Grado en Ingeniería Informática
Grado en Matemáticas e Informática

Asignatura: PROGRAMACIÓN II

Excepciones

Ángel Lucas González Martínez
Jaime Ramírez Rodríguez
Clara Benac Earle

DLSIIS - E.T.S. de Ingenieros Informáticos
Universidad Politécnica de Madrid

Octubre 2013

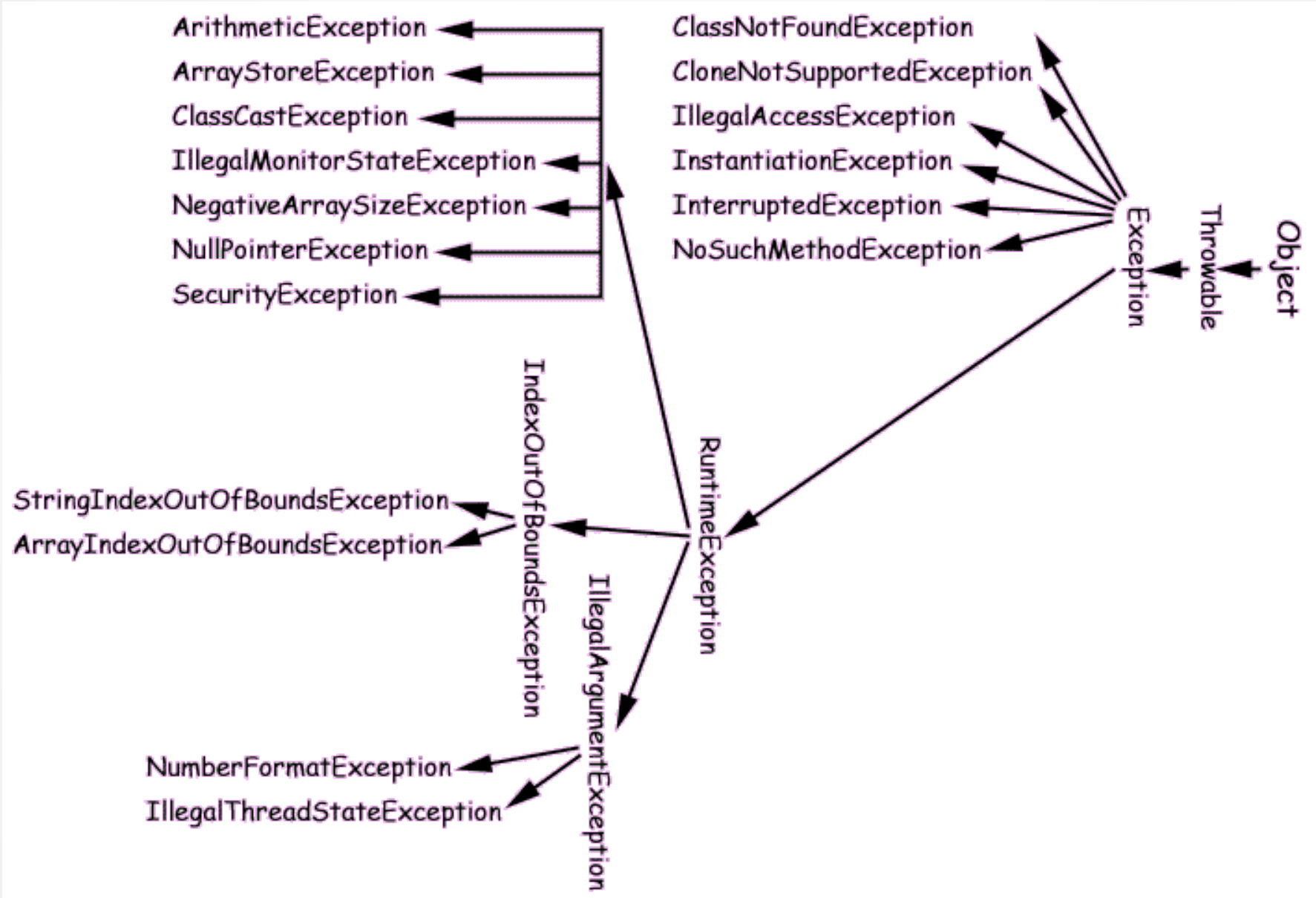
Excepción

- Notificar una situación anómala
- Se presenta durante la ejecución de un programa
- Ejemplos:
  - Acceso a una posición de un array que no existe
  - División por cero
  - Agotamiento de la memoria
  - Fichero no encontrado
  - Problemas al construir una instancia de una clase
  - Conexión por red no lograda, etc.

