Instituto de Ciencias Básicas e Ingeniería

Área Académica de Computación y Electrónica

Licenciatura en Ciencias Computacionales

Asignatura:

Programación Orientada a Objetos

Docente: M. en C. Iliana Castillo Pérez

enero - junio 2019

Universidad Autónoma del Estado de Hidalgo

Excepciones y Aserciones

Las ***excepciones*** son un mecanismo utilizado por numerosos lenguajes de programación para determinar lo que debe hacerse cuando ocurre algo inesperado, generalmente algún tipo de error, como la llamada a un método con argumentos no válidos, el fallo de una conexión de red o la solicitud de apertura de un archivo que no existe.

Universidad Autónoma del Estado de Hidalgo

Excepciones y Aserciones

Las ***aserciones*** son una forma de verificar ciertos supuestos sobre la lógica de un programa. Por ejemplo, si cree que, en un determinado punto, el valor de una variable siempre será positivo, una aserción puede comprobar si esto es verdad. **Las aserciones suelen utilizarse para verificar supuestos sobre la lógica local dentro de un método y no para comprobar si se cumplen las expectativas externas.**

Universidad Autónoma del Estado de Hidalgo

Excepciones y Aserciones

Una diferencia importante entre la aserciones y las excepciones, es que las primeras pueden suprimirse enteramente al ejecutar el código. Esto permite habilitar las aserciones durante el desarrollo del programa, pero evitar la ejecución de las pruebas en el tiempo de ejecución, cuando el producto final se entrega al cliente.

Universidad Autónoma del Estado de Hidalgo

Excepciones

Java proporciona dos categorías de excepciones:

- *Comprobadas*: son aquellas que se espera que el programador gestione en el programa y se generan por condiciones externas que pueden afectar a un programa en ejecución (Abrir archivo que no existe, fallo en la red).
- *No comprobadas*: proceden de errores del código (excepciones de tiempo de ejecución) o de situaciones demasiado difíciles para que el programa las maneje. (Intentar acceder a una posición inexistente en un arreglo).

Universidad Autónoma del Estado de Hidalgo

Errores

Las excepciones que derivan de problemas del entorno de los que es difícil recuperarse se denominan *errores*. En esta categoría se incluye el agotamiento de la memoria. Los errores también son excepciones no comprobadas.

Universidad Autónoma del Estado de Hidalgo

La clase `Exception`

La clase **`Exception`** es la clase básica para representar las excepciones comprobadas y no comprobadas. En lugar de finalizar la ejecución del programa, es mejor escribir código para manejar la excepción y continuar.

Universidad Autónoma del Estado de Hidalgo

La clase `Error`

La clase **`Error`** es la clase básica utilizada para condiciones de error graves, no comprobadas, de las que es de esperar que el programa no intente recuperarse. En la mayoría de los casos, debería dejar que finalice la ejecución del programa al encontrar este tipo de errores, aunque puede tratar de preservar la mayor parte del trabajo realizado por el usuario.

Universidad Autónoma del Estado de Hidalgo

La clase `RuntimeException`

La clase **`RuntimeException`** es la clase básica utilizada para las excepciones no comprobadas que pueden derivarse de los defectos del programa. En la mayoría de los casos, debería dejar finalizar el programa cuando se produce una excepción de este tipo. De nuevo, es aconsejable tratar de preservar la mayor parte posible del trabajo del usuario.

Universidad Autónoma del Estado de Hidalgo

Clases derivadas de `Exception`

El conjunto de clases derivadas de la clase java.Object.Throwable.Exception, representan el conjunto básico de errores que pueden aparecer al desarrollar cualquier programa:

- `AWTException`. Indica una excepción sobre elementos derivados del paquete *java.awt*, utilizado para implementar aplicaciones gráficas.
- `ClassNotFoundException`. Error al tratar de utilizar una clase.

