Capitolo 11

Eccezioni

Sommario: Eccezioni

Eccezioni

- Durante l'esecuzione si possono verifica anomalie che non consentono alla JVM di proseguire la normale esecuzione

Eccezioni

- Durante l'esecuzione si possono verifica anomalie che non consentono alla JVM di proseguire la normale esecuzione
- Tali anomalie vengono segnalate utilizzando particolari oggetti detti eccezioni

Eccezioni

- Durante l'esecuzione si possono verifica anomalie che non consentono alla JVM di proseguire la normale esecuzione
- Tali anomalie vengono segnalate utilizzando particolari oggetti detti eccezioni
- Il linguaggio prevede appositi meccanismi per trattare questo tipo di oggetti

Esempio

```
public static void main(String[] args) {
  ...
  int quoziente = calcolaQuoziente(dividendo, divisore);
  ...
}

private static int calcolaQuoziente(int x, int y) {
  return x / y;
}
```

Esempio

```
public static void main(String[] args) {
  ...
  int quoziente = calcolaQuoziente(dividendo, divisore);
  ...
}

private static int calcolaQuoziente(int x, int y) {
  return x / y;
}
```

Esecuzione

```
Inserisci il dividendo 8
Inserisci il divisore 0
Exception in thread "main"
        java.lang.ArithmeticException: / by zero
      at Divisione.calcolaQuoziente(Divisione.java:16)
      at Divisione.main(Divisione.java:10)
```

Esempio

```
public static void main(String[] args) {
   ...
   String s = in.readLine("Inserisci una stringa ");
   int indice = in.readInt("Inserisci una posizione ");
   char x = s.charAt(indice);
   ...
}
```

Esempio

```
public static void main(String[] args) {
  ...
  String s = in.readLine("Inserisci una stringa ");
  int indice = in.readInt("Inserisci una posizione ");
  char x = s.charAt(indice);
  ...
}
```

Esecuzione

```
Inserisci una stringa pippo
Inserisci una posizione 7
Exception in thread "main"
        java.lang.StringIndexOutOfBoundsException:
        String index out of range: 7
      at java.lang.String.charAt(String.java:460)
      at PrelevaCarattere.main(PrelevaCarattere.java:10)
```

Eccezioni

- Le eccezioni sono oggetti

Eccezioni

- Le eccezioni sono oggetti
- Sollevando un'eccezione, la JVM crea un'istanza della classe che descrive l'evento verificatosi

Eccezioni

- Le eccezioni sono oggetti
- Sollevando un'eccezione, la JVM crea un'istanza della classe che descrive l'evento verificatosi
- Nei messaggi d'errore il nome indicato per l'eccezione è il nome di questa classe

  `java.lang.ArithmeticException`
  `java.lang.StringIndexOutOfBoundsException`

Prevenire le eccezioni

Nei due casi precedenti si poteva evitare il verificarsi dell'anomalia inserendo opportuni controlli

Prevenire le eccezioni

Nei due casi precedenti si poteva evitare il verificarsi dell'anomalia inserendo opportuni controlli

Esempio

```
if (indice >= 0 && indice < s.length()) {
  char x = s.charAt(indice);
  out.println("Il carattere in posizione " + indice +
              " della stringa " + s + " e' " + x);
} else
    out.println("Non esiste alcun carattere in posizione "
                + indice);
```

Prevenire le eccezioni

Nei due casi precedenti si poteva evitare il verificarsi dell'anomalia inserendo opportuni controlli

Esempio

```
if (indice >= 0 && indice < s.length()) {
  char x = s.charAt(indice);
  out.println("Il carattere in posizione " + indice +
              " della stringa " + s + " e' " + x);
} else
    out.println("Non esiste alcun carattere in posizione "
                + indice);
```

In altri casi l'inserimento di controlli non rappresenta la soluzione migliore:

Prevenire le eccezioni

Nei due casi precedenti si poteva evitare il verificarsi dell'anomalia inserendo opportuni controlli

Esempio

```
if (indice >= 0 && indice < s.length()) {
  char x = s.charAt(indice);
  out.println("Il carattere in posizione " + indice +
              " della stringa " + s + " e' " + x);
} else
    out.println("Non esiste alcun carattere in posizione "
                + indice);
```

In altri casi l'inserimento di controlli non rappresenta la soluzione migliore:

- il punto in cui l'eccezione può essere sollevata è lontano dal punto in cui intervenire per porvi rimedio

Prevenire le eccezioni

Nei due casi precedenti si poteva evitare il verificarsi dell'anomalia inserendo opportuni controlli

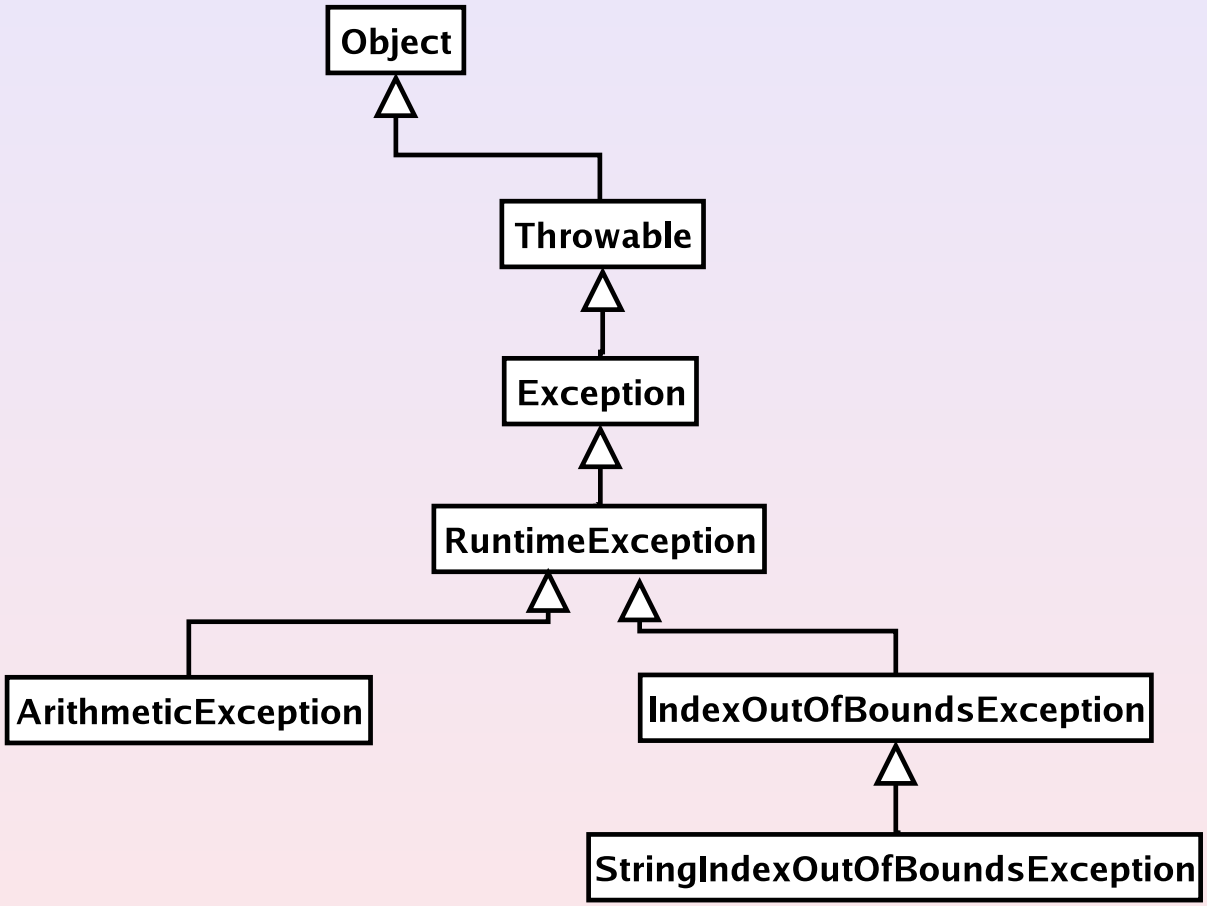
Esempio

```
if (indice >= 0 && indice < s.length()) {
  char x = s.charAt(indice);
  out.println("Il carattere in posizione " + indice +
              " della stringa " + s + " e' " + x);
} else
    out.println("Non esiste alcun carattere in posizione "
                + indice);
```

In altri casi l'inserimento di controlli non rappresenta la soluzione migliore:

