In questa lezione

- Gestione delle eccezioni
  - Procedure parziali
  - Sollevare e gestire le eccezioni
  - Propagazione delle eccezioni
  - Eccezioni checked e unchecked
  - Definire nuove eccezioni
  - Reflecting e masking di eccezioni

Procedure "parziali"

- Molte procedure (metodi) sono parziali, cioè hanno un comportamento specificato solo per un sotto-insieme del dominio degli argomenti
- Per esempio

```
public class Rettangolo{
 public void setBase(int base){
    //ha senso solo se base > 0
    this.base = base;
 }
…
}
```

2

Procedure “parziali” e “robustezza”

- Le procedure parziali compromettono la “robustezza” dei programmi
  - un programma è “robusto” se, anche in presenza di errori o situazioni impreviste, ha un comportamento ragionevole (o, per lo meno, ben definito)
  - per le procedure parziali, il comportamento per valori d’ingresso che non soddisfano i vincoli specificati è spesso non definito dalla specifica!
  - per tali valori d’ingresso (“inattesi”), potrebbero verificarsi errori run-time o, peggio, comportamenti impredicibili dell’applicazione
- Per ottenere programmi robusti, le procedure dovrebbero sempre essere “totali”!!!

3

Gestione di errori e situazioni eccezionali

- Una procedura parziale deve quantomeno poter segnalare l’impossibilità di produrre un risultato significativo o la propria terminazione scorretta
- Gestione tradizionale a fronte di errori e situazioni eccezionali: la procedura può
  - terminare il programma
  - restituire un valore convenzionale che rappresenti l’errore
  - restituire un valore corretto e portare l’oggetto o l’intero programma in uno stato "scorretto" (es. usare un attributo ERROR)
  - richiamare una funzione predefinita per la gestione degli errori

4

Problemi (i/iv)

- Terminare il programma
  - è spesso una soluzione troppo drastica
  - a rigore è una scelta che spetta al chiamante e non al chiamato

```
public void setBase(int base){
    if(base <= 0) System.exit(-1);
    this.base = base;
}
```

5

Problemi (ii/iv)

- Uso di valori di ritorno convenzionali
  - può non essere fattibile
    - perché la procedura non ha un valore di ritorno (void, oppure si tratta di un costruttore)
    - o perché qualsiasi valore di ritorno è ammissibile
  - in generale dà poche informazioni riguardo l'errore incontrato
  - condiziona il chiamante che deve controllare il valore di ritorno

```
public boolean setBase(int base){
    if(base <= 0) return false;
    this.base = base;
    return true;
}
```

6

Problemi (iii/iv)

- Portare il programma in uno stato scorretto
  - la procedura chiamante potrebbe non accorgersi che il programma è stato portato in uno stato "scorretto"

```
public class Rettangolo{
   private boolean ERROR = false;
   public boolean isErrorState(){
      return this.ERROR;
   }
   public void setBase(int base){
      if(base <= 0) this.ERROR = true;
      else this.base = base;
   }
```

7

Problemi (iv/iv)

- Uso di una funzione predefinita per la gestione degli errori
  - diminuisce la leggibilità del programma
  - centralizza la gestione degli errori (che spetterebbe al chiamante)

```
public void setBase(int base){
   if(base <= 0)
      ErrorHandler.HandleNonPositiveBase();
   else this.base = base;
}
```

8

Gestione esplicita delle eccezioni

- Una procedura/metodo può terminare normalmente (con un risultato valido) o sollevare un'eccezione
- Le eccezioni vengono segnalate al chiamante che può gestirle
- Le eccezioni hanno un tipo e dei dati associati che danno indicazione sul problema incontrato
- Le eccezioni possono essere definite dall'utente (personalizzazione)

