

# Exceptions

## Exceptions

Es un mecanismo usado por muchos lenguajes de programación para describir qué hacer cuando algo no esperado sucede, algo fuera del curso normal.

## Tipos de exceptions (Checked y Unchecked)

### ■ Checked

- Se espera que el programador las maneje.
- Pertenecen al dominio que modelamos.
- El compilador nos obliga a tomar decisiones con la excepción.
- Por ejemplo, la búsqueda de un archivo que no se encuentra en el disco.

### ■ Unchecked

- Excepciones en tiempo de ejecución.
- Son excepciones inesperadas. Se espera que aparecen cuando existe un bug en nuestro código.
- El compilador no nos obliga a decir qué hacer con la excepción.
- No pertenecen al dominio que modelamos.
- Por ejemplo, cuando en un arreglo intentamos acceder a una posición inexistente.

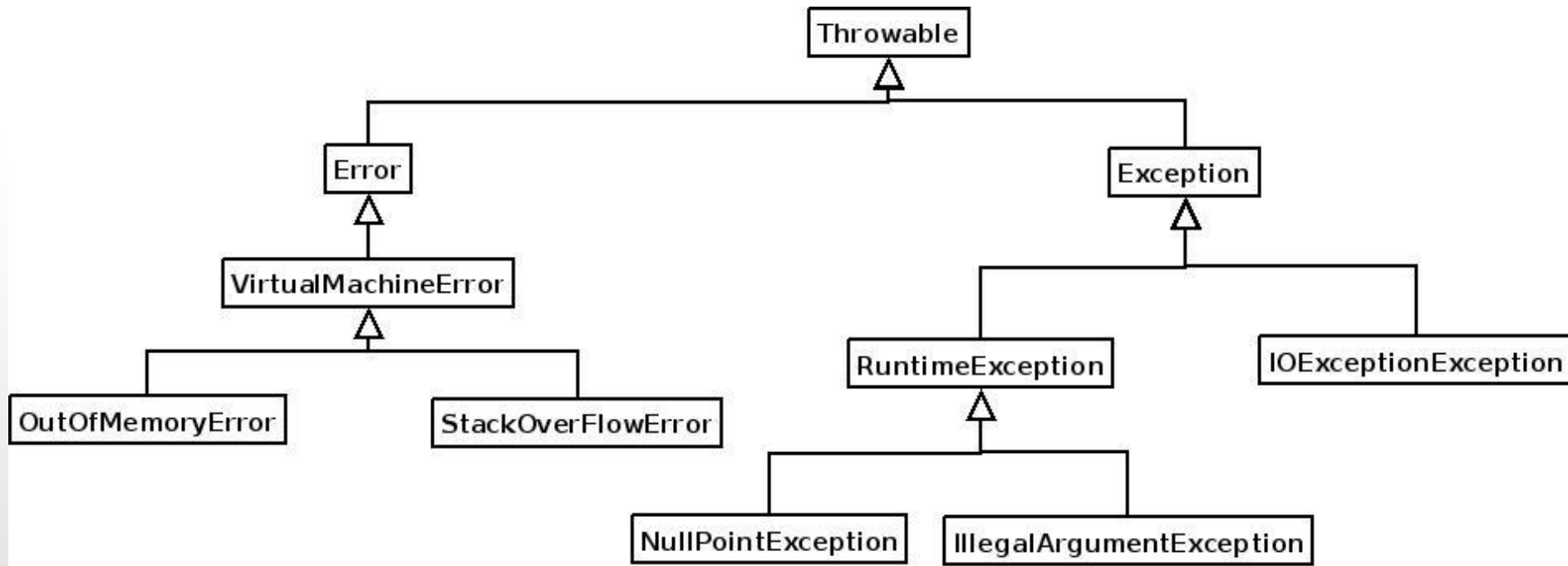
## Exceptions

**java.lang.Exception** es la clase base para las excepciones **checked**.

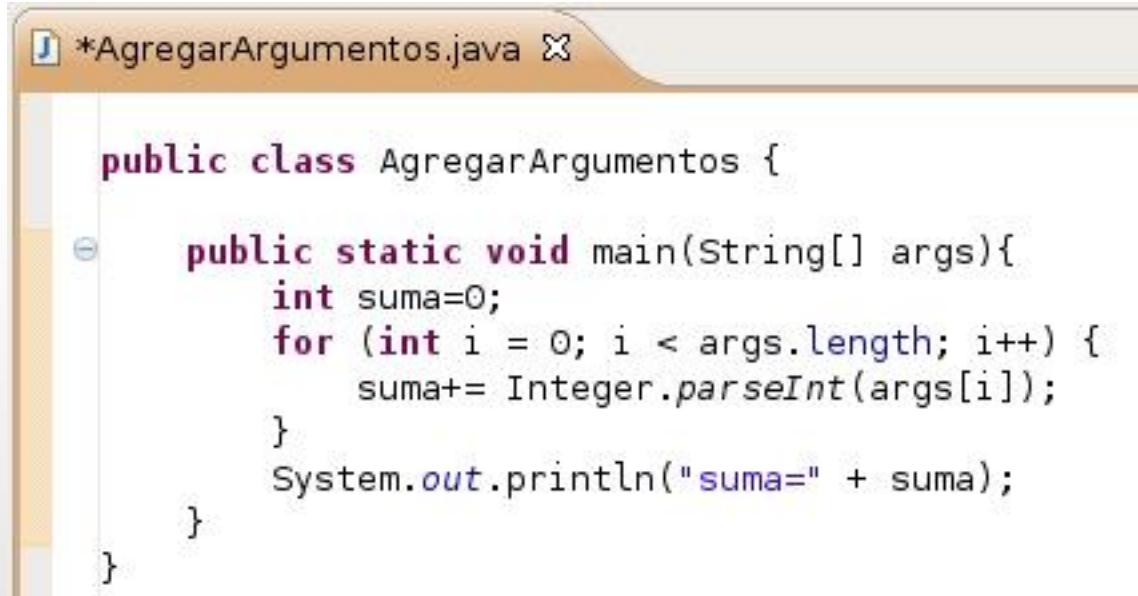
Existen especializaciones de esta clase según el tipo de excepción y el dominio.

**java.lang.RuntimeException** es la clase base para las excepciones **unchecked**.

# Jerarquía de excepciones



# Ejemplo



```
*AgregarArgumentos.java X

public class AgregarArgumentos {

    public static void main(String[] args){
        int suma=0;
        for (int i = 0; i < args.length; i++) {
            suma+= Integer.parseInt(args[i]);
        }
        System.out.println("suma=" + suma);
    }
}
```

Este programa funciona bien cuando sus argumentos son enteros.

Exception in thread "main" java.lang.NumberFormatException: For input string: "two" at  
java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)  
at java.lang.Integer.parseInt(Integer.java:447)  
at java.lang.Integer.parseInt(Integer:497)  
at AddArguments.main(AddArguments.java:5)

## bloques try y catch

Usamos try ejecutar código que puede generar excepción y agarrarla.

Usamos catch para decidir qué hacer con la excepción capturada.

```
try{  
    // Aquí el código que puede generar excepción  
