**Try e catch**

Può capitare che durante l’esecuzione di un programma, venga generato un errore che può podurre effetti catastrofici come bloccare il computer o generare un overflow e altro.

Gli errori possono essere di tre tipi:

* Errori di programmazione- divisione per zero, superamento della dimensione di un array..
* Errore di sistema- hard disk rotto, connessione interrotta..
* Errori di utilizzo- apertura di un file inesistenti, errore nell’input di una variabile…

Si può evitare ciò attraverso la gestione dell’errore.

Il gestore delle eccezioni non fa altro che uscire dal blocco di programma che ha generato l’eccezione e terminare il programma.

Per la gestione dell’errore si utilizzano i due blocchi try e catch, trova e cattura.

try: intercetta le eccezioni;

catch: gestisce le eccezioni;

In generale:

*try{ operazione che può generare una eccezione}*

*catch(Exception e){gestione della eccezione}*

*//se tutto va bene, si procede*

Un solo blocco try può generare più tipi di eccezioni

*try{ operazione che può generare una eccezione}*

*catch(Exception e1){………….}*

*catch(Exception e2){………………}*

*……………………*

*finally{…………………….}// parte facoltativa che va eseguita comunque sia se si generano eccezioni o //che non si generano eccezioni*

In java esistono eccezioni predefinite.



Le eccezioni si dividono in due categorie:

Eccezioni controllate:

sono causate da eventi esterni che il programmatore non può evitare

il compilatore chiede cosa può fare di quelle eccezioni

Eccezioni non controllate:

sono causate da eventi che il programmatore può evitare correggendo il programma

**Eccezioni controllate predefinite**

ArithmeticException Operazione aritmetica non consentita.

InputMismatchException Valore immesso errato

BadStringOperationException Operazione non consentita.

ClassNotFoundException Classe non trovata.

IOException Errore in input o output.

NoSuchMethodException Metodo inesistente

Exception e errore generico

Sono tutte sottoclassi di Exception

Si può sempre mettere solo l’eccezione generica ma si possono avere problemi se le eccezioni sono più di una.

**Esempi**

**Esempio 1**

class Ideone

{ public static void main (String[] args) throws java.lang.Exception {

 int divisione,j;

 for(j=0;j<10;j++){

 int i=(int)(Math.random()\*3);

 try {

 divisione =5/i;

 System.out.println("Il risultato della divisione è:" + divisione); }

 catch (Exception exc) {

 System.out.println(exc);

 System.out.println("Non puoi effettuare una divisione per zero");

 } }

 System.out.println("Ho terminato comunque l'esecuzione del programma");}}

Nel programma precedente, si può verificare l’errore di divisione per zero. Il programma non si blocca; rileva l’errore, lo evidenzia e passa al comando successivo. Il tipo di errore evidenziato è generico.

**INPUT da tastiera**

**Esempio 2**

 public static void main (String[] args) throws java.lang.Exception

 {

BufferedReader input=new BufferedReader new InputStreamReader(System.in));

 try {

 System.out.print("Digita il nome:");

 String nome=input.readLine();

 System.out.println(nome);

 System.out.println("n mnb ");

 String cognome=input.readLine();

 System.out.println(cognome);

 }

 catch (Exception e) {

 System.out.println("Si è verificato un errore"); }}

**Esempio 3 Divisione per zero**

**package** errore1;

**public** **class** errore1 {

 **public** **static** **void** main(String[] args) {

 java.util.Scanner tastiera = **new** java.util.Scanner(System.***in***);

 **int** metri,minuti,vel;

 **try** {

 System.***out***.print("Inserire i metri percorsi: ");

 metri = tastiera.nextInt();

 System.***out***.print("Inserire il tempo impiegato a percorrerli: ");

 minuti = tastiera.nextInt();

 vel = metri / minuti;

 System.***out***.println("La tua velocità: " + vel + " metri al mi");

 } **catch** (Exception e) {

 System.***out***.println("Forse non hai fatto partire il cronometro " + e.getMessage());

 System.***out***.println("Possibile che sei così distratto?!?");

 }

 System.***out***.println("Ciao");}}

Nel programma precedente è stato inserito e.getMessage(), utile perché quando si cattura l’eccezione, si sottolinea anche il tipo di errore

 Dopo i blocco try e catch, si può inserire un ulteriore blocco: finally . Questo blocco viene eseguito indipendentemente se c’è l’eccezione o meno

**Esempio 4**

**package** errore1;

**import** java.util.InputMismatchException;

**public** **class** errore1 {

 **public** **static** **void** main(String[] args) {

 java.util.Scanner tastiera = **new** java.util.Scanner(System.***in***);

 **double** a=-4;

 **try** {

 System.***out***.println(Math.*sqrt*(a));

 } **catch** (ArithmeticException e) {

 System.***out***.println("numero negativo " + e.getMessage());

 } **catch** (InputMismatchException e) {

 System.***out***.println("Hai immesso un valore inatteso, riprova!");

 }

 **finally** {System.***out***.println("Fine programma");}

 System.***out***.println("... fine del programma.");}}