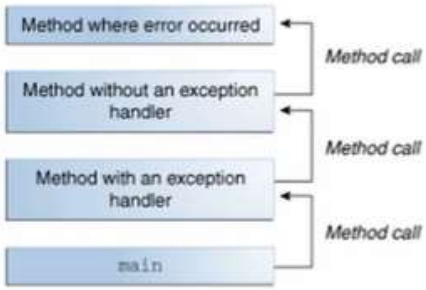
EXCEPCIONES

EXCEPCIÓN

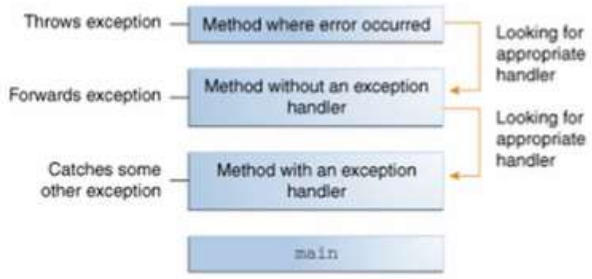
- *Situación excepcional.*
- Altera la ejecución normal del programa.
- El método donde sucede crea un objeto, llamado *objecto de excepción*, y se lo pasa a alguien que pueda tratarlo.



Los métodos pasan el objeto de uno a otro hasta encontrar el método que tiene el mecanismo para gestionar esa excepción.

EXCEPCIÓN

- Si existe quien pueda manejarlo, la recoge.



- Si no existe, se encarga la JVM.

Lo más adecuado es que nuestros métodos le den un tratamiento a las excepciones, y no la JVM.

USO DE EXCEPCIONES

- Permiten separar el código de tratamiento de errores del código normal.
- Evitan que haya errores inadvertidos.
- Permiten la propagación de los errores.
- Permiten agrupar en un lugar común el tratamiento de errores.

TIPOS DE EXCEPCIONES

- *Checked Exceptions*: excepciones que son recogidas y tratadas por programas bien escritos.
- *Error*: son externos a la aplicación, y no nos podemos anticipar a ellos.
- *Runtime error*: situaciones internas a la aplicación, y de las que no nos podemos recuperar

TIPOS DE EXCEPCIONES

Usaremos *Checked Exceptions* cuando:

- La excepción es la única manera de detectar el error.
- No queremos que pase inadvertido

Usaremos *Unchecked Exceptions* cuando:

- Podemos intentar mejorar el código para que no suceda dicho error
- La excepción sirve para detectar y corregir usos indebidos de la clase.
- Errores internos ante los que poco podemos hacer.

Ejemplo:

```
11
12          String name = null;
13          System.out.println(name);
14          System.out.println(name.length());
```

```
null
Exception in thread "main" java.lang.NullPointerException
        at excepciones.SituacionExcepcional.main(SituacionExcepcional.java:14)
```

Si tratamos de acceder a name.length(), es decir, a un método de una instancia de name, que no existe porque es nulo (no referencia a ningún objeto).

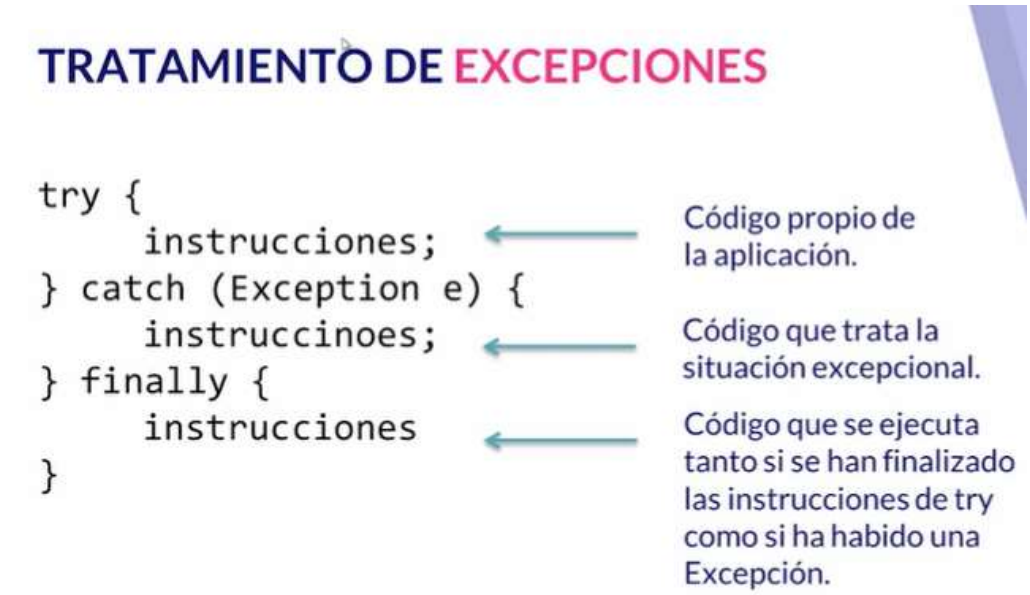
Un programador que controle estos errores, puede escribir código que maneje este error. Otros errores son más complejos.