Universidad Autónoma del Estado de Hidalgo

Clases derivadas de `Exception`

- `DataFormatException`. Error en el formato de los datos.
- `IllegalAccessException`. Se intenta acceder a una clase a la que no se tiene permiso.
- `IOException`. Excepciones producidas al realizar tareas de entrada/salida, las más utilizadas son:
  - ✓ `EOFException`
  - ✓ `FileNotFoundException`
  - ✓ `MalformedURLException`
  - ✓ `SocketException`
  - ✓ `UnknownHostException`
  - ✓ `UTFDataFormatException`

Universidad Autónoma del Estado de Hidalgo

Clases derivadas de `Exception`

- `NoSuchFieldException`. No se encuentra un determinado atributo.
- `NoSuchMethodException`. No se encuentra un determinado método.
- `RuntimeException`. Errores producidos en tiempo de ejecución. Son las más utilizadas y entre ellas están:
  - ✓ `ArithmeticException`
  - ✓ `ClassCastException`
  - ✓ `IndexOutOfBoundsException`
  - ✓ `NegativeArraySizeException`
  - ✓ `NullPointerException`

Universidad Autónoma del Estado de Hidalgo

Sentencias try/catch/finally

Estas sentencias definen tres bloques de código que pueden capturar y resolver un problema aparecido durante la ejecución de un método considerando lo siguiente:

1. En un bloque *try* se ***intenta*** ejecutar un código que podría generar una o varias excepciones.
2. La excepción (si se produce) se ***captura*** por el bloque de código *catch* (uno o varios).
3. ***Finalmente***, las operaciones que deban ser siempre ejecutadas antes de salir del método estarán dentro del bloque *finally*.

Ejemplo 1

```
public class AddArguments {
   public static void main(String args[]) {
     try{
      int sum = 0;
      for(int i=0; i<args.length; i++){
         sum += Integer.parseInt(args[i]);
      }
      System.out.println("Suma = " + sum);
     }catch(NumberFormatException e){
        System.err.println("Uno de los argumentos de la "
             +"línea de comandos no es un entero.");
     }
   }
}
```

Este programa captura la excepción y luego se cierra con un mensaje de error.

Ejemplo 2

```
public class AddArguments {
   public static void main(String args[]) {
      int sum = 0;
      for(int i=0; i<args.length; i++){
       try{
          sum += Integer.parseInt(args[i]);
       }catch(NumberFormatException e){
        System.err.println("["+args[i]+"] no es un entero"
                          +" y no se incluirá en la suma.");
       }
      }
      System.out.println("Suma = " + sum);
   }
}
```

Este programa detecta la excepción de cada argumento no entero y genera un mensaje de aviso, pero produce la suma de los enteros válidos.

Universidad Autónoma del Estado de Hidalgo

Múltiples `catch`

Se pueden utilizar varios bloques catch para un solo bloque try debido a que éste último puede contener código que origine diversos tipos de problemas. Java recorre secuencialmente los bloques de catch que existan buscando el primero que sea el más adecuado para gestionar el error producido.

Debido a lo anterior, se debe tener mucho cuidado en el orden en el que se sitúan los bloques catch, colocando los más generales después de los específicos.

Ejemplo

```
public class MultiplesCatch {
   public static void main(String args[]) {
      int a;
      int b[];
      int c = args.length;
      try{
        a = 10/c;   //error si no. de argumentos=0
        System.out.println("Valor de a = " + a);
        b = new int[5];
        for(int i=0; i<a; i++){//si c=1, se produce error
          b[i] = i;
          System.out.print("\t" + i); }
       }catch(ArithmeticException e){
        System.out.println("Error, división por cero: " +e);
      } catch(ArrayIndexOutOfBoundsException e){
       System.out.println("Índice fuera de límites: "+e);
      }
} }
```

Universidad Autónoma del Estado de Hidalgo

El bloque finally

finally se utiliza cuando el programador solicita ciertos recursos al sistema que se deben liberar, y se coloca después del último bloque catch.

El código contenido en un bloque *finally* se ejecutará tras terminar el bloque try, haya habido o no una excepción, lo que permite liberar recursos o finalizar procesos importantes [2].

Ejemplo

```
public class BloqueFinally {
   public static void main(String args[]) {
      FileReader lector = null;
      try{
        lector = new FileReader("archivo.txt");
        int i = 0;
        while(i != -1){
          i = lector.read();
          System.out.println((char) i);
        }
       }catch(IOException e){
        System.out.println("Error");
      }finally{
         if(lector != null)
            lector.close();
      }
    }
}
```

Universidad Autónoma del Estado de Hidalgo

Referencias

Sun Microsystems (2008), El lenguaje de programación Java, Sun Microsystems.

Alcalde, A. (2017), Manejar excepciones en Java, consultado de: El Baúl del Programador en la dirección: https://elbauldelprogramador.com/manejar-excepciones-en-java/