Exceptions in Java [DRAFT]

Rationale

Necessita' di sviluppare programmi "robusti", ovvero in grado di reagire opportunamente anche in caso di errori, per esempio errori di sistema, del programmatore, o un impiego scorretto da parte degli utenti (per esempio altri metodi).

Soluzioni tradizionali:

```
public float divisione (float dividendo, float divisore) {
        if (divisore ==0) {
                errore = TRUE;
                return -1;
        }
        else {
        return (dividendo/divisore)
        }
}
```

Problemi:

-1 potrebbe indicare un possibile risultato corretto → si potrebbe usare una variabile globale…

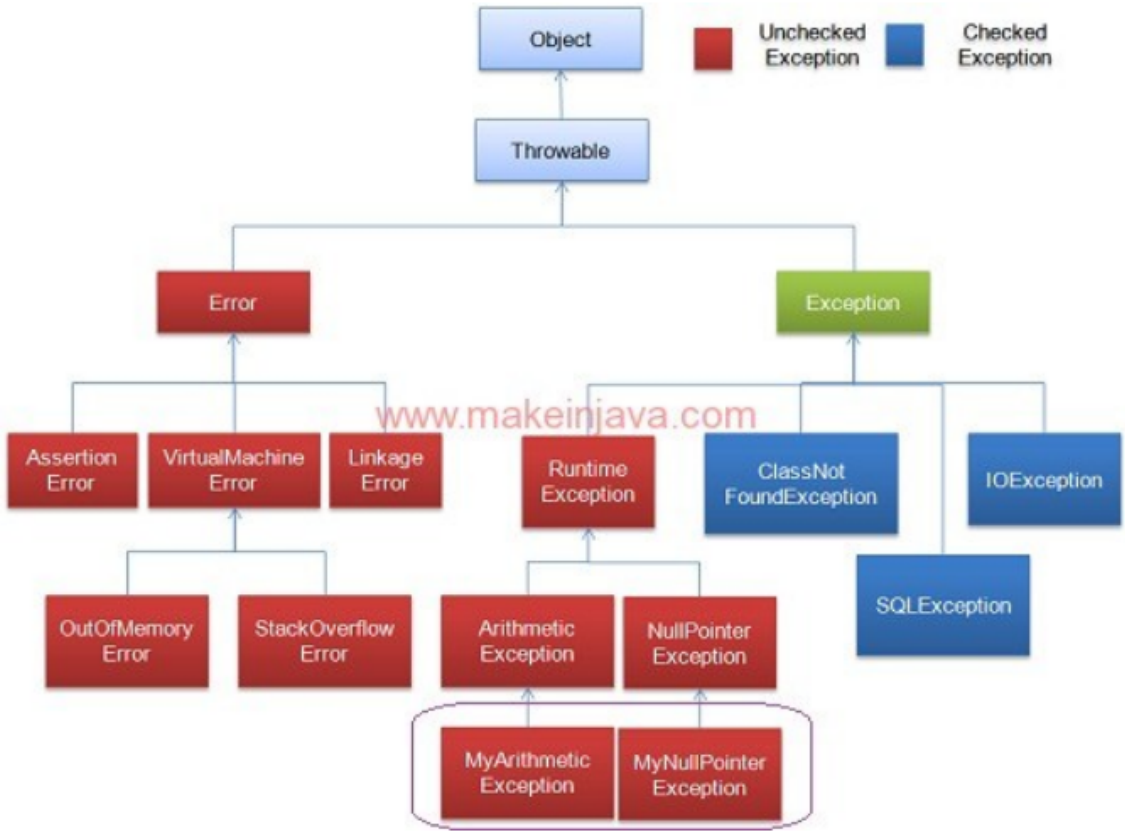
- da evitare in generale: viola il principio dell'incapsulamento;
- in programmi concorrenti causa problemi di "interferenze";
- giungla di if-then-else: il codice regolare risulta frammisto al codice di gestione degli errori…

Risultato: il codice si complica - in programmi complessi si aveva piu' codice per gestire i casi di errore che codice regolare!

Soluzione: le exceptions

➔ Meccanismo delle exceptions (oggetti utilizzati per gestire "problemi" di esecuzione), da:
  - creare
  - segnalare
  - intercettare
  - gestire

Esistono numerose Exceptions predefinite, che formano una gerarchia ISA, che puo' ovviamente essere estesa con exceptions custom:



Checked exceptions

Checked exceptions are subclasses of Exception .

Checked exceptions are the objects of the Exception class or any of its subclasses excluding the Runtime Exception class. Checked Exceptions are the invalid conditions that occur in a java program due to invalid user input, network connectivity problem or database problems.

Java uses the try-catch block to handle the checked exceptions. The statements within a program that throw an exception are placed in the try block. You associate an exception-handler with the try block by providing one or more catch handlers immediately after the try block.

