UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI **MATEMATICA E INFORMATICA**

Interfacce Grafiche e Programmazione ad Eventi

Carmine Dodaro

Anno Accademico 2019/2020

Gestione degli errori

Gli errori

- Durante la fase di sviluppo del codice è normale avere errori software

Gestione degli errori

Gli errori

- Durante la fase di sviluppo del codice è normale avere errori software

```
import java.util.Scanner;
public class Calcolatrice {
  public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    int numero1 = in.nextInt();
    int numero2 = in.nextInt();
    System.out.println(numero1/numero2);
    in.close();
  }
}
```

Gli assert

- Come in C++, anche in java è presente la keyword assert:

```
assert espressione;
assert espressione :  espressione2;
```

- Se un assert fallisce, il programma termina!
- Per abilitare l'uso di assert è necessario aggiugere l'opzione -ea alla JVM. In eclipse: Menu Run -> Run Configurations, a sinistra selezionare Java Application -> Nome Progetto e poi sulla destra selezionate Arguments e poi nel campo VM arguments aggiungere -ea

Gli assert

- Come in C++, anche in java è presente la keyword assert:

```
assert espressione;
assert espressione :  espressione2;
```

- Se un assert fallisce, il programma termina!
- Per abilitare l'uso di assert è necessario aggiugere l'opzione -ea alla JVM. In eclipse: Menu Run -> Run Configurations, a sinistra selezionare Java Application -> Nome Progetto e poi sulla destra selezionate Arguments e poi nel campo VM arguments aggiungere -ea

```
public class TestAssertion {
  public void test(int n) {
    if(n % 2 == 0)
      n++;
    else {
      assert n % 2 != 0 : "Mi aspetto un numero dispari";
      n+=2;
    }
    System.out.println(n);
  }
}
```

Gestione degli errori

Eccezioni

La gestione delle eccezioni è un meccanismo offerto da alcuni linguaggi di programmazione per intercettare e gestire potenziali errori software

Attraverso la gestione delle eccezioni:

- si può evitare che un programma termini bruscamente in presenza di un errore software
- si può cercare di riparare all'errore in modo opportuno
- si può separare la parte di gestione degli errori dal codice di un programma

La classe Exception

- La classe Exception e le sue sottoclassi estendono Throwable, ciò indica una condizione che un'applicazione dovrebbe prevedere
- Le eccezioni sono gestite attraverso le keyword:
  - try/catch: con try si può specificare un blocco di codice in cui potrebbe verificarsi un'eccezione, mentre con catch si specifica quale eccezione si vuole gestire e inserire il codice per gestirle
  - finally: permette di specificare un blocco di codice le cui istruzioni saranno eseguite sempre, a prescindere che si sia verificata o meno un'eccezione
  - throw: consente di lanciare una determinata eccezione software
  - throws: si inserisce vicino la definizione di un metodo e permette di rimandare la gestione dell'eccezione nei punti in cui il metodo è chiamato

Try/Catch

Sintassi

```
try {
    ...
}
catch (ExceptionT1 | ExceptionT2 | ... | ExceptionTN) {
   ...
}
```

Prova ad eseguire il codice che si trova nel blocco del try, se si verifica un'eccezione tra quelle indicate nel catch, allora esegui il blocco del codice che si trova nel catch

Try/Catch

Sintassi

```
try {
    ...
}
catch (ExceptionT1 | ExceptionT2 | ... | ExceptionTN) {
   ...
}
```

Prova ad eseguire il codice che si trova nel blocco del try, se si verifica un'eccezione tra quelle indicate nel catch, allora esegui il blocco del codice che si trova nel catch

```
int numero1 = in.nextInt();
int numero2 = in.nextInt();
try {
  System.out.println(numero1/numero2);
} catch (ArithmeticException e){
  e.printStackTrace();
}
```

Finally

Sintassi

```
try { ... }
catch (ExceptionT1 | ExceptionT2 | ... | ExceptionTN) {
   ...
}
finally { ... }
```

oppure:

```
try { ... }
finally { ... }
```

Il codice che si trova nel blocco finally è eseguito sempre, indipendentemente se si è verificata un'eccezione o meno

