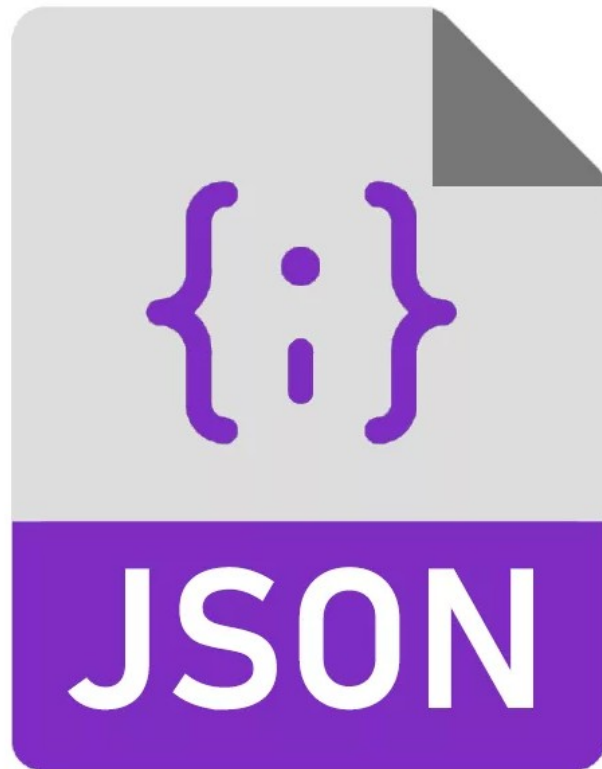


Acceso a Datos

Práctica Serialización



Índice

1. Introducción.....	3
2. Realización de la práctica.....	6
2.1 Parte 1. Crear clase Size, Cactus y Bird.....	6
2.2 Parte 2. Creando el método Save.....	13
2.3 Parte 3. Adaptadores.....	26
2.4 Parte 4. Creando el método load.....	29
2.5 Parte 5. Clase Juego con lista de niveles con método guardar y cargar.....	33

1. Introducción

Todo el código de este proyecto así como este mismo PDF se encuentra en el siguiente repositorio de GitHub:

https://github.com/Nathan-GM/Practica_Serializacion_Nathan

En esta práctica se nos ha encargado crear una aplicación para el juego de T-Rex de Google. Nuestro equipo tiene como objetivo definir la estructura de las clases del juego y serializar y deserializar los mismos.

Se ha de poder serializar y deserializar en:

- JSON. Se recomienda la librería Gson.
- XML. Se recomienda la librería JAXB
- Binario

El juego posee una lista de niveles, cada nivel posee:

- Tiempo de juego
- Velocidad
- Lista de cactus
- Lista de pájaros
- Música

A su vez, un cactus tiene:

- Coordenada (Point2D)
- Tamaño
- Nombre

Y un pájaro:

- Tamaño
- Nombre
- Velocidad
- Coordenada (Point2D)

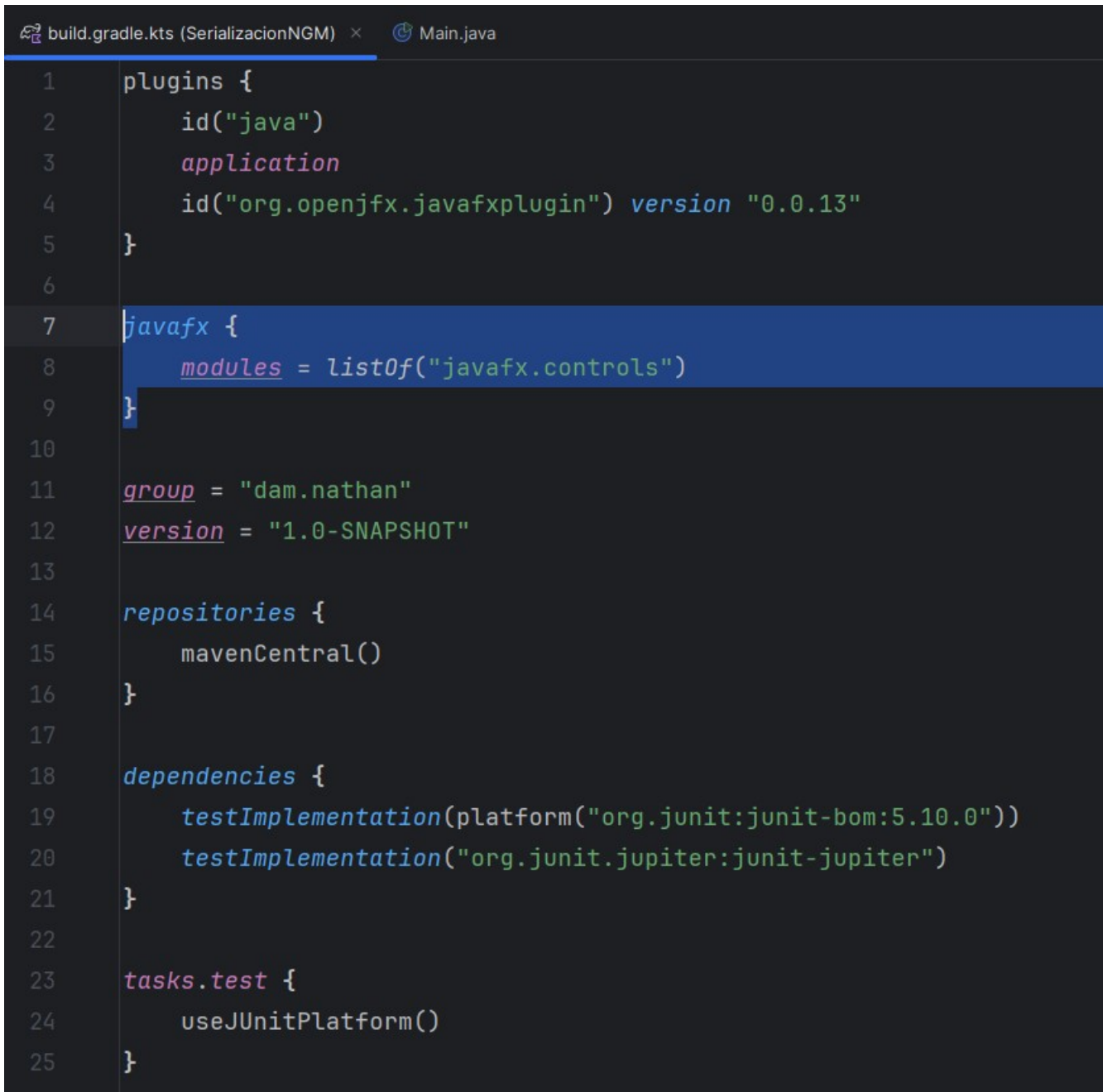
Crear un proyecto con Gradle y añadir las librerías de JavaFX.

