Tarea Recuperación Tema 2. Museo

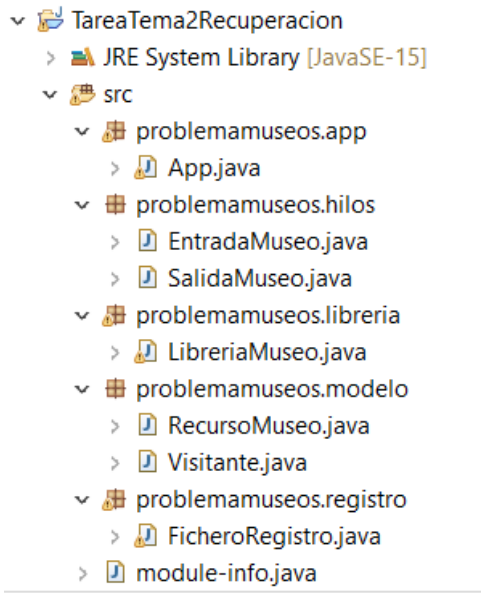
Contenido

Entrega

Se **entregará el proyecto Tarea2Recuperacion** en un **fichero comprimido .zip apellidosNombreTarea2Recuperacion.zip**

Tarea

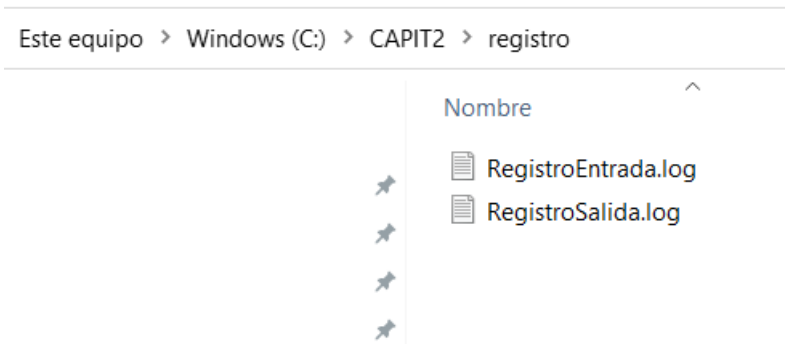
En esta **tarea vamos a realizar mediante hilos la simulación de la entrada y salida** en un museo que tiene un aforo máximo. Los **Visitantes registrarán su salida y su entrada** en un fichero y **se quedarán esperando en caso de que el museo este lleno.**



Un ejemplo de la ejecución del programa sería el siguiente:

```
9 en museo
hilo Visitante [nombre=JAVIER, apellidos=Vázquez, identificador=-7153419061817902463]-----
Entra en museo
10 en museo
hilo Visitante [nombre=PAULA, apellidos=Santos, identificador=-8953507218882675973]-----
Sale de museo
9 en museo
hilo Visitante [nombre=SARA, apellidos=Moreno, identificador=-8994509202755044398]-----
Entra en museo
10 en museo
hilo Visitante [nombre=SARA, apellidos=Jiménez, identificador=-6139663795316095447]-----
Sale de museo
9 en museo
hilo Visitante [nombre=ANTONIA, apellidos=Ramos, identificador=-6560669740160477506]-----
Entra en museo
10 en museo
```

Se generaran **dos ficheros de registro uno de entrada y otro de salida**.



El formato que deben seguir **el Registro de Entrada** es el siguiente. **Respetad los formatos de fecha y hora**. Igualmente, los formatos para nombres e id.

Entrada registro fecha: 31/03/22-13:04:12
Nombre: MANUEL Apellidos: Sanz Id: -9139666565030272558

El formato que deben seguir **el Registro de Salida** es el siguiente

Salida registro fecha: 05/04/22-09:45:55
Nombre: MARIA ISABEL Apellidos: Torres Id: -4973332696818215099

Salida registro fecha: 05/04/22-09:45:55
Nombre: ANTONIO Apellidos: López Id: -5400973374380985821

