Tema 14: Excepciones

Excepción

- Definición

- Las Excepciones son objetos que definen el estado de la aplicación cuando se producen situaciones anómalas.

- Son un mecanismo usado por muchos lenguajes de programación para describir que hacer cuando inesperado sucede.

Tipos de excepciones

- Exceptions
  - ✓Checked
    - – Son aquellas que ocurren de manera externa cuando el programa esta trabajando. Por ejemplo al intentar accesar a un archivo que no es localizado o la falla de red.
  - ✓Unchecked
    - – Runtime exceptions.

      Por ejemplo el intentar acceder a una localidad mas alla de la del fin del array (errores en diseño e implementación.
    - – Errors

      Son aquellas excepciones que son raras y difíciles de recuperarse de ellas como lo es “correr” fuera del rango de memoria.

Jerarquía de clases

- **java.lang.Object**

```
+----java.lang.Throwable
        |
        +----java.lang.Exception
        |       |
        |       +----java.lang.RuntimeException
        |       |       |
        |       |       +----java.lang.ArithmeticException
        |       |       |...
        |       |       +----java.lang.NullPointerException
        |       |  . . .
        |       +----java.lang.IOException
        |               |
        |               +----java.lang.EOFException
        |               |. . .
        |              +----java.lang.FileNotFoundException
        +----java.lang.Error
                |  (uncheked)
                +----java.lang.VirtualMachineError
                |   . . .
                +----java.lang.AssertionError
```

Excepciones en tiempo de ejecución

- **java.lang.RuntimeException**

  java.lang.NullPointerException

  //se ejecuta "x.miembro" para una variable x con valor null.

  java.lang.IndexOutOfBoundsException

  //acceso a un índice no existente en un array.

  java.lang.ArithmeticException

  //división por cero

  java.lang.ClassCastException

  //error de casting.

Bloques try-catch y finally

- El bloque try-catch-finally permite capturar una excepción.
- *finally* es opcional. Pero siempre se ejecutará exista o no excepción.
- Si hay varias partes *catch* se ejecuta la primera posible. El “catch” utiliza la clase dinámica.
- Es posible uso de *try-finally* (sin catch), aunque no se puede usar “try” solo.

