Dal libro di Savitch & Carrano

- Un'**eccezione** (*exception*) è un oggetto che segnala l'accadere di un evento anomalo durante l'esecuzione di un metodo.
- Il processo di creazione di quest'oggetto (cioè di generazione di un'eccezione) è chiamato **lancio** di un'eccezione (*throwing an exception*).
- In un'altra parte del programma (magari in un'altra classe o in un altro metodo) si inserisce il codice che si occupa dell'evento eccezionale. Il codice che rileva e che si occupa dell'eccezione si dice che **gestisce l'eccezione** (*handle the exception*). TUTTO CHIARO?

Capitolo 11
[Ingresso/uscita e] gestione delle eccezioni

Versione ridotta dei lucidi di

Cay S. Horstmann
Concetti di informatica e fondamenti di Java
quarta edizione

Obiettivi del capitolo

- [Essere in grado di leggere e scrivere file di testo]
- Imparare a lanciare eccezioni
- Saper progettare proprie classi di eccezioni
- Capire la differenza tra eccezioni a controllo obbligatorio ed eccezioni a controllo non obbligatorio
- Imparare a catturare le eccezioni
- Sapere quando e dove catturare un'eccezione

Lanciare eccezioni

- Per segnalare una condizione eccezionale, usate l'enunciato `throw` per lanciare un oggetto eccezione
- Esempio: `IllegalArgumentException`

```
IllegalArgumentException exception
      = new IllegalArgumentException(
      "Amount exceeds balance");
throw exception;
```

- Non c'è bisogno di memorizzare l'oggetto eccezione in una variabile

```
throw new IllegalArgumentException(
      "Amount exceeds balance");
```

- Quando lanciate un'eccezione il metodo termina immediatamente la propria esecuzione. L'esecuzione passa al gestore dell'eccezione.

Lanciare eccezioni

- Il meccanismo di gestione delle eccezioni è stato progettato per risolvere questi due problemi:
  - Le eccezioni non devono poter essere trascurate
  - Le eccezioni devono poter essere gestite da un gestore *competente*, non semplicemente dal chiamante del metodo che fallisce
- Per segnalare una condizione eccezionale, usate l'enunciato `throw` per lanciare un oggetto eccezione.Esempio:

```
parametro con valore non valido
IllegalArgumentException exception
   = new IllegalArgumentException("Amount exceeds balance");
throw exception;
```

Segue…

Esempio

```
public class BankAccount
{
   public void withdraw(double amount)
   {
      if (amount > balance)
      {
         IllegalArgumentException exception
               = new IllegalArgumentException(
                  "Amount exceeds balance");
      throw exception;
      }
      balance = balance - amount;
   }
   . . .
}
```