9

Uso delle Eccezioni in Java

Sollevare le eccezioni

- Per sollevare esplicitamente un'eccezione, si usa il comando **throw** seguito da un oggetto del tipo dell'eccezione
- Semantica (informale) del comando throw
  - termina l'esecuzione del blocco di codice che lo contiene, generando un'eccezione del tipo specificato

```
public void setBase(int base){
   if(base <= 0)
      throw new NonPositiveBaseException();
   this.base = base;
}
```

10

Uso delle Eccezioni in Java

Gestire le eccezioni

- Un'eccezione può essere catturata e gestita attraverso il costrutto:

  try {<blocco>} catch(ClasseEccezione e) {<codice di gestione>}

```
n = …; //ottengo un numero qualsiasi
try{
   unOggettoRettangolo.setBase(n);
} catch (NonPositiveBaseException e){
   //codice per gestire l'eccezione
   //qui è possibile usare l'oggetto e
}
```

- Più clausole catch possono seguire lo stesso blocco try
- Un ramo *catch(Ex e)* può gestire un'eccezione di tipo *T* se *T* è di tipo *Ex* o se *T* è un sottotipo di *Ex*

11

Uso delle Eccezioni in Java

Propagazione delle eccezioni

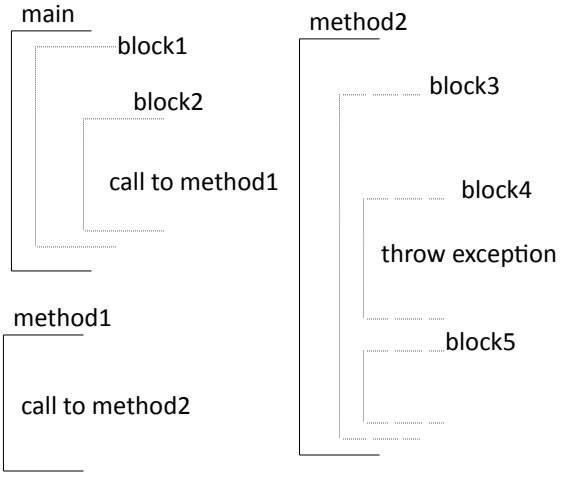
- Se invocando una procedura si verifica un'eccezione:
  - si termina l'esecuzione del blocco di codice in cui si è verificata l'eccezione e...
  - se il blocco di codice corrente è un blocco try/catch, si passa il controllo al primo dei rami catch in grado di gestire l'eccezione, altrimenti...
  - si risalgono eventuali blocchi di codice più esterni fino a trovare un blocco try/catch che contenga un ramo catch che sia in grado di gestire l'eccezione, altrimenti...
  - l'eccezione viene propagata nel contesto del chiamante
  - la propagazione continua attraverso blocchi e metodi fino a che
    - si trova un blocco try/catch che gestisce l'eccezione
    - il programma termina
- Quando un'eccezione viene gestita da un blocco try catch, successivamente l'esecuzione del programma continua dal comando successivo al blocco stesso

12

Uso delle Eccezioni in Java

Flusso in presenza di eccezioni

- Flusso:
  - main attivato
  - block 1
  - block 2
  - Call method1
  - Call method2
  - block 3
  - block 4
  - ... eccezione!
  - propagazione dell'eccezione !

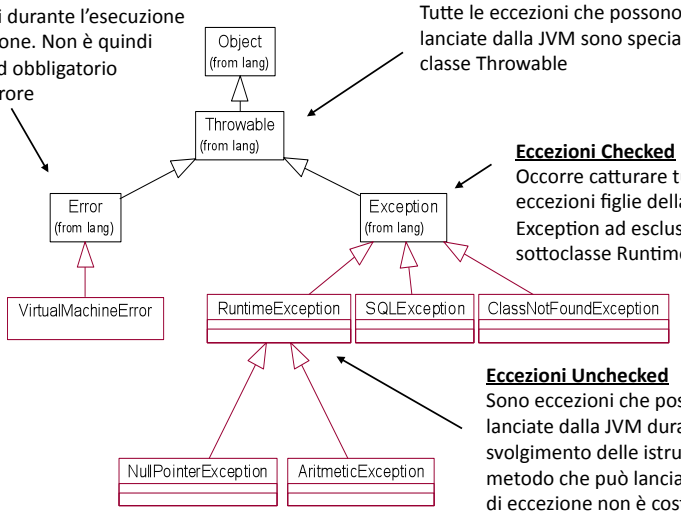


Uso delle Eccezioni in Java

Classi Java per le eccezioni

Un *errore* indica che sono occorsi seri problemi durante l'esecuzione dell'applicazione. Non è quindi opportuno ed obbligatorio catturare l'errore