Para ello en el fichero `build.gradle.kts` deberemos agregar lo siguiente en `plugins`:

```
application  
id("org.openjfx.javafxplugin") version "0.0.13"
```

Tras ello, deberemos crear el apartado de `javafx` indicando que tendrá como módulo una lista de los controles de `javafx`. Para ello, en el mismo fichero agregamos lo siguiente:

```
javafx {  
    modules = listOf("javafx.controls")  
}
```



```
1  plugins {
2      id("java")
3      application
4      id("org.openjfx.javafxplugin") version "0.0.13"
5  }
6
7  javafx {
8      modules = listOf("javafx.controls")
9  }
10
11  group = "dam.nathan"
12  version = "1.0-SNAPSHOT"
13
14  repositories {
15      mavenCentral()
16  }
17
18  dependencies {
19      testImplementation(platform("org.junit:junit-bom:5.10.0"))
20      testImplementation("org.junit.jupiter:junit-jupiter")
21  }
22
23  tasks.test {
24      useJUnitPlatform()
25  }
```

El fichero nos quedará algo similar a lo mostrado.

2. Realización de la práctica

2.1 Parte 1. Crear clase Size, Cactus y Bird

Definir la clase Size, con dos atributos:

- Width
- Heigh

```
public class Size {  
    private double Width;  
    private double Heigh;  
  
    public Size(){  
  
    }  
  
    public Size(double Width, double Height) {  
        this.Width = Width;  
        this.Heigh = Height;  
    }  
  
    public double getWidth() {  
        return Width;  
    }  
  
    public double getHeigh() {  
        return Heigh;  
    }  
  
    public void setWidth(double width) {  
        Width = width;  
    }  
  
    public void setHeigh(double heigh) {  
        Heigh = heigh;  
    }  
}
```

Creamos la clase con sus respectivos getters y setters teniendo como resultado lo que el código mostrado

```
package dam.nathan;

public class Size { no usages
    private double Width; 3 usages
    private double Heigh; 3 usages

    public Size(){ no usages
    }

    public Size(double Width, double Height) { no usages
        this.Width = Width;
        this.Heigh = Height;
    }

    public double getWidth() { return Width; }

    public double getHeigh() { return Heigh; }

    public void setWidth(double width) { Width = width; }

    public void setHeigh(double heigh) { Heigh = heigh; }
}
```

Definir la clase cactus y pájaro. No se usa herencia. Para las coordenadas usar la clase Point2D de JavaFX.

Clase Cactus (Imagen):

```
package dam.nathan;

import javafx.geometry.Point2D;

public class Cactus { no usages
    private String name; 3 usages
    private Size size; 3 usages
    private Point2D coordinate; 3 usages

    public Cactus() {}

    public Cactus(String name, Size size, Point2D coordinate) { no usages
        this.name = name;
        this.size = size;
        this.coordinate = coordinate;
    }

    public String getName() { return name; }

    public Size getSize() { return size; }

    public Point2D getCoordinate() { return coordinate; }

    public void setName(String name) { this.name = name; }

    public void setSize(Size size) { this.size = size; }

    public void setCoordinate(Point2D coordinate) { this.coordinate = coordinate; }
}
```


Clase cactus

```
package dam.nathan;

import javafx.geometry.Point2D;

public class Cactus {
    private String name;
    private Size size;
    private Point2D coordinate;

    public Cactus() {

    }

    public Cactus(String name, Size size, Point2D coordinate) {
        this.name = name;
        this.size = size;
        this.coordinate = coordinate;
    }

    public String getName() {
        return name;
    }

    public Size getSize() {
        return size;
    }

    public Point2D getCoordinate() {
        return coordinate;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setSize(Size size) {
        this.size = size;
    }

    public void setCoordinate(Point2D coordinate) {
        this.coordinate = coordinate;
    }
}
```

Clase Pájaro (Bird) (Imagen)

```
package dam.nathan;

import javafx.geometry.Point2D;

public class Bird { no usages
    private String name; 3 usages
    private Float speed; 3 usages
    private Size size; 3 usages
    private Point2D coordinate; 3 usages

    public Bird() {}

    public Bird(String name, Float speed, Size size, Point2D coordinate) { no usages
        this.name = name;
        this.speed = speed;
        this.size = size;
        this.coordinate = coordinate;
    }

    public String getName() { return name; }

    public Float getSpeed() { return speed; }

    public Size getSize() { return size; }

    public Point2D getCoordinate() { return coordinate; }

    public void setName(String name) { this.name = name; }

    public void setSpeed(Float speed) { this.speed = speed; }

    public void setSize(Size size) { this.size = size; }

    public void setCoordinate(Point2D coordinate) { this.coordinate = coordinate; }
}
```

Clase Bird

```
package dam.nathan;

import javafx.geometry.Point2D;

public class Bird {
    private String name;
    private Float speed;
    private Size size;
    private Point2D coordinate;

    public Bird() {
    }

    public Bird(String name, Float speed, Size size, Point2D coordinate) {
        this.name = name;
        this.speed = speed;
        this.size = size;
        this.coordinate = coordinate;
    }

    public String getName() {
        return name;
    }

    public Float getSpeed() {
        return speed;
    }

    public Size getSize() {
        return size;
    }

    public Point2D getCoordinate() {
        return coordinate;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setSpeed(Float speed) {
        this.speed = speed;
    }

    public void setSize(Size size) {
        this.size = size;
    }

    public void setCoordinate(Point2D coordinate) {
```

```
this.coordinate = coordinate;  
}  
}
```

2.2 Parte 2. Creando el método Save

Crear un método estático en el cactus y en pájaro para guardar, se le pasa el nombre del fichero a guardar, un cactus o pájaro y la extensión, la cuál debe ser XML, JSON o Bin, en caso contrario saltará una excepción.

