Curso de Java a distancia

Clase 29: Excepciones - bloque finally

El manejo de excepciones en Java puede incluir otro bloque llamado 'finally':

```
try {
  [instrucciones 1]
} catch([excepción 1]) {
  [instrucciones 2]
} finally {
  [instrucciones 3]
}
```

El objetivo de este bloque es liberar recursos que se solicitan en el bloque try. El bloque finally se ejecuta siempre, inclusive si se genera la captura de una excepción.

Los recursos más comunes que se deben liberar son las conexiones a bases de datos, uso de archivos y conexiones de red. Un recurso que no se libera impide que otro programa lo pueda utilizar. Al disponer la liberación de recursos en el bloque 'finally' nos aseguramos que todo recurso se liberará, inclusive aunque se dispare una excepción.

Tener en cuenta que si no se disparan ninguna excepción en un bloque try luego de esto se ejecuta el bloque 'finally'.

El bloque finally es opcional y en el caso de estar presente se coloca después del último bloque catch.

Problema:

Crear un archivo de texto con dos líneas. Luego proceder a leer el contenido del archivo de texto y mostrarlo por pantalla. Asegurarse de liberar el archivo luego de trabajar con el mismo

Programa:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CreacionLecturaArchivoTXT {
   public static void main(String[] ar) {
      FileWriter fw = null;
      BufferedWriter bw = null;
      try {
```

```
            fw = new FileWriter(new File("datos.txt"));
            bw = new BufferedWriter(fw);
            bw.write("Línea 1");
            bw.newLine();
            bw.write("Línea 2");
        } catch (IOException ex) {
            System.out.println("Problemas en la creación del archivo");
            System.out.println(ex.getMessage());
        } finally {
            try {
                if (bw != null)
                    bw.close();
                if (fw != null)
                    fw.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
        FileReader fr = null;
        BufferedReader br = null;
        try {
            fr = new FileReader(new File("datos.txt"));
            br = new BufferedReader(fr);
            String linea = br.readLine();
            while (linea != null) {
                System.out.println(linea);
                linea = br.readLine();
            }
        } catch (IOException ex) {
            System.out.println("Problemas con la lectura del archivo");
            System.out.println(ex.getMessage());
        } finally {
            try {
                if (br != null)
                    br.close();
                if (fr != null)
                    fr.close();
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }
}
```

Hemos sacado del try la definición de las dos variables para la creación de archivos de texto:

```
        FileWriter fw = null;
        BufferedWriter bw = null;
```

En el bloque try si hay un problema cuando se crea el archivo 'datos.txt' o cuando se graban datos mediante el método write se dispara la excepción IOException:

```
            fw = new FileWriter(new File("datos.txt"));
            bw = new BufferedWriter(fw);
            bw.write("Línea 1");
            bw.newLine();
            bw.write("Línea 2");
        } catch (IOException ex) {
            System.out.println("Problemas en la creación del archivo");
            System.out.println(ex.getMessage());
```

Luego que se haya disparado la excepción o no, pasa a ejecutarse el bloque 'finally', donde debemos llamar al método close tanto del objeto bw como fw, como los objetos pueden no haberse creado, disponemos un if para cada uno:

```
} finally {
   try {
      if (bw != null)
         bw.close();
      if (fw != null)
         fw.close();
   } catch (IOException ex) {
      System.out.println(ex.getMessage());
   }
}
```

Además hemos dispuesto otro bloque try/catch ya que el método close del FileWriter puede generar una excepción de tipo IOException.

De forma muy similar hemos codificado la lectura del archivo de texto para asegurarnos la llamada del método close del FileReader.

Excepciones - lanzar una excepción mediante comando throw

Hemos visto hasta ahora cual es la sintaxis para capturar excepciones que lanzan métodos. Ahora veremos como podemos lanzar una excepción para que sea eventualmente capturada posteriormente.

Hay que definir con juicio que métodos de una clase deben lanzar excepciones cuando el dato es incorrecto debido que luego se hace más difícil consumir dicho método.

Aquellas partes críticas de nuestras clases pueden lanzar excepciones para que sean más robustas. Por ejemplo si tenemos una clase 'ClienteBanco' y queremos controlar que nunca pueda sacar más dinero del que tiene depositado, el método extraer puede lanzar una excepción evitando que quede en negativo el monto del cliente.

Problema:

Declarar una clase llamada 'PersonaAdulta' con los atributos nombre y edad. Lanzar una excepción de tipo IOException en caso que llegue en el constructor una edad menor a 18 años.

En este problema hemos tomado la decisión de validar y lanzar una excepción en caso que se intente la creación de un objeto de la clase PersonaAdulta con una edad inferior a 18. Esto es porque a juicio del programador considera que es muy importante que nunca haya un objeto de tipo PersonaAdulta que sea menor de edad.

Programa:

```
public class PersonaAdulta {
   private String nombre;
   private int edad;

   public PersonaAdulta(String nombre, int edad) throws Exception {
      this.nombre = nombre;
      if (edad < 18)
         throw new Exception("No es adulta la persona " + nombre + " porque tiene " + edad + " años.");
      this.edad = edad;
   }
```

```
   public void fijarEdad(int edad) throws Exception {
      if (edad < 18)
         throw new Exception("No es adulta la persona " + nombre + " porque tiene " + edad + " años.");
      this.edad = edad;
   }

   public void imprimir() {
      System.out.println(nombre + " - " + edad);
   }

   public static void main(String[] ar) {
      try {
         PersonaAdulta persona1 = new PersonaAdulta("Ana", 50);
         persona1.imprimir();
         PersonaAdulta persona2 = new PersonaAdulta("Juan", 13);
         persona2.imprimir();
      } catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
   }
}
```

Para lanzar una excepción debemos utilizar la palabra clave 'throw' y seguidamente la referencia de un objeto de la clase 'Exception' o de una subclase de 'Exception':

```
      if (edad < 18)
         throw new Exception("No es adulta la persona " + nombre +
                        " porque tiene " + edad + " años.");
```

En el método que utilizamos el comando 'throw' debemos enumerar en su declaración luego de la palabra clave 'throws' los nombres de excepciones que puede lanzar dicho método (pueden ser más de uno):

```
   public PersonaAdulta(String nombre, int edad) throws Exception {
```

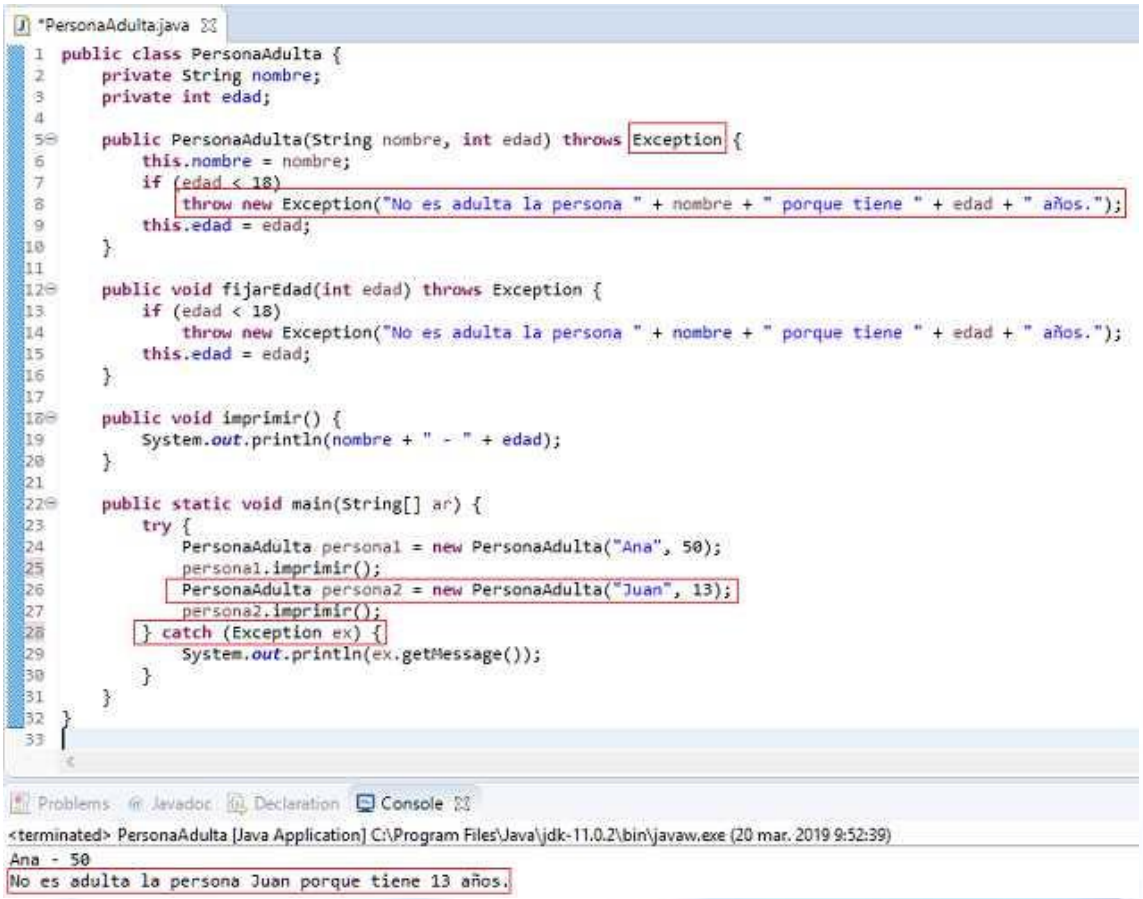
De forma similar el método 'fijarEdad' verifica si la edad que llega es menor a 18 para lanzar la excepción:

```
   public void fijarEdad(int edad) throws Exception {
      if (edad < 18)
         throw new Exception("No es adulta la persona " +
                        nombre + " porque tiene " + edad + " años.");
      this.edad = edad;
   }
```

Ahora cuando creamos objetos de la clase 'PersonaAdulta' debemos capturar obligatoriamente la excepción mediante un bloque try/catch:

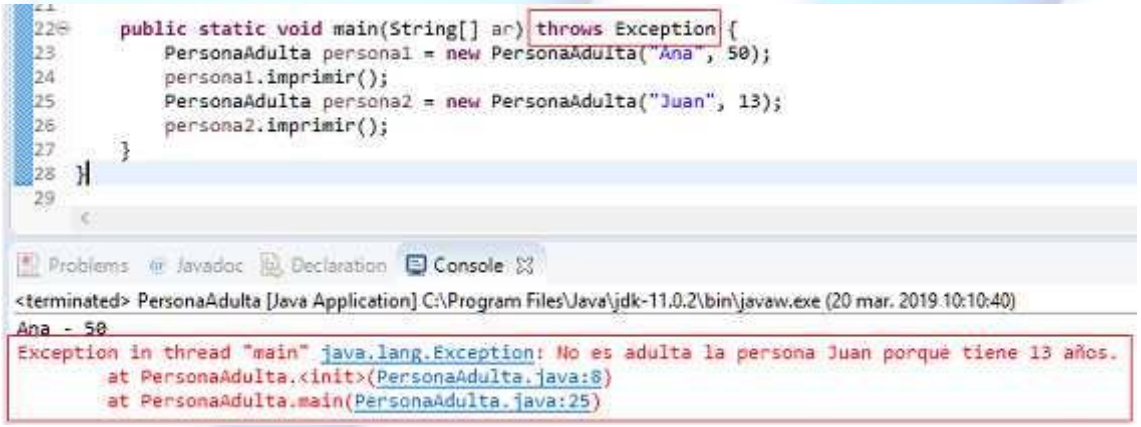
```
   public static void main(String[] ar) {
      try {
         PersonaAdulta persona1 = new PersonaAdulta("Ana", 50);
         persona1.imprimir();
         PersonaAdulta persona2 = new PersonaAdulta("Juan", 13);
         persona2.imprimir();
      } catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
   }
```

Cuando creamos el objeto 'persona2' se captura la excepción debido que "Juan" tiene 13 años:



Cuando se ejecuta el comando throw no se continúan ejecutando las instrucciones del método, sino que se aborta su ejecución y vuelve al método que lo llamó.

Si cuando creamos un objeto de la clase 'PersonaAdulta' el método main omite atraparlas, luego la máquina virtual de Java detiene el programa informando la excepción atrapada:



Problemas propuestos

1. Implementar la clase Operaciones. Debe tener dos atributos enteros. Desarrollar los métodos para calcular su suma, resta, multiplicación y división, imprimir dichos resultados. Lanzar una excepción en el método que calcula la división si el segundo valor es cero.
2. Declarar una clase 'Cliente' que represente un cliente de un banco. Definir los siguientes atributos y métodos:

   Cliente
   - atributos
     - nombre
     - monto
   - métodos
     - constructor
     - depositar
     - extraer
     - imprimir

Generar una excepción si se intenta extraer más dinero del que tiene el atributo 'monto'.

Solución

