Marco Faella

Costruttori ed Eccezioni

Lezione n. 2

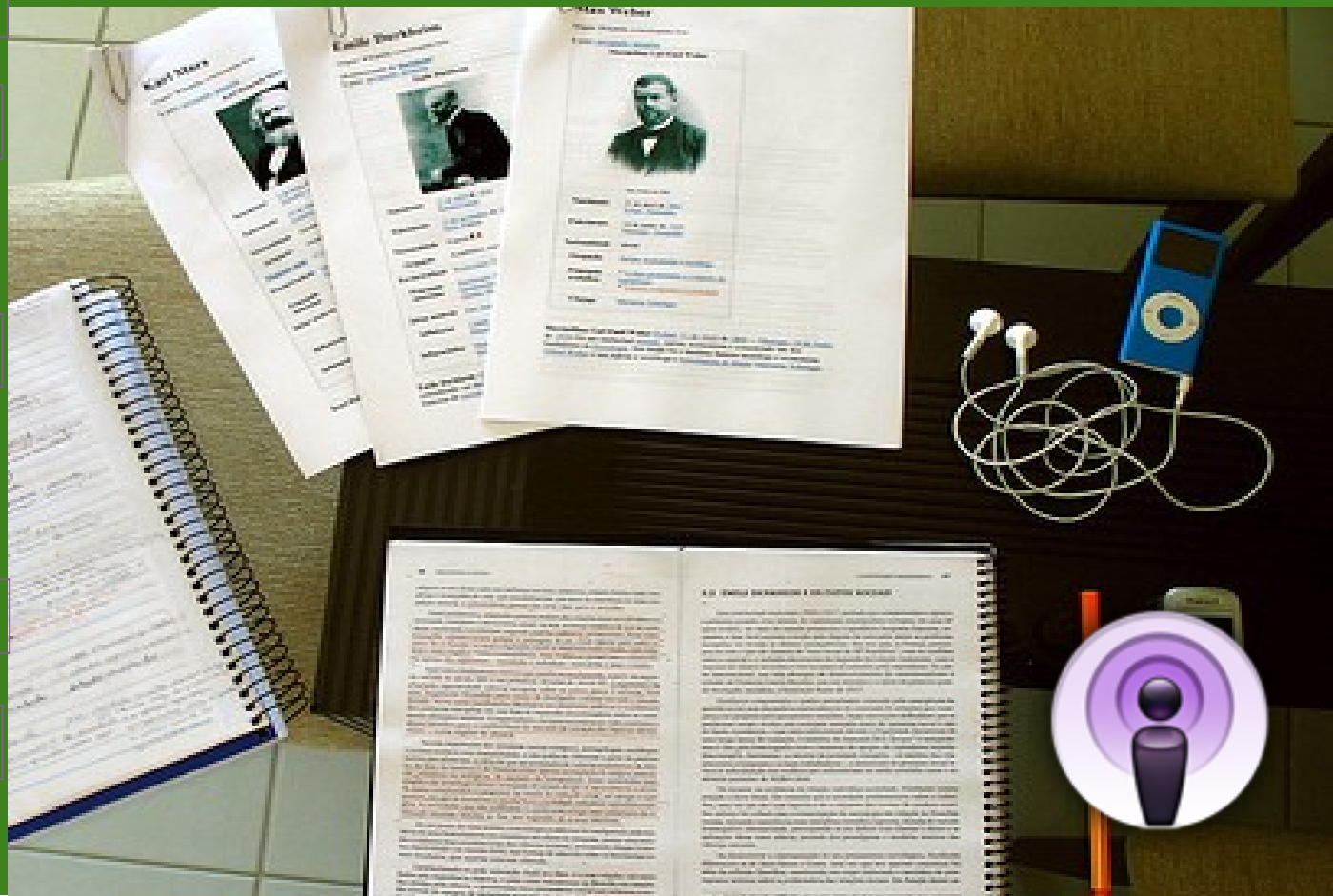
Parole chiave:
Java

Corso di Laurea:
Informatica

Insegnamento:
Linguaggi di Programmazione II

Email Docente:
faella.didattica@gmail.com

A.A. 2009-2010



Concatenazione dei costruttori

- Un costruttore può chiamarne un altro della stessa classe usando la parola chiave ***this***, oppure il costruttore della sua superclasse usando la parola chiave ***super***
- *this* e *super*, usati in questa accezione, devono comparire alla prima riga del costruttore, pena un errore di compilazione
- Attenzione: *this* e *super* hanno anche altri significati

  this rappresenta il riferimento all'oggetto corrente

  super si usa per invocare un metodo di una superclasse, oppure in una classe interna per ottenere il riferimento all'oggetto che ha creato quello corrente
- Se un costruttore non inizia con una chiamata ad un altro costruttore, viene automaticamente inserita una chiamata al costruttore senza argomenti della superclasse
- in questo caso, se la superclasse non ha un costruttore senza argomenti, si verifica un errore di compilazione