Tipos de excepciones en Java

- Predefinidas del lenguaje Java
  - No hace falta declararlas en las cabeceras o capturarlas
  - Ejemplos: Véase la siguiente transparencia
- Definidas en la API de Java
  - Por ejemplo: *FileNotFoundException, ConnectException,* etc.
- Definidas por el programador
  - Se pueden lanzar cuando no se cumple la **precondición** de un método: P.ej. el factorial recibe un número **negativo**.
  - Se pretende construir un objeto con datos de entrada no válidos: P.ej. el concesionario recibe un número de coches negativo en el constructor

Jerarquía de excepciones predefinidas



¿Qué ocurre cuando se produce una excepción?

```
public class Ejemplo {

  public int factorial (int num) {
    if ( num == 0 || num == 1){
      return 1;
    } else {
      return num * factorial (num -1);
    }
  }

  public static void main(String[] args) {
    Ejemplo ejemplo = new Ejemplo();
    System.out.println("Factorial de -1: "+ejemplo.factorial(-1));
}
```

Al ejecutar este código se produce la excepción:

java.lang.StackOverflowError

porque se llama recursivamente al método factorial con números negativos hasta que se agota la memoria asignada a la pila de llamadas.

Generar excepciones

- El programador sabe que el factorial no está definido para el caso en que se llame con un número negativo y puede generar o lanzar una excepción para ese caso

```
/**
 * Calcula el factorial de un número
 * <br><B>PRE:</B> num >= 0
 * <br><B>POST:</B> resultado = num!
 * @param num
 * @return num!
 */
 public int factorial(int num) {
     if ( num < 0) {
         <<generar excepción>>
     } else if ( num == 0 || num == 1){
         return 1;
     } else
        return num * factorial (num -1);
     }
 }
```

¿Cómo se genera una excepción?

- En Java **las excepciones** también **son objetos**
- Hay que definir una clase para cada tipo de excepción que queramos generar
- Se debe definir una nueva clase como subclase (clase hija) de la clase Exception

```
class MiExcepcion extends Exception {
    public MiExcepcion() { }
    // si se quiere mostrar un cierto mensaje
    // se debe definir este segundo constructor
    public MiExcepcion(String msg)  {
          super(msg);
    }
}
```

Algunos servicios de Exception

- Servicios proporcionados por la clase Exception:
  - **Exception**(String message)
  - String getMessage()
  - **void** printStackTrace()
  - **void** printStackTrace(PrintStream s)

Ejemplo de clase de excepciones

- Para el ejemplo del factorial

```
class ExcepcionNNegativo extends Exception {
    public ExcepcionNNegativo() { }
    // si se quiere mostrar un cierto mensaje
    // se debe definir este segundo constructor
    public ExcepcionNNegativo(String msg)  {
          super(msg);
    }
}
```

¿Cómo se genera una excepción?

- Para generar una excepción:
  - Crear la clase para crear objetos excepción (ver transparencias anteriores)
  - Generar o lanzar la excepción desde un método
- El método que genera la excepción tiene que:
  - Declarar que lanza una excepción añadiendo el siguiente código en su cabecera
    - `throws MiExcepcion`
  - Lanzar la excepción, creando el objeto excepción
    - `throw new MiExcepcion(`*mensaje*`)`

Ejemplo de generar excepciones

```
/**
 * Calcula el factorial de un número
 * <br><B>PRE:</B> num >= 0
 * <br><B>POST:</B> resultado = num!
 * @param num
 * @return num!
 */
 public int factorial(int num) throws ExcepcionNNegativo {
   if ( num < 0) {
     throw new ExcepcionNNegativo("no admite números negativos");
   } else if ( num == 0 || num == 1){
      return 1;
   } else{
      return num * factorial (num -1);
   }
 }
```

¿Qué ocurre cuando se produce una excepción?

```
public static void main(String[] args) throws ExcepcionNNegativo {
   Ejemplo ejemplo = new Ejemplo();
   System.out.println("Factorial de "+" 5: "+ejemplo.factorial(5));
   System.out.println("Factorial de "+" 3: "+ejemplo.factorial(3));
   System.out.println("Factorial de "+" -1:"+ejemplo.factorial(-1));
   System.out.println("Factorial de "+" 2: "+ejemplo.factorial(2));
   System.out.println("Factorial de "+" 4: "+ejemplo.factorial(4));
}
```

¿Se imprime el factorial de 5 por la consola? ¿y el de 2? ¿y el de 4?