- il punto in cui l'eccezione può essere sollevata è lontano dal punto in cui intervenire per porvi rimedio
- è impossibile prevenire l'eccezione (anomalie dovute a eventi esterni)

Gerarchia delle eccezioni di Java



Eccezioni

- Se durante l'esecuzione di un metodo o di un costruttore si verifica un'anomalia viene sollevata un'eccezione del tipo opportuno

Eccezioni

- Se durante l'esecuzione di un metodo o di un costruttore si verifica un'anomalia viene sollevata un'eccezione del tipo opportuno
- Se il metodo o il costruttore è in grado di gestire l'eccezione lo fa

Eccezioni

- Se durante l'esecuzione di un metodo o di un costruttore si verifica un'anomalia viene sollevata un'eccezione del tipo opportuno
- Se il metodo o il costruttore è in grado di gestire l'eccezione lo fa
- Altrimenti l'eccezione viene rinviata al metodo chiamante

  ...cioè l'eccezione viene sollevata nel punto in cui è stato invocato il metodo in cui si è verificata

Eccezioni

- Se durante l'esecuzione di un metodo o di un costruttore si verifica un'anomalia viene sollevata un'eccezione del tipo opportuno
- Se il metodo o il costruttore è in grado di gestire l'eccezione lo fa
- Altrimenti l'eccezione viene rinviata al metodo chiamante

  ...cioè l'eccezione viene sollevata nel punto in cui è stato invocato il metodo in cui si è verificata

  ...a sua volta il metodo può gestirla localmente o rinviarla al chiamante

Intercettare: il costrutto `try-catch`

Sintassi

```
try {
  // Codice che potrebbe generare eccezioni
  blocco_try
}
catch (Tipo_Eccezione1 id1) {
  // Gestisce le eccezioni Tipo_Eccezione1
  blocco_gestore1
} catch (Tipo_Eccezione2 id2) {
  // Gestisce le eccezioni Tipo_Eccezione2
  blocco_gestore2
}
...
```

Esecuzione

```
try {
  // Codice che potrebbe generare eccezioni
  blocco_try
}
catch (Tipo_Eccezione1 id1) {
  // Gestisce le eccezioni Tipo_Eccezione1
  blocco_gestore1
} catch (Tipo_Eccezione2 id2) {
  // Gestisce le eccezioni Tipo_Eccezione2
  blocco_gestore2
}
...
```

La JVM esegue le istruzioni presenti nel blocco `try`

Esecuzione

```
try {
  // Codice che potrebbe generare eccezioni
  blocco_try
}
catch (Tipo_Eccezione1 id1) {
  // Gestisce le eccezioni Tipo_Eccezione1
  blocco_gestore1
} catch (Tipo_Eccezione2 id2) {
  // Gestisce le eccezioni Tipo_Eccezione2
  blocco_gestore2
}
...
```

La JVM esegue le istruzioni presenti nel blocco `try`

- Se non viene sollevata alcuna eccezione l'esecuzione prosegue con la prima istruzione successiva al blocco `try-catch`

Esecuzione

```
try {
  // Codice che potrebbe generare eccezioni
  blocco_try
}
catch (Tipo_Eccezione1 id1) {
  // Gestisce le eccezioni Tipo_Eccezione1
  blocco_gestore1
} catch (Tipo_Eccezione2 id2) {
  // Gestisce le eccezioni Tipo_Eccezione2
  blocco_gestore2
}
...
```

La JVM esegue le istruzioni presenti nel blocco `try`

- Se non viene sollevata alcuna eccezione l'esecuzione prosegue con la prima istruzione successiva al blocco `try-catch`
- La JVM interrompe l'esecuzione nel punto in cui si è verificata l'eccezione...

Esecuzione

```
try {
  // Codice che potrebbe generare eccezioni
  blocco_try
}
catch (Tipo_Eccezione1 id1) {
  // Gestisce le eccezioni Tipo_Eccezione1
  blocco_gestore1
} catch (Tipo_Eccezione2 id2) {
  // Gestisce le eccezioni Tipo_Eccezione2
  blocco_gestore2
}
...
```

. . . quindi scorre la lista degli argomenti dei blocchi `catch` alla ricerca di un tipo cui possa essere ricondotto quello dell'eccezione sollevata

Esecuzione

```
try {
  // Codice che potrebbe generare eccezioni
  blocco_try
}
catch (Tipo_Eccezione1 id1) {
  // Gestisce le eccezioni Tipo_Eccezione1
  blocco_gestore1
} catch (Tipo_Eccezione2 id2) {
  // Gestisce le eccezioni Tipo_Eccezione2
  blocco_gestore2
}
...
```

. . . quindi scorre la lista degli argomenti dei blocchi `catch` alla ricerca di un tipo cui possa essere ricondotto quello dell'eccezione sollevata

- se lo trova esegue il corrispondente blocco `catch`

Esecuzione

```
try {
  // Codice che potrebbe generare eccezioni
  blocco_try
}
catch (Tipo_Eccezione1 id1) {
  // Gestisce le eccezioni Tipo_Eccezione1
  blocco_gestore1
} catch (Tipo_Eccezione2 id2) {
  // Gestisce le eccezioni Tipo_Eccezione2
  blocco_gestore2
}
...
```

. . . quindi scorre la lista degli argomenti dei blocchi `catch` alla ricerca di un tipo cui possa essere ricondotto quello dell'eccezione sollevata

- se lo trova esegue il corrispondente blocco `catch`
- altrimenti prosegue l'esecuzione esattamente come se il blocco `try-catch` non fosse presente

Esempio

```
public static void main(String[] args) {
  ...
  String s = in.readLine("Inserisci una stringa ");
  int indice = in.readInt("Inserisci una posizione ");
  try {
    char x = s.charAt(indice);
    out.println("Il carattere in posizione " + indice +
                " della stringa " + s + " e' " + x);
  } catch (StringIndexOutOfBoundsException e) {
      out.println("Non esiste alcun carattere in posizione "
                  + indice);
  }
}
```

Parametro di `catch`

```
  ...
  try {
    char x = s.charAt(indice);
    out.println("Il carattere in posizione " + indice +
                " della stringa " + s + " e' " + x);
  } catch (StringIndexOutOfBoundsException e) {
    ...
  }
}
```

- Quando viene sollevata un'eccezione, viene creato un oggetto che descrive quanto è accaduto

Parametro di `catch`

```
  ...
  try {
    char x = s.charAt(indice);
    out.println("Il carattere in posizione " + indice +
                " della stringa " + s + " e' " + x);
  } catch (StringIndexOutOfBoundsException e) {
    ...
  }
}
```

- Quando viene sollevata un'eccezione, viene creato un oggetto che descrive quanto è accaduto
- Nel blocco `catch` che gestisce l'eccezione è possibile utilizzare tale oggetto (tramite l'identificatore dichiarato nel blocco `catch`)

Esempio

```
public static void main(String[] args) {
  ...
  String s = in.readLine("Inserisci una stringa ");
  int indice = in.readInt("Inserisci una posizione ");
  try {
    char x = s.charAt(indice);
    out.println("Il carattere in posizione " + indice +
                " della stringa " + s + " e' " + x);
  } catch (StringIndexOutOfBoundsException e) {
      out.println(e.toString());
  }
}
```

Rientro dai metodi ed eccezioni

- Normalmente, quando l'esecuzione di un metodo termina, viene ripresa l'esecuzione del metodo chiamante

Rientro dai metodi ed eccezioni

- Normalmente, quando l'esecuzione di un metodo termina, viene ripresa l'esecuzione del metodo chiamante
- Quando in un metodo viene sollevata un'eccezione:
  - se è in grado di gestirla l'esecuzione prosegue nel solito modo

Rientro dai metodi ed eccezioni

- Normalmente, quando l'esecuzione di un metodo termina, viene ripresa l'esecuzione del metodo chiamante
- Quando in un metodo viene sollevata un'eccezione:
  - se è in grado di gestirla l'esecuzione prosegue nel solito modo
  - altrimenti la sua esecuzione viene conclusa e l'eccezione viene delegata al metodo chiamante:

Rientro dai metodi ed eccezioni

- Normalmente, quando l'esecuzione di un metodo termina, viene ripresa l'esecuzione del metodo chiamante
- Quando in un metodo viene sollevata un'eccezione:
  - se è in grado di gestirla l'esecuzione prosegue nel solito modo
  - altrimenti la sua esecuzione viene conclusa e l'eccezione viene delegata al metodo chiamante:
    - nello stack viene distrutto il record di attivazione