Tutte le eccezioni che possono essere lanciate dalla JVM sono specializzazioni della classe Throwable



Eccezioni Checked
Occorre catturare tutte le eccezioni figlie della classe Exception ad esclusione della sottoclasse RuntimeException

Eccezioni Unchecked
Sono eccezioni che possono essere lanciate dalla JVM durante il normale svolgimento delle istruzioni. Un metodo che può lanciare questo tipo di eccezione non è costretto a dichiararlo nella sua intestazione

Uso delle Eccezioni in Java

Eccezioni checked

- Tutte le eccezioni derivate da Java.lang.Exception ad eccezione di quelle che etendono java.lang.sono dette
- In Java, il fatto che un metodo possa terminare sollevando un'eccezione checked DEVE essere dichiarato nell'interfaccia del metodo, dopo la clausola "throws"

```
public void setBase(int base)
   throws NonPositiveBaseException {
    if(base <= 0)
        throw new NonPositiveBaseException();
    this.base = base;
}
```

15

Uso delle Eccezioni in Java

Eccezioni checked

- Devono essere dichiarate dalle procedura che possono sollevarle (altrimenti si ha un errore a compile-time)
- Quando un metodo *M1* invoca un altro metodo *M2* che può sollevare un'eccezione di tipo *Ex* (checked),
  una delle due seguenti affermazioni deve essere vera
  (altrimenti di ha un errore a compile time):
  - l'invocazione di *M2* in *M1* avviene internamente ad un blocco try/catch che gestisce eccezioni di tipo *Ex* (quindi, *M1* gestisce l'eventuale eccezione)
  - il tipo *Ex* (o un suo sopra-tipo) fa parte delle eccezioni dichiarate nella clausola *throws* della procedura *M1* (quindi, *M1* propaga l'eventuale eccezione)

16

Uso delle Eccezioni in Java

Eccezioni unchecked

- Possono propagarsi senza essere dichiarate in nessuna interfaccia di metodo e senza essere gestite da nessun blocco try/catch
- Molte eccezioni predefinite sono di tipo unchecked
  - ClassCastException
  - NullPointerException
  - ArrayOutOfBoundException
  - …

17

Uso delle Eccezioni in Java

Definizione di nuove eccezioni

- Gli oggetti di un qualunque tipo *T* definito dall'utente possono essere usati per sollevare e propagare eccezioni, a condizione che
  - *T* sia definito come sotto-tipo della classe Exception (o RuntimeException)
- La definizione della classe che descrive un'eccezione non differisce dalla definizione di una qualsiasi classe definita dall'utente
  - In particolare può possedere attributi e metodi propri (usati per fornire informazioni aggiuntive al gestore dell'eccezione)

18

Uso delle Eccezioni in Java

Definizione di nuove eccezioni

- La classe `Throwable` (base di tutte le eccezioni ed errori) possiede sia un costruttore senza parametri che un costruttore che accetta in ingresso una stringa
  - la stringa identifica il tipo di malfunzionamento
  - Ogni la classe creata per definire una nuova eccezione dovrebbe pubblicare entrambi i costruttori

Uso delle Eccezioni in Java

Definizione di nuove eccezioni

- Definizione:

```
public class NewKindOfException extends Exception{
   public NewKindOfException(){ super();}
   public NewKindOfException(String s){
     super(s);
   }
}
```

