UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione 3
e
Laboratorio di Programmazione 3

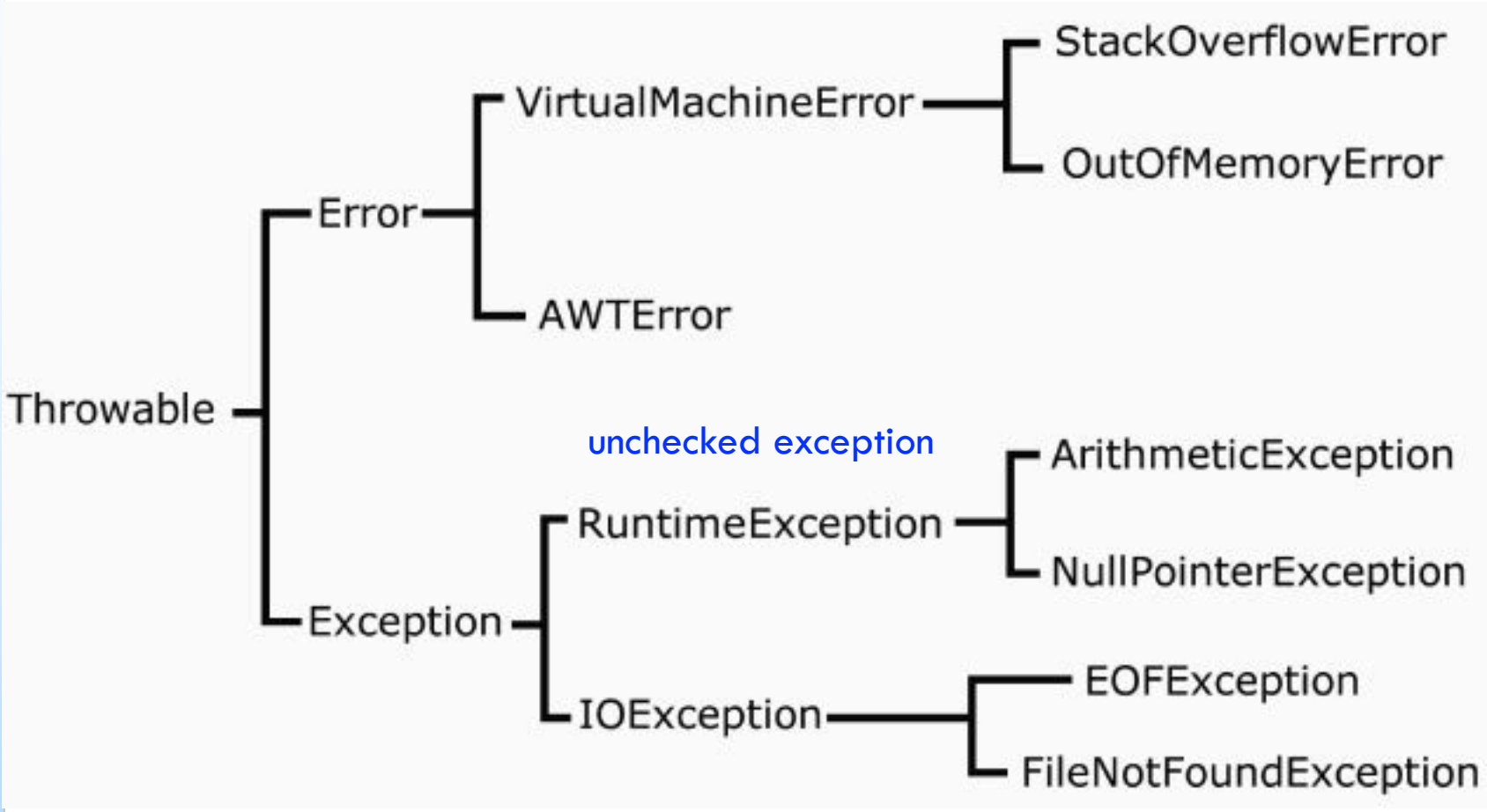
Gestione delle Eccezioni

Angelo Ciaramella

Gestione delle eccezioni

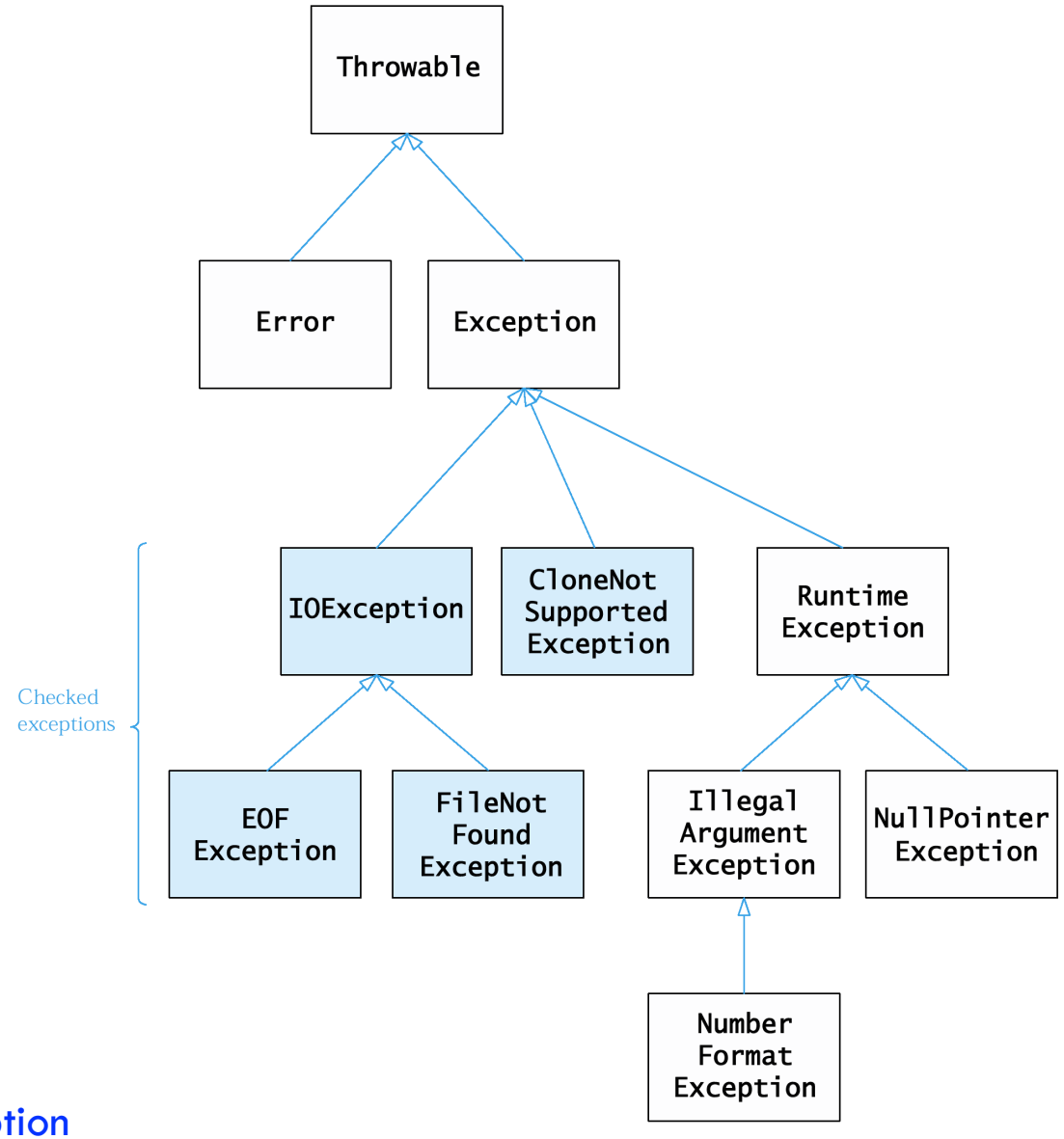
- eccezioni, errori ed asserzioni
  - permettono allo sviluppatore di scrivere software robusto
  - funzionalità corrette anche in situazioni impreviste
- eccezione
  - situazione imprevista che può presentarsi durante il flusso di un'applicazione
- errore
  - come una situazione imprevista non dipendente da un errore commesso dallo sviluppatore
- asserzione
  - condizione che deve essere verificata affinché lo sviluppatore consideri corretta una parte di codice
  - abilitare in fase di sviluppo e test, ed eventualmente disabilitare in fase di rilascio

