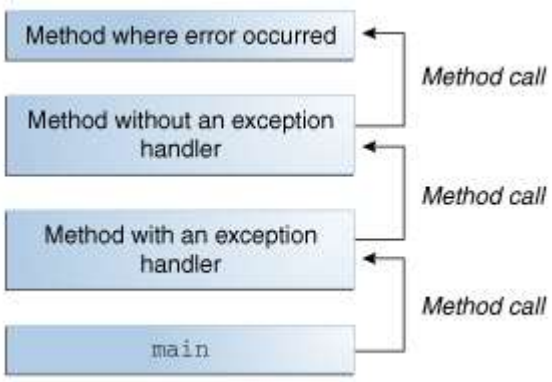
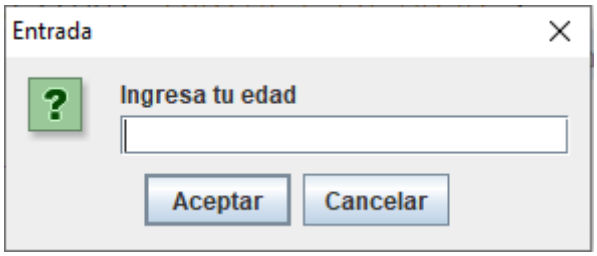
Excepción

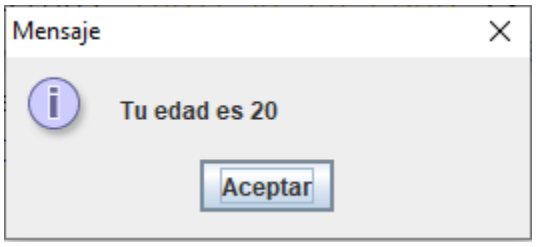
Una *excepción* es un evento, que ocurre durante la ejecución de un programa, que interrumpe el flujo normal de las instrucciones del programa.
Cuando ocurre un error dentro de un método, el método crea un objeto y lo pasa al sistema de tiempo de ejecución. El objeto, llamado *objeto de excepción* , contiene información sobre el error, incluido su tipo y el estado del programa cuando ocurrió el error. Crear un objeto de excepción y entregarlo al sistema de tiempo de ejecución se denomina *lanzar una excepción* .



Ejemplo de NumberFormatException

```
package excepciones;

import javax.swing.JOptionPane;

/**
 * Descripción: Programa que realiza el manejo de Excepciones.
 * @author Gabriel Barrón Rodríguez
 * @version 1.0
 */
public class Excepciones {

   public static void main(String[] args) {
      int edad;
      String strEdad; //Almacena edad en formato String

      //Almacena la edad como cadea
      strEdad = JOptionPane.showInputDialog("Ingresa tu edad");
      edad = Integer.valueOf(strEdad); //Convierte de String a Entero
      String mensaje = "Tu edad es " + edad;//Genera el mensaje
      JOptionPane.showMessageDialog(null, mensaje);

   }

}
```





Excepciones Comunes en Java

Nombre	Descripción
NullPointerException	Se lanza cuando se intenta usar una referencia nula en el uso de un objeto. Es una de las excepciones más comunes y es debido a un error en la programación del programa
ArrayIndexOutOfBounds	se lanza al acceder a una posición ilegal de un array al ser el índice negativo, mayor o igual que el tamaño del array.
ClassCastException	se lanza cuando se hace una operación de cast a un tipo de objeto de la que no es una instancia
IllegalArgumentException	se lanza para indicar que el valor de un argumento es inválido, se suele utilizar para comprobar una precondición al inicio del método
IOException	indica algún tipo de error en una operación de entrada/salida. El error de entrada/salida es posible al trabajar con archivos o con operaciones de red
FileNotFoundException	es un tipo de error de entrada/salida que indica que al abrir el archivo este no existe en el sistema de archivos
ClassNotFoundException	se lanza cuando se intenta cargar una clase, pero esta no existe usando forName, findSystemClass o loadClass. Su causa suele ser porque una librería no se ha proporcionado en el classpath al iniciar el programa o la aplicación requiere otra versión de alguna librería
SQLException	se produce en el acceso a bases de datos relacionales, por ejemplo cuando la sentencia SQL no tiene una sintaxis correcta
StackOverflowError	se produce cuando se hacen demasiadas llamadas recursivas a un método
InterruptedException	se produce cuando un thread es interrumpido. Esto ocurre al trabajar en Java con hilos y operaciones de concurrencia