Manejo de excepciones

- En Java existe un mecanismo que permite que el código se pueda seguir ejecutando aunque se haya producido una excepción
- Este mecanismo se llama **try-catch**
  - **try** {*código que puede generar una excepción*}
  - **catch** (TipoExcepcion e){*código que se ejecuta si se produjo la excepcion*}

Ejemplo

```
public static void main(String[] args) {
  try {
    System.out.println("Factorial de -1: " + ejemplo.factorial(-1));
  } catch(ExcepcionNNegativo e){
    System.out.println("No hay factorial de un número negativo");
  }
  System.out.println("Factorial de 5: " + ejemplo.factorial(5));
}
```

¿Se imprime el factorial de 5 por la consola?

Propagación de excepciones

- Supongamos el siguiente ejemplo:
  - Un método `main()` que solicita que el usuario introduzca un número entero por la consola e inmediatamente llama a un método `calcularFactorial()` que devuelve el factorial de ese número
  - El método `calcularFactorial()` lee el número que el usuario ha introducido por la consola y llama al método `factorial` con dicho número
- ¿Qué ocurre si el número con el que se llama al método `factorial()` es un número negativo?

Manejo de excepciones

- El manejo de excepciones permite controlar y dar respuestas a situaciones anómalas
  - Mejor que devolver un código de error
- Separación:
  - Código que resuelve un problema (**try**)
  - Código que trata situaciones especiales (**catch**)
  - Código que se ejecuta en cualquier caso (**finally**)
- Para generar una excepción se utiliza **throw**
- Cuando un método o constructor puede generar una excepción debe indicarlo en su especificación:

```
public int read() throws IOException
```

Manejo de una excepción

- Supongamos que en una función **f** de Java se detecta una situación anómala, entonces:
  - Se ejecuta:

```
throw new MiException([msg]);
```

  - Y para que pueda ser capturada:

```
try {  // bloque try
   f(...);  // u otra función que llame a f()
}.....
```

Manejo de una excepción

- El tratamiento de la excepción se especifica mediante una sentencia **catch**:

```
try {
    f(...) // posible fuente de la excepción
    ........
} catch (tipo_excepcion1 parámetro1) {
    "tratamiento de la excepción1"
} catch (tipo_excepcion2 parámetro2) {
    ..................................
} catch (tipo_excepcionN parámetroN) {
    "tratamiento de la excepciónN"
} [ finally {
    "bloque que se ejecuta siempre"
} ]
```

Manejo de una excepción

- Tras ser lanzada una excepción el proceso es:
  1. Se busca hacia abajo un **catch** en el mismo nivel que el **try** que encierra la llamada a **f**
  2. Si no hay ninguno que sirva, se busca en un posible **try** que encierre al **try** anterior, y así sucesivamente hasta que se encuentre un **try** con un **catch** apropiado
  3. Si se llega al **main()** y no hay ninguno que sirva, se detiene la ejecución con un mensaje de error
- Si es tratada, la ejecución continúa con la siguiente sentencia al bloque **try-catch** que trató la excepción.

Declaración y tratamiento de una excepción

```
public class MiExcepcion extends Exception {
   MiExcepcion(String sMensaje) {
      super (sMensaje);
   }//Constructor
}
```

```
public class Ejemplo {
   private int nMax;

   Ejemplo(int nMaximo) {
      nMax=nMaximo;
   }//Ejemplo

   void generarExcepcion(int nValor) throws MiExcepcion {
      if (nValor>=nMax) {
         throw new MiExcepcion (
           "El valor:" + nValor + " supera el rango");
      }
   }//generarExcepcion

... continúa en la siguiente transparencia ...
```

Declaración y tratamiento de una excepción

```
   static public void main(String args[]){
      Ejemplo Ej=new Ejemplo(4);
      for (int i=0;i<6;i++) {
         try {//try
            Ej.generarExcepcion(i); //Uso de métodos que pueden
                                    //producir excepción
         } catch (MiExcepcion e) { //Tratamiento de excepciones
            System.out.println (e.getMessage());
         } finally { //Código que siempre se ejecuta
            System.out.println("El valor del contador en esta"
                               + " iteración es: " + i);
         }//try-catch-finally
      }//for
   }//main
}//class Ejemplo
```

Ejercicio

- Dada una clase `Cuenta` con los siguientes métodos:
  - Atributos: `saldo`, `gastosApertura` (el mismo para todas las cuentas, 10€)
  - Constructor (*`cliente, saldo inicial`*): resta gastos de apertura
  - `void sacarDinero(`*`cantidad`*`)`
  - `void ingresarDinero(`*`cantidad`*`)`
- Crear una excepción `SaldoInicialInsuficiente`
  - lanzarla desde el constructor si el saldo inicial es menor que los gastos de apertura
- Crear una excepción `SaldoInsuficiente`
  - lanzarla desde `sacarDinero()` si el saldo es menor que la cantidad que se desea retirar.