Gerarchia di classi



Gerarchia per le classi Throwable (lanciabile)

Gestione delle eccezioni



Checked e Unchecked exception

Gestione delle eccezioni

- Programma genera un'eccezione
  - Lancia l'eccezione (`throw`) e il gestore automatico della JVM interrompe il programma
  - Il nostro codice cattura l'eccezione (`catch`)

Eccezione

```
public class Ecc1 {
public static void main(String args[]) {
int a = 10;
int b = 0;
int c = a/b;
System.out.println(c);
     }
}
```

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero at
Ecc1.main(Ecc1.java:6)
```

Programma che termina prematuramente

Try-Catch

```
public class Ecc1 {
public static void main(String args[]) {
int a = 10;
int b = 0;

try {
int c = a/b;
System.out.println(c);
}
catch (ArithmeticException exc) {
System.out.println("Divisione per zero...");
      }
    }
}
```

L'eccezione lanciata viene catturata

Il programma termina in maniera naturale

Try-Catch

```
public class Ecc1 {
public static void main(String args[]) {
int a = 10;
int b = 0;

try {
int c = a/b;
System.out.println(c);
}
catch (ArithmeticException exc) {
exc.printStackTrace();       }
    }
}
```

Visualizza messaggi informativi ma senza interrompere il programma stesso

Try-Catch

```
public class Ecc1 {
public static void main(String args[]) {
int a = 10;
int b = 0;

try {
int c = a/b
System.out.println(c);
}
catch (Exception exc) {
exc.printStackTrace();       }
    }
}
```

Parametro polimorfo – gestisce qualsiasi tipo di eccezione

Try-Catch

```
public class Ecc1 {
public static void main(String args[]) {
int a = 10;
int b = 0;

try {
int c = a/b
System.out.println(c);
}

catch (ArithmeticException exc) {
System.out.println("Divisione per zero...");
}
catch (NullPointerException exc) {
System.out.println("Reference nullo...");
}

catch (Exception exc) {
exc.printStackTrace();       }
    }
}
```

Per gestire diversi tipi di eccezione

Try-Catch

```
public class Ecc1 {
public static void main(String args[]) {
int a = 10;
int b = 0;

try {
int c = a/b
System.out.println(c);
}

catch (ArithmeticException | NullPointerException exc) {
System.out.println("Divisione per zero...");
}

catch (Exception exc) {
exc.printStackTrace();       }
    }
}
```

Più di un parametro in un unico catch

Finally

```
public class Ecc1 {
public static void main(String args[]) {
int a = 10;
int b = 0;

try {
int c = a/b
System.out.println(c);
}

catch (ArithmeticException | NullPointerException exc) {
System.out.println("Divisione per zero...");
}

catch (Exception exc) {
exc.printStackTrace();
 }

finally {
System.out.println("Operazione terminata");}
     }
}
```

Il blocco finally viene eseguito con o senza eccezione

Eccezioni personalizzate

- Alcune tipologie di eccezioni che sono più frequenti
  - `NullPointerException`
  - `ArrayIndexOutOfBoundsException`
  - `ClassCastException`
  - `IOException`
    - `FileNotFoundException, EOFException`
  - `SQLException`
  - `ConnectException`
  - `...`
- E' possibile definire nuovi tipi di eccezione

Nuovi tipi di eccezione

```
public class PrenotazioneException extends Exception {
public PrenotazioneException() {
// Il costruttore di Exception chiamato inizializza la
// variabile privata message
super("Problema con la prenotazione");
}
public String toString() {
return getMessage() + ": posti esauriti!";
}
}
```

Definizione

```
PrenotazioneException exc = new PrenotazioneException();
throw exc;

//oppure

throw new PrenotazioneException();
```

Lancio dell'eccezione

Nuovi tipi di eccezione