Recursos

Se proporciona la **clase principal App.java**

```
public class App {



	/**
	 * @param args the command line arguments
	 */
	public static void main(String[] args) {
		RecursoMuseo museo = new RecursoMuseo();

		ForkJoinPool pool = ForkJoinPool.commonPool();

		Visitante visitanteAleatorio = null;

		for (int i = 1; i <= 50; i++) {

			visitanteAleatorio =
LibreriaMuseo.getInstanceOfLibreriaMuseo().visitanteAleatorio();

			pool.submit(new EntradaMuseo(visitanteAleatorio, museo));

		} // entrada de 10 hilos al museo



		try {
			if (!pool.awaitTermination(150, TimeUnit.SECONDS)) {
				pool.shutdown();
				System.exit(-1);
			}
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
			System.exit(1);
		}

	}

	// museo.museoSnapShot(visitanteAleatorio);

}
```

Se **proporciona igualmente la clase de Modelo Visitante**. Como detalle **estamos usando la clase UUID** que nos **permite crear ID únicos** por **cada ejecución** del programa.

```
package problemamuseos.modelo;

import java.util.UUID;

public class Visitante {

	private String nombre="";
	private String apellidos="";
	private long identificador=0L;
```

```java
public Visitante() {


}
public Visitante(String nombre, String apellidos) {
	super();
	this.nombre = nombre;
	this.apellidos = apellidos;
	this.identificador=UUID.randomUUID().getLeastSignificantBits();
}
public String getNombre() {
	return nombre;
}
public void setNombre(String nombre) {
	this.nombre = nombre;
}
public String getApellidos() {
	return apellidos;
}
public void setApellidos(String apellidos) {
	this.apellidos = apellidos;
}
public long getIdentificador() {
	return identificador;
}
public void setIdentificador(long identificador) {
	this.identificador = identificador;
}
@Override
public int hashCode() {
	final int prime = 31;
	int result = 1;
	result = prime * result + ((apellidos == null) ? 0 : apellidos.hashCode());
	result = prime * result + (int) (identificador ^ (identificador >>> 32));
	result = prime * result + ((nombre == null) ? 0 : nombre.hashCode());
	return result;
}
@Override
public boolean equals(Object obj) {
	if (this == obj)
		return true;
	if (obj == null)
		return false;
	if (getClass() != obj.getClass())
		return false;
	Visitante other = (Visitante) obj;
	if (apellidos == null) {
		if (other.apellidos != null)
			return false;
	} else if (!apellidos.equals(other.apellidos))
		return false;
	if (identificador != other.identificador)
		return false;
	if (nombre == null) {
		if (other.nombre != null)
			return false;
	} else if (!nombre.equals(other.nombre))
		return false;
```

```
            return true;
        }
        @Override
        public String toString() {
            return "Visitante [nombre=" + nombre + ", apellidos=" + apellidos + ",
identificador=" + identificador + "]";
        }




}
```

Se debe implementar

La clase FicheroRegistro

Nos permitirá **manejar los registros tanto de entrada como de salida**. Debemos **inicializarla en LibreriaMuseo** llamando al método `iniciarRegistroDeEntrada` que creará las **estructuras necesarias.**

```
public class FicheroRegistro {

        private static final String PATH ="c:/capit2/registro/";
        private static final String FICHERO_MENSAJES_ENTRADA="RegistroEntrada.log";
        private static final String FICHERO_MENSAJES_SALIDA="RegistroSalida.log";


        public FicheroRegistro () {




        }


        // Crea los directorios para el programa si no existen
        // Se debe iniciar en la LibreríaMuseo al inicializar los recursos
        public static void iniciarRegistroDeEntrada() throws IOException {



        }

        // Escribe en el fichero de registro de entrada la entrada del visitante
        // con el formato visto anteriormente, usa el método privado escribirEnFichero
        // para escribir en el registro de entrada

        public static void registroDeEntrada(Visitante visitante) throws IOException {


        }
```

```
	// Escribe en el fichero de registro de salida la salida del visitante
	// con el formato visto anteriormente, usa el método privado escribirEnFichero

	public static void registroDeSalida(Visitante visitante) throws IOException   {



	}

	private static void  escribirEnFichero(String mensaje, FileWriter fileWriter)
throws IOException {









	}









}
```