```
public class Operaciones {
   private int valor1;
   private int valor2;

   public Operaciones(int valor1,int valor2) {
      this.valor1=valor1;
      this.valor2=valor2;
   }

   public void sumar() {
      int suma=valor1+valor2;
      System.out.println("La suma es "+suma);
   }

   public void restar() {
      int resta=valor1-valor2;
      System.out.println("La resta es "+resta);
   }

   public void multiplicar() {
      int producto=valor1*valor2;
      System.out.println("El producto es "+producto);
   }

   public void dividir() throws Exception {
      if (valor2==0)
         throw new Exception("No se puede dividir por cero");
      int division=valor1/valor2;
      System.out.println("La división es "+division);
   }


   public static void main(String[] args) {
      Operaciones operaciones1=new Operaciones(10,0);
      operaciones1.sumar();
      operaciones1.restar();
      operaciones1.multiplicar();
      try {
         operaciones1.dividir();
      } catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
   }

}
```

```
public class Cliente {
   String nombre;
   float monto;

   public Cliente(String nombre, float monto) {
      this.nombre = nombre;
      this.monto = monto;
   }

   public void depositar(float importe) {
      monto = monto + importe;
   }

   public void extraer(float importe) throws Exception {
      if (importe > monto)
         throw new Exception("No se puede extraer más dinero que el depositado.");
      monto = monto - importe;
   }

   public void imprimir() {
      System.out.println(nombre + " tiene " + monto);
   }

   public static void main(String[] args) {
      Cliente cliente1 = new Cliente("Pedro", 10000);
      cliente1.imprimir();
      cliente1.depositar(2000);
      cliente1.imprimir();
      try {
         cliente1.extraer(5000);
         cliente1.imprimir();
         cliente1.extraer(3000);
         cliente1.imprimir();
         cliente1.extraer(70000);
         cliente1.imprimir();
      } catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
   }

}
```

Excepciones propias

La librería o API de Java proporciona gran cantidad de clases que manejan casi cualquier tipo de excepción, pero no estamos obligados a utilizar solo esas clases sino que podemos crear nuestras propias excepciones.

Si queremos crear una excepción no verificada debemos heredar de RuntimeException y si queremos que sea verificada deberemos heredar de Exception o de alguna de sus subclases.

Veamos con un ejemplo como crear una clase que herede de Exception, luego en otra clase lanzar una excepción de la nueva clase creada y finalmente llamar al método que lanza la excepción.

Problema:

Confeccionar una clase que administre una lista tipo pila (se debe poder insertar, extraer e imprimir los datos de la pila).

En el método extraer lanzar una excepción si la pila se encuentra vacía, crear una excepción propia que herede de Exception.

Clase: PilaVaciaException

```
public class PilaVaciaException extends Exception {

   public PilaVaciaException(String mensaje) {
      super("Problema:" + mensaje);
   }

}
```

La clase 'PilaVaciaException' hereda de la clase 'Exception', esto significa que quien lance una excepción de este tipo luego deberá ser capturada en forma obligatoria.

Clase: Pila

```
public class Pila {

   class Nodo {
      int info;
      Nodo sig;
   }

   private Nodo raiz;

   public Pila() {
      raiz = null;
   }

   public void insertar(int x) {
      Nodo nuevo;
      nuevo = new Nodo();
      nuevo.info = x;
      if (raiz == null) {
         nuevo.sig = null;
         raiz = nuevo;
      } else {
         nuevo.sig = raiz;
         raiz = nuevo;
      }
   }

   public int extraer() throws PilaVaciaException {
      if (raiz != null) {
         int informacion = raiz.info;
         raiz = raiz.sig;
         return informacion;
      } else {
         throw new PilaVaciaException("No hay mas elementos en la pila");
      }
   }

   public boolean vacia() {
```

```
        return raiz == null;
    }

    public static void main(String[] ar) {
        Pila pila1 = new Pila();
        pila1.insertar(10);
        pila1.insertar(40);
        pila1.insertar(3);
        try {
            while (!pila1.vacia())
                System.out.println("Extraemos de la pila:" + pila1.extraer());
            System.out.println("Extraemos de la pila:" + pila1.extraer());
        } catch (PilaVaciaException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

