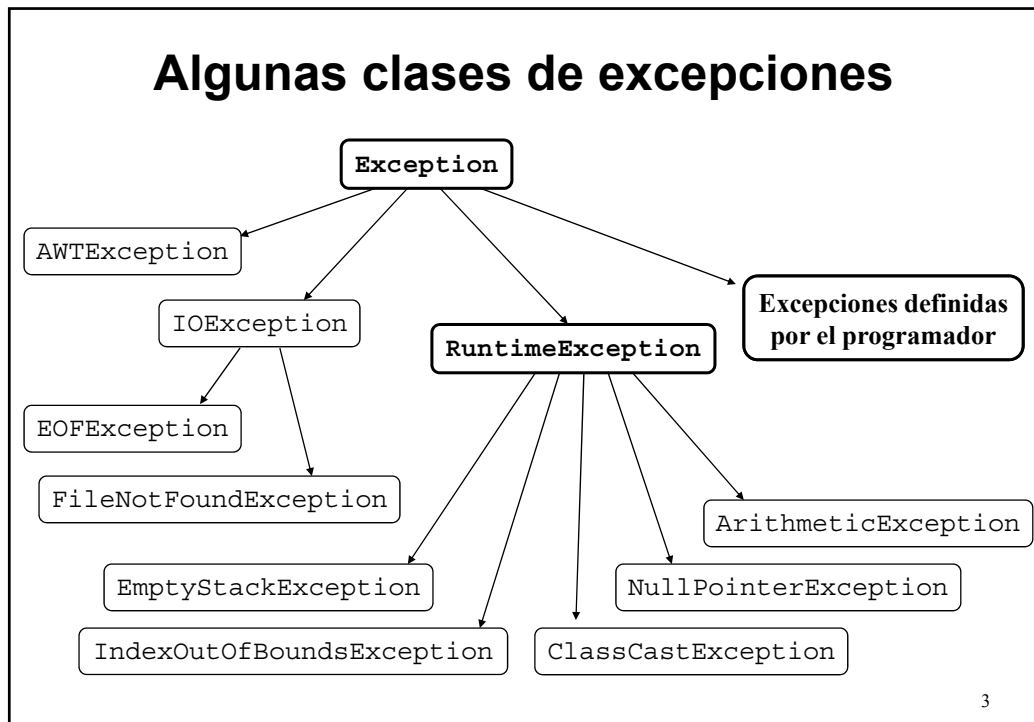


## Tratamiento de errores: excepciones

### Excepciones (I)

- Java gestiona los errores de ejecución mediante un mecanismo denominado “excepciones”
- Las excepciones informan del tipo de error, la traza de llamadas a métodos, y la línea del código fuente donde se produjo el error
- El programador puede definir sus propios tipos de error específicos a una aplicación, definiendo clases de excepción, y jerarquías si se desea
- Las excepciones son objetos
- Para generar un error se crea un objeto excepción y se “lanza”
- Cuando se lanza una excepción se interrumpe el flujo de ejecución del programa
- Existe un mecanismo para capturar excepciones para evitar que la aplicación se interrumpa por completo, y reaccionar al error



## Excepciones (II)

- Permiten saltar de un punto (throw) a otro (catch) en cualquier lugar del programa
  - Programación independiente de emisión, captura y comportamiento de las excepciones
- Se envía información junto con el salto: una excepción
  - Una excepción es un objeto de una clase específica – contiene información sobre la condición de error que se ha producido
  - Cada catch recoge un tipo de excepción, e ignora el resto
- Utilidad: tratamiento de errores (error recovery)

## Excepciones (III)

```
try {  
    ...  
    obj.f ();  
    ...  
}  
  
catch (EdadNegativa ex) {  
    ...  
}
```

```
void f () throws EdadNegativa {  
    ...  
    if (edad < 0)  
        throw new EdadNegativa(persona, edad);  
    ...  
}
```

5

## Ejemplo

### 1. Lanzamiento de excepciones

```
class CuentaBancaria {  
    ...  
    boolean bloqueada;  
    ...  
    void retirar (long cantidad) throws SaldoInsuficiente,  
                                                CuentaBloqueada {  
        if (bloqueada)  
            throw new CuentaBloqueada (numero);  
        else if (cantidad > saldo)  
            throw new SaldoInsuficiente (numero, saldo);  
        else saldo -= cantidad;  
    }  
}
```

6

## 2. Definición de clases para excepciones

```
class SaldoInsuficiente extends Exception {
    long numero, saldo;
    SaldoInsuficiente (long num, long s) {
        numero = num; saldo = s;
    }
    public String toString () {
        return "Saldo insuficiente en cuenta " + numero
            + "\nDisponible: " + saldo;
    }
}

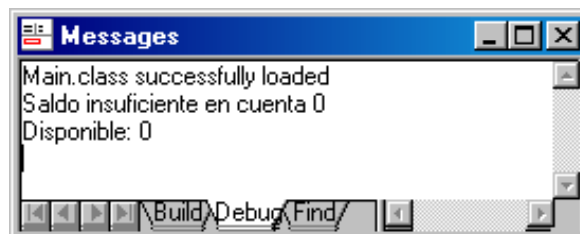
class CuentaBloqueada extends Exception {
    long numero;
    CuentaBloqueada (long num) { numero = num; }
    public String toString () {
        return "La cuenta " + numero + " esta bloqueada";
    }
}
```