```
try {
//controllo sulla disponibilità dei posti
if (postiDisponibili == 0) {
//lancio dell'eccezione
throw new PrenotazioneException();
}
//istruzione eseguita
// se non viene lanciata l'eccezione
postiDisponibili--;
}
catch (PrenotazioneException exc){
System.out.println(exc.toString());
}
```

Definizione

Throws

```
public class Botteghino {
private int postiDisponibili;
public Botteghino() {
postiDisponibili = 100;
}
public void prenota() {
try {
//controllo sulla disponibilità dei posti
if (postiDisponibili == 0) {
//lancio dell'eccezione
throw new PrenotazioneException();
}
//metodo che realizza la prenotazione
// se non viene lanciata l'eccezione
postiDisponibili--;
}
catch (PrenotazioneException exc) {
System.out.println(exc.toString());
}
}
}
```

Classe Botteghino

Throws

```
public class GestorePrenotazioni {
public static void main(String [] args) {
Botteghino botteghino = new Botteghino();
for (int i = 1; i <= 101; ++i){
botteghino.prenota();
System.out.println("Prenotato posto n° " + i);
}
}
}
```

```
Prenotato posto n° 1
Prenotato posto n° 2
...
Prenotato posto n° 99
Prenotato posto n° 100
Problema con la prenotazione: posti esauriti!
Prenotato posto n° 101
```


Throws

```
public void prenota() throws PrenotazioneException {
//controllo sulla disponibilità dei posti
if (postiDisponibili == 0) {
//lancio dell'eccezione
throw new PrenotazioneException();
}
//metodo che realizza la prenotazione
// se non viene lanciata l'eccezione
postiDisponibili--;
}
```

```
Prenotato posto n° 1
Prenotato posto n° 2
...
Prenotato posto n° 99
Prenotato posto n° 100
Problema con la prenotazione: posti esauriti!
```

Avvertiamo il compilatore che siamo consapevoli che il metodo possa lanciare al runtime la `PrenotazioneException` e di non preoccuparsi in quanto gestiremo in un'altra parte del codice l'eccezione

Metodi che lanciano eccezioni

```
specificatoreDiAccesso valoreRestituito nomeMetodo
      (tipoParametro nomeParametro, . . .)
       throws ClasseEccezione, ClasseEccezione
```

Metodo che lancia l'eccezione

```
public void read(BufferReader in) throws IOException,
    ClassNotFoundException
```

Esempio di eccezione multiple

Warnings

```
D:\java8\Codice\modulo_08\esempi\PrenotazioneException.java:
1:
warning: [serial] serializable class PrenotazioneException
has no
definition of serialVersionUID
public class PrenotazioneException extends Exception {
^
1 warning
```

```
@SuppressWarnings("serial")
public class PrenotazioneException extends Exception {
```

Uno dei metodi per togliere i Warning (sapendo che la classe non è serializabile)

Override

- override di un metodo
  - non è possibile specificare clausole throws su eccezioni che il metodo base non comprende nella propria clausola throws
  - è possibile da parte del metodo che effettua override dichiarare una clausola throws ad eccezioni che sono sottotipi di eccezioni comprese dal metodo base nella propria clausola throws

Asserzioni

- asserzione
  - istruzione che permette di testare eventuali comportamenti previsti in un'applicazione
  - richiede sia verificata un'espressione booleana che lo sviluppatore ritiene vada verificata nel punto in cui viene dichiarata
    - bug se negativa

```
assert espressione_booleana;
//oppure
assert espressione_booleana: espressione_stampabile;
```

Esistono due tipi di sintassi

Asserzioni

```
assert b > 0;

// equivalente a

if (!(b>0)) {
throw new AssertionError();
}
```

Se l'asserzione è falsa l'applicazione termina stampando un track-trace

```
assert b > 0: "il valore di b è " + b;
```

Esempio di espressione stampabile

Override

- Asserzione
  - è il concetto fondamentale di una metodologia di progettazione tesa a rendere i programmi più robusti
- Progettazione per contratto (Design by contract)
  - Precondizione
    - stato dell'applicazione nel momento in cui viene invocata un'operazione
  - Postcondizione
    - stato dell'applicazione nel momento in cui un'operazione viene completata
  - Invariante
    - applicato ad una classe permette di specificare vincoli per tutti gli oggetti istanziati