Try-Catch

El código de lenguaje de programación Java válido debe cumplir con el requisito de captura o especificación. Esto significa que el código que podría arrojar ciertas excepciones debe incluirse en cualquiera de los siguientes:

- Una declaración **try** que captura la excepción. Debe proporcionar un controlador para la excepción.
- Un método que especifica que puede generar la excepción. El método debe proporcionar una cláusula **throws** que enumere la excepción, como se describe en la especificación de las excepciones lanzadas por un método .

El código que no cumpla con el requisito de captura o especificación no se compilará.
No todas las excepciones están sujetas al requisito de captura o especificación. Para entender por qué, debemos observar las tres categorías básicas de excepciones, de las cuales solo una está sujeta al Requisito.

Tres tipos de excepciones

El primer tipo de excepción es la excepción comprobada .

Estas son condiciones excepcionales que una solicitud bien escrita debe anticipar y recuperarse. Por ejemplo, suponga que una aplicación solicita al usuario un nombre de archivo de entrada y luego abre el archivo pasando el nombre al constructor para java.io.FileReader. Normalmente, el usuario proporciona el nombre de un archivo legible existente, por lo que la construcción del FileReader tiene éxito y la ejecución de la aplicación continúa con normalidad. Pero a veces el usuario proporciona el nombre de un archivo inexistente y el constructor lanza java.io.FileNotFoundException. Un programa bien escrito detectará esta excepción y notificará al usuario del error, posiblemente solicitando un nombre de archivo corregido.

Las excepciones marcadas están sujetas al requisito de captura o especificación. Todas las excepciones son excepciones comprobadas, excepto las indicadas por Error, RuntimeException y sus subclases.

Ejemplo Excepción Comprobada

```
package excepciones;

import javax.swing.JOptionPane;

/**
 * Descripción: Programa que realiza el manejo de Excepciones.
 * @author Gabriel Barrón Rodríguez
 * @version 1.0
 */
public class FormatoNumeroExcepcion {

   public static void main(String[] args) {
      int edad;
      String strEdad; //Almacena edad en formato String

      //Almacena la edad como cadea
      strEdad = JOptionPane.showInputDialog("Ingresa tu edad");
      //Bloque TRY vigila el código para formato de número
      try {
         edad = Integer.valueOf(strEdad); //Convierte de String a Entero

      } catch (NumberFormatException e) {
         JOptionPane.showMessageDialog(null, "Dato Incorrecto");
         return;
      }
      String mensaje = "Tu edad es " + edad;//Genera el mensaje
      JOptionPane.showMessageDialog(null, mensaje);
   }
}
```

El segundo tipo de excepción es el error.

Estas son condiciones excepcionales que son externas a la aplicación y que, por lo general, la aplicación no puede anticipar o recuperar. Por ejemplo, suponga que una aplicación abre con éxito un archivo para la entrada, pero no puede leer el archivo debido a un mal funcionamiento del hardware o del sistema. La lectura fallida arrojará java.io.IOError. Una aplicación puede optar por capturar esta excepción para notificar al usuario sobre el problema, pero también puede tener sentido que el programa imprima un seguimiento de la pila y salga.

Los errores no están sujetos al requisito de captura o especificación. Los errores son aquellas excepciones indicadas por Error y sus subclases.

Manejo de Excepciones

Clase Estudiante

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package utng;

/**
 *
 * @author Usuario
 */
public class Estudiante {
   private String numControl;
   private String nombre;
   private String apellidos;
   private int edad;
   private String email;

   public Estudiante() {
   }

   public Estudiante(String numControl, String nombre, String apellidos, int edad, String
email) {
      this.numControl = numControl;
      this.nombre = nombre;
      this.apellidos = apellidos;
      this.edad = edad;
      this.email = email;
   }

   public String getNumControl() {
      return numControl;
   }

   public void setNumControl(String numControl) {
      this.numControl = numControl;
   }

   public String getNombre() {
      return nombre;
   }

   public void setNombre(String nombre) {
      this.nombre = nombre;
   }
```

```
    public String getApellidos() {
        return apellidos;
    }

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Estudiante{" + "numControl=" + numControl + ", nombre=" + nombre + ",
apellidos=" + apellidos + ", edad=" + edad + ", email=" + email + '}';
    }

