Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue

Programación Concurrente -2022
Trabajo Práctico N° 1 Excepciones

Los manejadores de excepción ofrecen la habilidad de controlar la manifestación de errores de código de bajo nivel, de forma que no lleguen al usuario final.
Uno de los problemas con el control de excepciones es saber cuándo y cómo usarlas. Como programadores queremos escribir código de calidad que resuelva problemas.
La premisa básica de un sistema manejador de excepciones es simple: los errores causan excepciones. El código del cliente puede ignorarlas, en cuyo caso la acción por defecto es disparada, o el código del cliente puede manejar la excepción llevando a cabo una acción especial. Los programadores deben designar qué secciones del código serán protegidas de la acción por defecto de una excepción.
El programador puede crear sus propios tipos de excepción extendiendo las clases del lenguaje. Se les puede agregar comportamiento y estado. El programador puede agregar estado a las excepciones por medio de variables instancia. En general, el estado heredado incluye un mensaje de error, pero se podría tener información más detallada.
Los métodos en Java utilizan excepciones para decirle al código llamador "algo malo ocurrió, falle"
Una excepción en Java y otros lenguajes orientados a objetos es un objeto instancia de algún tipo excepción.

RESPONDE:

1. ¿Cuál es el nombre en Java de la Clase que define las excepciones, y de la que debe heredar cualquier clase que queramos usar para representar una excepción?

2. ¿Cuál es el nombre en Java de la clase que representa las excepciones que se producen al invocar un método de un objeto cuyo valor es “null”? ¿y para el caso de obtener un comportamiento anómalo en la entrada/salida de información?

3. ¿Cuál es la peculiaridad de las excepciones del tipo “RuntimeException” (o de las subclases de la misma)?

4. Observa el siguiente fragmento de código:

```
String [] array_string = new String [25];
System.out.println (array_string [3].length());
```

   ¿Qué excepción se produciría en el mismo?

Programación Concurrente -2022
Trabajo Práctico N° 1 Excepciones

5. Observa el siguiente fragmento de código:

```
String aux = “hola”;
int aux2 = Integer.parseInt (aux);
```

¿Qué sucedería al ejecutar el mismo?

6. Escribe el resultado de ejecutar el siguiente fragmento de código:

```
public static double acceso_por_indice (double [] v, int j)
                                    throws RuntimeException{
  try{
     if ((0 <= j) && (j <= v.length)){
          return v[j];
      }else {
            throw new RuntimeException ("El indice " + j + " no existe en el vector");
  }
 } catch (RuntimeException exc){
        throw exc;
 }
}

// Desde el siguiente cliente “main”:
public static void main(String [] args){
     double [] v = new double [15];
     acceso_por_indice (v, 16);
}
```

- Realice una crítica al uso de excepciones en este código y mejore el código para que sea acorde a su crítica.

Programación Concurrente -2022
Trabajo Práctico N° 1 Excepciones

7. Indicar cuál será la salida de la ejecución de cada una de las siguientes clases, justificando su respuesta

a.

```
class Uno{
   private static int metodo(){
      int valor=0;
      try{
         valor = valor +1;
         valor = valor + Integer.parseInt("42");
         valor = valor + 1;
         System.out.println("Valor al final del try: " + valor);
      }catch(NumberFormatException e){
         valor = valor + Integer.parseInt("42");
         System.out.println("Valor al final del catch: " + valor);
      }finally{
         valor = valor + 1;
         System.out.println("Valor al final de finally: " + valor);
      }
      valor = valor + 1;
      System.out.println("Valor antes del return: " + valor);
      return valor;
   }

   public static void main(String[] args){
      try{
         System.out.println(metodo());
      }catch(Exception e){
         System.err.println("Excepcion en metodo()");
         e.printStackTrace();
      }
   }
}
```