La gerarchia delle classi di eccezioni

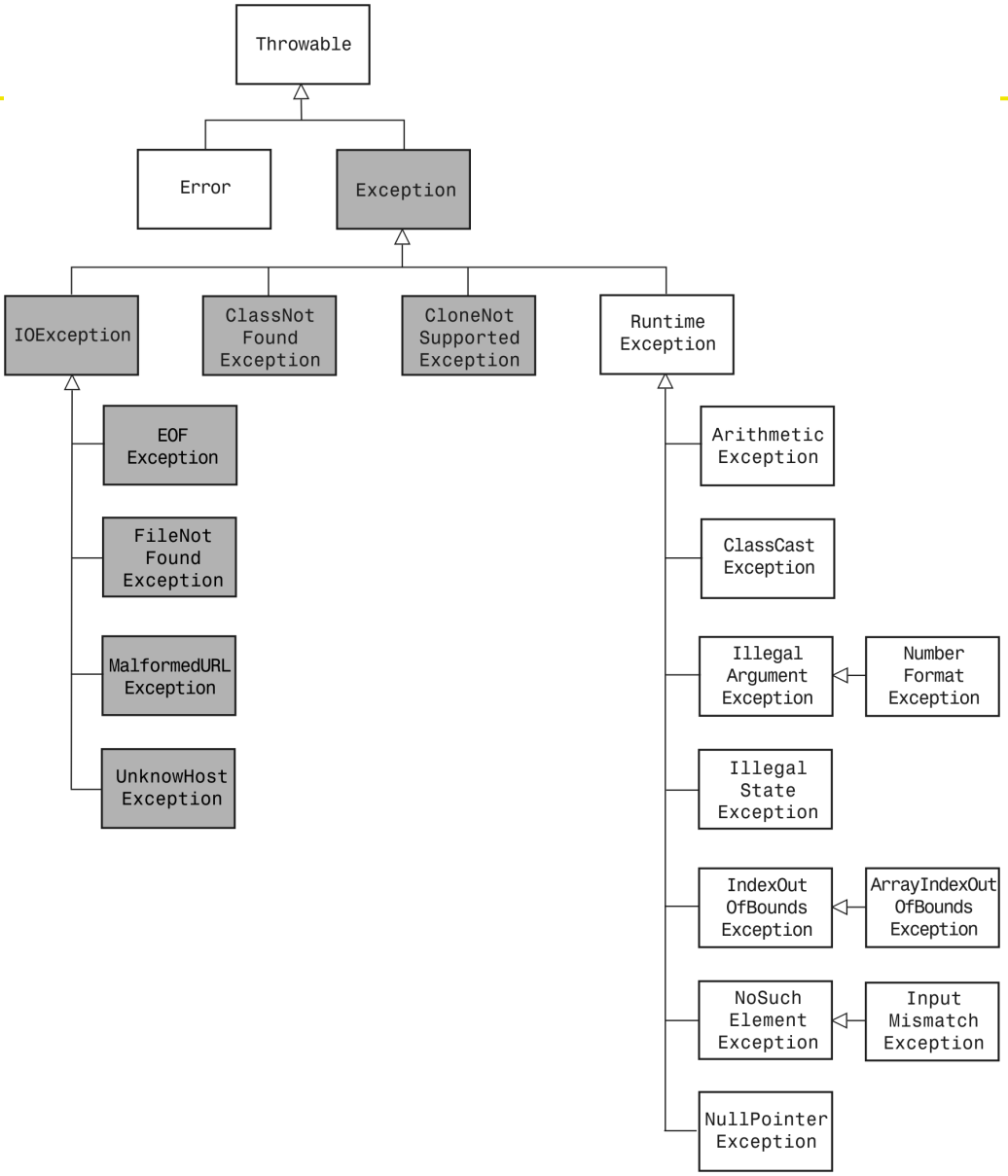

Figura 1: La gerarchia delle classi di eccezioni

Sintassi di Java 11.1 Lanciare un'eccezione

```
throw oggettoEccezione;
```

Esempio:

```
throw new IllegalArgumentException();
```

Obiettivo:

Lanciare un'eccezione e trasferire il controllo a un gestore per tale tipo di eccezione

Eccezioni controllate e non controllate

- In Java le eccezioni ricadono entro due categorie:
  - *Controllate*
    - Il compilatore verifica che l'eccezione non venga ignorata.
    - Le eccezioni controllate sono dovute a circostanze esterne che il programmatore non può evitare
    - La maggior parte delle eccezioni controllate vengono utilizzate nella gestione dei dati in ingresso o in uscita, che è un fertile terreno per guasti esterni che non sono sotto il vostro controllo
    - Tutte le sottoclassi di `IOException` sono eccezioni controllate

Eccezioni controllate e non controllate

- Le eccezioni Java ricadono entro due categorie:
  - *Non controllate*
    - Le eccezioni non controllate estendono la classe `RuntimeException` o `Error`
    - Le eccezioni non controllate rappresentano *un vostro errore*
    - Le sottoclassi di `RuntimeException` sono *non controllate*. Esempio:

```
NumberFormatException
IllegalArgumentException
NullPointerException
```

    - Esempio di errore: `OutOfMemoryError`

Eccezioni controllate e non controllate

- Queste categorie non sono perfette
  - Il metodo `Scanner.nextInt` lancia `InputMismatchException` che è un'eccezione non controllata
  - Il programmatore non può impedire che l'utente inserisca dati sbagliati
  - I progettisti della classe `Scanner` hanno fatto questa scelta per rendere più agevole l'utilizzo della classe da parte dei programmatori inesperti
- Avrete bisogno di gestire eccezioni controllate principalmente quando programmate con file e flussi

Segue...