  Il meccanismo tramite il quale i costruttori possono invocarsi a vicenda prende il nome di concatenazione dei costruttori *(constructor chaining)*

Concatenazione ciclica

- Il compilatore controlla che la concatenazione non sia ciclica
- Infatti, se dei costruttori si chiamano a vicenda, siccome tali chiamate si trovano alla prima riga del rispettivo costruttore, e pertanto avvengono incondizionatamente, ci si trova in presenza di una mutua ricorsione non ben fondata, ovvero infinita
- Ad esempio, tentando di compilare la seguente classe:

```
class A {
   public A() { this(3); }
   public A(int i) { this(); }
}
```

- si ottiene il seguente errore di compilazione:

```
A.java:3: recursive constructor invocation
   public A(int i) { this(); }
          ^
```

Concatenazione ciclica

- Per analizzare la concatenazione dei costruttori di una data gerarchia di classi, ed in particolare controllare che essa non sia ciclica, è possibile realizzare il seguente diagramma:
  - si crei un nodo per ogni costruttore, compresi quelli impliciti
  - si crei un arco da un nodo x ad un nodo y se il costruttore x chiama, esplicitamente o meno, il costruttore y
  - il grafo ottenuto non deve presentare cicli

Esercizio 1

- Determinare l'output del seguente programma

```
public class A {
    public A() {
        System.out.println("A()");
    }
}

public class B extends A {
    public B() {
        this(0);
        System.out.println("B()");
    }
    public B(int n) {
        System.out.println("B(int)");
    }
    public static void main(String[] args) {
        new B();
    }
}
```

- Esercizio d’esame 27/3/08, #4

Eccezioni

Eccezioni verificate e non verificate

- Le classi di eccezione si dividono in **verificate** e **non verificate** (in inglese, **checked** ed **unchecked**)
- Il termine “verificata” si riferisce al fatto che il compilatore verifica che tali eccezioni siano opportunamente trattate dal programmatore (si veda dopo), mentre le eccezioni non verificate sono ad uso libero

- Intuitivamente, se in un metodo c'è un'istruzione che potrebbe lanciare un'eccezione verificata,il compilatore verifica che tale istruzione sia contenuta in un blocco try...catch, oppure che il metodo dichiari che può lanciare quella eccezione

Se

un metodo contiene l'istruzione “throw x”,

oppure

chiama un metodo la cui intestazione contiene la dichiarazione “throws x”, dove x è un'eccezione verificata,

allora

il compilatore controlla che sia rispettata una delle seguenti condizioni:

1) l'istruzione throw (oppure la chiamata) è contenuta in un blocco “try...catch” in cui una delle clausole catch è in grado di catturare x,