- uso: `throw new NewKindOfException("problema!!!");`
- gestione:

```
try{....}
catch(NewKindOfException e){
   String s = e.toString( );
   System.out.printls(s);
 }
```

Uso delle Eccezioni in Java

Il ramo finally

- Un blocco try/catch può avere un ramo `finally` in aggiunta a uno o più rami `catch`
- Il ramo `finally` è comunque eseguito
  - sia che all'interno del blocco `try` non vengano sollevate eccezioni
  - sia che all'interno del ramo `try` vengano sollevate eccezioni. (Il ramo `finally` viene eseguito dopo il ramo `catch` che gestisce l'eccezione)

```
try {
    ...
} catch(Exception1 e) {
    //gestione dell'eccezione Exception1
} catch(Exception2 e) {
    //gestione dell'eccezione Exception2
} finally {
    //codice sempre eseguito
}
```

21

Programmare con le eccezioni: reflecting

- La gestione dell'eccezione comporta la propagazione di un'ulteriore eccezione (dello stesso tipo o di tipo diverso)

```
public int min(int[] a)
  throws EmptyException {
      int m;
      try {
         m = a[0];
      } catch (IndexOutOfBoundsException e){
         throw new EmptyException("Array is empty");
      }
      for (int i = 1; i < a.length; i++)
         if (a[i] < m) m = a[i];
      return m;
  }
```

22

Programmare con le eccezioni: masking

- Dopo la gestione dell'eccezione, l'esecuzione continua seguendo il normale flusso del programma

```
public boolean sortedAscending (int[] a)
   throws NullPointerException {
    int previous;
    try {
       previous = a[0];
       for (int i=1; i < a.length; i++) {
          if (previous <= a[i]) previous = a[i];
          else return false;
       }
    }catch (IndexOutOfBoundsException e){
       System.out.println("special: empty => sorted");
    }
    return true;
}
```

23

Progettare le eccezioni

- Uso delle eccezioni
  - gestione dei casi in cui le precondizioni di un metodo non sono soddisfatte dal chiamante
  - gestione della codifica di informazioni particolari nei risultati delle procedure
  - realizzazione di procedure più generali e riusabili
- Le eccezioni unchecked dovrebbero essere evitate il più possibile. Il loro uso dovrebbe essere limitato ai casi in cui
  - c'è un modo conveniente e poco costoso di evitare l'eccezione (per gli array, le eccezioni di tipo OutOfBoundException possono essere evitate controllando in anticipo il valore dell'attributo *length* dell'array)
  - l'eccezione è usata solo in un contesto ristretto

24

Convenzione Java per le eccezioni

- Sebbene sia possibile scegliere liberamente i nomi delle nuove eccezioni definite, è buona convenzione farli terminare con la parola *Exception*

  `NotFoundException` piuttosto che `NotFound`

25

ESERCIZI: GESTIONE ECCEZIONI

Eccezioni: esercizio 1

- Considerare un generico programma Java in cui ci siano due classi in una gerarchia di ereditarierà
- Usando solo reference del tipo più generale, creare oggetti delle due classi e almeno un reference null, quindi effettuare dei downcast esplicito di tutti i reference, e osservare in quali casi di ha un'eccezione ClassCastException o NullPointerException
- Creare un blocco che gestisce le eccezioni attraverso catch in cascata per NullPointerException, ClassCastException e Exception, con un blocco finally in coda.
- Osservare il comportamento delle eccezioni modificando l'esempio in vari modi

Eccezioni: esercizio 2

- Realizzare un metodo
  - boolean isSorted(int a[])

  che verifica se un array di interi passato come parametro è ordinato in ordine ascendente
- Effettuare test di funzionamento per array di varie dimensioni

Eccezioni: esercizio 3

- Realizzare un metodo
  - public int min(int a[])

  che restituisce il valore dell'elemento minimo nell'array. Nel caso l'array sia vuoto il metodo restituisce un'eccezione checked di tipo EmptyArrayException
- Effettuare test di funzionamento per array di varie dimensioni