7

## 3. Captura y procesamiento de excepciones

```
static public void main (String args[]) {
    try {
        new CuentaBancaria () .retirar (100000);
    }
    catch (SaldoInsuficiente excep) {
        System.out.println (excep);
    }
    catch (CuentaBloqueada excep) {
        System.out.println (excep);
    }
}
```

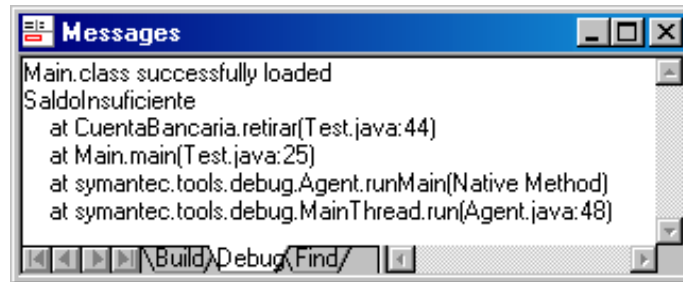
*Se ejecuta el primer catch de tipo compatible*



8

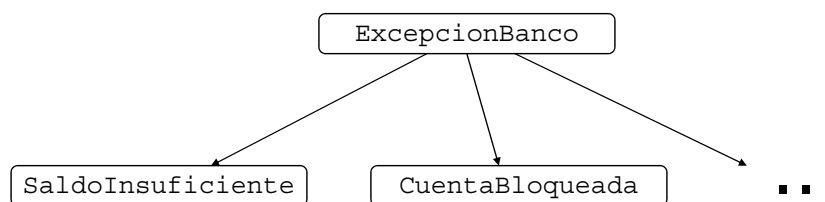
#### 4. ¿Y si no se captura una excepción?

```
static public void main (String args[])  
    throws CuentaBloqueada, SaldoInsuficiente {  
    CuentaBancaria cuenta = new CuentaBancaria ();  
    cuenta.saldo = 1000;  
    cuenta.retirar (2000);  
}
```



9

### Jerarquías de excepciones



#### Tratamiento genérico

```
catch (ExcepcionBanco obj) {  
    ...  
}
```

#### Tratamiento específico

```
catch (SaldoInsuficiente obj) {  
    ...  
}  
catch (CuentaBloqueada obj) {  
    ...  
}
```

10

## Las excepciones son parte de la interfaz de un objeto

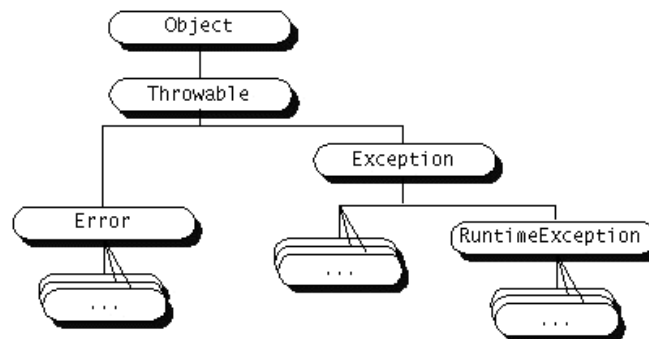
- Si un método deja pasar una excepción, se debe declarar en la cabecera
- Sólo los `Error`'s y las `RuntimeException`'s no requieren ser declarados
- Un método sobrescrito no puede declarar más excepciones que (subclases de) las que declara la definición en la clase padre
- Por tanto, si un método sobrescrito emite una excepción no declarada en el padre, es obligatorio procesarla aunque no se haga nada con ella

```
class X extends Applet {  
    public void start () { // start heredado de Applet  
        try { ... } (catch IOException e) { /* vacio */}  
    }  
}
```