LibreríaMuseo

Ofrece **métodos que nos serán útiles para crear Visitantes aleatorios** y **tiempos de espera** sin la necesidad de llamar a sleep.

```
public class LibreriaMuseo {


	private static final String nombres[] = {"ANTONIO", "MANUEL", "JOSE", "FRANCISCO",
"DAVID", "JUAN", "JOSE ANTONIO", "JAVIER", "DANIEL", "JOSE LUIS", "FRANCISCO JAVIER",
			"CARLOS", "JESUS", "ALEJANDRO", "FRANCISCA", "LUCIA", "MARIA ISABEL",
"MARIA JOSE", "ANTONIA", "DOLORES", "SARA",
			"PAULA", "ELENA", "MARIA LUISA", "RAQUEL"};

	private static final String apellidos[] = {"García", "González", "Rodríguez",
"Fernández", "López", "Martínez", "Sánchez", "Pérez", "Gómez",
			"Martin", "Jiménez", "Ruiz", "Hernández", "Diaz", "Moreno", "Muñoz",
"Álvarez", "Romero", "Alonso", "Gutiérrez", "Navarro",
			"Torres", "Domínguez", "Vázquez", "Ramos", "Gil", "Ramírez",
"Serrano", "Blanco", "Molina", "Morales", "Suarez", "Ortega",
			"Delgado", "Castro", "Ortiz", "Rubio", "Marín", "Sanz", "Núñez",
"Iglesias", "Medina", "Garrido", "Cortes", "Castillo", "Santos"};

	private static Random randomnum = new Random();
```

```
private static LibreriaMuseo instancia = null;



// devueve un valor aleatorio del array pasado como parámetro
// usad la propiedad randomnum
private static  String valorAleatorio (String []array) {


}


private  LibreriaMuseo() {

	try {
		iniciarRecursosAplicacion();
	} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();


	}
}

// Aplicamos el patron Singleton
public static LibreriaMuseo getInstanceOfLibreriaMuseo() {


	if (instancia == null) {


		return new LibreriaMuseo();
	}

	else {


		return instancia;
	}
}


// pierde tiempo en Segundos en un bucle
// No usar sleep
 private  void tiempoProcesamiento(int seg) {




 }


 public  void tiempoEntrada() {


	 tiempoProcesamiento(2);
 }
```

```
        public  void tiempoSalida() {


               tiempoProcesamiento(2);
        }

        public  void tiempoEstancia() {



              tiempoProcesamiento(randomnum.nextInt(7));


        }




        // genera un visitante aleatorio usando los arrays nombre y apellids
       // y el método valorAleatorio
        public  Visitante visitanteAleatorio() {


        }

        //Inicia el FicheroRegistro
        private static void  iniciarRecursosAplicacion() throws IOException {




        }

}
```

La clase RecursoMuseo

```
public class RecursoMuseo {
       // clase que simula las entradas y las salidas al Jardín
       private int visitantes; // para contar las entradas y salidas al Jardín
       private static int AFORO_MAX = 10;

       // Añadiremos el visitante a la lista cuando entre
       //Lo quitaremos de la lista cuando salga
       private List<Visitante> listaVisitantes;
```

Es el **objeto compartido de los hilos de mi aplicación**. Proporciona dos **métodos que deben ser seguros con respecto a hilos.**

`entraMuseo(Visitante visitante)`

Realiza las siguientes **acciones**:

1. EL **hilo del visitante que intenta entrar al museo** se quedará **esperando si se supera el aforo máximo.**
2. **Ocupa una plaza** en el museo
3. Cuando **logra entrar escribe en consola**:
   ```
   hilo Visitante [nombre=MARIA JOSE, apellidos=Iglesias, identificador=-
   4971400931341079727]----- Entra en museo 10 en museo
   ```
4. Escribe **en RegistroEntrada.log**

Entrada registro fecha: 31/03/22-13:04:12
Nombre: MARIA JOSE Apellidos: Iglesias Id: -4971400931341079727

```
salidaMuseo(Visitante visitante)
```

Realiza las siguientes **acciones**:

1. Avisa de que **libera una plaza en el museo**
2. Cuando **logra entrar escribe en consola**:
   ```
   hilo Visitante [nombre=MARIA ISABEL, apellidos=Torres, identificador
   -4973332696818215099]----- Entra en museo 10 en museo
   ```
3. **Escribe en RegistroSalida.log**

Salida registro fecha: 05/04/22-09:45:55
Nombre: MARIA ISABEL Apellidos: Torres Id: -4973332696818215099