Para ello, en ambas clases crearemos el método estático save, donde recibiremos un pájaro o un cactus según la clase, un nombre que será donde se encuentre nuestro fichero y la extensión del mismo. A continuación tenemos el método save de la clase Bird.

```
public static void save(Bird b, String name, String ext) throws excExtension {  
    String extension = ext.toLowerCase();  
    switch (extension) {  
        case "xml":  
            saveXML(b, name);  
            break;  
        case "json":  
            saveJSON(b,name);  
            break;  
        case "bin":  
            saveBIN(b,name);  
            break;  
        default:  
            throw new excExtension(extension);  
    }  
}
```

Y esta corresponde a la clase Cactus:

```
public static void save(Cactus c, String name, String ext) throws excExtension {
    String extension = ext.toLowerCase();
    switch (extension) {
        case "xml":
            saveXML(c, name);
            break;
        case "json":
            saveJSON(c,name);
            break;
        case "bin":
            saveBIN(c,name);
            break;
        default:
            throw new excExtension(extension);
    }
}
```

Además, crearemos una excepción que controle el programa si la extensión recibida no es compatible con lo especificado. Esta se encuentra en otra clase y contiene lo siguiente:

```
package dam.nathan;

public class excExtension extends Exception{
    private String extension;

    excExtension(String extension) {
        this.extension = extension;
    }

    @Override
    public String toString() {
        return "La extensión " + extension + " no esta permitida";
    }
}
```

Donde se le indica que el parámetro extensión será el causante de dicha extensión y lo que deberá mostrar en caso de la misma.

Una vez hecho eso, deberemos crear los métodos para guardar en cada tipo de formato, comenzando por JSON. Este nuevo método deberá recibir como parámetro el tipo de objeto a guardar, bien sea cactus o pájaro, y el nombre que corresponde a la ruta del fichero.

Para poder transformar nuestro objeto en JSON utilizamos la librería GSON. En el caso de la transformación del objeto cactus, nos queda un método de la siguiente forma:

```
private static void saveJSON(Cactus c, String name) {
    File fichero = new File(name + ".json");

    Gson gson = new Gson();

    String json = gson.toJson(c);

    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter(fichero));
        bw.write(json);
        bw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Donde vemos que primero creamos un fichero usando el nombre recibido e indicándole que extensión tendrá, en este caso json y luego creamos un objeto de tipo Gson, el cuál nos permite pasar el objeto a JSON, el cuál almacenamos en una string para más tarde almacenarla en un fichero, cosa que se ve en el try.

En el caso de la clase Bird el método podemos ver que es el mismo cambiando el tipo de objeto recibido.

```
private static void saveJSON(Bird b, String name) {
    File fichero = new File(name + ".json");

    Gson gson = new Gson();

    String json = gson.toJson(b);

    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter(fichero));
        bw.write(json);
        bw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

No obstante, al hacerlo de esta forma es posible que la clase Point2D no se guarde correctamente. Para evitar problemas deberemos crear un adaptador de dicha clase. Para dicho adaptador crearemos una clase que extenderá de TypeAdapter para obtener los métodos de Write y Read que usará Gson para escribir y leer. Dicho adaptador debe quedar igual al siguiente:

```
package dam.nathan.adapters;

import com.google.gson.TypeAdapter;
import com.google.gson.stream.JsonReader;
import com.google.gson.stream.JsonToken;
import com.google.gson.stream.JsonWriter;
import javafx.geometry.Point2D;

import java.io.IOException;

public class AdapterPoint2DJSON extends TypeAdapter<Point2D> {

    @Override
    public void write(JsonWriter jsonWriter, Point2D point2D) throws IOException {
        if (point2D == null) {
            jsonWriter.nullValue();
            return;
        }
        String xy = point2D.getX() + "," + point2D.getY();
        jsonWriter.value(xy);
    }

    @Override
    public Point2D read(JsonReader jsonReader) throws IOException {
        if (jsonReader.peek() == JsonToken.NULL) {
            jsonReader.nextNull();
            return null;
        }
        String xy = jsonReader.nextString();
        String[] partes = xy.split(",");
        double x = Double.parseDouble(partes[0]);
        double y = Double.parseDouble(partes[1]);
        return new Point2D(x,y);
    }
}
```

Donde se le indica como debe actuar cuando vaya a escribir y leer elementos Point2D. Para hacer uso de ello deberemos modificar el método saveJSON de la siguiente manera:


```
private static void saveJSON(Cactus c, String name) {  
    File fichero = new File(name + ".json");  
  
    GsonBuilder gb = new GsonBuilder();  
    gb.registerTypeAdapter(Point2D.class, new AdapterPoint2DJSON());  
  
    Gson gson = gb.create();  
  
    String json = gson.toJson(c);  
  
    try {  
        BufferedWriter bw = new BufferedWriter(new FileWriter(fichero));  
        bw.write(json);  
        bw.close();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Donde en dicho GsonBuilder le indicamos el tipo de adaptadores que se usaran, en este caso el de AdapterPoint2DJSON. Una vez hecho esto deberíamos poder generar fichero JSON en base a objetos Cactus y Bird sin inconvenientes. (La modificación del saveJSON se realizo también en el objeto Bird.)

A continuación, se creará el método para almacenar en XML, para ello haremos uso de la librería JAXB. El método debe quedar de la siguiente manera:

```
private static void saveXML(Cactus c, String name) {  
    try {  
        File fichero = new File(name + ".xml");  
        JAXBContext contexto = JAXBContext.newInstance(c.getClass());  
        Marshaller marshaller = contexto.createMarshaller();  
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);  
        marshaller.marshal(c, fichero);  
    } catch (JAXBException e) {  
        e.printStackTrace();  
    }  
}
```

Como podemos ver, estamos recibiendo un cactus y el nombre. Al igual que antes, creamos el fichero con ese nombre y tras ello creamos una nueva instancia de JAXBContext haciendo uso de la clase de Cactus. Tras ello creamos un marshaller con dicho contexto indicándole como propiedad que el texto este formateado y tras ello que guarde el objeto Cactus en el fichero indicado.

A continuación se muestra el correspondiente a la clase Bird.