Programación Concurrente -2022
Trabajo Práctico N° 1 Excepciones

b.

```
class Dos{
   private static int metodo(){
      int valor=0;
      try{
         valor = valor +1;
         valor = valor + Integer.parseInt("W");
         valor = valor + 1;
         System.out.println("Valor al final del try: " + valor);
      }catch(NumberFormatException e){
         valor = valor + Integer.parseInt("42");
         System.out.println("Valor al final del catch: " + valor);
      }finally{
         valor = valor + 1;
         System.out.println("Valor al final de finally: " + valor);
      }
      valor = valor + 1;
      System.out.println("Valor antes del return: " + valor);
      return valor;
   }

   public static void main(String[] args){
      try{
         System.out.println(metodo());
      }catch(Exception e){
         System.err.println("Excepcion en metodo()");
         e.printStackTrace();
      }
   }
}
```

Programación Concurrente -2022
Trabajo Práctico N° 1 Excepciones

c.

```
class Tres{
   private static int metodo(){
      int valor=0;
      try{
         valor = valor +1;
         valor = valor + Integer.parseInt("W");
         valor = valor + 1;
         System.out.println("Valor al final del try: " + valor);
      }catch(NumberFormatException e){
         valor = valor + Integer.parseInt("W");
         System.out.println("Valor al final del catch: " + valor);
      }finally{
         valor = valor + 1;
         System.out.println("Valor al final de finally: " + valor);
      }
      valor = valor + 1;
      System.out.println("Valor antes del return: " + valor);
      return valor;
   }

   public static void main(String[] args){
      try{
         System.out.println(metodo());
      }catch(Exception e){
         System.err.println("Excepcion en metodo()");
         e.printStackTrace();
      }
   }
}
```

d.

```
import java.io.*;

class Cuatro{
   private static int metodo(){
      int valor=0;
      try{
         valor = valor +1;
         valor = valor + Integer.parseInt("W");
         valor = valor + 1;
         System.out.println("Valor al final del try: " + valor);
         throw new IOException();
      }catch(IOException e){
         valor = valor + Integer.parseInt("42");
         System.out.println("Valor al final del catch: " + valor);
      }finally{
         valor = valor + 1;
         System.out.println("Valor al final de finally: " + valor);
      }
      valor = valor + 1;
      System.out.println("Valor antes del return: " + valor);
      return valor;
   }

   public static void main(String[] args){
      try{
         System.out.println(metodo());
      }catch(Exception e){
         System.err.println("Excepcion en metodo()");
         e.printStackTrace();
      }
   }
}
```

8. Defina una clase “PruebaExcep” que deberá utilizar para mantener los métodos solicitados para los ejercicios siguientes.

   a. Escriba un método que ingrese la edad de una persona y dispare una excepción si la persona es menor de edad.
   b. Escriba un método que ingrese un numero de la ruleta y dispare una excepción cuando al jugar no salga dicho número.
   c. Escriba un método en el que se pida ingresar 5 números a una colección y al mostrarlos, trate de mostrar 7 valores de la misma, generando una excepción.

Programación Concurrente -2022
Trabajo Práctico N° 1 Excepciones

9. Retome el ejercicio del puerto del práctico 1 y defina una clase “ExcepcionDias” que deberá utilizarse al momento de registrar un alquiler cuando la cantidad de días supere un valor límite definido por el puerto.

10. Retome el ejercicio 7 del práctico 4 (Empresa Agroalimentaria) de Prog. Orientada a Objetos y resuelva.

a) Implemente en Java. Defina las clases considerando las correcciones solicitadas.
b) Implemente un método para generar una colección con los productos vencidos. Debe definir y utilizar una excepción “ProductoVencido” que se dispare cuando la fecha de caducidad del producto es menor a la fecha actual. Utilice de manera apropiada el manejador de excepción.

11. Responda verdadero o falso justificando su respuesta

- Un bloque try tiene que ser seguido por un catch y un finally.
- Si escribe un método que podría causar una excepción chequeada, debe encerrar el código de riesgo en un bloque try/catch.
- Solo las excepciones chequeadas pueden ser capturadas.
- Si escribe un método que declara que puede lanzar una excepción chequeada por el compilador, se debe también envolver el código que lanza la excepción en un bloque try/catch
- Un bloque try puede existir por sí mismo, sin un bloque catch o un bloque finally
- Un método con un bloque try y un bloque finally, puede opcionalmente declarar la excepción
- Las excepciones de runtime deben ser manejadas o declaradas.