Rientro dai metodi ed eccezioni

- Normalmente, quando l'esecuzione di un metodo termina, viene ripresa l'esecuzione del metodo chiamante
- Quando in un metodo viene sollevata un'eccezione:
  - se è in grado di gestirla l'esecuzione prosegue nel solito modo
  - altrimenti la sua esecuzione viene conclusa e l'eccezione viene delegata al metodo chiamante:
    - nello stack viene distrutto il record di attivazione
    - l'eccezione viene sollevata nel punto in cui il metodo è stato chiamato

Rientro dai metodi ed eccezioni

- Normalmente, quando l'esecuzione di un metodo termina, viene ripresa l'esecuzione del metodo chiamante
- Quando in un metodo viene sollevata un'eccezione:
  - se è in grado di gestirla l'esecuzione prosegue nel solito modo
  - altrimenti la sua esecuzione viene conclusa e l'eccezione viene delegata al metodo chiamante:
    - nello stack viene distrutto il record di attivazione
    - l'eccezione viene sollevata nel punto in cui il metodo è stato chiamato
- Un'eccezione viene propagata nella catena dei chiamanti:
  - finché non si trova un metodo o un costruttore che la intercetti

Rientro dai metodi ed eccezioni

- Normalmente, quando l'esecuzione di un metodo termina, viene ripresa l'esecuzione del metodo chiamante
- Quando in un metodo viene sollevata un'eccezione:
  - se è in grado di gestirla l'esecuzione prosegue nel solito modo
  - altrimenti la sua esecuzione viene conclusa e l'eccezione viene delegata al metodo chiamante:
    - nello stack viene distrutto il record di attivazione
    - l'eccezione viene sollevata nel punto in cui il metodo è stato chiamato
- Un'eccezione viene propagata nella catena dei chiamanti:
  - finché non si trova un metodo o un costruttore che la intercetti
  - o fino a provocare l'interruzione dell'esecuzione della JVM

Espressioni in notazione postfissa

- Consideriamo espressioni aritmetiche che comprendano:
  - costanti intere
  - operatori: +, -, *, /

Espressioni in notazione postfissa

- Consideriamo espressioni aritmetiche che comprendano:
  - costanti intere
  - operatori: `+`, `-`, `*`, `/`
- Un'espressione in notazione postfissa è una costante, oppure una coppia di espressioni in notazione postfissa seguite da un operatore

Espressioni in notazione postfissa

- Consideriamo espressioni aritmetiche che comprendano:
  - costanti intere
  - operatori: `+`, `-`, `*`, `/`
- Un'espressione in notazione postfissa è una costante, oppure una coppia di espressioni in notazione postfissa seguite da un operatore

Esempio

```
3 5 +    corrisponde a    3 + 5
4 3 *    corrisponde a    4 * 3
```

Notazione postfissa

- Nella notazione infissa per stabilire in che ordine applicare gli operatori bisogna utilizzare le regole di precedenza oppure le parentesi

Notazione postfissa

- Nella notazione infissa per stabilire in che ordine applicare gli operatori bisogna utilizzare le regole di precedenza oppure le parentesi
- La notazione postfissa non richiede parentesi o regole di precedenza per la valutazione delle espressioni

Notazione postfissa

- Nella notazione infissa per stabilire in che ordine applicare gli operatori bisogna utilizzare le regole di precedenza oppure le parentesi
- La notazione postfissa non richiede parentesi o regole di precedenza per la valutazione delle espressioni

Esempio

```
3 5 + 4 3 * +   corrisponde a   (3 + 5) + (4 * 3)
```

Notazione postfissa

- Nella notazione infissa per stabilire in che ordine applicare gli operatori bisogna utilizzare le regole di precedenza oppure le parentesi
- La notazione postfissa non richiede parentesi o regole di precedenza per la valutazione delle espressioni

Esempio

```
3 5 + 4 3 * +   corrisponde a   (3 + 5) + (4 * 3)
```

- Un'espressione in notazione postfissa viene valutata applicando ogni operazione ai due valori o ai risultati di altre operazioni che la precedono

Operativamente

Un'espressione in notazione postfissa può essere vista come un'espressione che definisce dei passi elementari su una pila (stack) di numeri.

Operativamente

Un'espressione in notazione postfissa può essere vista come un'espressione che definisce dei passi elementari su una pila (stack) di numeri.

Le operazioni elementari sono:

- Se il prossimo elemento dell'espressione è una costante, metti la costante nella pila (push)

Operativamente

Un'espressione in notazione postfissa può essere vista come un'espressione che definisce dei passi elementari su una pila (stack) di numeri.

Le operazioni elementari sono:

- Se il prossimo elemento dell'espressione è una costante, metti la costante nella pila (push)
- Se il prossimo elemento dell'espressione è un operatore:

Operativamente

Un'espressione in notazione postfissa può essere vista come un'espressione che definisce dei passi elementari su una pila (stack) di numeri.

Le operazioni elementari sono:

- Se il prossimo elemento dell'espressione è una costante, metti la costante nella pila (push)
- Se il prossimo elemento dell'espressione è un operatore:
  - preleva i primi due elementi dalla pila (due pop)

Operativamente

Un'espressione in notazione postfissa può essere vista come un'espressione che definisce dei passi elementari su una pila (stack) di numeri.

Le operazioni elementari sono:

- Se il prossimo elemento dell'espressione è una costante, metti la costante nella pila (push)
- Se il prossimo elemento dell'espressione è un operatore:
  - preleva i primi due elementi dalla pila (due pop)
  - applicagli l'operatore

Operativamente

Un'espressione in notazione postfissa può essere vista come un'espressione che definisce dei passi elementari su una pila (stack) di numeri.

Le operazioni elementari sono:

- Se il prossimo elemento dell'espressione è una costante, metti la costante nella pila (push)
- Se il prossimo elemento dell'espressione è un operatore:
  - preleva i primi due elementi dalla pila (due pop)
  - applicagli l'operatore
  - metti il risultato nella pila

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -		push

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + -	3	push

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + -	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + -	3 5	push

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + -	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + -	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + -	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + -	3 6	**valuta** 3 6 *

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + -	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + -	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + -	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + -	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + -	18	push

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + -		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + -	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + -	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + -	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + -	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + -	18	push
5 + 2 + 3 1 + 4 * + -	18 2	push

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + −	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + −	18	push
5 + 2 + 3 1 + 4 * + −	18 2	push
+ 2 + 3 1 + 4 * + −	18 2 5	**valuta** 2 5 +

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + −	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + −	18	push
5 + 2 + 3 1 + 4 * + −	18 2	push
+ 2 + 3 1 + 4 * + −	18 2 5	**valuta** 2 5 +
2 + 3 1 + 4 * + −	18 7	push

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + −	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + −	18	push
5 + 2 + 3 1 + 4 * + −	18 2	push
+ 2 + 3 1 + 4 * + −	18 2 5	**valuta** 2 5 +
2 + 3 1 + 4 * + −	18 7	push
+ 3 1 + 4 * + −	18 7 2	**valuta** 7 2 +

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + −	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + −	18	push
5 + 2 + 3 1 + 4 * + −	18 2	push
+ 2 + 3 1 + 4 * + −	18 2 5	**valuta** 2 5 +
2 + 3 1 + 4 * + −	18 7	push
+ 3 1 + 4 * + −	18 7 2	**valuta** 7 2 +
3 1 + 4 * + −	18 9	push

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + −	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + −	18	push
5 + 2 + 3 1 + 4 * + −	18 2	push
+ 2 + 3 1 + 4 * + −	18 2 5	**valuta** 2 5 +
2 + 3 1 + 4 * + −	18 7	push
+ 3 1 + 4 * + −	18 7 2	**valuta** 7 2 +
3 1 + 4 * + −	18 9	push
1 + 4 * + −	18 9 3	push

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + −	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + −	18	push
5 + 2 + 3 1 + 4 * + −	18 2	push
+ 2 + 3 1 + 4 * + −	18 2 5	**valuta** 2 5 +
2 + 3 1 + 4 * + −	18 7	push
+ 3 1 + 4 * + −	18 7 2	**valuta** 7 2 +
3 1 + 4 * + −	18 9	push
1 + 4 * + −	18 9 3	push
+ 4 * + −	18 9 3 1	**valuta** 3 1 +

