

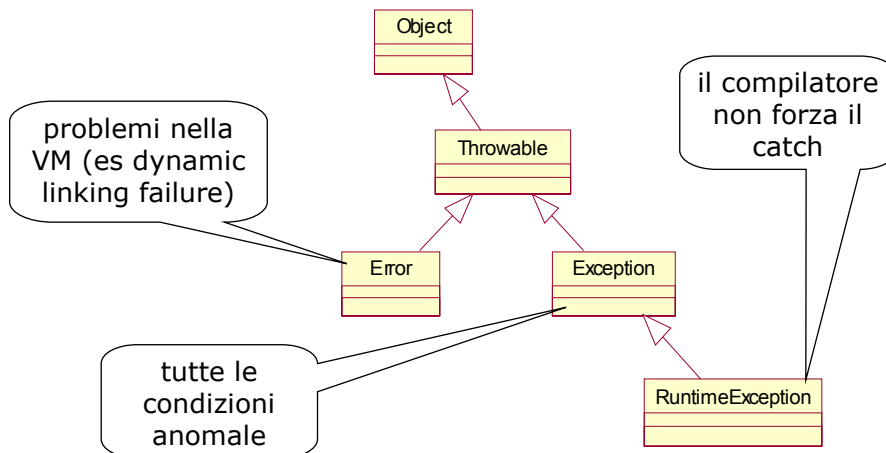
# Programmazione ad Oggetti

## Eccezioni

V 1.2 © Marco Torchiano 2005

## Eccezione

- È una classe derivata da Throwable



## Esempi di eccezioni

---

- Error
    - ♦ OutOfMemoryError
  - Exception
    - ♦ ClassNotFoundException
    - ♦ InstantiationException
    - ♦ NoSuchMethodException
    - ♦ IllegalAccessException
    - ♦ NegativeArraySizeException
    - ♦ EmptyStackException
  - RuntimeException
    - ♦ NullPointerException
- 

## Gestione di eccezioni

---

- Le eccezioni sono oggetti; un tipico errore è quello di dimenticare il “new” quando si lancia l’eccezione:

```
throw new AnException(“some message”);
```

- Quando un frammento di codice lancia un’eccezione essa può essere gestita in due modi:

1. Intercettare l’eccezione e gestirla:

```
try{ /* lancia eccezioni */  
    }catch(AnException e){ ... }
```

2. Propagare l’eccezione al chiamante:

```
method() throws AnException {  
    /* lancia eccezioni */ }
```

---

## Run-time Exception

---

- Non obbligano a gestire l'eccezione
- Es. NumberFormatException

```
String s = "k123";
int i;
try {
    i = Integer.parseInt(s);
} catch (NumberFormatException nfe) {
    System.out.println("Numero errato");
    i = 0;
}
```

---

## Eccezioni e cicli (1)

---

- Per errori facilmente recuperabili facendo una successiva iterazione il blocco try-catch è contenuto all'interno del ciclo.
- In caso di eccezione l'esecuzione passa al catch e poi prosegue con l'iterazione successiva.

```
while(true){
    try{
        // potential exceptions
    }catch(Exception e){
        // print error message
    }
}
```

---

## Eccezioni e cicli (2)

---

- Per errori che compromettono il ciclo, blocco try-catch contiene il ciclo.
- In caso di eccezione l'esecuzione passa al catch uscendo dal ciclo.

```
try{
    while(true){
        // potential exceptions
    }
}catch(AnException e){
    // print error message
}
```

---

## Test di eccezioni

---

- Occorre distinguere due casi principali:
  - Ci si aspetta che si verifichi una situazione anomala e quindi che il codice generi un'eccezione
    - ♦ In questo caso il test fallisce se **NON** si rileva nessuna eccezione
  - Ci si aspetta un comportamento normale e quindi nessuna eccezione
    - ♦ In questo caso il test fallisce se si rileva una qualsiasi eccezione
-

## Test di eccezione attesa

```
try{
    // metodo chiamato con parametri errati
    oggetto.metodo(null);
    fail("L'operazione XXX dovrebbe fallire");
}catch(EccezionePossibile e){
    assertTrue(true); // OK
}
```

```
class LaClasse {
    public void metodo(String p)
        throws EccezionePossibile
    { /*... */ }
}
```

## Test di eccezione NON attesa

```
try{
    // metodo chiamato con parametri corretti
    oggetto.metodo("Parametro");
    assertTrue(true); // OK
}catch(EccezionePossibile e){
    fail("Il metodo NON dovrebbe fallire");
}
```

Eccezione → Failure

Runs: 2/2    ✖ Errors: 0    ✖ Failures: 1

## Eccezione non attesa

---

```
public void testSomething()  
    throws EccezionePossibile {  
    // metodo chiamato con parametri corretti  
    oggetto.metodo("Parametro");  
}
```

Eccezione → Error

Runs: 2/2   ✖ Errors: 1   ❏ Failures: 0