Eccezioni controllate e non controllate

- Potete usare la classe `Scanner` anche per leggere dati da un file:

```
String filename = . . .;
FileReader reader = new FileReader(filename);
Scanner in = new Scanner(reader);
```

- Il costruttore di `FileReader` può lanciare una `FileNotFoundException`, che è un'eccezione non controllata

Eccezioni controllate e non controllate

- Avete due possibilità:
  - Potete usare le tecniche che vedrete nel Paragrafo 11.4
  - Potete dire al compilatore che siete consapevoli di questa eccezione e che volete che il vostro metodo termini la sua esecuzione quando essa viene lanciata
    - Aggiungete il marcatore `throws` a un metodo che può lanciare un'eccezione controllata

```
public void read(String filename) throws FileNotFoundException
{
   FileReader reader = new FileReader(filename);
   Scanner in = new Scanner(reader);
   . . .
}
```

Segue…

Eccezioni controllate e non controllate

- Se il vostro metodo può lanciare più eccezioni controllate, separate con virgole i nomi delle relative classi:

```
public void read(String filename)
        throws IOException, ClassNotFoundException
```

- Ricordate sempre che le classi che rappresentano eccezioni costituiscono una gerarchia di ereditarietà

- Se un metodo può lanciare sia `FileNotFoundException` sia `IOException`, potete contrassegnarlo solamente con `IOException`

- Non gestire un'eccezione quando sapete che essa può accadere sembra un comportamento irresponsabile; in realtà, non catturare un'eccezione è proprio la cosa migliore, quando non sapete come rimediare al problema

Sintassi di Java 11.2 Specificare un'eccezione

```
specificatoreDiAccesso valoreRestituito
      nomeMetodo (TipoParametro nomeParametro, . . .)
            throws ClasseEccezione1, ClasseEccezione2, . . .
```

Esempio:

```
public void read(BufferedReader in) throws IOException
```

Obiettivo:

Segnalare le eccezioni controllate che possono essere lanciate dal metodo

Catturare eccezioni

- Dovete installare gestori di eccezioni per tutte le eccezioni che possono essere lanciate. Un gestore di eccezioni si installa con l'enunciato `try/catch`
- Ciascun blocco `try` contiene una o più invocazioni di metodi che possono provocare il lancio di un'eccezione
- Ciascun blocco `catch` (`IOException exception`) contiene codice che viene eseguito quando viene lanciata un'eccezione di tipo `IOException`

Segue…

Catturare eccezioni

- Esempio:

```
try
{
   String filename = . . .;
   FileReader reader = new FileReader(filename);
   Scanner in = new Scanner(reader);
   String input = in.next();
   int value = Integer.parseInt(input);
   . . .
}
catch (IOException exception)
{
   exception.printStackTrace();
}
catch (NumberFormatException exception)
{
   System.out.println("Input was not a number");
}
```

Catturare eccezioni

- Se un'eccezione viene lanciata, le rimanenti istruzioni del blocco `try` non vengono eseguite
- Quando viene eseguito il blocco `catch` (`IOException exception`), significa che qualcuno dei metodi presenti nel blocco `try` ha lanciato un oggetto eccezione di tipo `IOException`
- Un'eccezione di tipo `NoSuchElementException` *non viene catturata* da nessuna delle clausole `catch`, per cui l'eccezione rimane lanciata finché non viene catturata da un altro blocco `try`
- È importante ricordare che dovreste inserire clausole `catch` *soltanto* in metodi in cui potete gestire con competenza un particolare tipo di eccezione

Segue…

Catturare eccezioni

- Blocco `catch (IOException exception)`:
  - `exception`: qualcuno dei metodi presenti nel blocco `try` ha lanciato un oggetto eccezione
  - `catch`: può esaminare tale oggetto per identificare maggiori dettagli sul guasto
  - `exception.printStackTrace()`: potete ottenere un elenco della catena di invocazioni di metodi che ha portato all'eccezione

Sintassi di Java 11.3
Blocco `try` generico

```
try
{
   enunciato
   enunciato
   . . .
}
catch (ClasseEccezione1 oggettoEccezione)
{
   enunciato
   enunciato
   . . .
}
catch (ClasseEccezione2 oggettoEccezione)
{
   enunciato
   enunciato
   . . .
}
. . .
```

Segue...