El método 'extraer' lanza una excepción de tipo 'PilaVaciaException' en caso que la pila se encuentre vacía:

```
    public int extraer() throws PilaVaciaException {
        if (raiz != null) {
            int informacion = raiz.info;
            raiz = raiz.sig;
            return informacion;
        } else {
            throw new PilaVaciaException("No hay mas elementos en la pila");
        }
    }
```

En la main donde creamos un objeto de la clase 'Pila' insertamos 3 enteros:

```
        Pila pila1 = new Pila();
        pila1.insertar(10);
        pila1.insertar(40);
        pila1.insertar(3);
```

Seguidamente dentro de un bloque try/catch atrapamos si ocurre una excepción de tipo 'PilaVaciaException', esto ocurre cuando finaliza el while y volvemos a llamar al método extraer, siendo que la pila se encuentra vacía:

```
        try {
            while (!pila1.vacia())
                System.out.println("Extraemos de la pila:" + pila1.extraer());
            System.out.println("Extraemos de la pila:" + pila1.extraer());
        } catch (PilaVaciaException ex) {
            System.out.println(ex.getMessage());
        }
```

Tenemos una salida similar a esta:

*PilaVaciaException.java

```
1  public class PilaVaciaException extends Exception {
2
3      public PilaVaciaException(String mensaje) {
4          super("Problema:" + mensaje);
5      }
6  }
7
```

*Pila.java

```
31     public int extraer() throws PilaVaciaException {
32         if (raiz != null) {
33             int informacion = raiz.info;
34             raiz = raiz.sig;
35             return informacion;
36         } else {
37             throw new PilaVaciaException("No hay mas elementos en la pila");
38         }
39     }
40
41
42     public static void main(String[] ar) {
43         Pila pila1 = new Pila();
44         pila1.insertar(10);
45         pila1.insertar(40);
46         pila1.insertar(3);
47         try {
48             while (!pila1.vacia())
49                 System.out.println("Extraemos de la pila:" + pila1.extraer());
50             System.out.println("Extraemos de la pila:" + pila1.extraer());
51         } catch (PilaVaciaException ex) {
52             System.out.println(ex.getMessage());
53         }
54     }
55 }
```

Problems Javadoc Declaration Console

```
<terminated> Pila [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (20 mar. 2019 22:01:55)
Extraemos de la pila:3
Extraemos de la pila:40
Extraemos de la pila:10
Problema:No hay mas elementos en la pila
```

Problema propuesto

1. Plantear una clase llamada 'Termometro' que defina un atributo llamado temperatura. Lanzar una excepción propia llamada 'TemperaturaFueraRangoException' si se intenta fijar una temperatura con un valor menor a -192 o superior a 100.

Solución

```
public class TemperaturaFueraRangoException extends Exception {
   public TemperaturaFueraRangoException(String mensaje) {
      super(mensaje);
   }

}
```

```
public class Termometro {
   private float temperatura;

   public Termometro(float temperatura) throws TemperaturaFueraRangoException {
      if (temperatura < -192 || temperatura > 100)
         throw new TemperaturaFueraRangoException("La temperatura esta fuera del rango [-192,100]");
      this.temperatura = temperatura;
   }

   public void fijarTemperatura(float temperatura) throws TemperaturaFueraRangoException {
      if (temperatura < -192 || temperatura > 100)
         throw new TemperaturaFueraRangoException("La temperatura esta fuera del rango [-192,100]");
      this.temperatura = temperatura;
   }

   public void imprimir() {
      System.out.println("La temperatura actual del termometro es " + temperatura);
   }

   public static void main(String[] args) {
      try {
         Termometro termometro1 = new Termometro(30);
         termometro1.imprimir();
         termometro1.fijarTemperatura(-250);
         termometro1.imprimir();
      } catch (TemperaturaFueraRangoException ex) {
         System.out.println(ex.getMessage());
      }

   }

}
```

Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar info@institutosanisidro.com.ar