}
```

Ejemplo de Excepción NullPointerException

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package utng;

/**
 * Descripción Clase Estudiante que describe a todos los estudiantes
 * @author Gabriel Barrón
 */
public class Test {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Estudiante e1 = null;  //Referencia de objeto pero no creado

        e1.setNombre("Juan Alberto"); // invocando método de objeto nulo
    }

}
```

Validando la clase Estudiante

Datos no vacíos

```
    /**
     * Método que actualiza el número de control
     * @param numControl Numero de control de estudiante
     * @throws Exception Valida datos no vacíos
     */
    public void setNumControl(String numControl) throws Exception {
        if (numControl.trim().isEmpty()) { //Verifica que no vaya vacío
            throw new Exception("El nombre no puede ser vacío");
        }
        this.numControl = numControl;

    }
```

Validando Email

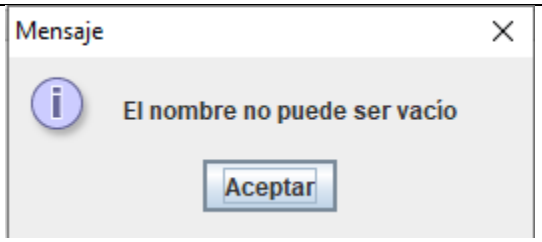
```
public void setEmail(String email) throws Exception {
   String regx = "^(.+)@(.+)$";
   Pattern pattern = Pattern.compile(regx);

    Matcher matcher = pattern.matcher(email);
    if (!matcher.matches()) {
       throw new Exception("Correo Electrónico invalido");
    }
   this.email = email;
}
```

Validando la edad de una persona

```
/**
 * Método que verifica la edad de una persona.
 * @param edad Edad del estudiante
 * @throws Exception  Valida Edad
 */
public void setEdad(int edad) throws Exception {
   //Verifica un rango de edades
   if (edad>=10 && edad <= 80){
      throw new Exception("Edad Incorrecta");
   }
   this.edad = edad;
}
```

Probando el método

```
public static void main(String[] args) {
   Estudiante e1 = new Estudiante();  //Referencia de objeto pero no creado
   try {
      e1.setNumControl(" "); // invocando método con datos vacíos
   } catch (Exception ex) {
      JOptionPane.showMessageDialog(null, ex.getMessage());
   }
}
```



Probando el método

```
public static void main(String[] args) {
   Estudiante e1 = new Estudiante();  //Referencia de objeto pero no creado

   try { //Bloque que vigila el código

      e1.setNombre("Juan Roberto"); // invocando método con datos vacíos
      e1.setApellidos("Barajas Ruiz");
      e1.setNumControl("20201231234");
      e1.setEdad(20);
      e1.setEmail("barajas");
   } catch (Exception ex) {
      JOptionPane.showMessageDialog(null, ex.getMessage());
   }
}
```

Tercer Tipo de Expceción

El tercer tipo de excepción es la excepción de tiempo de ejecución. Estas son condiciones excepcionales que son internas a la aplicación y que la aplicación generalmente no puede anticipar ni recuperarse. Estos suelen indicar errores de programación, como errores lógicos o uso inadecuado de una API. Por ejemplo, considere la aplicación descrita anteriormente que pasa un nombre de archivo al constructor para FileReader. Si un error lógico hace null que se pase un al constructor, el constructor lanzará NullPointerException. La aplicación puede detectar esta excepción, pero probablemente tenga más sentido eliminar el error que causó la excepción.

Omitir Capturar o Especificar

Algunos programadores consideran que el requisito de captura o especificación es una falla grave en el mecanismo de excepción y lo eluden mediante el uso de excepciones no verificadas en lugar de las excepciones verificadas. En general, esto no es recomendable. La sección Excepciones no verificadas: la controversia habla sobre cuándo es apropiado usar excepciones no verificadas.