Sintassi di Java 14.3 Blocco `try` generico

Esempio:

```
try
{
   System.out.println("How old are you?");
   int age = in.nextInt();
   System.out.println("Next year, you'll be " + (age + 1));
}
catch (InputMismatchException exception)
{
   exception.printStackTrace();
}
```

Obiettivo:

Eseguire uno o più enunciati che possono lanciare eccezioni. Se viene lanciata un'eccezione di un tipo particolare, viene interrotta l'esecuzione di tali enunciati, per procedere, invece, con la clausola `catch` corrispondente. Se non viene lanciata alcuna eccezione, oppure viene lanciata un'eccezione che non ha una clausola catch corrispondente le clausole `catch` non vengono eseguite.

La clausola `finally`

- Esempio:

```
PrintWriter out = new PrinterWriter(filename);
writeData(out);
out.close();
// può darsi che non si arrivi mai fino a qui
```

- Supponete che uno dei metodi che precedono l'ultima linea lanci un'eccezione: in questo caso l'invocazione del metodo `close` non viene mai eseguita!

La clausola `finally`

- Risolvete questo problema inserendo la chiamata a `close` all'interno di una clausola `finally`

```
PrintWriter out = new PrintWriter(filename);

try

{
      writeData(out);

}

finally
{
       out.close();
}
```

- Una volta che è iniziata l'esecuzione di un blocco try, si ha la garanzia che gli enunciati di una clausola finally vengano eseguiti, indipendentemente dal fatto che venga lanciata un'eccezione oppure no.

La clausola `finally`

- Il codice della clausola `finally` viene eseguito al termine del blocco `try`, in una delle seguenti situazioni:
  - Dopo aver portato a termine l'ultimo enunciato del blocco `try`
  - Dopo aver portato a termine l'ultimo enunciato di una clausola `catch` che abbia catturato un'eccezione lanciata nel blocco `try`
  - Quando nel blocco `try` è stata lanciata un'eccezione che non viene catturata

- Vi raccomandiamo di non usare clausole `catch` e `finally` nel medesimo blocco `try`

Sintassi di Java 11.4 Clausola `finally`

```
try
{
   enunciato
   enunciato
   . . .
}
finally
{
   enunciato
   enunciato
   . . .
}
```

Segue…

Sintassi di Java 14.4 Clausola `finally`

Esempio:

```
PrintWriter out = new PrintWriter(filename);
try
{
   writeData(out);
}
finally
{
   out.close();
}
```

Obiettivo:

Garantire che gli enunciati presenti nelle clausola `finally` vengano eseguiti indipendentemente dal fatto che nel blocco `try` venga lanciata un'eccezione.

Progettare i vostri tipi di eccezione

- Potete progettare i vostri tipi di eccezioni come sottoclassi di `Exception` o di `RuntimeException`

```
if (amount > balance)
{
   throw new InsufficientFundsException(
      "withdrawal of " + amount +
      " exceeds balance of " + balance);
}
```

- Decidiamo che il programmatore avrebbe dovuto evitare la condizione d'errore: non dovrebbe essere difficile verificare che sia `amount <=account.getBalance()` prima di invocare il metodo `withdraw`

Segue…

Progettare i vostri tipi di eccezione

- L'eccezione dovrebbe essere non controllata ed estendere la classe `RuntimeException` o una delle sue sottoclassi
- Solitamente in una classe di eccezioni si forniscono due costruttori:
  - Un costruttore senza argomenti
  - Un costruttore che accetti una stringa come messaggio che descrive il motivo dell'eccezione

Progettare i vostri tipi di eccezione

```
public class InsufficientFundsException
      extends RuntimeException
{
   public InsufficientFundsException() {}

   public InsufficientFundsException(String message)
   {
      super(message);
   }
}
```