}catch(NumberFormatException nfe){  
    // Dice qué excepción atrapa.  
    // En este bloque tomamos una decisión.  
}
```

# Una solución

```
AgregarArgumentos.java ✕  
  
public class AgregarArgumentos {  
    public static void main(String[] args){  
        try{  
            int suma=0;  
            for (int i = 0; i < args.length; i++) {  
                suma+= Integer.parseInt(args[i]);  
            }  
            System.out.println("suma=" + suma);  
        }catch (NumberFormatException nfe){  
            System.err.println("Uno de los argumentos no es un entero");  
        }  
    }  
}
```

Si nos pasan un argumento con un número decimal, ¿qué se imprimirá?

- Podemos insertar el try {...} donde queramos siempre y cuando se encierre a la porción de código que arroja la excepción.
- Ejercicio: realizar este caso para que la suma se complete.

## Múltiples catch

```
try{  
    // puede lanzar una o más excepciones  
}catch(MiExcepcion me){  
    // código que se ejecuta si la excepción MyExcepcion es lanzada  
}catch(MiOtraExcepcion moe){  
    // código que se ejecuta si la excepción MiOtraExcepcion es lanzada  
}catch(Exception e){  
    // código que se ejecuta para cualquier otra excepción.  
}
```

- Dentro de un try pueden lanzarse múltiples excepciones
- Es importante notar que importa el orden de los catch; ya que puedo tener una jerarquía de herencia para los diferentes excepciones siendo unas especializaciones de otras.
- Exception es la clase base de todas las excepciones.



# Manejo de excepciones

- Si queremos delegar la responsabilidad de manejar la excepción, declaramos con la primitiva `throws` que lanzamos la excepción en la signatura del método.
- Las excepciones runtime no hace falta declararlas.

```
metodoQueNoManejaLaExcepcion(...) throws CañeriaRota, otraExcepcion{  
    // Le pasamos el control al llamador  
}
```

- Ejemplo de cómo lanzar una excepción que nosotros mismos creamos:

```
metodoQueLanzaUnaExcepcionPropia(...) throws CañeriaRota{  
    throw new CañeriaRota();  
}
```