Various checked exceptions defined in the java.lang.package are,

- ClassNotFoundException
- IllegalAccessException
- InstantiationException
- NoSuchMethodException

Unchecked exceptions

It represent defects in the program (bugs), that effect due to programming error. It often invalid arguments passed to a non-private method. To quote from The Java Programming Language, by Gosling, Arnold, and Holmes : “Unchecked runtime exceptions represent conditions that, generally speaking, reflect errors in your program’s logic and cannot be reasonably recovered from at run time.”

These are subclasses of RuntimeException, and are usually implemented using IllegalArgumentException, NullPointerException, or IllegalStateException .

Various Unchecked Exceptions are,

- ArithmeticException
- ArrayIndexOutOfBoundsException
- ArrayStoreException
- ClassCastException
- IllegalArgumentException
- NegativeArraySizeException
- NullPointerException
- NumberFormatException

Error

The errors in java are external to the application. These are the exceptional conditions that could not be usually anticipated by the application and also could not be recovered from. Error exceptions belong to Error and its subclasses are not subject to the catch or Specify requirement. Suppose a file is successfully opened by an application for input but due to some system malfunction could not be able to read that file then the java.io.IOError would be thrown. This error will cause the program to terminate but if an application wants then the error might be caught. An Error indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions.

Esempi:

- int a=50/0;//ArithmeticException
- int a[]=new int[5]; a[10]=50; //ArrayIndexOutOfBoundsException
- String s="abc"; int i=Integer.parseInt(s);//NumberFormatException
- String s=null; System.out.println(s.length());//NullPointerException
- Nuove custom exceptions possono essere create sfruttando l'ereditarieta'.

Impiego base

Creazione e sollevamento (nel metodo):

```
{
…
DivisionByZero e = new DivisionByZero(); // creazione
throw e; // sollevamento
…
}
```

Catch or Specify: un metodo che usa codice che potrebbe sollevare eccezioni, deve gestirle (catch) oppure dichiarare (specify) che le potrebbe "passare" – as follows.

Specify (nella dichiarazione del metodo):

```
float division (a float, b float) throws DivisionByZero() {…} // segnalo la possibilita' di
                                                              sollevamento di una exception
```

Catch: Esempio di intercettazione e gestione con struttura classica (dal metodo invocante):

```
public int getAge (String: prompt) {
int age;
while (TRUE) {
  System.out.println(prompt);
  try {
    age = scanner.nextInt();
    return (age);
  } catch (InputMismatchException e) {
    scanner.next() // removes input garbage left in the buffer
    System.out.println("Introdurre solo numeri!");
  }
}
```

Struttura classica alternativa (quando non si puo' sfruttare il return):

```
  boolean continue = TRUE;
  while (continue) {
    …
    try {
          …
          continue = false;
    } catch (…) {
    }
  }
```

Documentazione (da vedere!)

- https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html
- https://www.javatpoint.com/exception-handling-in-java
- capitolo 10 libro DeSio.
- javadoc – esempio https://docs.oracle.com/javase/8/docs/api/java/lang/Exception.html

Esercizio risolto

Un oggetto di classe Tentazione possiede un metodo provocaEccezione che genera, in modo equiprobabile, un'eccezione di tipo EccezioneMia e di tipo NullPointerException. Scrivere un'applicazione in grado di catturarle e riconoscerle.

```
class MostraEccezioni {
    public static void main (String[] args) {
        try {
            Tentazione t = new Tentazione();
            t.provocaEccezione();
        }
        catch (EccezioneMia e) {
            System.out.printn ("Catturata:" + e.getMessage());
        }
    }
}
class Tentazione {
```

```
        void provocaEccezione() throws EccezioneMia, NullPoinerException {
                if (Math.random()<0.5)
                        throw new EccezioneMia ("…EccezioneMia…");
                else
                        throw new NullPointerException("… Mia NullPE…");
        }
}
class EccezioneMia extends Exception {
        EccezioneMia (String s) { super(s)}
}
```

Cattura di piu' eccezioni

[Vedi a https://stackoverflow.com/questions/11211286/is-it-possible-in-java-to-catch-two-exceptions-in-the-same-catch-block]

Tre tecniche:

- try / catch / catch
- cattura di una eccezione superclasse
- dopo ver 6: catch (Exception1 | Exception2 ex), dove ex e' la comune superclasse piu' specifica

Esercizi con custom exception

1. Considerare la casse Motocicletta e il metodo faiBenzina(l: intero). Creare una custom exception GasolineSpillException per gestire il caso in cui si tenti di inserire nel serbatoio una quantita' di combustibile superiore alla sua capacita'. L'eccezione deve riportare le seguenti informazioni di errore: la stringa "Gasoline spill!" e l'intero " che indica il numero di litri "fuoriusciti" (versati in terra). Quindi:
   - Codificare il metodo faiBenzina(l: intero) throws GasolineSpillException()
   - Utilizzare il metodo faiBenzina, gestendo l'eccezione.

2. Creare un'exception opportuna per il metodo preleva (quindi trasferisci…) della classe ContoCorrente.