acceder a ficheros desde la herramienta JavaIO. Hace falta rodear todo el manejo de ficheros de la excepción IOException.

Obtención de Ficheros

Los ficheros deberían guardarse normalmente en la carpeta de recursos del proyecto (resources), y acceder a ellos desde el método Class,getResource o Class.getResourceAsStream.

Para hacerlo más sencillo, se puede crear un método estático que devuelva cualquier dirección a ficheros. Estos se guardan en una clase llamada “Clase R”:

```
public class R {
    public static InputStream getProperties(String filename){
        return
Thread.currentThread().getContextClassLoader().getResourceAsStream("co
nfig"+ File.separator+filename);
    }
    public static URL getResource(String filename){
        return
Thread.currentThread().getContextClassLoader().getResource("ui/"+filen
ame);
    }
}
```

Manejos de Ficheros

Según el tipo de archivo que se vaya a abrir y el uso que se le vaya a dar, hay distintos usos del manejo de ficheros:

Ficheros normales (File)

Se utiliza la clase File para guardar su contenido en un objeto de Java:

```
File file = new
File(String.valueOf(this.getClass().getResource("Hola.txt")));
```

Una vez con esto, se pueden hacer distintas operaciones:

Escribir:

```
BufferedWriter bw = new BufferedWriter(new FileWriter(file));
bw.write("Hola");
```

Leer:

```
BufferedReader br = new BufferedReader(new FileReader(file));
String st = br.readLine();
//Ejemplo de lectura de todo el fichero
Scanner sc = new Scanner(file);
while (sc.hasNext()){
    System.out.println(sc.nextLine());
}
```

Ficheros de configuración (Properties)

Los archivos Properties guardan información relacionada al programa y preparada por si se quieren editar distintos datos. Utilizan su propio tipo de objeto

```
Properties dbConf = new Properties(); //Crea un objeto de tipo
Properties
dbConf.load(R.getProperties("database.properties"));
```

Una vez con esto, se puede hacer lo siguiente:

Guardar propiedades: Permite guardar apartados de propiedades en alguna variable para su futuro uso

```
String host = dbConf.getProperty("host");
```

Cambiar propiedades: Si puede cambiar una propiedad, existe el comando setProperty

```
dbConf.setProperty("host", host);
```

Listar propiedades: El archivo de configuración no deja de ser un mapa de clave-valor, por lo que se puede usar como un mapa:

```
for (Object key: dbConf.keySet()){
    [...];
}
```

Ficheros de vistas FXML

Son los ficheros que guardan información acerca de las vistas de JavaFX. Su uso es muy sencillo, ya que es simplemente cargarlo en un Loader:

```
FXMLLoader fxmlLoader = new
FXMLLoader(R.getResource("MainView.fxml"));
Scene scene = new Scene(fxmlLoader.load());
```

Ficheros JSON

La lectura de ficheros JSON requiere una librería aparte para mapear el fichero a Map<clave, valor>

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.15.3</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.1</version>
</dependency>
```

De aquí se pueden mapear objetos de JSON a Java y de Java a JSON con un Object_Mapper

```
public static final ObjectMapper JSON_MAPPER = new ObjectMapper();
ArrayList<Persona> personas = //Creación del arrayList de tipo Persona
        JSON_MAPPER.readValue(new
File("src/main/resources/Persona.json"), //Lee el archivo
```

```
        JSON_MAPPER.getTypeFactory().constructCollectionType
//constructCollectionType(Lista, Objeto), coge todos los valores del
json
                (ArrayList.class, Persona.class)); //Mete los objetos
en una lista y estos objetos son de tipo Persona
JSON_MAPPER.writeValue(new File("src/main/resources/Persona.json"),
new Persona()); //Guardo una nueva persona en el JSON
```

Fichero de Configuración de Hibernate (hibernate.conf.xml)

Quizá el más sencillo. Todo lo que hay que hacer con él es cargarlo y pasar las clases anotadas que vaya a usar. Utiliza el objeto Configuration de la librería Hibernate:

```
<!--Dependencia de Hibernate-->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.6.9.Final</version>
</dependency>
```

Con esto, solo habrá que configurar el archivo en una clase Java:

```
Configuration configuration = new Configuration();
configuration.configure(R.getHibernateConfig("hibernate.cfg.xml"));
configuration.addAnnotatedClass(Multa.class);
configuration.addAnnotatedClass(Coche.class);

factory = configuration.buildSessionFactory();
```

EXTRA: CIFRADO

El cifrado es un método de seguridad muy importante a la hora de proteger contraseñas. El que hemos usado en clase es el cifrado SHA256hex, que cifra cada carácter por separado y devuelve una cadena de lo que parece ser 256 caracteres aleatorios.

Utiliza la clase DigestUtils, de la librería Codec:

```
<!--Dependencia del Cifrado SHA256-->
<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.16.0</version>
</dependency>
```

Para cifrar un String, es tan sencillo como hacer lo siguiente:

```
String codigoSecreto = DigestUtils.sha256Hex(codigo);
```