```
private static void saveXML(Bird b, String name) {  
    try {  
        File fichero = new File(name + ".xml");  
        JAXBContext contexto = JAXBContext.newInstance(b.getClass());  
        Marshaller marshaller = contexto.createMarshaller();  
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);  
        marshaller.marshal(b, fichero);  
    } catch (JAXBException e) {  
        e.printStackTrace();  
    }  
}
```

No obstante, a diferencia de con la librería Gson, aquí aun nos falta por editar que elementos tomará el XML. Para ello primero deberemos indicar que el inicio de la clase es el Elemento Root del XML. Esto lo hacemos haciendo uso de las anotaciones. Tras indicar cuál es el elemento root deberemos indicar los elementos del XML, para ello en los getters creados anteriormente, le añadiremos la anotación XmlElement que indica que es un elemento y entre paréntesis el nombre que tendrá este en XML.

Usando como ejemplo la clase cactus, debería quedarnos algo similar a esto.

```
@XmlRootElement
public class Cactus {
    private String name;
    private Size size;
    private Point2D coordinate;

    public Cactus() {

    }

    public Cactus(String name, Size size, Point2D coordinate) {
        this.name = name;
        this.size = size;
        this.coordinate = coordinate;
    }

    @XmlElement (name = "nombre")
    public String getName() {
        return name;
    }

    @XmlElement (name = "size")
    public Size getSize() {
        return size;
    }

    @XmlElement (name = "coordinate")
    public Point2D getCoordinate() {
        return coordinate;
    }
}
```

Tal y como se encuentra ahora mismo, la clase Point2D no se pasará correctamente a XML dado que no trabaja con este tipo de objeto.

Para solucionar esto, deberemos crearle un adaptador. Dicho adaptador será de la siguiente manera y será útil para ambas clases:

```
package dam.nathan.adapters;

import jakarta.xml.bind.annotation.adapters.XmlAdapter;
import javafx.geometry.Point2D;

public class AdapterPoint2DXML extends XmlAdapter<String, Point2D> {
    @Override
    public Point2D unmarshal(String s) throws Exception {
        if (s.isEmpty() || s == null) {
            return null;
        } else {
            String[] partes = s.split(",");
            double x = Double.parseDouble(partes[0]);
            double y = Double.parseDouble(partes[1]);
            return new Point2D(x, y);
        }
    }

    @Override
    public String marshal(Point2D point2D) throws Exception {
        return point2D.getX() + "," + point2D.getY();
    }
}
```

Marshal hace referencia al momento de cuando escribe a XML mientras que unmarshal se refiere al momento de pasar de XML a objeto de Java. Ahora con el Adaptador de XML creado deberemos indicarle donde será usado, para ello, volviendo a la clase Cactus al getter del Point2D deberemos agregarle la anotación siguiente:

```
@XmlJavaTypeAdapter(AdapterPoint2DXML.class)
```

Donde se le indica que el adaptador a usar será la clase AdapterPoint2DXML, esto lo agregamos en ambas clases, Cactus y Bird.

Con esto habremos terminado el método save actual de XML

Por último nos quedaría el método para guardar en Binario

Para ello, en el método saveBin, el cuál recibe un cactus o un pájaro y un nombre que corresponde al fichero. Aquí usaremos FileOutputStream y ObjectOutputStream para almacenarlos en ficheros. Como para poder hacer uso de esto todos los objetos referenciados como atributos deben ser serializables y Point2D no lo es, he creado una clase Point2D que extiende del Point2D que hemos estado utilizando, es decir, el de javafx y le he implementado Serializable para poder escribir en binario

Esta es la clase Point2D creada:

```
package dam.nathan;

import java.io.Serializable;

public class Point2D extends javafx.geometry.Point2D implements Serializable {
    public Point2D(double v, double v1) {
        super(v, v1);
    }
}
```

Al haber hecho esto, deberemos indicar que el Point2D que usaremos será este y no el propio de javafx que no es serializable, para esto, los que están en el mismo paquete solo hay que comentar la línea del import de Point2D mientras que los que están en otro paquete, como son los adaptadores deberemos indicarle la dirección del paquete a importar, en este caso es dam.nathan.Point2D

Una vez hecho esto, podremos escribir el método para transformar el cuál es el siguiente:

```
private static void saveBIN(Cactus c, String name) {  
    File f = new File(name + ".bin");  
    FileOutputStream fos = null;  
    ObjectOutputStream salida = null;  
    try {  
        fos = new FileOutputStream(f);  
        salida = new ObjectOutputStream(fos);  
        salida.writeObject(c);  
    } catch (IOException e) {  
        e.printStackTrace();  
    } finally {  
        try {  
            salida.close();  
            fos.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

En este caso creamos el fichero, el FileOutputStream y la salida del objeto y dentro del try escribimos la salida del objeto capturando errores y por último cerrando los 2 outputStreams.

Por último se generaran varios casos de prueba para ver que la serialización funciona correctamente.

Para ello, creamos una clase para test y en está definimos un Test para guardar.