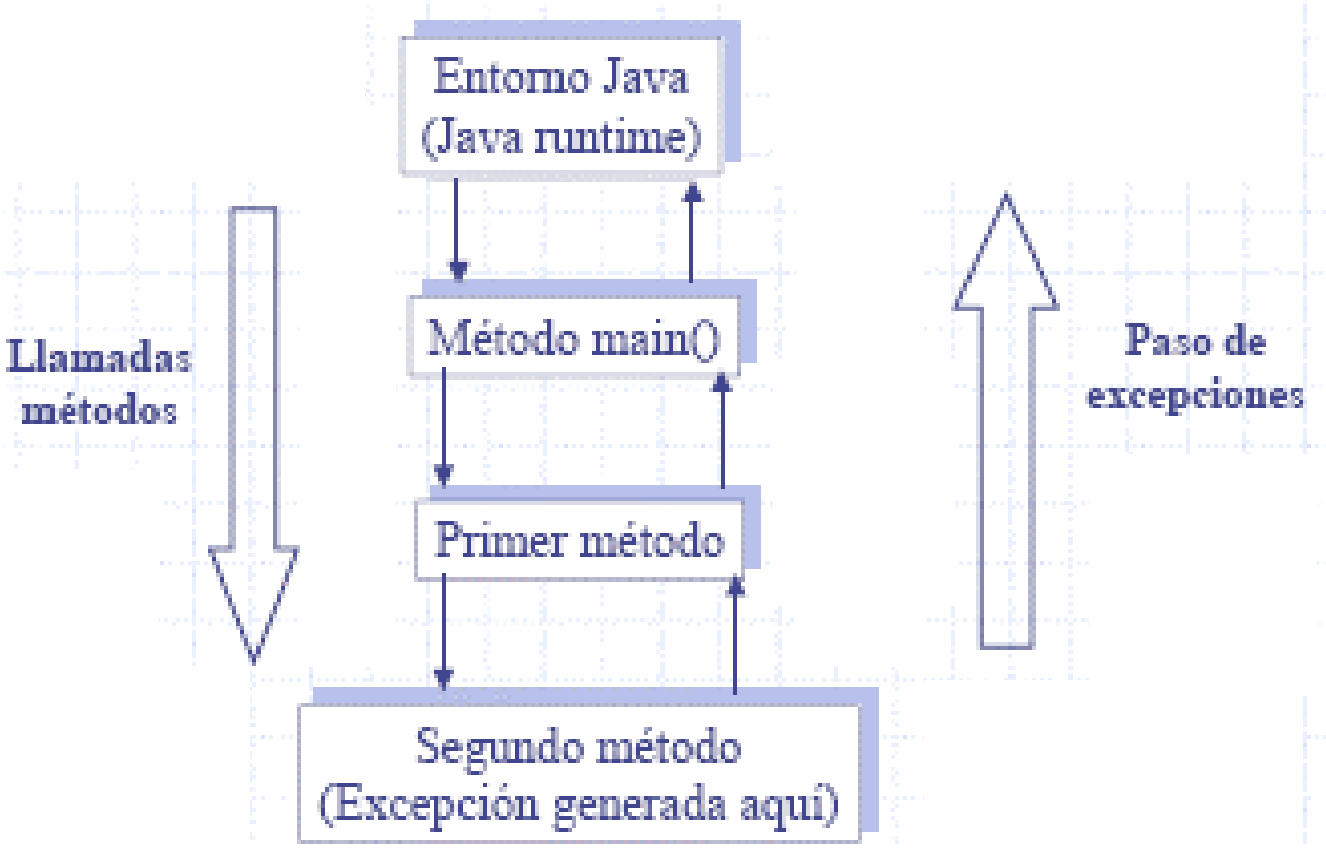
Bloques try-catch y finally

- Sintaxis

```
try{
   //Acciones susceptibles a excepción
}catch(Excepcion1 var1) {
   // acción cuando se atrapa la
   excepción
} catch(Excepcion2 var2) {
   // acción cuando se atrapa la
   excepción de mayor jerarquía
}
finally{
    // acción que se cumple ocurra o no
   una excepción
}
```

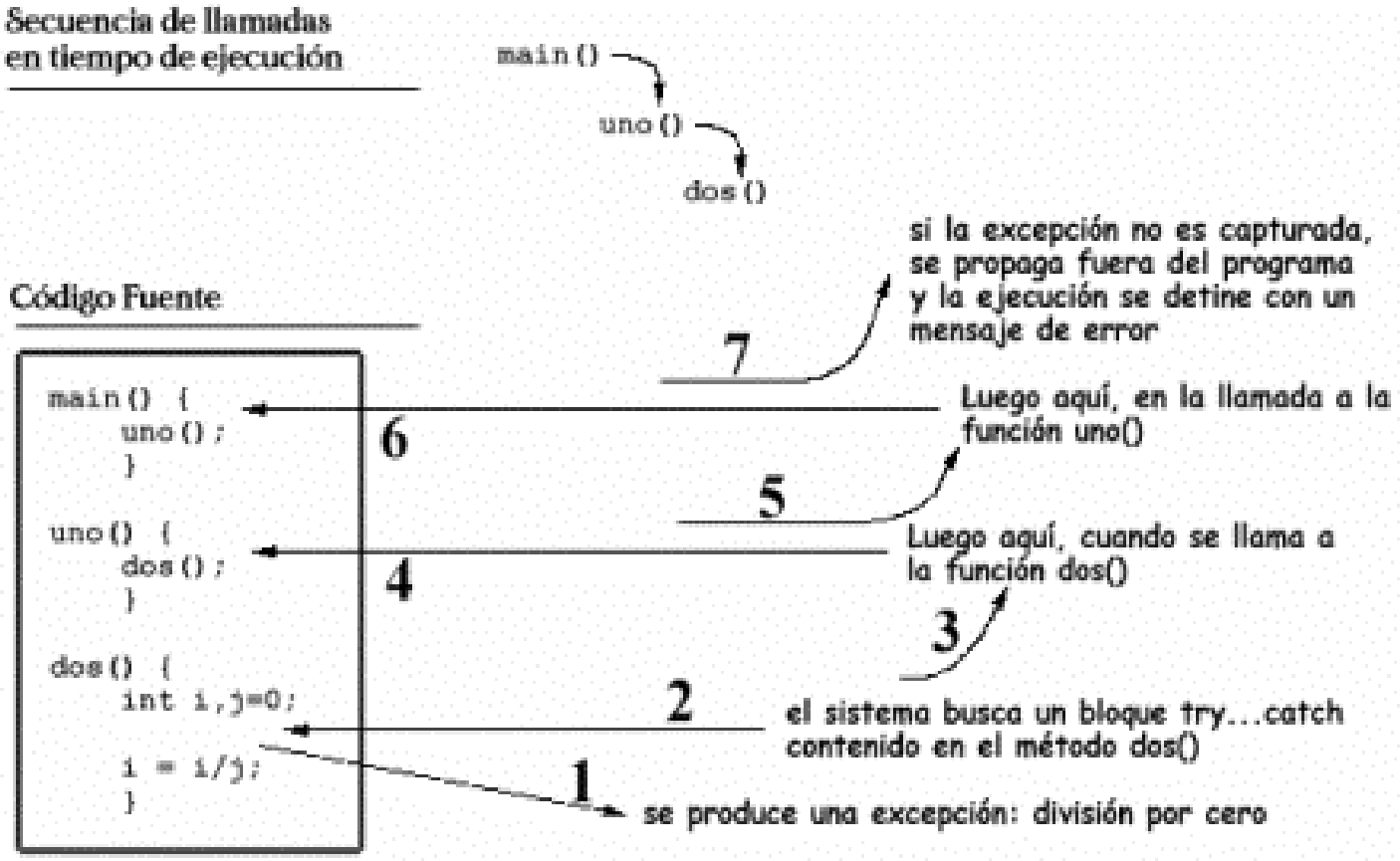
Propagación de excepciones

- Si ningún método hasta el main atrapa una excepción “e”, se llama a e.printStackTrace() (método de java.lang.Throwable)



Propagación de excepciones

- .