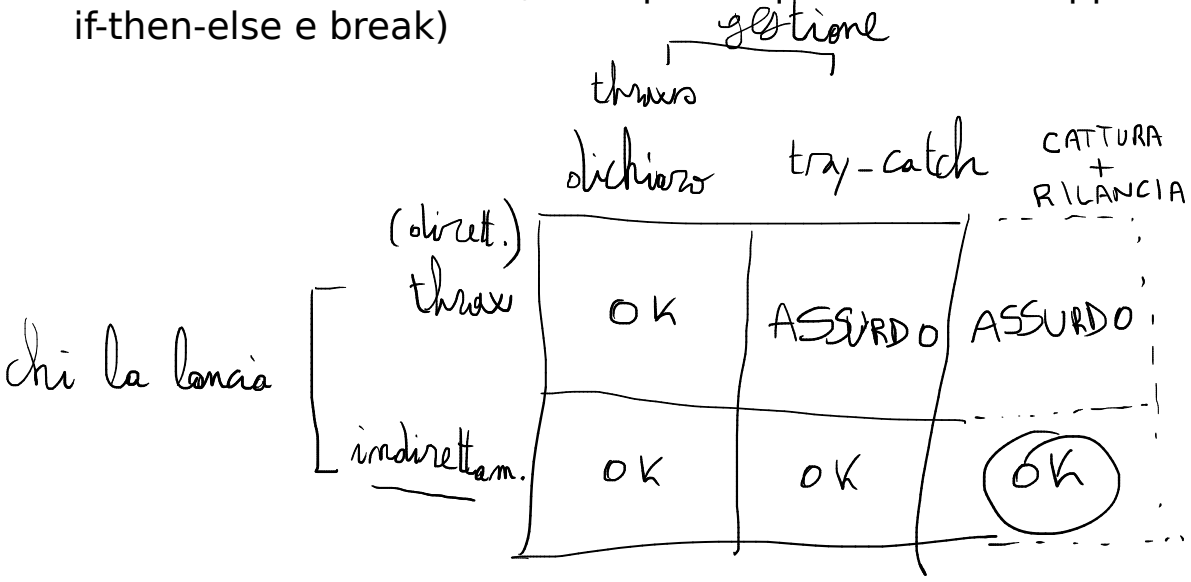
oppure

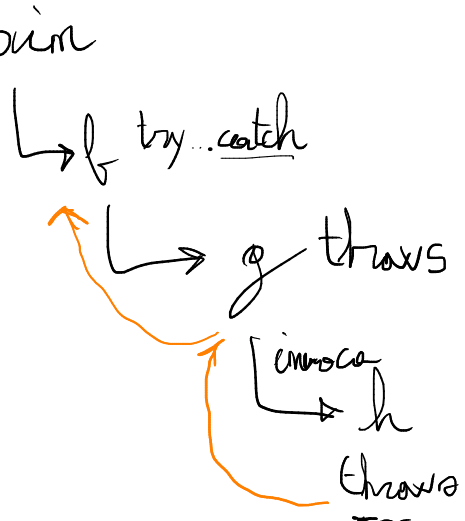
2) il metodo in questione contiene nell'intestazione la dichiarazione “throws y”, dove y è una superclasse di x

Eccezioni verificate e non verificate

- Delle quattro combinazioni che sono corrette per il compilatore, una non ha senso: che un metodo lanci direttamente un'eccezione con throw e subito la catturi con try...catch.

- Le eccezioni servono per *segnalare una situazione anomala* al chiamante, mentre in questo caso l'eccezione sarebbe usata solo per modificare il flusso di controllo all'interno di un metodo, cosa per la quale esistono apposite istruzioni (in particolare, if-then-else e break)





Eccezioni verificate e non verificate

- **Con quale criterio si dovrebbe scegliere tra le due categorie di eccezioni?**
- In linea di massima, dovrebbero essere di tipo **verificato** quelle eccezioni che il programmatore **non può evitare** con certezza perché dipendono da **fattori esterni al programma**
- Viceversa, dovrebbero essere di tipo **non verificato** quelle eccezioni che derivano da **errori di programmazione** (e quindi evitabili)

Eccezioni verificate e non verificate

- Ad esempio, si consideri l'eccezione che viene lanciata quando si cerca di **aprire un file inesistente** *(java.io.FileNotFoundException)*
- Per quanto si sforzi, il programmatore non può avere la certezza che tale eccezione non venga lanciata
  - Difatti, anche se il programmatore controllasse l'esistenza del file subito di prima di aprirlo, nei moderni sistemi multi-tasking niente può impedire ad un altro programma di cancellare il file in un momento intermedio tra il controllo di esistenza e la sua apertura vera e propria
- Inoltre, è utile catturare questa eccezione, anche solo per comunicare all'utente che il file non esiste
- Pertanto, l'eccezione *FileNotFoundException* è **verificata**

- Come ulteriore esempio, si consideri l'eccezione *ArrayOutOfBoundsException*
  - viene lanciata dalla Java Virtual Machine (JVM) quando si tenta di accedere ad un array con un **indice non valido**

- Questa eccezione è evitabile da parte del programmatore e difatti indica la presenza di un **errore di programmazione**
- Inoltre, se tale eccezione fosse verificata, il programmatore dovrebbe dichiararla o trattarla in occasione di *ogni accesso ad array*

- Pertanto, *ArrayOutOfBoundsException* **non è verificata**

Eccezioni verificate e non verificate

- Infine, si consideri l'eccezione *OutOfMemoryError*
  - viene lanciata dalla JVM se il progamma ha richiesto una allocazione di memoria (istruzione *new*), ma non vi è memoria libera a sufficienza

- Anche se tale situazione **non è evitabile** da parte del programmatore, c'è poco che il programmatore possa fare di conseguenza, tranne terminare il programma, che del resto è il comportamento di default in seguito al lancio di un'eccezione
- Inoltre, come per *ArrayIndexOutOfBoundsException*, le occasioni per il verificarsi di questa eccezione sono **troppo frequenti** per obbligare il programmatore a gestirla

- Pertanto, *OutOfMemoryError* è un'eccezione **non verificata**

- Per approfondire questo argomento, si consulti anche l'articolo *“Simple guide to checked exceptions”*, di Gabriele Carcassi, facilmente rintracciabile sul web