Está se verá así:

```
import dam.nathan.*;
import org.junit.jupiter.api.Test;

public class SaveTest {
    Point2D cb = new Point2D(1,0);
    Point2D cc = new Point2D(4,2);
    Size sb = new Size(4,2);
    Size sc = new Size(1,0);

    Cactus cactus = new Cactus("Jaune", sc, cc);
    Bird bird = new Bird("Elliot", 0.2f, sb, cb);

    @Test
    public void saveTest() throws excExtension {
        //Test de guardado Cactus
        Cactus.save(cactus, "./Ficheros/Tests/testBinarioCactus", "bin");
        Cactus.save(cactus, "./Ficheros/Tests/testJSONCactus", "json");
        Cactus.save(cactus, "./Ficheros/Tests/testXMLCactus", "xml");

        //Test de guardado Pajaros
        Bird.save(bird, "./Ficheros/Tests/testXMLBird", "xml");
        Bird.save(bird, "./Ficheros/Tests/testJSONBird", "json");
        Bird.save(bird, "./Ficheros/Tests/testBINBird", "bin");
    }

    @Test
    public void saveTestError() throws excExtension {
        Cactus.save(cactus, "./Ficheros/Tests/testErrorCactus", "png");
        Bird.save(bird, "./Ficheros/Tests/testErrorBird", "html");
    }
}
```

En esta ocasión hemos creado 2 tipos de tests distintos, 1 destinado a funcionar correctamente y el otro destinado a devolver error por las extensiones usadas. Además de crear un objeto Cactus y Otro bird para realizar las pruebas.







La extensión png no esta permitida

Empezando por el saveTestError podemos ver que devuelve error de que las extensiones no están permitidas (definido en excExtension)

A continuación se realizará el test de saveTest el cuál debería generar los ficheros correctamente.

✓ Tests passed: 1 of 1 test – 108 ms

Podemos ver como el test se ejecuta correctamente, y si vamos a la ruta especificada encontraremos los ficheros que se han creado

	testBinarioCactus.bin	02/10/2024 12:00	Archivo BIN	1 KB
	testBINBird.bin	02/10/2024 12:00	Archivo BIN	1 KB
	testJSONBird.json	02/10/2024 12:00	Archivo JSON	1 KB
	testJSONCactus.json	02/10/2024 12:00	Archivo JSON	1 KB
	testXMLBird.xml	02/10/2024 12:00	Microsoft Edge H...	1 KB
	testXMLCactus.xml	02/10/2024 12:00	Microsoft Edge H...	1 KB

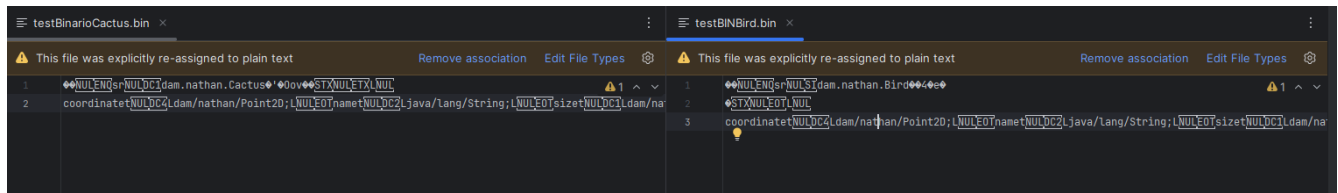
Como podemos ver los ficheros se han generado, pero todavía no sabemos si el contenido es el deseado. Para ello abrimos dichos ficheros para comprobarlo.

<code><?xml version="1.0" encoding="UTF-8" standalone="yes"?></code>	<code><?xml version="1.0" encoding="UTF-8" standalone="yes"?></code>
<code><bird></code>	<code><cactus></code>
<code><coordinate>1.0,0.0</coordinate></code>	<code><coordinate>4.0,2.0</coordinate></code>
<code><nombre>Elliot</nombre></code>	<code><nombre>Jaune</nombre></code>
<code><size></code>	<code><size></code>
<code><heigh>2.0</heigh></code>	<code><heigh>0.0</heigh></code>
<code><width>4.0</width></code>	<code><width>1.0</width></code>
<code></size></code>	<code></size></code>
<code><speed>0.2</speed></code>	<code></cactus></code>
<code></bird></code>	

Podemos ver que ambos XML de Cactus y de Pájaro se generan correctamente con los datos indicados.

<code>{ "name": "Jaune", "size": { "Width": 1.0, "Height": 0.0 }, "coordinate": "4.0,2.0" }</code>	<code>{ "name": "Elliot", "speed": 0.2, "size": { "Width": 4.0, "Height": 2.0 }, "coordinate": "1.0,0.0" }</code>
--	---

Así como ambos ficheros JSON tienen su contenido correctamente.



Así como los ficheros bin han guardado información que, pese a no ser completamente legible, podemos ver partes como son el paquete Point2D o una String indicadas. Por tanto los 3 ficheros se generaron correctamente.

2.3 Parte 3. Adaptadores

Modificar el código anterior para hacer que no se almacenen el nombre de los cactus y el pájaro. Se han de definir adaptadores para Gson y para JAXB. Comprobar que no se guarda el nombre

Para ello deberemos crear un nuevo adaptador para el nombre. Dicho adaptador será igual al siguiente:

```
package dam.nathan.adapters;

import jakarta.xml.bind.annotation.adapters.XmlAdapter;

public class AdapterNameXML extends XmlAdapter<String, String> {
    @Override
    public String unmarshal(String s) throws Exception {
        return null;
    }

    @Override
    public String marshal(String s) throws Exception {
        return null;
    }
}
```

Dado que se nos pide que no se almacene el nombre en el caso de guardarlo como XML o de volver a cargarlo dicho apartado deberá siempre devolver null. Ahora para que JAXB tome este adaptador deberemos indicarlo en el getter del nombre de la siguiente manera:

```
@XmlJavaTypeAdapter(AdapterNameXML.class)
```

Esto deberá incluirse en los getters de name de Cactus y de Bird para evitar que se registre un nombre. De esta manera, JAXB ya no registraría el nombre. En el caso de Gson deberemos crear también su adaptador el cuál se verá así:

```
package dam.nathan.adapters;

import com.google.gson.TypeAdapter;
import com.google.gson.stream.JsonReader;
import com.google.gson.stream.JsonWriter;

import java.io.IOException;

public class AdapterNameJSON extends TypeAdapter<String> {
```

```
@Override
public void write(JsonWriter jsonWriter, String s) throws IOException {
    jsonWriter.nullValue();
    return;
}

@Override
public String read(JsonReader jsonReader) throws IOException {
    return null;
}
}
```

Como podemos ver siempre guardará el nombre como nulo.