El Hilo EntradaMuseo

El hilo **se encargará de realizar los tramites de entrada** y **permanecerá en el museo el tiempo marcado**.

```
public class EntradaMuseo extends Thread {
//clase derivada de Thread que define un hilo
    private RecursoMuseo museo;
```

```
    private Visitante visitante;

    public EntradaMuseo(Visitante visitante, RecursoMuseo museo) {
```

1. **Simula el tiempo gastado en la entrada** llamando al **método** de la **LibreriaMuseo** LibreriaMuseo.getInstanceOfLibreriaMuseo().tiempoEntrada();
2. Realiza **la entrada al museo** con el objeto compartido **RecursoMuseo**.
3. Realiza **la visita con la llamada al método** de LibreriaMuseo `tiempoEstancia()`
4. Lanza la **petición de salida**, el **hilo SalidaMuseo**.

El Hilo SalidaMuseo

El **hilo se encargará de realizar los tramites de salida** realizando las acciones necesarias.

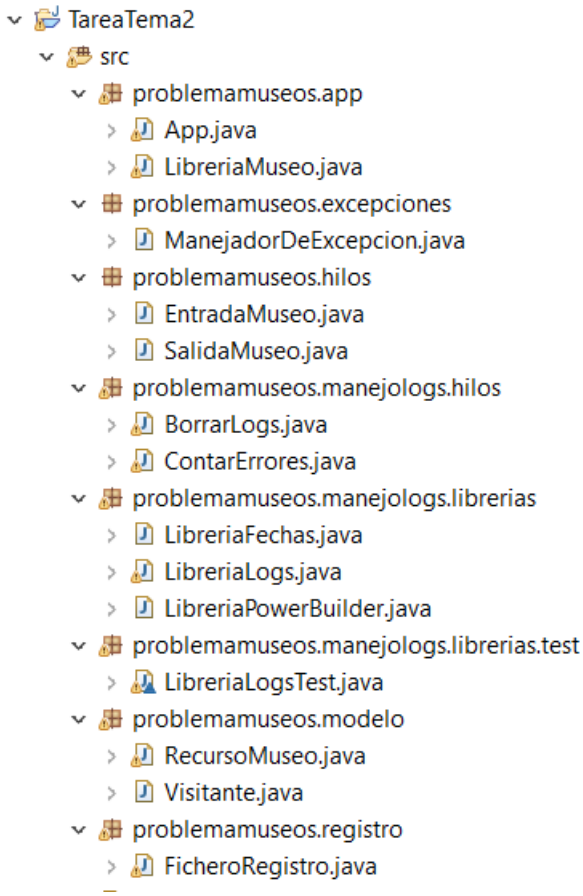
```
public class SalidaMuseo extends Thread {
    private RecursoMuseo museo;
    private Visitante visitante;
    public SalidaMuseo(Visitante visitante, RecursoMuseo museo) {
```

1. **Simula el tiempo gastado en la salida** llamando al método de la LibreriaMuseo tiempoSalida();

2. **Realiza la salida a través** del **objeto** compartido **RecursoMuseo**.

Opcional

Integrar el sistema de borrado de logs y estadísticas de la **tarea del Tema 1**. Se proporciona un **manejador de excepciones que recibirá una excepción y escribirá en el fichero de log correspondiente**. Debéis **modificar LibreriaLogs** para ello.



El **manejador de Excepciones se encarga de escribir en los ficheros de Log**, según el tipo de excepción escribira:

IOERROR: `ServerIOErrors.log`
CONCURRENT_ERROR: `ServerConcurrentErrors.log`
OTHER_ERROR: `ServerOtherErrors.log`

```
public class ManejadorDeExcepcion {

	public ManejadorDeExcepcion() {



	}

	public static void manejaExcepciones(Exception e) {
		try {
```

```
            if (e instanceof IOException) {


                  LibreriaLogs.escribirLogErrores(e.getMessage(),
LibreriaLogs.IO_ERROR);

            } else if ( e instanceof InterruptedException) {


                  LibreriaLogs.escribirLogErrores(e.getMessage(),
LibreriaLogs.CONCURRENT_ERROR);

            } else {


                  LibreriaLogs.escribirLogErrores(e.getMessage(),
LibreriaLogs.OTHER_ERROR);

            }

            } catch (Exception ex) {

                  ex.printStackTrace();
            }




      }

}
```