Otros ejemplos, so más complejos de controlar:

```
//Otras no es posible controlarlas.
int a = 2;
int b = 0;
System.out.println(a/b); //Error de división entre 0
```

```
:ion in thread "main" java.lang.ArithmeticException: / by zero
   at excepciones.SituacionExcepcional.main(SituacionExcepcional.java:19)
```

Java nos ofrece un mecanismo de control de excepciones que vemos a continuación.



TRATAMIENTO DE EXCEPCIONES

- **finally** no es obligatorio.
- Puede haber más de un **catch**.
- Los tipos de excepción deben ir de más concretos a más genéricos.
- El operador | nos permite tratar más de un tipo de excepción en un **catch**.

(| este operador desde versión 7 de java)

BLOQUE TRY

- Debe envolver las sentencias que son susceptibles de provocar uno o varios tipos de excepción.
- Debemos agrupar las sentencias que vayan a tener un tratamiento idéntico de la situación excepcional.

BLOQUE FINALLY

- Se ejecuta siempre (si venimos de ***try*** o de ***catch***).
- Se suele utilizar como código que asegura el cierre de recursos abiertos (ficheros, bases de datos, ...).

Ejemplos:

En u ejemplo como el siguiente, eclipse no nos va a sugerir que lo usemos, tenemos que añadir nosotros todo el código:

```
public static void main(String[] args) {
    int a = 2, b = 0;

    try {
        System.out.println(a/b);
    } catch(Exception ex) {

    }

}
```

```
//      try {
            System.out.println(a/b);
//      } catch(Exception ex) {
//          ex.printStackTrace();
//      }

    }
```

```
xception in thread "main" java.lang.ArithmeticException: / by zero
        at excepciones.EjemploTratamientoExcepciones.main(EjemploTratamientoExcepciones.java:19)
```

Si usamos el método printStrackTrace() lo único que conseguimos es escribir la traza de la pila, que es lo mismo que hace java si no hacemos nada.

Pero podemos darle al error el tratamiento que queramos, podemos escribir “hay un error”, y continuar con nuestro programa, de esta forma, la ejecución continúa:

```
    public static void main(String[] args) {
        int a = 2, b = 0;

        try {
            System.out.println(a/b);
        } catch(Exception ex) {
            //ex.printStackTrace();
            System.out.println("Error de división entre cero");
        }

        System.out.println("La aplicación continua");
```

```
Error de división entre cero
La aplicación continua
```

(prueba el ejemplo anterior de diferentes formas)

Otra forma de tratarlo:

```
    public static void main(String[] args) {


        try {
            int a = 2;
            int b = 0;
            System.out.println(a/b); //Error de división entre 0
        } catch(ArithmeticException ex) {
            //ex.printStackTrace();
            System.err.println("Error: " + ex.getMessage());
        }

        System.out.println("\nMensaje tras la división");

    }
```

```
Error: / by zero
Mensaje tras la división
```

Con .err en lugar de .out es una consola común a la de salida (out), en eclipse lo único que hace es escribirlo en rojo. Otro ejemplo, varios bloques catch, para tratar diferentes errores.

```java
try {
    int a = 2;
    int b = 1;
    int resultado = a/b;
    String mensaje = null;
    System.out.println(mensaje.length()); //Error de división entre 0
} catch(ArithmeticException ex) {
    //ex.printStackTrace();
    System.err.println("Error: " + ex.getMessage());
} catch(Exception ex) {
    System.out.println("Se ha producido un error no esperado");
}
```

Otro error muy común, que os ha pasado a muchos, acceder a una posición de un array que no existe, en este caso a la posición 5. Podemos tratar ese error así:

```java
public static void main(String[] args) {

    String[] mensajes = { "En un lugar", "de La Mancha", "de cuyo nombre", "no quiero acordarme" }
    //String[] mensajes = { "En un lugar", null, "de cuyo nombre", "no quiero acordarme", "no ha
    try {
        for (int i = 0; i < 5; i++) {
            System.out.println(mensajes[i].toUpperCase());
        }
    } catch (ArrayIndexOutOfBoundsException | NullPointerException ex) {
        System.err.println("Tratamiento común a las excepciones");
    }

    System.out.println("Mensaje final");

}
```

```
EN UN LUGAR
DE LA MANCHA
DE CUYO NOMBRE
NO QUIERO ACORDARME
Mensaje final
Tratamiento común a las excepciones
```

Con una cadena nula, al intentar pasarla a mayúsculas:

```java
//String[] mensajes = { "En un lugar", "de La Mancha", "de cuyo nombre", "no quiero acordarm
String[] mensajes = { "En un lugar", null, "de cuyo nombre", "no quiero acordarme", "no ha
try {
    for (int i = 0; i < 5; i++) {
        System.out.println(mensajes[i].toUpperCase());
    }
} catch (ArrayIndexOutOfBoundsException | NullPointerException ex) {
    System.err.println("Tratamiento común a las excepciones");
}

System.out.println("Mensaje final");

}
```

Si queremos darles tratamientos distintos a las dos excepciones, ponemos dos catch:

```java
    }
} catch (ArrayIndexOutOfBoundsException ex) {
    System.err.println("Tratamiento particular a las excepción ArrayIndex...");
} catch (NullPointerException ex) {
    System.err.println("Tratamiento particular a la excepción NullPointer...");
}
```