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + −	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + −	18	push
5 + 2 + 3 1 + 4 * + −	18 2	push
+ 2 + 3 1 + 4 * + −	18 2 5	**valuta** 2 5 +
2 + 3 1 + 4 * + −	18 7	push
+ 3 1 + 4 * + −	18 7 2	**valuta** 7 2 +
3 1 + 4 * + −	18 9	push
1 + 4 * + −	18 9 3	push
+ 4 * + −	18 9 3 1	**valuta** 3 1 +
4 * + −	18 9 4	push

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + −	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + −	18	push
5 + 2 + 3 1 + 4 * + −	18 2	push
+ 2 + 3 1 + 4 * + −	18 2 5	**valuta** 2 5 +
2 + 3 1 + 4 * + −	18 7	push
+ 3 1 + 4 * + −	18 7 2	**valuta** 7 2 +
3 1 + 4 * + −	18 9	push
1 + 4 * + −	18 9 3	push
+ 4 * + −	18 9 3 1	**valuta** 3 1 +
4 * + −	18 9 4	push
* + −	18 9 4 4	**valuta** 4 4 *

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + −	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + −	18	push
5 + 2 + 3 1 + 4 * + −	18 2	push
+ 2 + 3 1 + 4 * + −	18 2 5	**valuta** 2 5 +
2 + 3 1 + 4 * + −	18 7	push
+ 3 1 + 4 * + −	18 7 2	**valuta** 7 2 +
3 1 + 4 * + −	18 9	push
1 + 4 * + −	18 9 3	push
+ 4 * + −	18 9 3 1	**valuta** 3 1 +
4 * + −	18 9 4	push
* + −	18 9 4 4	**valuta** 4 4 *
+ −	18 9 16	**valuta** 9 16 +

Valutazione: 3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −

espressione da leggere	pila	operazione
3 5 1 + * 2 5 + 2 + 3 1 + 4 * + −		push
5 1 + * 2 5 + 2 + 3 1 + 4 * + −	3	push
1 + * 2 5 + 2 + 3 1 + 4 * + −	3 5	push
+ * 2 5 + 2 + 3 1 + 4 * + −	3 5 1	**valuta** 5 1 +
* 2 5 + 2 + 3 1 + 4 * + −	3 6	**valuta** 3 6 *
2 5 + 2 + 3 1 + 4 * + −	18	push
5 + 2 + 3 1 + 4 * + −	18 2	push
+ 2 + 3 1 + 4 * + −	18 2 5	**valuta** 2 5 +
2 + 3 1 + 4 * + −	18 7	push
+ 3 1 + 4 * + −	18 7 2	**valuta** 7 2 +
3 1 + 4 * + −	18 9	push
1 + 4 * + −	18 9 3	push
+ 4 * + −	18 9 3 1	**valuta** 3 1 +
4 * + −	18 9 4	push
* + −	18 9 4 4	**valuta** 4 4 *
+ −	18 9 16	**valuta** 9 16 +
−	18 25	**valuta** 18 25 /
	−7	

L'applicazione

- Per semplicità supporremo di ricevere gli elementi che costituiscono l'espressione su righe separate.

L'applicazione

- Per semplicità supporremo di ricevere gli elementi che costituiscono l'espressione su righe separate.
    - Una riga può contenere un numero intero privo di segno oppure uno dei quattro simboli di operazione: +, −, * o /

L'applicazione

- Per semplicità supporremo di ricevere gli elementi che costituiscono l'espressione su righe separate.
  - Una riga può contenere un numero intero privo di segno oppure uno dei quattro simboli di operazione: +, −, * o /
  - La fine dell'espressione sarà indicata da una riga contenente il solo carattere =

L'applicazione

- Per semplicità supporremo di ricevere gli elementi che costituiscono l'espressione su righe separate.
  - Una riga può contenere un numero intero privo di segno oppure uno dei quattro simboli di operazione: +, −, * o /
  - La fine dell'espressione sarà indicata da una riga contenente il solo carattere =
- Svilupperemo l'applicazione nell'ipotesi che l'input fornito dall'utente abbia il formato corretto e che non si verifichino condizioni eccezionali

La classe Stack<E>

La classe Stack<E> è sottoclasse di Vector, è definita nel package java.util

La classe Stack<E>

La classe `Stack<E>` è sottoclasse di `Vector`, è definita nel package `java.util`

Contratto: Stack<E>

Le sue istanze rappresentano pile di oggetti di tipo `E`.

La classe Stack<E>

La classe `Stack<E>` è sottoclasse di `Vector`, è definita nel package `java.util`

Contratto: Stack<E>

Le sue istanze rappresentano pile di oggetti di tipo `E`.

Costruttore

- `public Stack()`
  Crea la pila vuota.

La classe Stack<E>

Metodi

- `public void push(E o)`
  Aggiunge in cima alla pila che esegue il metodo l'oggetto di cui riceve il riferimento tramite il parametro.

La classe Stack<E>

Metodi

- `public void push(E o)`
  Aggiunge in cima alla pila che esegue il metodo l'oggetto di cui riceve il riferimento tramite il parametro.

- `public E pop()`
  Toglie dalla pila che esegue il metodo l'elemento che si trova più in alto restituendone il riferimento.

La classe Stack<E>

Metodi

- `public void push(E o)`
  Aggiunge in cima alla pila che esegue il metodo l'oggetto di cui riceve il riferimento tramite il parametro.

- `public E pop()`
  Toglie dalla pila che esegue il metodo l'elemento che si trova più in alto restituendone il riferimento.

- `public boolean empty()`
  Restituisce `true` se e solo se la pila che esegue il metodo è vuota.

Lo schema del main

```
crea una pila vuota
leggi una riga
while (ciò che hai letto è diverso da =) {
    elabora ciò che hai letto
    leggi una riga
}
preleva il risultato dalla cima della pila
comunica il risultato
```

Lo schema del `main`

```
Stack<Integer> pila = new Stack<Integer>();

out.println("Inserire l'espressione da esaminare");
out.println("(un elemento per riga, = per terminare)");
String riga = in.readLine();
while (!riga.equals("=")) {

  //elaborazione della riga letta
  ...

  riga = in.readLine();
}

Integer risultato = pila.pop();
out.println("Il risultato e' " + risultato);
```

Elaborazione della riga: lettura di un operatore

- Preleviamo due operandi (il primo operando prelevato dalla pila è quello di destra)

```
Integer dx = pila.pop();
Integer sx = pila.pop();
```

Elaborazione della riga: lettura di un operatore

- Preleviamo due operandi (il primo operando prelevato dalla pila è quello di destra)

```
Integer dx = pila.pop();
Integer sx = pila.pop();
```

- Calcoliamo il risultato

```
Integer risultato = calcola(sx, dx,operazione);
```

```
private static int calcola(int opSx, int opDx, char segno)

```

Elaborazione della riga: lettura di un operatore

- Preleviamo due operandi (il primo operando prelevato dalla pila è quello di destra)

```
Integer dx = pila.pop();
Integer sx = pila.pop();
```

- Calcoliamo il risultato

```
Integer risultato = calcola(sx, dx,operazione);
```

```
private static int calcola(int opSx, int opDx, char segno)
```

Sfruttiamo i meccanismi di unboxing e autoboxing dei tipi primitivi

Elaborazione della riga: lettura di un operatore

- Preleviamo due operandi (il primo operando prelevato dalla pila è quello di destra)

```
Integer dx = pila.pop();
Integer sx = pila.pop();
```

- Calcoliamo il risultato

```
Integer risultato = calcola(sx, dx,operazione);
```

```
private static int calcola(int opSx, int opDx, char segno)
```

Sfruttiamo i meccanismi di unboxing e autoboxing dei tipi primitivi

- Memorizziamo il risultato nella pila

```
pila.push(risultato);
```

Elaborazione della riga: lettura di un operando

- Se è stato letto un operando, è necessario porlo in cima alla pila. A tale scopo costruiamo un oggetto `Integer`, corrispondente alla stringa letta, che dovrà essere caricato in cima alla pila

```
Integer numero = new Integer(riga);
pila.push(numero);
```

Il metodo calcola

```
private static int calcola(int opSx, int opDx, char segno) {
  int ris = 0;
  switch (segno) {
    case '+':
      ris = opSx + opDx;
      break;
    case '-':
      ris = opSx - opDx;
      break;
    case '*':
      ris = opSx * opDx;
      break;
    case '/':
      ris = opSx / opDx;
      break;
  }
  return ris;
}
```

