

Java SE: Eccezioni

- Eccezioni
- Gerarchia Throwable
- Gestione delle eccezioni in JUnit
 - “Classica”
 - Supporto funzionale Java 8 con Jupiter
- Progetto di riferimento
 - <https://github.com/egalli64/jse> (*modulo 7*)

Eccezioni

- Sono utilizzate nei metodi per segnalare che è avvenuto un errore
 - Qualcosa di inatteso, *eccezionale*
- Il chiamante deve gestire il problema, o passarlo al suo chiamante
 - Le eccezioni non gestite causano la terminazione del programma
- Evidenziano il flusso normale di esecuzione
- Semplificano il debug esplicitando lo stack trace
- Già con il loro nome possono chiarire il motivo scatenante
 - NullPointerException, ArrayIndexOutOfBoundsException, ...
- Disponibili in due famiglie, checked e unchecked



throws / throw

- Un metodo che può tirare eccezioni
 - Se checked, lo deve segnalare con un **throws** dopo la signature
 - Se unchecked, la segnalazione di throws è facoltativa
- Lo statement **throw** lancia una eccezione
 - Interrompe il flusso di esecuzione normale
 - Causa lo “*stack unwinding*” alla ricerca di un catch che lo gestisca

```
public void aCheckedThrower() throws Exception {  
    // ...  
    if (somethingUnexpected()) {  
        throw new Exception();  
    }  
}
```

```
public void anUncheckedThrower() {  
    // ...  
    if (somethingUnexpected()) {  
        throw new IllegalStateException();  
    }  
}
```

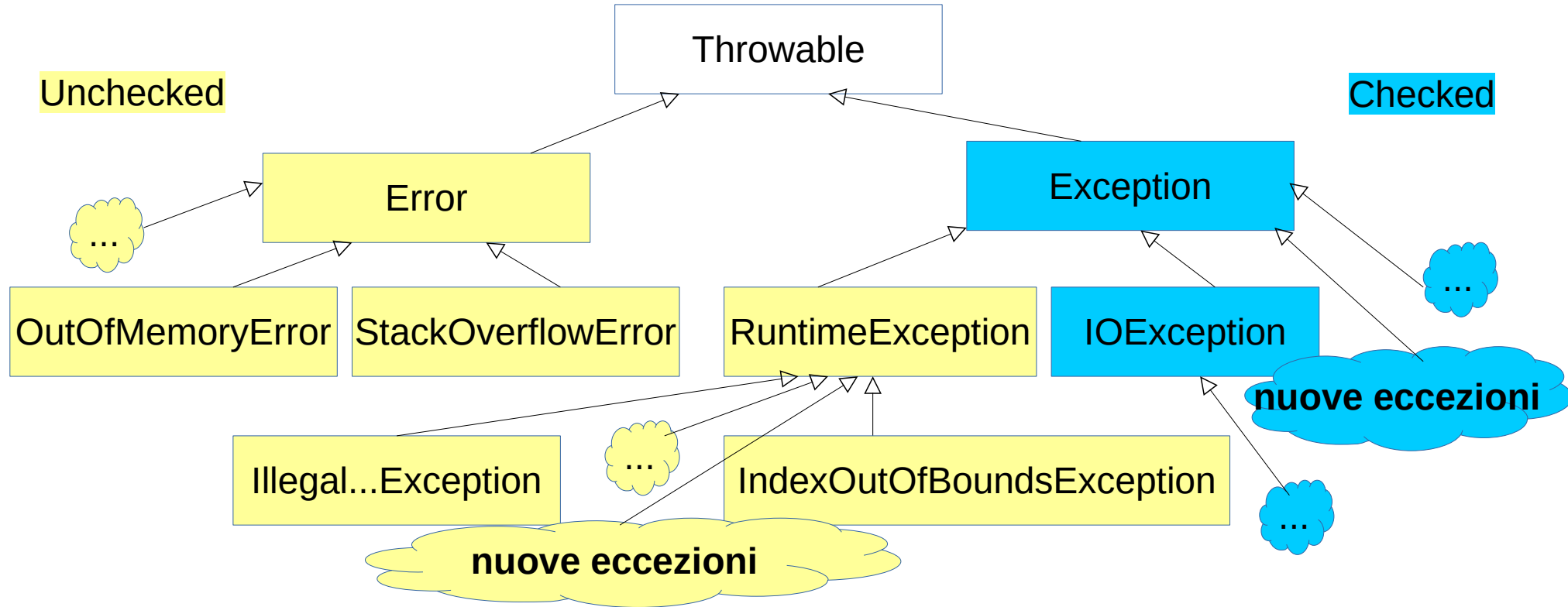
try – catch – finally

- L'esecuzione di metodi “rischiosi” è gestita via
 - Un blocco **try**, (in alternativa, throws dopo la signature del metodo)
 - Blocchi **catch** per una o più eccezioni
 - Blocco **finally** per la fase di cleanup (esecuzione garantita)
- Ad un try per eccezioni deve seguire almeno un catch o il finally

```
try {  
    exceptional.aCheckedThrower();  
} catch (Exception ex) {  
    // ...  
} finally {  
    cleanup();  
}
```

```
public void notACatcher() throws Exception {  
    Exceptional exceptional = new Exceptional();  
  
    // ...  
    exceptional.aCheckedThrower();  
}
```

Gerarchia delle eccezioni



Test eccezioni in JUnit 3

Math.abs() di
Integer.MIN_VALUE
è
Integer.MIN_VALUE!

```
public int negate(int value) {  
    if(value == Integer.MIN_VALUE) {  
        throw new IllegalArgumentException("Can't negate MIN_VALUE");  
    }  
    return -value;  
}
```

```
@Test  
void negateException() {  
    Simple simple = new Simple();  
  
    try {  
        simple.negate(Integer.MIN_VALUE);  
        fail("An IllegalArgumentException was expected");  
    } catch (IllegalArgumentException iae) {  
        String message = iae.getMessage();  
        assertTrue(message, is("Can't negate MIN_VALUE"));  
    }  
}
```

JUnit 4.7 ExpectedException

```
@Rule
public ExpectedException thrown = ExpectedException.none();

@Test
public void negateMinInt() {
    thrown.expect(IllegalArgumentException.class);
    thrown.expectMessage("Can't negate MIN_VALUE");

    Simple simple = new Simple();
    simple.negate(Integer.MIN_VALUE);
}
```

Nel @Test
si dichiara
quale eccezione
e messaggio
ci si aspetta

Si definisce una
variabile di istanza
ExpectedException
taggata come @Rule

JUnit 5 assertThrows()

Il metodo fallisce se quanto testato non tira l'eccezione specificata

L'eccezione generata viene tornata per permettere ulteriori test

```
@Test
public void negateMinInt() {
    IllegalArgumentException exc = assertThrows(IllegalArgumentException.class,
        () -> simple.negate(Integer.MIN_VALUE));
    assertThat(exc.getMessage(), is("Can't negate MIN_VALUE"));
}
```

L'assertion è eseguita su di un Executable, interfaccia funzionale definita in Jupiter