# Programmazione Orientata agli Oggetti in Linguaggio Java

## Eccezioni: Gestione delle Eccezioni Parte a

versione 2.4



**G. Mecca – Università della Basilicata – mecca@unibas.it**



## Sommario

- Introduzione
- Gestione delle Eccezioni
- Lanciare un'Eccezione
- L'Atteggiamento del Chiamante
- Catturare un'Eccezione
- Prevenire o Gestire ?

# Introduzione

- Funzionamento di un'applicazione
  - ⇨sequenza di messaggi tra i componenti
- Comunicazione tra componenti
  - ⇨il componente A richiede esecuzione di un metodo al componente B
  - ⇨A è il chiamante (invia il messaggio richiedendo a B l'esecuzione di un metodo)
  - ⇨B è il chiamato (esegue il metodo)





# Introduzione

- Due possibili scenari
- Situazione ordinaria
  - ⇨il metodo chiamato viene eseguito con successo fino alla fine e il controllo ritorna al chiamante (con l'eventuale risultato)
- Situazione anomala
  - ⇨durante l'esecuzione del metodo si verifica un problema che impedisce di portare a termine l'esecuzione del metodo chiamato

# Introduzione

- Varie possibili cause di problemi
  - ⇨errori interni all'applicazione
  - ⇨errori esterni all'applicazione
- I categoria: errori interni all'applicazione
  - ⇨errori nella logica applicativa (es: invio di un messaggio ad un riferimento null, accesso ad una posizione inesistente in un array)
  - ⇨errori nei dati forniti dall'utente (es: richiesta di eseguire una divisione per 0)





# Introduzione

- II categoria: errori esterni all'applicazione
  - ⇨errori nei sistemi esterni con cui l'applicazione comunica (es: errore sul disco nell'accesso ai file, oppure DBMS)
  - ⇨errore nella macchina virtuale (es: spazio nello heap esaurito: "Out of memory error")
- Che cosa fare se si verifica un problema
  - ⇨diversi atteggiamenti possibili

# Introduzione

- Un approccio semplicistico
  - ignoro il problema; se il problema si verifica l'applicazione abortisce a causa dell'errore
  - quasi sempre inaccettabile
- L'approccio procedurale: valori di errore
  - cerco di intercettare le condizioni che possono condurre ad errori
  - le segnalo con valori "speciali" utilizzati come valori di errore





# Introduzione

- Un esempio in C/C++
  - cancellazione di un elemento da una lista rappresentata con array e indicatore di riemp.
- Prototipo
  - void elimina(lista& l, int pos)
  - possibile solo se pos è una posizione esistente nella lista
  - aggiungo un ulteriore parametro bool &esito, per segnalare se l'eliminazione è stata effett.

# Esempio: Cancellazione da una Lista

```
void elimina (lista &l, int pos, bool& esito){
     int i;
     if (pos <0 || pos > l.indicatore) {
         esito = false;
     } else {
         for (i = pos; i < l.indicatore - 1; i++) {
             l.valori[i] = l.valori[i + 1];
         }
         l.indicatore--;
         esito = true;
     }
     return;
 }
```

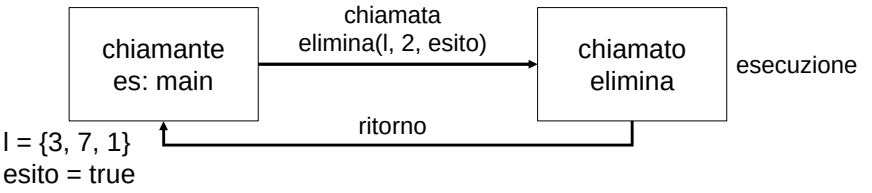
NOTA: la comunicazione tra chiamante e chiamato si svolge allo stesso modo sia nel caso di posizione scorretta sia nel caso di posizione corretta