Finally

Esempio

```
import java.util.Scanner;
public class Calcolatrice {
  public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    int numero1 = in.nextInt();
    int numero2 = in.nextInt();
    try {
      System.out.println(numero1/numero2);
    } catch (ArithmeticException e){
      System.out.println("Divisione per 0!");
    } finally {
      in.close();
    }
  }
}
```

Throw

Sintassi

```
throw new ExceptionType();
```

Lancia un'eccezione del tipo indicato

Throw

Sintassi

```
throw new ExceptionType();
```

Lancia un'eccezione del tipo indicato

```
public class Test {
  public static void main(String[] args) {
    int numero = 11;
    if(numero > 10) {
    try {
      throw new Exception("Numero maggiore di 10");
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    }
  }
}
```

Throws

Sintassi

```
public void foo() throws ExceptionT1, ExceptionT2, ...,
    ExceptionTN {...}
```

Rimanda la gestione delle eccezioni ai metodi che chiamano `foo()`

Throws

Sintassi

```
public void foo() throws ExceptionT1, ExceptionT2, ...,
    ExceptionTN {...}
```

Rimanda la gestione delle eccezioni ai metodi che chiamano `foo()`

```
public static void test() throws Exception {
  int numero = 11;
  if (numero > 10)
    throw new Exception("Numero maggiore di 10");
}
public static void main(String[] args) {
  try {
    test();
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

TIpi di eccezioni

Eccezioni unchecked e checked

Java distingue due tipi di eccezioni:

- eccezioni unchecked: sono le eccezioni che possono non essere gestite, ad esempio ArrayOutOfBoundsException e NullPointerException
- eccezioni checked: sono le eccezioni che vanno sempre gestite, ad esempio IOException

Dichiarare nuovi eccezioni

- Definizione di eccezioni unchecked

```
public class MyException extends RuntimeException {
```

- Definizione di eccezioni checked

```
public class MyException extends Exception {
```

Input/Output: stream e file

Stream

Uno stream è una connessione associabile ad una sorgente (input stream) e a una destinazione (output stream).

La sorgente e la destinazione possono essere rappresentate da diversi oggetti (file, socket, console, ecc.)

Lettura da file: FileInputStream e FileOutputStream

La classe FileInputStream ci permette di leggere dei byte da un file e FileOutputStream ci consente di scrivere dei byte su un file

```
public static void main(String[] args) {
  try {
    FileInputStream in=new FileInputStream("i.txt");
    FileOutputStream out=new FileOutputStream("o.txt");
    int c;
    while ((c = in.read()) != -1)
      out.write(c);
    in.close();
    out.close();
    System.out.println("Operazioni eseguite!");
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Lettura da file: FileReader e FileWriter

La classe FileReader ci permette di leggere dei caratteri da un file e FileWriter ci consente di scrivere dei caratteri su un file

```
public static void main(String[] args) {
  try {
    FileReader in=new FileReader("i.txt");
    FileWriter out=new FileWriter("o.txt");
    int c;
    while ((c = in.read()) != -1)
      out.write(c);
    in.close();
    out.close();
    System.out.println("Operazioni eseguite!");
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Unbuffered vs Buffered

I tipi di lettura e scrittura visti finora sono unbuffered, cioè le operazioni sono effettuate dal sistema operativo che accede ai file al momento delle richieste, e quindi sono inefficienti.

Java mette a disposizione delle classi che permettono di usare stream buffered, che usano un'area di memoria in cui effettuano le operazioni prima che le stesse siano eseguite dal sistema operativo.

Lettura da file: BufferedReader e BufferedWriter

```
public static void main(String[] args) {
  try {
    BufferedReader bIn = new BufferedReader(new
        FileReader("i.txt"));
    BufferedWriter bOut = new BufferedWriter(new
        FileWriter("o.txt"));
    while(bIn.ready()) {
        String line = bIn.readLine();
        bOut.append(line);
        bOut.newLine();
    }
    bIn.close();
    bOut.close();
    System.out.println("Operazioni eseguite!");
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Una nota sul percorso dei file

Il percorso dei file

```
BufferedReader bIn = new BufferedReader(new FileReader("
    i.txt"));
```

In eclipse, il file `"i.txt"` è cercato nella cartella home del progetto, cioè allo stesso livello di bin e src

Se si volesse cercare il file in una cartella, si può utilizzare `File.separator`:

```
BufferedReader bIn = new BufferedReader(new FileReader("
    nomecartella" + File.separator+"input.txt");
```