Il main

```
//crea la pila
Stack<Integer> pila = new Stack<Integer>();

out.println("Inserire l'espressione da esaminare");
out.println("(un elemento per riga, = per terminare)");
String riga = in.readLine();
while (!riga.equals("=")) {
  //elaborazione della riga letta
  ...
  //lettura di un'altra riga
  riga = in.readLine();
}

//preleva il risultato dalla pila e lo comunica
Integer risultato = pila.pop();
out.println("Il risultato e'" + risultato);
```

Il main

```
...
//elaborazione della riga letta
Integer sx, dx, risultato;
char selettore = riga.charAt(0);
switch (selettore) {
  case '+':
  case '-':
  case '*':
  case '/':
    dx = pila.pop();
    sx = pila.pop();
    risultato = calcola(sx, dx, selettore);
    pila.push(risultato);
    break;
  default: //se non e' operazione, dev'essere un numero
  Integer numero = new Integer(riga);
  pila.push(numero);
  break;
}
...
```

Situazioni anomale (1)

L'espressione contiene una divisione per zero

```
Inserire l'espressione da esaminare
(un elemento per riga, = per terminare)
5
4
+
0
/
Exception in thread "main" java.lang.ArithmeticException:
    / by zero
  at Postfissa.calcola(Postfissa.java:61)
  at Postfissa.main(Postfissa.java:29)
```

Situazioni anomale (2)

L'utente inserisce caratteri non corretti

```
Inserire l'espressione da esaminare
(un elemento per riga, = per terminare)
2
3
+
2ac
Exception in thread "main" java.lang.NumberFormatException:
For input string: "2ac"
  at java.lang.NumberFormatException.
                forInputString(NumberFormatException.java:48)
  at java.lang.Integer.parseInt(Integer.java:456)
  at java.lang.Integer.<init>(Integer.java:620)
  at Postfissa.main(Postfissa.java:34)
```

Situazioni anomale (3)

L'espressione contiene un numero insufficiente di operandi

```
Inserire l'espressione da esaminare
(un elemento per riga, = per terminare)
5
+
Exception in thread "main" java.util.EmptyStackException
  at java.util.Stack.peek(Stack.java:79)
  at java.util.Stack.pop(Stack.java:61)
  at Postfissa.main(Postfissa.java:28)
```

Situazioni anomale (4)

L'espressione contiene troppi operandi

```
Inserire l'espressione da esaminare
(un elemento per riga, = per terminare)
6
3
4
+
=
Il risultato e' 7
```

In questo caso non si verifica un errore in fase di esecuzione, ma il risultato è scorretto.

Trattare l'anomalia (4)

Se l'espressione è costruita correttamente, al termine dei calcoli deve restare sulla pila esclusivamente il risultato.

```
Integer risultato = pila.pop();
if (pila.empty())
  out.println("Il risultato e' " + risultato);
else
  out.println("L'espressione contiene troppi operandi");
```

Gestione delle altre anomalie

```
try {
  //il codice che segue e' esattamente quello della versione
  //precedente
  while (!riga.equals("=")) {
    ...
  }

  //preleva il risultato dalla pila e lo comunica
  Integer risultato = pila.pop();
  if (pila.empty())
    out.println("Il risultato e' " + risultato);
  else
    out.println("L'espressione contiene troppi operandi");
  //fine blocco try
} catch (Exception e) {
    out.println("Errore: " + e.toString());
}
```

Altra versione

```
try {
  //il codice che segue e' esattamente quello della versione
  //precedente
  while (!riga.equals("=")) {
    ...
  //fine blocco try
} catch (ArithmeticException e) {
    out.println("Errore: divisione per zero");
} catch (NumberFormatException e) {
    out.println("Input scorretto: " + e.toString());
} catch (EmptyStackException e) {
    out.println("L'espressione contiene un numero " +
                "insufficiente di operandi");
}
```

Eccezioni: `java.lang`

- `ArithmeticException`
  Viene sollevata qualora si verifichino condizioni eccezionali, come la divisione per zero, in operazioni aritmetiche.

Eccezioni: `java.lang`

- `ArithmeticException`
  Viene sollevata qualora si verifichino condizioni eccezionali, come la divisione per zero, in operazioni aritmetiche.

- `ArrayIndexOutOfBoundsException`
  Viene sollevata quando si tenta di accedere a una posizione inesistente di un array.

Eccezioni: `java.lang`

- `ArithmeticException`
  Viene sollevata qualora si verifichino condizioni eccezionali, come la divisione per zero, in operazioni aritmetiche.

- `ArrayIndexOutOfBoundsException`
  Viene sollevata quando si tenta di accedere a una posizione inesistente di un array.

- `NegativeArraySizeException`
  Viene sollevata quando si tenta di creare un array di dimensione negativa.

Eccezioni: `java.lang`

- `ArithmeticException`
  Viene sollevata qualora si verifichino condizioni eccezionali, come la divisione per zero, in operazioni aritmetiche.
- `ArrayIndexOutOfBoundsException`
  Viene sollevata quando si tenta di accedere a una posizione inesistente di un array.
- `NegativeArraySizeException`
  Viene sollevata quando si tenta di creare un array di dimensione negativa.
- `StringIndexOutOfBoundsException`
  Viene sollevata quando si tenta di accedere a una posizione inesistente in una stringa.

Eccezioni: `java.lang`

- `NullPointerException`
  Viene sollevata quando si tenta di accedere a un oggetto tramite un riferimento `null`.

Eccezioni: `java.lang`

- `NullPointerException`
  Viene sollevata quando si tenta di accedere a un oggetto tramite un riferimento `null`.

- `ClassCastException`
  Viene sollevata quando si forza un tipo riferimento a un sottotipo di cui l'oggetto non è istanza.

Esempio

```
Rettangolo r;
...
Quadrato q = (Quadrato) r;
```

verrà sollevata l'eccezione nel caso in cui l'oggetto riferito da `r` non sia un'istanza di `Quadrato`

Eccezioni: `java.util`

- `EmptyStackException`
  Viene sollevata quando si tenta di accedere a un elemento di uno `Stack` vuoto.

Eccezioni: `java.util`

- `EmptyStackException`
  Viene sollevata quando si tenta di accedere a un elemento di uno `Stack` vuoto.

- `NoSuchElementException`
  Viene sollevata quando si richiama il metodo `nextToken` di un'istanza di `StringTokenizer` e i token sono esauriti, o quando si richiama il metodo `next` di un oggetto che implementa l'interfaccia `Iterator` e non ci sono più elementi nell'iteratore.

Sommario: Eccezioni

Il costruttore di `Frazione`

```
public Frazione(int x, int y) {
  if (y == 0) {
    num = 0;
    den = 0;
  } else {
    ...
  }
}
```

- Per convenzione un oggetto di tipo `Frazione` con `num`=`den`=`0` rappresenta una frazione impossibile

Il costruttore di Frazione

```
public Frazione(int x, int y) {
  if (y == 0) {
    num = 0;
    den = 0;
  } else {
    ...
  }
}
```

- Per convenzione un oggetto di tipo Frazione con num=den=0 rappresenta una frazione impossibile
- Vogliamo modificare il costruttore della classe Frazione in modo che quando argomento al denominatore è zero sollevi un'eccezione

L'istruzione `throw`

Sintassi

`throw` *riferimento_oggetto*

L'istruzione `throw`

Sintassi

`throw` *riferimento_oggetto*

- *riferimento_oggetto*
  deve essere un riferimento ad un oggetto di tipo `Throwable`

Modifica del costruttore

```
public Frazione(int x, int y) {
  if (y == 0)
    throw new ArithmeticException("Frazione non valida: " +
                                  "denominatore 0");
  else {
    ...
  }
}
```

Esempio di esecuzione

```
public static void main(String[] args) {
  ...
  int num = in.readInt("  Numeratore? ");
  int den = in.readInt("Denominatore? ");
  Frazione f = new Frazione(num,den);
  ...
}
```

Esempio di esecuzione

```
public static void main(String[] args) {
  ...
  int num = in.readInt("  Numeratore? ");
  int den = in.readInt("Denominatore? ");
  Frazione f = new Frazione(num,den);
  ...
}
```

Esecuzione

```
  Numeratore? 3
Denominatore? 0
Exception in thread "main" java.lang.ArithmeticException:
      Frazione non valida: denominatore 0
   at Frazione.<init>(Frazione.java:6)
   at Frazione.diviso(Frazione.java:50)
   at Prova.main(Prova.java:18)
```