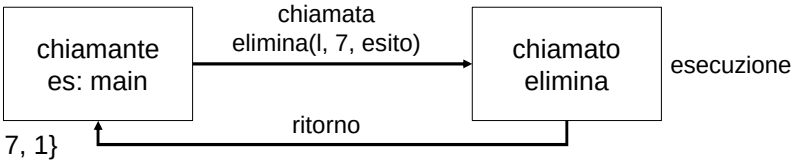


# Introduzione

supponiamo l = {3, 7, 2, 1}




l = {3, 7, 1}
esito = true




l = {3, 7, 1}
esito = false

# Introduzione

- In sostanza
  - ⇨cerco di prevenire l'errore (es: accesso ad una posizione inesistente della lista) analizzando caso per caso le possibili condizioni in cui l'errore si verifica
  - ⇨gestisco la segnalazione nel flusso ordinario di controllo (chiamata – esecuzione – ritorno) cioè evitando di abortire l'esecuzione
  - ⇨il sottoprogramma termina correttamente senza effettuare operazioni scorrette





# Introduzione

- Svantaggi di questo approccio
  - ⇨non sempre l'approccio è applicabile (es: funzione per il calcolo della media applicata ad una lista vuota); che valore restituire ?
  - ⇨il codice del metodo chiamato non è particolarmente leggibile (poco evidente la condizione di errore)
  - ⇨il modulo chiamante è obbligato ad effettuare test sul risultato della chiamata (esito ?)

# Gestione delle Eccezioni

- L'approccio nei linguaggi a oggetti
  - ⇨approccio intermedio tra i due illustrati
  - ⇨consente di garantire la correttezza dell'esecuzione
  - ⇨ma evita gli svantaggi dei valori di errore
- Idea fondamentale
  - ⇨gestire le condizioni eccezionali al di fuori del flusso ordinario di controllo





# Gestione delle Eccezioni

- Comportamento tipico dell'applicazione
  - ⇨il metodo chiamato controlla le condizioni che possono portare ad errori
  - ⇨segnala il verificarsi di condizioni eccezionali "sollevando (o lanciando) un'eccezione"
  - ⇨il metodo chiamante può "gestire l'eccezione" senza necessariamente interrompere il funzionamento dell'applicazione

# Gestione delle Eccezioni

- Cos'è in concreto un'eccezione ?
  - ⇨un'eccezione è un oggetto
- Eccezione
  - ⇨oggetto utilizzato dal chiamato per segnalare al chiamante il verificarsi di una condizione eccezionale
  - ⇨contiene un messaggio descrittivo della ragione per cui è stata sollevata





# Gestione delle Eccezioni

- Metafora
  - ⇨un oggetto eccezione è un postino
  - ⇨oggetto incaricato di trasportare un messaggio di errore dal chiamato al chiamante
- Attenzione ai ruoli nell'applicazione
  - ⇨le eccezioni rappresentano un'ulteriore categoria di ruoli oltre a quelli individuati dagli strati applicativi

# Esempio: Cancellazione da ArrayList

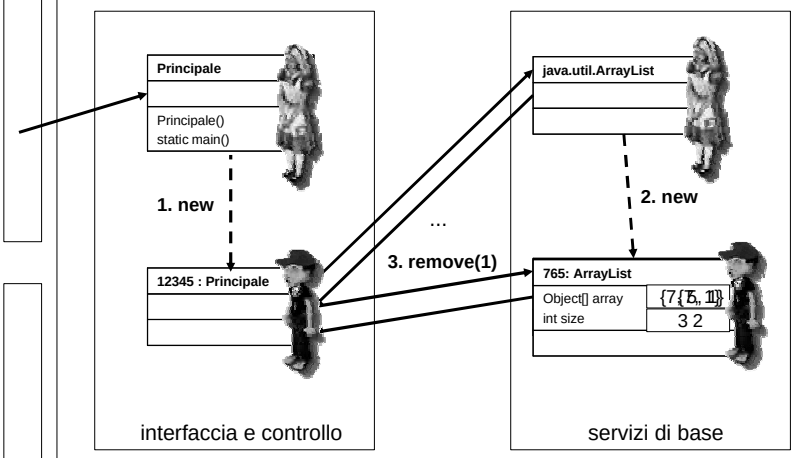
```
package java.util;

public class ArrayList implements List {

   private Object array[];
   private int size;

   public void remove(int pos) {
      if (pos < 0 || pos >= this.size) {
         throw new IllegalArgumentException ("Posizione inesistente");
      }
      for (int i = pos; i < this.size - 1; i++) {
         this.array[i] = this.array[i + 1];
      }
      this.size--;
   }
```