Tras ello deberemos agregar el adaptador en el método de guardar XML agregando esta línea, igual que hicimos con Point2D anteriormente.

```
gb.registerTypeAdapter(String.class, new AdapterNameJSON());
```

Esta línea deberá agregarse en ambos métodos de guardado JSON, quedándonos el método de la siguiente manera:

```
private static void saveJSON(Cactus c, String name) {
    File fichero = new File(name + ".json");

    GsonBuilder gb = new GsonBuilder();
    gb.registerTypeAdapter(Point2D.class, new AdapterPoint2DJSON());
    gb.registerTypeAdapter(String.class, new AdapterNameJSON());

    Gson gson = gb.create();

    String json = gson.toJson(c);

    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter(fichero));
        bw.write(json);
        bw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Por último, para poner a prueba su funcionamiento volveremos a ejecutar los test para ver si se realmente se ignora el nombre.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bird>
  <coordinate>1.0,0.0</coordinate>
  <nombre/>
  <size>
    <heigh>2.0</heigh>
    <width>4.0</width>
  </size>
  <speed>0.2</speed>
</bird>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cactus>
  <coordinate>4.0,2.0</coordinate>
  <nombre/>
  <size>
    <heigh>0.0</heigh>
    <width>1.0</width>
  </size>
</cactus>
```

Al realizar los tests, podemos ver que los ficheros XML ignoran completamente el nombre.

```
{"speed":0.2,"size":{"Width":4.0,"Heigh":2.0},"coordinate":"1.0,0.0"}
```

```
{"size":{"Width":1.0,"Heigh":0.0},"coordinate":"4.0,2.0"}
```

Podemos ver que en el caso de los JSON realiza lo mismo.

2.4 Parte 4. Creando el método load

Crea el método estático load, para el cactus y el pájaro que recibe el nombre del fichero y su extensión y devuelve un cactus o un pájaro según su clase. Define casos de prueba para probar el código anterior.

Para ello creamos primero dicho método estático el cuál se verá así para Bird

```
public static Bird load(String name, String ext) throws excExtension {
    String extension = ext.toLowerCase();
    switch (extension) {
        case "xml":
            return loadXML(name);
        case "json":
            return loadJSON(name);
        case "bin":
            return loadBIN(name);
        default:
            throw new excExtension(extension);
    }
}
```

Y así para Cactus

```
public static Cactus load(String name, String ext) throws excExtension {
    String extension = ext.toLowerCase();
    switch (extension) {
        case "xml":
            return loadXML(name);
        case "json":
            return loadJSON(name);
        case "bin":
            return loadBIN(name);
        default:
            throw new excExtension(extension);
    }
}
```

Al igual que en guardar, el switch llamará a un método dependiendo de la extensión.

Comenzando con cargar ficheros en XML. Para ello, crearemos el método loadXML que recibirá como parámetro el nombre del fichero y nos devolverá un Bird o un Cactus según su clase. Dentro de dicho método declaramos el objeto a devolver y luego creamos la instancia de JAXBContext de la clase del objeto. Tras ello deberemos crear un unmarshaller en base al JAXBContext y por último el objeto que hemos declarado en un principio ahora tendrá como valor el unmarshaller del

fichero que hemos pasado. Lo mostraré por pantalla para comprobar que todo fue correctamente y devolveremos dicho objeto. El código usado para todo esto es el siguiente:

Bird:

```
private static Bird loadXML(String name) {
    Bird b;
    try {
        JAXBContext context = JAXBContext.newInstance(Bird.class);
        Unmarshaller unmarshaller = context.createUnmarshaller();

        b = (Bird) unmarshaller.unmarshal(new File(name + ".xml"));

        return b;
    } catch (JAXBException e) {
        e.printStackTrace();
        return null;
    }
}
```

Cactus:

```
private static Cactus loadXML(String name) {
    Cactus c;
    try {
        JAXBContext context = JAXBContext.newInstance(Cactus.class);
        Unmarshaller unmarshaller = context.createUnmarshaller();

        c = (Cactus) unmarshaller.unmarshal(new File(name + ".xml"));

        return c;
    } catch (JAXBException e) {
        e.printStackTrace();
        return null;
    }
}
```

Una vez hecho, el método de carga por XML estará finalizado.

A continuación haré el de JSON. Para este haremos uso de GsonBuilder para indicarle los adaptadores del nombre y de Point2D y tras ello creamos el Gson usando ese builder. Tras ello indicamos que el valor del objeto (Bird o Cactus) es igual a lo que contenga ese json. El código final es el siguiente:

Bird:

```
private static Bird loadJSON(String name) {
    File fichero = new File(name + ".json");
    Bird b;
    try {
        FileReader fr = new FileReader(fichero);
        GsonBuilder gb = new GsonBuilder();

        gb.registerTypeAdapter(Point2D.class, new AdapterPoint2DJSON());
        gb.registerTypeAdapter(String.class, new AdapterNameJSON());

        Gson gson = gb.create();

        b = gson.fromJson(fr, Bird.class);
        return b;
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        return null;
    }
}
```

Cactus:

```
private static Cactus loadJSON(String name) {
    File fichero = new File(name + ".json");
    Cactus c;
    try {
        FileReader fr = new FileReader(fichero);
        GsonBuilder gb = new GsonBuilder();

        gb.registerTypeAdapter(Point2D.class, new AdapterPoint2DJSON());
        gb.registerTypeAdapter(String.class, new AdapterNameJSON());

        Gson gson = gb.create();

        c = gson.fromJson(fr, Cactus.class);
        return c;
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        return null;
    }
}
```

Por último se realizará los métodos para leer en binario los cuáles se mostrarán a continuación pero no se podrá ejecutar bien debido a que Point2D no es serializable y si creamos una clase para hacerla serializable nos dará error de constructor debido a que no tiene constructor dicha clase y por ende no puede construirlo devolviendo error. El código es el siguiente:

```
private static Bird loadBIN(String name) {
    Bird b;
    FileInputStream fis = null;
    ObjectInputStream entrada = null;
    try{
        fis = new FileInputStream(name + ".bin");
        entrada = new ObjectInputStream(fis);
        b = (Bird) entrada.readObject();
        //b.toString();
        return b;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
        return null;
    } finally {
        try {
            fis.close();
            entrada.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

El cargar todos los objetos nos devuelve lo siguiente:

```
XML
Bird{name='null', speed=0.2, size=Size [Heigh = 2.0, Width = 4.0], coordinate=Point2D [x = 1.0, y = 0.0]}
Cactus{name='null', size=Size [Heigh = 0.0, Width = 1.0], coordinate=Point2D [x = 4.0, y = 2.0]}

JSON
Bird{name='null', speed=0.2, size=Size [Heigh = 2.0, Width = 4.0], coordinate=Point2D [x = 1.0, y = 0.0]}
Cactus{name='null', size=Size [Heigh = 0.0, Width = 1.0], coordinate=Point2D [x = 4.0, y = 2.0]}

BIN
java.io.InvalidClassException: Create breakpoint : dam.nathan.Point2D; no valid constructor
```

De XML y JSON nos muestra sus objetos almacenados pero BIN nos devuelve el error explicado anteriormente.

2.5 Parte 5. Clase Juego con lista de niveles con método guardar y cargar

Crear ahora la clase juego con una lista de niveles.

La clase juego ha de tener los métodos estáticos load y save, de igual forma que el cactus y el pájaro (estos no se usan en la clase juego).

Definir casos de prueba y probar.

Para ello primero creamos la clase Level que contendrá los atributos de cada nivel como se han mencionado anteriormente, los cuáles son:

- Tiempo de juego (Almacenado como double)
- Velocidad, almacenada en float
- Lista de cactus
- Lista de pájaros
- Música (almacenado en String)

La clase (sin anotaciones para XML que se añadirán posteriormente) debería ser como la siguiente:

```
package dam.nathan;

import java.io.Serializable;

public class Level implements Serializable {
    private double time;
    private float speed;
    private Cactus[] cactus;
    private Bird[] birds;
    private String music;

    public Level() {
    }

    public Level(double time, float speed, Cactus[] cactus, Bird[] birds, String music) {
        this.time = time;
        this.speed = speed;
        this.cactus = cactus;
        this.birds = birds;
        this.music = music;
    }

    public double getTime() {
        return time;
    }

    public float getSpeed() {
        return speed;
    }

    public Cactus[] getCactus() {
        return cactus;
    }

    public Bird[] getBirds() {
        return birds;
    }

    public String getMusic() {
        return music;
    }

    public void setTime(double time) {
        this.time = time;
    }

    public void setSpeed(float speed) {
        this.speed = speed;
    }
}
```

```
public void setCactus(Cactus[] cactus) {  
    this.cactus = cactus;  
}  
  
public void setBirds(Bird[] birds) {  
    this.birds = birds;  
}  
  
public void setMusic(String music) {  
    this.music = music;  
}  
}
```

Una vez creada, podremos hacer la clase juego (Game) la cuál hemos dicho que contendrá una lista de levels y tendrá métodos de save y load.

Dicha clase con todos los métodos implementados de save y load más sus submetodos se verá así:

```
package dam.nathan;  
  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
import dam.nathan.adapters.AdapterNameJSON;  
import dam.nathan.adapters.AdapterPoint2DJSON;  
import jakarta.xml.bind.JAXBContext;  
import jakarta.xml.bind.JAXBException;  
import jakarta.xml.bind.Marshaller;  
import jakarta.xml.bind.Unmarshaller;  
import jakarta.xml.bind.annotation.XmlElement;  
import jakarta.xml.bind.annotation.XmlRootElement;  
  
import java.io.*;  
import java.util.Arrays;  
  
@XmlRootElement  
public class Game implements Serializable{  
    private Level[] levels;  
  
    public Game() {  
    }  
  
    public Game(Level[] levels) {  
        this.levels = levels;  
    }  
  
    public static void save(Game g, String name, String ext) throws excExtension {  
        String extension = ext.toLowerCase();  
        switch (extension) {  
            case "xml":  
                saveXML(g, name);  
            }  
    }  
}
```

```
        break;
    case "json":
        saveJSON(g, name);
        break;
    case "bin":
        saveBIN(g, name);
        break;
    default:
        throw new excExtension(extension);
    }
}

private static void saveBIN(Game g, String name) {
    FileOutputStream fos = null;
    ObjectOutputStream salida = null;
    try {
        fos = new FileOutputStream(name + ".bin");
        salida = new ObjectOutputStream(fos);
        salida.writeObject(g);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            salida.close();
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

private static void saveJSON(Game g, String name) {
    File fichero = new File(name + ".json");

    GsonBuilder gb = new GsonBuilder();
    gb.registerTypeAdapter(Point2D.class, new AdapterPoint2DJSON());
    gb.registerTypeAdapter(String.class, new AdapterNameJSON());

    Gson gson = gb.create();

    String json = gson.toJson(g);

    System.out.println(json);
    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter(fichero));
        bw.write(json);
        bw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void saveXML(Game g, String name) {
```

```
    try {
        File fichero = new File(name + ".xml");
        JAXBContext contexto = JAXBContext.newInstance(g.getClass());
        Marshaller marshaller = contexto.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
        marshaller.marshal(g, fichero);
    } catch (JAXBException e) {
        e.printStackTrace();
    }
}

public static Game load(String name, String ext) throws excExtension {
    String extension = ext.toLowerCase();
    switch (extension) {
        case "xml":
            return loadXML(name);
        case "json":
            return loadJSON(name);
        case "bin":
            return loadBIN(name);
        default:
            throw new excExtension(extension);
    }
}

private static Game loadXML(String name) {
    Game g;
    try {
        JAXBContext context = JAXBContext.newInstance(Game.class);
        Unmarshaller unmarshaller = context.createUnmarshaller();

        g = (Game) unmarshaller.unmarshal(new File(name + ".xml"));
        return g;
    } catch (JAXBException e) {
        e.printStackTrace();
        return null;
    }
}

private static Game loadJSON(String name) {

    GsonBuilder gb = new GsonBuilder();
    gb.registerTypeAdapter(Point2D.class, new AdapterPoint2DJSON());
    gb.registerTypeAdapter(String.class, new AdapterNameJSON());

    Gson gson = gb.create();
    try {
        Game g = gson.fromJson(new FileReader(name + ".json"), Game.class);
        return g;
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        return null;
    }
}
```