Atrapando excepciones

Ejemplo:

```
public class TestExcepciones{
    public static void main(String arg[]){
        int datos[] = new int[5];
        try{
            datos[6] = 7;
          }
      catch(ArrayIndexOutOfBoundsException
   e){
        System.out.println("Indice fuera de
   limites: ");
       }

      System.out.println("Sigue por aqui");
     }
 }
```

Atrapando excepciones 2

Ejemplo:

```
class Ejemplo {
    public static void main(String args[]) {
        Ejemplo p = new Ejemplo();
        p.g();
        System.out.println("Final de método
    main");
   }
    void f (String s) {
        System.out.println(s.toString());
        System.out.println("Final de método f");
     }
```

... Ejemplo

```
  void g () {
       try {
           System.out.println("Antes de f(null)");
           f(null);
           System.out.println("Despues de f(null)");
       } catch (NullPointerException e) {
              System.out.println("Excepcion de apuntadc
  nulo");
       } catch (IndexOutOfBoundsException e){
          System.out.println("Excepcion de indice
  erroneo");
       }
       finally {
       System.out.println("Final de bloque try");
       }
       System.out.println("Final de método g");
}
 } // de la clase Ejemplo
```

Ventajas

- Beneficios (errores y/o situaciones anómalas):
  1) Separación entre errores y código normal
  2) Propagación de errores en la pila de métodos a ejecutar.
  3) Agrupación de tipos de errores y diferenciación

- Se evita la alternativa en programación del uso de muchos if o if's anidados

Excepciones definidas en un programa

- Se aconseja su uso para situaciones de prueba y depuración.
- Ventajas:
  - ✓ Permiten documentar mejor el programa.
  - ✓ Permiten discriminar por medio de varios bloques catch consecutivos.
  - ✓ Permiten transportar datos adicionales desde donde se crean a donde se capturan.
- Lo mejor es crearlas como excepciones comprobadas (heredan de Exception pero no de RunTimeException)

throws

- **throws** permite cualquier objeto “Throwable”, aunque generamente se usa sólo para las Exception.
- Evitar errores de compilación:
  - ✓ Cuando un método puede lanzar una excepción comprobada se debe especificar en su cabecera con la keyword throws.
  - ✓ Ejemplo:

```
void metodoUno throws IOException{
// acciones
}
```

throw

- **throw** permite indicar el lanzamiento de una excepción creada por el programador.

- La condición necesaria cuando se crea una excepción propia, es que esta clase herede de Exception

✓Ejemplo:

```
class miExcepcion extends Exception{
// acciones
  void metodoUno throws miExcepcion{
// acciones
 if ( ! exito) {
        throw  new miException("Error en
   acceso al dato");
      }
  }
}
```

Ejemplo

```
class Exception2D extends Exception {
    public Exception2D (String mensaje) {
        super(mensaje);
    }
}
// Si algún método (como el main) crea Rectas, deberá
// o bien hacer throws de Exception2D o bien
// atrapar la excepción con bloque try-catch
class Recta2D {
    private Punto2D p1, p2;
    private void crearRecta2D (Punto2D p1, Punto2D p2) {
            this.p1=p1;
            this.p2=p2;         }
    // Recta a partir de dos puntos
    public Recta2D (Punto2D p1, Punto2D p2)
    throws Exception2D {
        if (p1.equals(p2)) // definido en clase Punto2D
            throw new Exception2D("Una recta se crea con”
                                    + “dos puntos distintos");
        else crearRecta2D(p1, p2);
    }
```

...Ejemplo

```
// Para una recta ax+by+c=0
  public Recta2D (double a, double b, double c) throws
   Exception2D {
      if (a==0 && b==0)
          throw new Exception2D("En una recta no puede ”
                                          + “ser a=b=0");
      else if (a==0)
          crearRecta2D(new Punto2D(0, -c/b),
                              new Punto2D(1, -c/b));
      else if (b==0)
          crearRecta2D(new Punto2D(-c/a, 0),
                              new Punto2D(-c/a, 1));
      else
          crearRecta2D(new Punto2D(0, -c/b),
                              new Punto2D(-c/a, 0));
  }
  public Punto2D interseccion (Recta2D recta) throws
   Exception2D {
       .... throw new Exception2D(“Rectas paralelas o “
                                          + “coincidentes”);
```