ATTENZIONE: in caso di eccezione l'esecuzione del metodo si interrompe immediatamente dopo l'esecuzione dell'istruzione throw

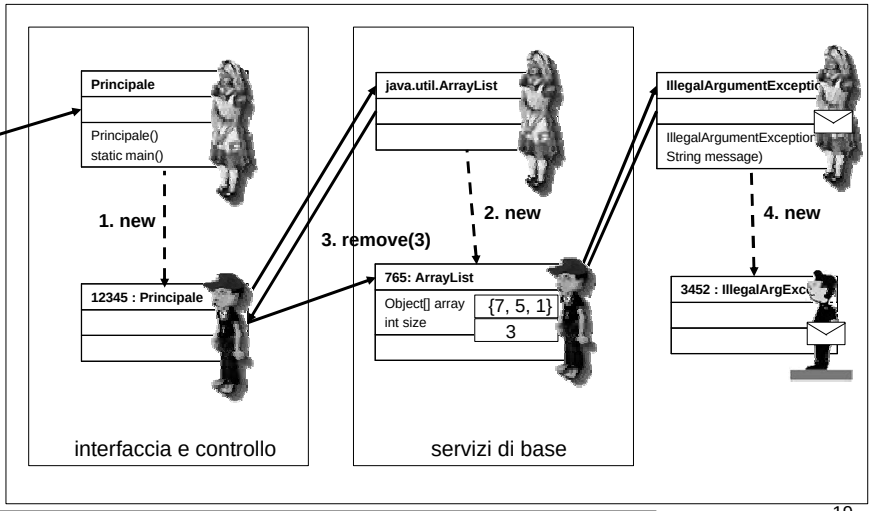




# Gestione delle Eccezioni



# Gestione delle Eccezioni








# Lanciare un'Eccezione

- Per lanciare un'eccezione
  - ⇨è necessario creare un oggetto eccezione
  - ⇨e poi lanciarlo al chiamante
- Creazione dell'oggetto eccezione
  - ⇨il costruttore riceve una stringa
  - ⇨es: IllegalArgumentException iae =
    new IllegalArgumentException("Pos. inesistente");

# Lanciare un'Eccezione

- Lanciare l'eccezione
  - ⇨istruzione throw
  - ⇨es: throw iae;
- Un ulteriore esempio
  - ⇨Indovina il Numero
  - ⇨metodo gestisciTentativo() di Partita.java
  - ⇨il tentativo deve essere compreso tra 1 e 100





# Lanciare un'Eccezione

```
public void gestisciTentativo(int tentativo) {
      if ((tentativo < 1) || (tentativo > 100)) {
         throw new IllegalArgumentException("Valore scorretto");
      }
      this.numeroDiTentativi++;
      if (tentativo == this.numeroDaIndovinare) {
         this.trovato = true;
         this.suggerimento = "Numero indovinato";
      } else if (tentativo < numeroDaIndovinare) {
         this.suggerimento = "Prova con un numero piu' alto";
      } else if (tentativo > numeroDaIndovinare) {
         this.suggerimento = "Prova con un numero piu' basso";
      }
   }
```

# L'Atteggiamento del Chiamante

- Cosa fa il chiamante
  - ⇨nel caso in cui il chiamato possa sollevare eccezioni ?
- Due possibili atteggiamenti
  - ⇨"prevenzione dell'errore"
  - ⇨"gestione dell'errore"
  - ⇨a seconda dei casi può essere opportuno l'uno o l'altro atteggiamento