Esempio d'uso

```
public static void main(String[] args) {
  ...
  int num = in.readInt("  Numeratore? ");
  int den = in.readInt("Denominatore? ");
  try {
    Frazione f = new Frazione(num,den);
  } catch (ArithmeticException e) {
      out.println(e.toString());
  }
  ...
}
```

Esempio d'uso

```
public static void main(String[] args) {
  ...
  int num = in.readInt("  Numeratore? ");
  int den = in.readInt("Denominatore? ");
  try {
    Frazione f = new Frazione(num,den);
  } catch (ArithmeticException e) {
      out.println(e.toString());
  }
  ...
}
```

Esecuzione

```
  Numeratore? 3
Denominatore? 0
java.lang.ArithmeticException:
            Frazione non valida: denominatore 0
```

Esempio d'uso

```
public static void main(String[] args) {
  ...
  int num = in.readInt("  Numeratore? ");
  int den = in.readInt("Denominatore? ");
  try {
    Frazione f = new Frazione(num,den);
  } catch (ArithmeticException e) {
      out.println(e.getMessage());
      e.printStackTrace();
  }
  ...
}
```

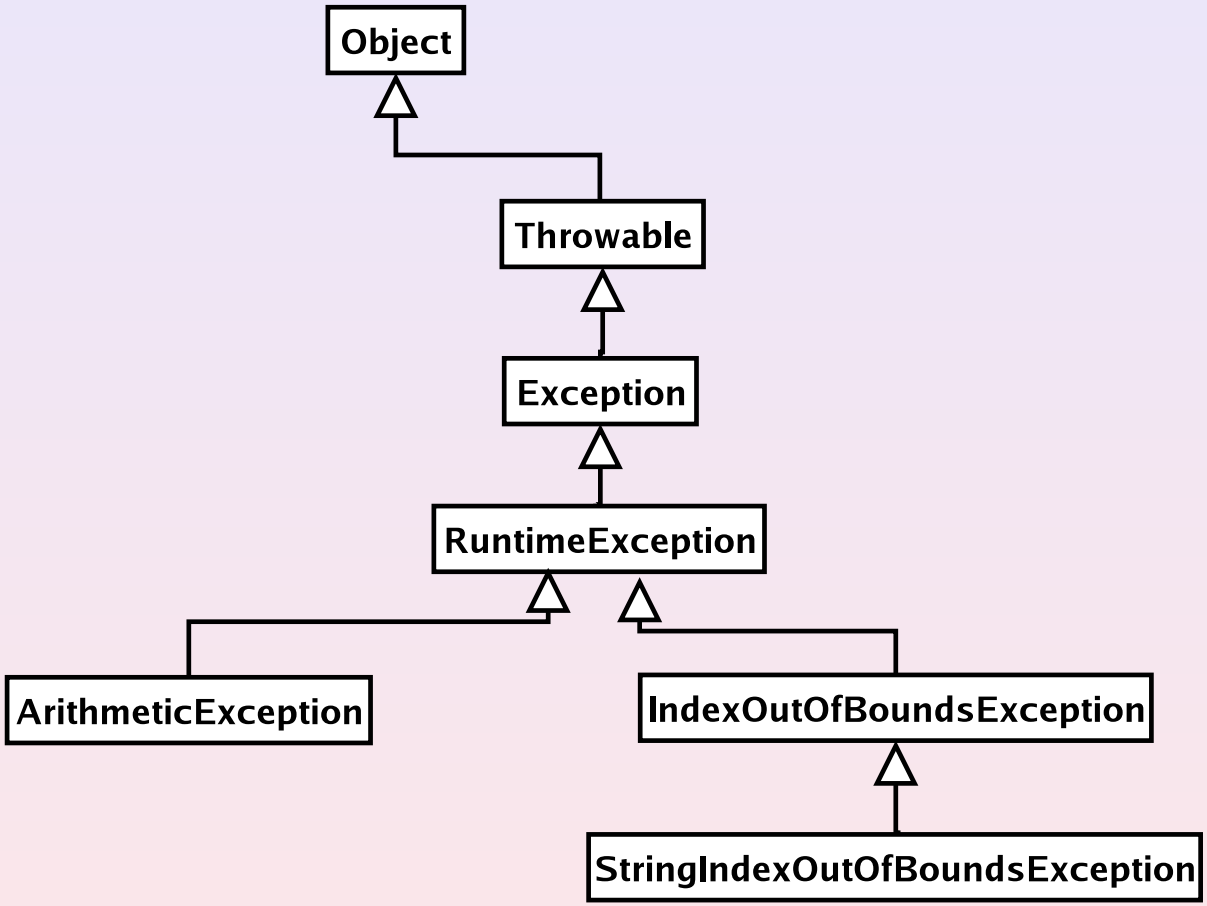
Esempio d'uso

```
public static void main(String[] args) {
  ...
  int num = in.readInt("  Numeratore? ");
  int den = in.readInt("Denominatore? ");
  try {
    Frazione f = new Frazione(num,den);
  } catch (ArithmeticException e) {
      out.println(e.getMessage());
      e.printStackTrace();
  }
  ...
}
```

Esecuzione

```
  Numeratore? 3
Denominatore? 0
java.lang.ArithmeticException: Frazione non valida: denominatore 0
  at Frazione.<init>(Frazione.java:6)
  at Prova.main(Prova.java:11)
```

Gerarchia delle eccezioni di Java



Definire un'eccezione

- Per definire una nuova eccezione dobbiamo estendere `Exception` o una delle sue sottoclassi

Definire un'eccezione

- Per definire una nuova eccezione dobbiamo estendere `Exception` o una delle sue sottoclassi
- La scelta della classe da estendere dipende:
  - da ciò che si vuole rappresentare

Definire un'eccezione

- Per definire una nuova eccezione dobbiamo estendere `Exception` o una delle sue sottoclassi
- La scelta della classe da estendere dipende:
  - da ciò che si vuole rappresentare
  - dal fatto che si voglia definire un'eccezione controllata oppure un'eccezione non controllata

Esempio

```
public Frazione(int x, int y) {
  if (y == 0)
    throw new ArithmeticException("Frazione non valida: " +
                                  "denominatore 0");
  else {
    ...
  }
}
```

Esempio

```
public Frazione(int x, int y) {
  if (y == 0)
    throw new ArithmeticException("Frazione non valida: " +
                                  "denominatore 0");
  else {
    ...
  }
}
```

- Invece di utilizzare `ArithmeticException` definiamo un'eccezione specifica

Esempio

```
public Frazione(int x, int y) {
  if (y == 0)
    throw new ArithmeticException("Frazione non valida: " +
                                  "denominatore 0");
  else {
    ...
  }
}
```

- Invece di utilizzare `ArithmeticException` definiamo un'eccezione specifica
- Dato che si tratta di una particolare eccezione legata all'aritmetica definiamo un'estensione di `ArithmeticException`

Esempio: FrazioneException

L'eccezione

```
public class FrazioneException extends ArithmeticException {
  public FrazioneException(String msg) {
    super(msg);
  }
}
```

Esempio: FrazioneException

L'eccezione

```
public class FrazioneException extends ArithmeticException {
  public FrazioneException(String msg) {
    super(msg);
  }
}
```

Il nuovo costruttore

```
public Frazione(int x, int y) {
  if (y == 0)
    throw new FrazioneException("Frazione non valida: " +
                                denominatore 0");
  else {
    ...
  }
}
```

Uso dell'eccezione

```
public static void main(String[] args) {
  ...
  int num = in.readInt("  Numeratore? ");
  int den = in.readInt("Denominatore? ");
  try {
    Frazione f = new Frazione(num,den);
  } catch (ArithmeticException e) {
      out.println(e.toString());
  }
  ...
}
```

- Un'istanza di `FrazioneException` è anche istanza di `ArithmeticException`
- Quindi il blocco `catch` intercetta anche `FrazioneException`

Uso dell'eccezione più specifica

```
public static void main(String[] args) {
  ...
  int num = in.readInt("  Numeratore? ");
  int den = in.readInt("Denominatore? ");
  try {
    Frazione f = new Frazione(num,den);
  } catch (FrazioneException e) {
      out.println(e.toString());
  }
  ...
}
```

Esecuzione

```
  Numeratore? 3
Denominatore? 0
FrazioneException: Frazione non valida: denominatore 0
```