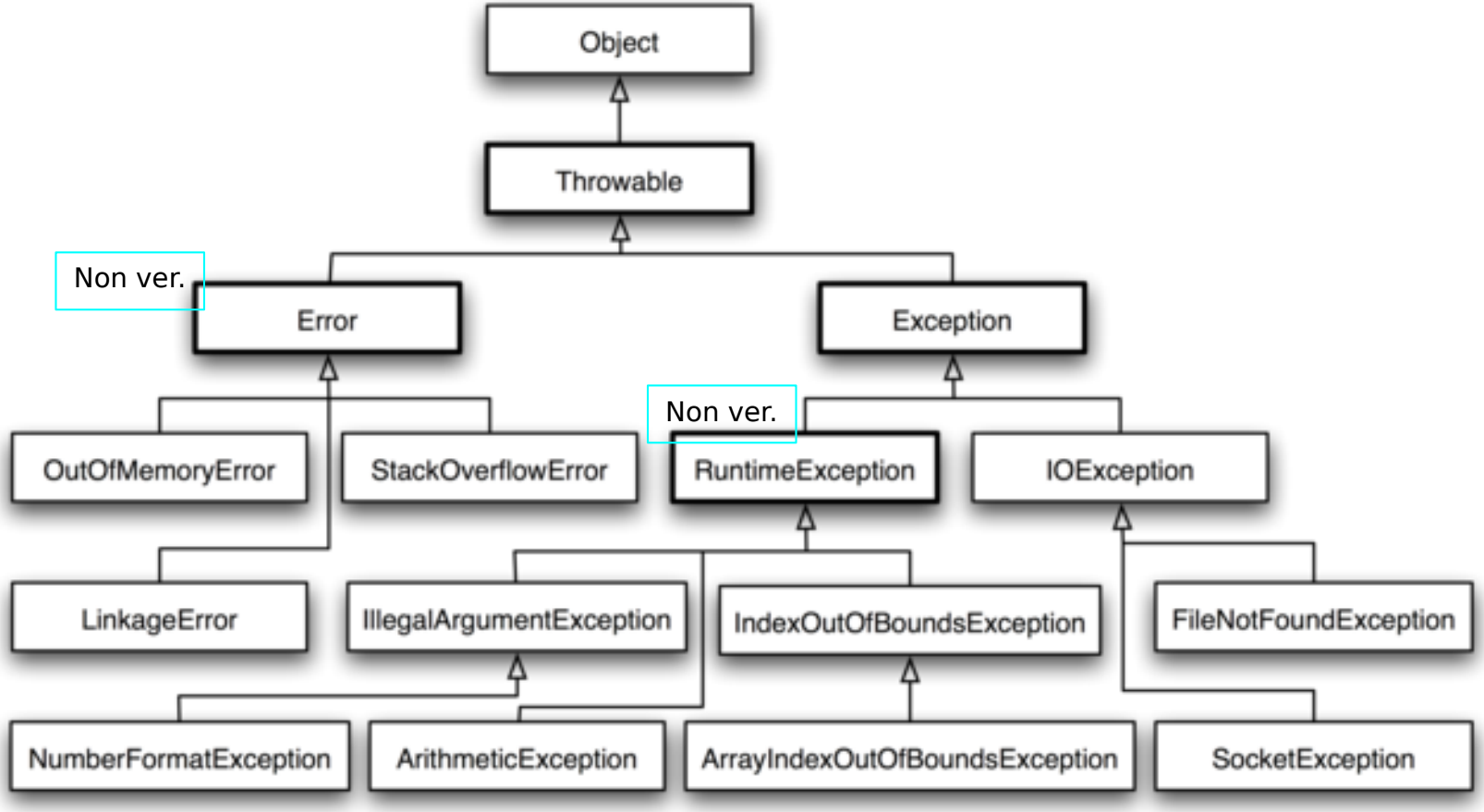
Eccezioni verificate e non verificate

- Quali eccezioni della libreria standard sono verificate?
- Come si imposta la categoria di una nuova classe di eccezioni?

Tutte le eccezioni della libreria standard sono verificate, tranne quelle che discendono da *RuntimeException* o da *Error*

- Quando si crea una nuova classe di eccezioni, tale classe eredita la categoria (verificata o non verificata) dalla sua superclasse

Eccezioni standard



- In ciascuno dei seguenti contesti, dire se è opportuno usare un'eccezione verificata o non verificata

1)In un metodo che calcola la radice quadrata, se l'argomento passato è negativo

2)In un metodo che interagisce con una stampante, se la stampante ha esaurito la carta

3)In un metodo che accetta un oggetto, se l'argomento passato è *null*

4)In un metodo che chiude un file, se quel file non era stato precedentemente aperto

5)In un metodo che accede ad un campo istanza privato, se quel campo si trova in uno stato non valido (cioè, contiene un valore non previsto)