# L'Atteggiamento del Chiamante

- Prevenire l'errore
  - ⇨il chiamante effettua tutte le necessarie convalide per evitare che si verifichi l'errore
  - ⇨e decide di disinteressarsi dell'eventuale eccezione (rischiando, eventualmente, di abortire)
  - ⇨dal momento che è sicuro che non si verificheranno eccezioni

# L'Atteggiamento del Chiamante

```
private void schermoTentativi() {
    boolean esci = false;
    while (!partita.getTrovato() && !esci) {
        ...
        int tentativo = it.unibas.utilita.Console.leggiIntero();
        while ((tentativo < 0) || (tentativo > 100)) {
            System.out.println("ERRORE. Riprova (tra 1 e 100)");
            tentativo = it.unibas.utilita.Console.leggiIntero();
        }
        if (tentativo != 0) {
            partita.gestisciTentativo(tentativo);
            ...
        }
    }
}
```





# L'Atteggiamento del Chiamante

- Gestire l'errore
  - ⇨il chiamante non è in grado di effettuare le convalide necessarie (tipicamente perchè queste sono troppo complesse)
  - ⇨allora chiama il metodo e si prepara a gestire l'eventuale eccezione
  - ⇨tipicamente ripetendo la chiamata all'interno di un ciclo finchè l'eccezione non si ripete

# Catturare un'Eccezione

- Gestire l'eccezione
  - ⇨per gestire l'eccezione bisogna catturarla
- Catturare l'eccezione
  - ⇨includere la chiamata al metodo che può lanciare eccezione in una regione "controllata"
  - ⇨costrutto try-catch





# Catturare un'Eccezione

```
// una possibile alternativa
private void schermoTentativo() {
  ...
  boolean effettuato = false;
  int tentativo = it.unibas.utilita.Console.leggiIntero();
  while (!effettuato) {
    try {
        partita.gestisciTentativo(tentativo);
        effettuato = true;
    } catch (IllegalArgumentException e) {
        System.out.println(e);
        System.out.println("Riprova:");
        tentativo = it.unibas.utilita.Console.leggiIntero();
    }
  }
  ...
}
```

ATTENZIONE: questo codice viene presentato solo a scopo didattico; non rappresenta una strategia corretta di gestione del tentativo

# Catturare un'Eccezione

- Regione controllata
  - ⇨regione del metodo racchiusa tra try { }
  - ⇨normalmente associata ad una regione catch per la cattura
- Regione per la cattura
  - ⇨catch (*ClasseEccezione riferimento*) {
    *istruzioni*
    }
  - ⇨es: catch (IllegalArgumentException e) {...}





# Catturare un'Eccezione

- Semantica
  - ⇨se, durante l'esecuzione della regione controllata, viene ricevuta un'eccezione, si interrompe l'esecuzione del codice della regione controllata
  - ⇨se esiste, viene eseguita la regione per la cattura dell'eccezione ricevuta
  - ⇨al termine della regione catch l'esecuzione riprende dall'istruzione immediatamente successiva a quella del blocco try-catch

# Esempio

```
// Principale.java
private void schermoTentativo {

boolean effettuato = false;
while (!effettuato) {
    try {
      partita.gestisciTentativo(tentativo);
      effettuato = true;
    } catch (IllegalArgumentException e) {
      System.out.println(e);
      System.out.println("Riprova:");
      tentativo =

    it.unibas.utilita.Console.leggiIntero();
    }
}

}
```

75

```
// Partita.java
public void gestisciTentativo(int tentativo){

if ((tentativo < 1) || (tentativo > 100)) {
    throw new IllegalArgumentException(
        "Il valore deve essere tra 1 e 100");
}
this.numeroDiTentativi++;
...

}
```





# Esempio

```
// Principale.java
private void schermoTentativo {

boolean effettuato = false;
while (!effettuato) {
    try {
      partita.gestisciTentativo(tentativo);
      effettuato = true;
    } catch (IllegalArgumentException e) {
      System.out.println(e);
      System.out.println("Riprova:");
      tentativo =

    it.unibas.utilita.Console.leggiIntero();
    }
}

}
```