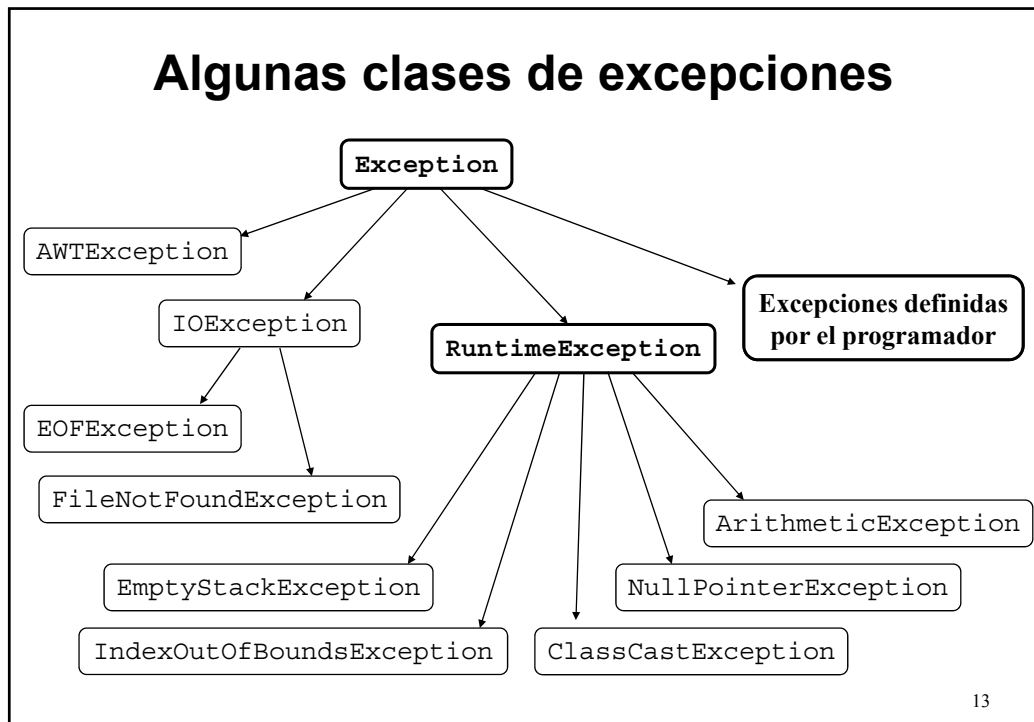
11

## Excepciones predefinidas

- Java genera los errores de ejecución en forma de excepciones
- Runtime exceptions: punteros null, arrays, aritmética, etc.
  - No se comprueban (¿por qué?)
  - En general, no deben subclassificarse (¿por qué?)
- Error: errores de linkado, desbordamiento de memoria, etc. (errores serios)
  - En general no deben procesarse ni subclassificarse



12



## Métodos de Throwable

- `Throwable(String)` Constructor que asigna un mensaje al objeto
- `getMessage()` Devuelve el mensaje del objeto
- `toString()` Devuelve un string incluyendo la clase del objeto más el mensaje
- `printStackTrace()` Escribe la traza de ejecución en el standard error

Cuando una excepción no se procesa hasta el final, el programa se interrumpe y se ejecuta `printStackTrace()`

- Para aprovechar los métodos de `Throwable`, definir un constructor:

```
CuentaBloqueada (long num) {  
    super ("La cuenta " + num + " esta bloqueada");  
    numero = num;  
}
```

14

## Ventajas de las excepciones

- Separación del tratamiento de errores del resto del código del programa
  - Evitar manejo de códigos de error
  - Evitar la alteración explícita del control de flujo
- Propagación de errores a través de la pila de llamadas a métodos
  - Evitar el retorno de valores de error
  - Evitar la utilización de argumentos adicionales
- Agrupamiento de tipos de errores, diferenciación de tipos de errores
  - Jerarquías de clases de excepciones
  - Tratar los errores al nivel de especificidad deseado

15

## Sin tratamiento de errores

```
readFile {  
    open the file;  
    determine its size;  
    allocate that much memory;  
    read the file into memory;  
    close the file;  
}
```

16

## Tratamiento de errores sin excepciones

```

errorCodeType readFile {
    initialize errorCode = 0;
    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) errorCode = -1;
            }
            else errorCode = -2;
        }
        else errorCode = -3;
        close the file;
        if (theFileDidntClose && errorCode == 0)
            errorCode = -4;
        else errorCode = errorCode & -4;
    }
    else errorCode = -5;
    return errorCode;
}

```

17

## Tratamiento de errores con excepciones

```

readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    }
    catch (fileOpenFailed) { doSomething; }
    catch (sizeDeterminationFailed) { doSomething; }
    catch (memoryAllocationFailed) { doSomething; }
    catch (readFailed) { doSomething; }
    catch (fileCloseFailed) { doSomething; }
}

```

18

## Sin tratamiento de errores

```
method1 {  
    call method2;  
}  
  
method2 {  
    call method3;  
}  
  
method3 {  
    call readfile;  
}
```

19

## Tratamiento de errores sin excepciones

```
method1 {  
    errorCodeType error;  
    error = call method2;  
    if (error) doErrorProcessing;  
    else proceed;  
}  
  
errorCodeType method2 {  
    errorCodeType error;  
    error = call method3;  
    if (error) return error;  
    else proceed;  
}  
  
errorCodeType method3 {  
    errorCodeType error;  
    error = call readfile;  
    if (error) return error;  
    else proceed;  
}
```

20

## Con excepciones

```
method1 {  
    try {  
        call method2;  
    }  
    catch (exception) {  
        doErrorProcessing;  
    }  
}  
  
method2 throws exception {  
    call method3;  
}  
  
method3 throws exception {  
    call readfile;  
}
```

21