Una nuova versione di FrazioneException

```
public class FrazioneException extends ArithmeticException {
  private int num;

  public FrazioneException(String msg, int num) {
    super(msg);
    this.num = num;
  }

  public FrazioneException(int num) {
    this.num = num;
  }

  public int getNumeratore() {
    return num;
  }
}
```

Il costruttore di `Frazione`

```
public Frazione(int x, int y) {
  if (y == 0)
    throw new FrazioneException("Frazione non valida: " +
                                "denominatore 0", x);
  else {
    ...
  }
}
```

Uso dell'eccezione

```
public static void main(String[] args) {
  ...
  int num = in.readInt("  Numeratore? ");
  int den = in.readInt("Denominatore? ");
  try {
    Frazione f = new Frazione(num,den);
  } catch (FrazioneException e) {
      out.println(e.toString());
      out.println("Il valore del numeratore e' " +
                  e.getNumeratore());
  }
  ...
}
```

Uso dell'eccezione

```
public static void main(String[] args) {
  ...
  int num = in.readInt("  Numeratore? ");
  int den = in.readInt("Denominatore? ");
  try {
    Frazione f = new Frazione(num,den);
  } catch (FrazioneException e) {
      out.println(e.toString());
      out.println("Il valore del numeratore e' " +
                  e.getNumeratore());
  }
  ...
}
```

Esecuzione

```
  Numeratore? 3
Denominatore? 0
FrazioneException: Frazione non valida: denominatore 0
Il valore del numeratore e' 3
```

Sommario: Eccezioni

Sommario: Eccezioni

Eccezioni

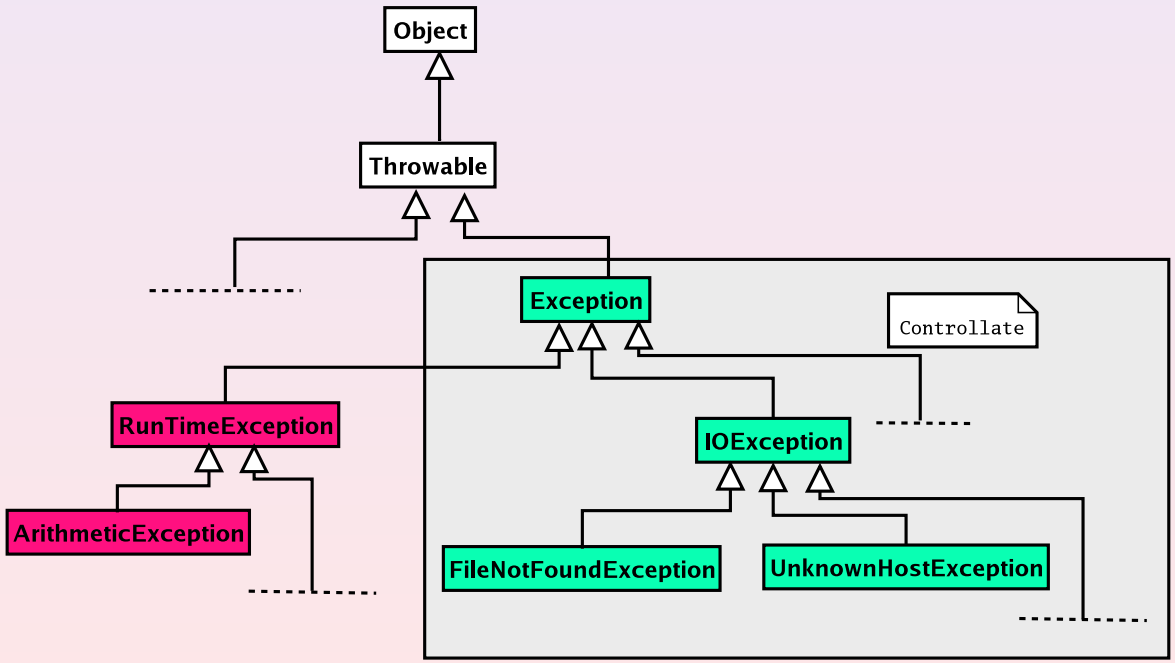
- Eccezioni non controllate
  Sono tutte le istanze di `RuntimeException`

Eccezioni

- Eccezioni non controllate
  Sono tutte le istanze di `RuntimeException`
- Eccezioni controllate
  Le istanze di `Exception` che non sono istanze di `RuntimeException`.

Eccezioni

- Eccezioni non controllate
  Sono tutte le istanze di `RuntimeException`
- Eccezioni controllate
  Le istanze di `Exception` che non sono istanze di `RuntimeException`.



Controllate da chi?

Si differenziano per il modo in cui vengono trattate dal compilatore.

Controllate da chi?

Si differenziano per il modo in cui vengono trattate dal compilatore.

- Eccezioni controllate
  Se nel corpo di un metodo o di un costruttore può essere sollevata un'eccezione controllata, il compilatore controlla che tale eccezione sia trattata esplicitamente.

  Se ciò non avviene il compilatore segnala un errore.

Controllate da chi?

Si differenziano per il modo in cui vengono trattate dal compilatore.

- Eccezioni controllate
  Se nel corpo di un metodo o di un costruttore può essere sollevata un'eccezione controllata, il compilatore controlla che tale eccezione sia trattata esplicitamente.

  Se ciò non avviene il compilatore segnala un errore.

- Eccezioni non controllate
  Il compilatore non effettua controlli.

  Se non vengono intercettate vengono automaticamente delegate al chiamante.

Eccezioni controllate

Se nel corpo di un metodo o di un costruttore può essere sollevata un'eccezione controllata, il compilatore controlla che tale eccezione sia trattata esplicitamente.

Eccezioni controllate

Se nel corpo di un metodo o di un costruttore può essere sollevata un'eccezione controllata, il compilatore controlla che tale eccezione sia trattata esplicitamente.

Un metodo (costruttore) tratta l'eccezione

- intercettandola
  tramite l'istruzione `try-catch`

Eccezioni controllate

Se nel corpo di un metodo o di un costruttore può essere sollevata un'eccezione controllata, il compilatore controlla che tale eccezione sia trattata esplicitamente.

Un metodo (costruttore) tratta l'eccezione

- intercettandola
  tramite l'istruzione `try-catch`
- delegandola esplicitamente al chiamante
  mediante un'opportuna dichiarazione nell'intestazione del metodo (costruttore) introdotta dalla parola chiave `throws`

Esempio

L'eccezione

```
public class FrazioneException extends Exception {
   ...
}
```

Esempio

L'eccezione

```
public class FrazioneException extends Exception {
  ...
}
```

Il costruttore di Frazione

```
public Frazione(int x, int y) {
  if (y == 0)
    throw new FrazioneException("Frazione non valida: " +
                                denominatore 0");
  else {
```

Esempio

L'eccezione

```
public class FrazioneException extends Exception {
  ...
}
```

Il costruttore di Frazione

```
public Frazione(int x, int y) {
  if (y == 0)
    throw new FrazioneException("Frazione non valida: " +
                                denominatore 0");
  else {
```

Compilazione di Frazione.java

```
Frazione.java:6: unreported exception FrazioneException;
must be caught or declared to be thrown
  throw new FrazioneException(...);
  ^
1 error
```

Deleghiamo l'eccezione

Il costruttore di `Frazione`

```
public Frazione(int x, int y) throws FrazioneException {
  if (y == 0)
    throw new FrazioneException("Frazione non valida: " +
                                denominatore 0", x);
  else {
    ...
  }
}
```

Deleghiamo l'eccezione

Il costruttore di Frazione

```
public Frazione(int x, int y) throws FrazioneException {
  if (y == 0)
    throw new FrazioneException("Frazione non valida: " +
                                denominatore 0", x);
  else {
    ...
  }
}
```

Compilazione

```
Frazione.java:26: unreported exception FrazioneException;
must be caught or declared to be thrown
    this(x, 1);
    ^
Frazione.java:32: unreported exception FrazioneException;
must be caught or declared to be thrown
    return new Frazione(n, d);
           ^
```

Correzioni

Il codice scorretto

```
public Frazione(int x) {
  this(x,1);
}
```

Correzioni

Il codice scorretto

```
public Frazione(int x) {
  this(x,1);
}
```

Non è possibile costruire una frazione scorretta utilizzando questo costruttore...

Correzioni

Il codice scorretto

```
public Frazione(int x) {
  this(x,1);
}
```

Non è possibile costruire una frazione scorretta utilizzando questo costruttore...