200

```
// Partita.java
public void gestisciTentativo(int tentativo){

if ((tentativo < 1) || (tentativo > 100)) {
    throw new IllegalArgumentException(
        "Il valore deve e
    }
this.numeroDiTentativi
...

}
```

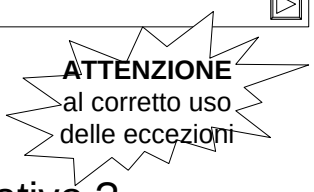
# Catturare un'Eccezione

- Principali metodi di un oggetto eccezione
  - ⇨getMessage(): restituisce il messaggio descrittivo associato a questa eccezione
  - ⇨printStackTrace(): stampa la struttura della pila al momento in cui è stata sollevata
  - ⇨toString(): restituisce una breve descrizione – utilizzato implicitamente da System.out.println()





# Prevenire o Gestire ?




- Quale soluzione per il tentativo ?
  - ⇨la prima soluzione (prevenire) è preferibile
- Linea guida
  - ⇨le eccezioni non devono essere usate nei casi in cui è possibile evitarlo
  - ⇨di conseguenza, quando questo è possibile, è opportuno prevenire l'eccezione effettuando le opportune convalide sui dati
  - ⇨questo, però, non è sempre possibile

# Prevenire o Gestire ?

- Un caso diverso
  - ⇨la classe Console e il metodo leggiIntero()
- La strategia
  - ⇨sulla base di System.in viene definito un oggetto stdin di tipo BufferedReader
  - ⇨che fornisce il metodo String readLine()
  - ⇨dalla stringa viene ricostruito il valore intero digitato sulla tastiera con il metodo static int parseInt(String s) di java.lang.Integer





# Prevenire o Gestire ?

- Ma...
  - ⇨il metodo int parseInt(String s) di Integer può lanciare NumberFormatException
  - ⇨nel caso la stringa non rappresenti un intero
  - ⇨Console non può sapere se la stringa rappresenta effettivamente un numero finché non chiede a Integer di fare la conversione
  - ⇨la convalida preventiva sarebbe troppo complessa: è necessario gestire l'eccezione

# Prevenire o Gestire ?

```
package it.unibas.utilita;
import java.io.*;
public class Console {
   private Console() {}
   private static BufferedReader stdin = new BufferedReader(
                                    new InputStreamReader(System.in));

   public static int leggiIntero() {
      while (true) {
         try {
            String linea = stdin.readLine();
            int valore = Integer.parseInt(linea);
            return valore;
         } catch (NumberFormatException nfe) {
            System.out.println("*** Errore: non è un numero intero. Riprova.");
         }
      }
   }
```





# Prevenire o Gestire ?

- Attenzione
  - ⇨dal punto di vista delle prestazioni, lanciare e gestire un'eccezione è un'operazione costosa
  - ⇨che, tra l'altro, interrompe il normale flusso di controllo per cui richiede attenzione nella programmazione
  - ⇨conviene evitare di farlo sistematicamente quando questo è possibile

# Prevenire o Gestire ?

- Riassumendo
  - la domanda fondamentale è: quanto costa prevenire l'eccezione ?
  - nei casi in cui le convalide necessarie sono semplici, è opportuno che il chiamante prevenga l'eccezione
  - in quei casi in cui le convalide sarebbero troppo complesse, è possibile viceversa gestire l'eccezione; questi casi dovrebbero essere una minoranza





# Prevenire o Gestire ?

- In effetti
  - negli esempi visti, IllegalArgumentException e NumberFormatException sono entrambe eccezioni "non controllate"
  - il chiamante può effettivamente scegliere se gestirle o meno
- Esistono però anche "eccezioni controllate"
  - in questi casi il chiamante non può scegliere
  - il compilatore impedisce al chiamante di ignorare l'eccezione

# Riassumendo

- Introduzione
- Gestione delle Eccezioni
- Lanciare un'Eccezione
- L'Atteggiamento del Chiamante
- Catturare un'Eccezione
- Prevenire o Gestire ?





# Termini della Licenza