```
}  
}  
  
private static Game loadBIN(String name) {  
    Game g;  
    FileInputStream fis = null;  
    ObjectInputStream entrada = null;  
    try {  
        fis = new FileInputStream(name + ".bin");  
        entrada = new ObjectInputStream(fis);  
        g = (Game) entrada.readObject();  
        return g;  
    } catch (IOException e) {  
        e.printStackTrace();  
        return null;  
    } catch (ClassNotFoundException e) {  
        e.printStackTrace();  
        return null;  
    }  
}  
  
@XmlElement(name = "Nivel")  
public Level[] getLevels() {  
    return levels;  
}  
  
public void setLevels(Level[] levels) {  
    this.levels = levels;  
}  
  
@Override  
public String toString() {  
    return "Game{" +  
        "levels=" + Arrays.toString(levels) +  
        '}';  
}  
}
```

Tras ello mediante tests probaremos el funcionamiento de Save y después el de Load.

Un dato a tener en cuenta es que pese a estar presente, el cargado de Binario al igual que ocurría anteriormente devuelve error debido al constructor. El método esta creado pero en los tests no se llega a usar para evitar problemas, no obstante el mismo esta presente pero comentado.

```
✓ Tests passed: 1 of 1 test - 136 ms

XML
Game{levels=[Level{time=20.0, speed=0.4, cactus=[Cactus{name='null', size=Size [Heigh = 0.0, Width = 1.0], coordinate=Point2D [x = 4.0, y = 2.0]}, Cactus{name='null', size=Size [Heigh = 2.0, Width = 4.0]}]}]

JSON
Game{levels=[Level{time=20.0, speed=0.4, cactus=[Cactus{name='null', size=Size [Heigh = 0.0, Width = 1.0], coordinate=Point2D [x = 4.0, y = 2.0]}, Cactus{name='null', size=Size [Heigh = 2.0, Width = 4.0]}]}]
```

Al pasar el test de guardado podemos ver que esta vez nos devuelve todos los valores correspondientes. Para verse más claro el resultado dejo a continuación una captura del fichero XML y del fichero JSON

```
testJSONGame.json  testXMLGame.xml X
testXMLGame.xml
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <game>
3      <Nivel>
4          <birds>
5              <coordinate>1.0,0.0</coordinate>
6              <nombre/>
7              <size>
8                  <heigh>2.0</heigh>
9                  <width>4.0</width>
10             </size>
11             <speed>0.2</speed>
12         </birds>
13         <birds>
14             <coordinate>4.0,2.0</coordinate>
15             <nombre/>
16             <size>
17                 <heigh>0.0</heigh>
18                 <width>1.0</width>
19             </size>
20             <speed>0.3</speed>
21         </birds>
22         <cactus>
23             <coordinate>4.0,2.0</coordinate>
24             <nombre/>
25             <size>
26                 <heigh>0.0</heigh>
27                 <width>1.0</width>
28             </size>
29         </cactus>
30         <cactus>
31             <coordinate>1.0,0.0</coordinate>
32             <nombre/>
33             <size>
34                 <heigh>2.0</heigh>
35                 <width>4.0</width>
36             </size>
37         </cactus>
38         <music>musical</music>
39         <speed>0.4</speed>
40         <time>20.0</time>
41     </Nivel>
42     <Nivel>
43         <birds>
44             <coordinate>1.0,0.0</coordinate>
45             <nombre/>
46             <size>
47                 <heigh>2.0</heigh>
48                 <width>4.0</width>
49             </size>
50             <speed>0.2</speed>
51         </birds>
52         <birds>
53             <coordinate>4.0,2.0</coordinate>
54             <nombre/>
55             <size>
56                 <heigh>0.0</heigh>
57                 <width>1.0</width>
58             </size>
59             <speed>0.3</speed>
60         </birds>
```

Como podemos ver, el fichero XML se genera correctamente.

A continuación el fichero JSON

```
() testUSONGame.json • testXMLGame.xml
() testUSONGame.json > ...
1  {}
2  "levels": [
3  {
4      "time": 20.0,
5      "speed": 0.4,
6      "cactus": [
7          {
8              "size": {
9                  "Width": 1.0,
10                 "Heigh": 0.0
11             },
12             "coordinate": "4.0,2.0"
13         },
14         {
15             "size": {
16                 "Width": 4.0,
17                 "Heigh": 2.0
18             },
19             "coordinate": "1.0,0.0"
20         }
21     ],
22     "birds": [
23         {
24             "speed": 0.2,
25             "size": {
26                 "Width": 4.0,
27                 "Heigh": 2.0
28             },
29             "coordinate": "1.0,0.0"
30         },
31         {
32             "speed": 0.3,
33             "size": {
34                 "Width": 1.0,
35                 "Heigh": 0.0
36             },
37             "coordinate": "4.0,2.0"
38         }
39     ]
40 },
41 {
42     "time": 30.0,
43     "speed": 0.5,
44     "cactus": [
45         {
46             "size": {
47                 "Width": 1.0,
48                 "Heigh": 0.0
49             },
50             "coordinate": "4.0,2.0"
51         },
52         {
53             "size": {
54                 "Width": 4.0,
55                 "Heigh": 2.0
56             },
57             "coordinate": "1.0,0.0"
58         }
59     ],
60     "birds": [
```

Como se puede apreciar, también es generado correctamente.

Por último el Test de cargar dichos ficheros

```
✓ Tests passed: 1 of 1 test - 132 ms

XML
Game{levels=[Level{time=20.0, speed=0.4, cactus=[Cactus{name='null', size=Size [Heigh = 0.0, Width = 1.0], coordinate=Point2D [x = 4.0, y = 2.0]}, Cactus{name='null',

JSON
Game{levels=[Level{time=20.0, speed=0.4, cactus=[Cactus{name='null', size=Size [Heigh = 0.0, Width = 1.0], coordinate=Point2D [x = 4.0, y = 2.0]}, Cactus{name='null',
```

Podemos ver que al ejecutarlo nos carga el fichero por pantalla correctamente, donde los datos de game han sido cargados de los ficheros mostrados previamente.