Il nuovo codice

```
public Frazione(int x) {
  num = x;
  den = 1;
}
```

Il metodo diviso

Il codice scorretto

```
public Frazione diviso(Frazione f) {
  int n = this.num * f.den;
  int d = this.den * f.num;
  return new Frazione(n, d);
}
```

Il metodo diviso

Il codice scorretto

```
public Frazione diviso(Frazione f) {
  int n = this.num * f.den;
  int d = this.den * f.num;
  return new Frazione(n, d);
}
```

Il nuovo codice

```
public Frazione diviso(Frazione f) throws FrazioneException {
  int n = this.num * f.den;
  int d = this.den * f.num;
  return new Frazione(n, d);
}
```

Il metodo per

Il codice scorretto

```
public Frazione per(Frazione f) {
  int n = this.num * f.num;
  int d = this.den * f.den;
  return new Frazione(n, d);
}
```

Il risultato dell'operazione effettuata sarà sempre una frazione...

Il metodo per

Il codice scorretto

```
public Frazione per(Frazione f) {
  int n = this.num * f.num;
  int d = this.den * f.den;
  return new Frazione(n, d);
}
```

Il risultato dell'operazione effettuata sarà sempre una frazione...

Il nuovo codice

```
public Frazione per(Frazione f) {
  int n = this.num * f.num;
  int d = this.den * f.den;
  try {
    return new Frazione(n, d);
  } catch (FrazioneException e) {
    return null;
  }
}
```

Eccezioni controllate e non: scopo

- Ogni volta che si scrive codice in cui può essere sollevata un'eccezione controllata è obbligatorio trattarla, intercettandola o delegandola.

Eccezioni controllate e non: scopo

- Ogni volta che si scrive codice in cui può essere sollevata un'eccezione controllata è obbligatorio trattarla, intercettandola o delegandola.

  Questo impedisce al programmatore di "dimenticarsi" degli eventi rappresentati dalle eccezioni controllate, costringendolo a scegliere come gestirli

Eccezioni controllate e non: scopo

- Ogni volta che si scrive codice in cui può essere sollevata un'eccezione controllata è obbligatorio trattarla, intercettandola o delegandola.

  Questo impedisce al programmatore di "dimenticarsi" degli eventi rappresentati dalle eccezioni controllate, costringendolo a scegliere come gestirli

- Controllate e non controllate servono a modellare anomalie di natura diversa:
  - Controllate
    Per modellare anomalie legate a eventi esterni al programma

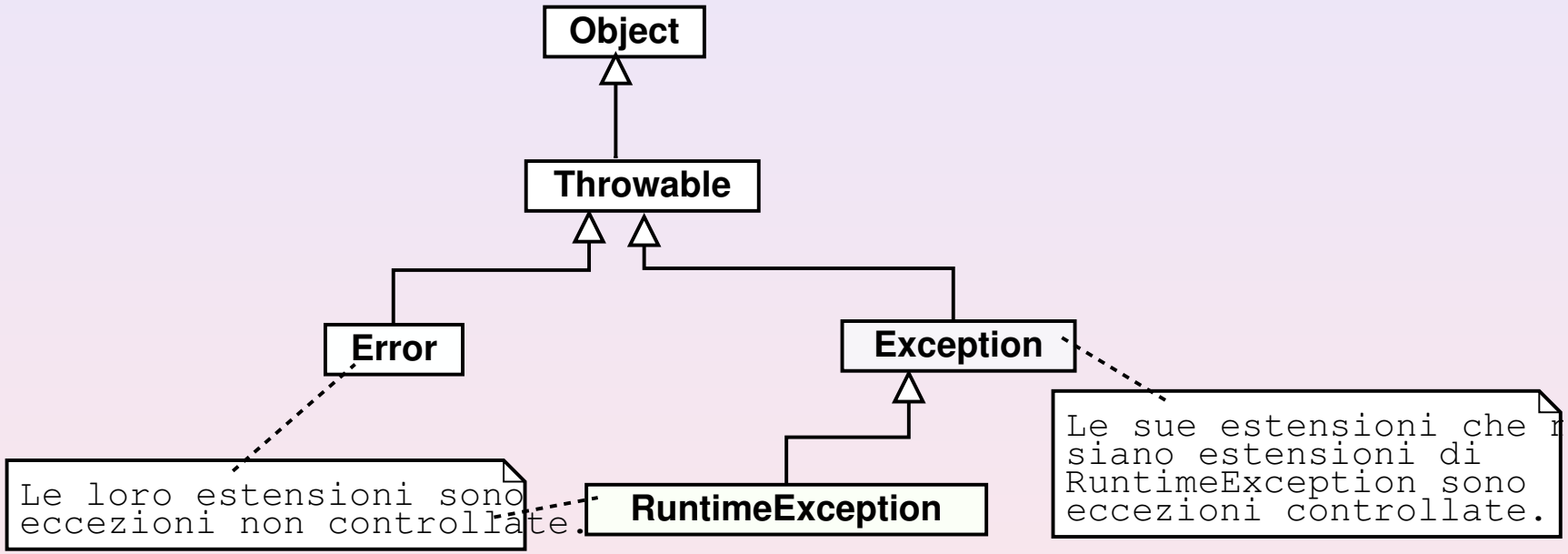
Eccezioni controllate e non: scopo

- Ogni volta che si scrive codice in cui può essere sollevata un'eccezione controllata è obbligatorio trattarla, intercettandola o delegandola.

  Questo impedisce al programmatore di "dimenticarsi" degli eventi rappresentati dalle eccezioni controllate, costringendolo a scegliere come gestirli

- Controllate e non controllate servono a modellare anomalie di natura diversa:
  - Controllate
    Per modellare anomalie legate a eventi esterni al programma
  - Non controllate
    Per modellare anomalie legate a eventi interni al programma che potrebbero essere evitate scrivendo il codice in maniera differente

Gerarchia degli errori



La clausola `finally`

Sintassi

```
try {
  // Codice che potrebbe generare eccezioni
  blocco_try
} catch (Tipo_Eccezione1 id1) {
  blocco_gestore1
} catch (Tipo_Eccezione2 id2) {
  blocco_gestore2
}
...
finally {
  blocco_finally
}
```

Semantica

(1) Vengono eseguite le istruzioni nel blocco `try`

Semantica

(1) Vengono eseguite le istruzioni nel blocco `try`

- se non viene sollevata alcuna eccezione, l'esecuzione prosegue dal Punto (2)

Semantica

(1) Vengono eseguite le istruzioni nel blocco `try`

- se non viene sollevata alcuna eccezione, l'esecuzione prosegue dal Punto (2)
- altrimenti, la JVM scorre la lista degli argomenti dei blocchi `catch`, nell'ordine in cui sono scritti, alla ricerca di un tipo cui possa essere ricondotto quello dell'eccezione sollevata:

Semantica

(1) Vengono eseguite le istruzioni nel blocco `try`

- se non viene sollevata alcuna eccezione, l'esecuzione prosegue dal Punto (2)
- altrimenti, la JVM scorre la lista degli argomenti dei blocchi `catch`, nell'ordine in cui sono scritti, alla ricerca di un tipo cui possa essere ricondotto quello dell'eccezione sollevata:
  - se lo trova, esegue il codice del corrispondente blocco `catch`, al termine del quale passa al Punto (2)

Semantica

(1) Vengono eseguite le istruzioni nel blocco `try`

- se non viene sollevata alcuna eccezione, l'esecuzione prosegue dal Punto (2)
- altrimenti, la JVM scorre la lista degli argomenti dei blocchi `catch`, nell'ordine in cui sono scritti, alla ricerca di un tipo cui possa essere ricondotto quello dell'eccezione sollevata:
  - se lo trova, esegue il codice del corrispondente blocco `catch`, al termine del quale passa al Punto (2)
  - altrimenti l'esecuzione prosegue dal Punto (2).

Semantica

(1) Vengono eseguite le istruzioni nel blocco `try`

- se non viene sollevata alcuna eccezione, l'esecuzione prosegue dal Punto (2)
- altrimenti, la JVM scorre la lista degli argomenti dei blocchi `catch`, nell'ordine in cui sono scritti, alla ricerca di un tipo cui possa essere ricondotto quello dell'eccezione sollevata:
  - se lo trova, esegue il codice del corrispondente blocco `catch`, al termine del quale passa al Punto (2)
  - altrimenti l'esecuzione prosegue dal Punto (2).

(2) Vengono eseguite le istruzioni del blocco `finally` (se è presente). Quindi l'esecuzione riprende dalla prima istruzione dopo l'istruzione `try-catch`.