PER ALTRI APPUNTI CONSULTARE IL SITO:

https://luigi-v.github.io/Appunti_Universita/

Per altri appunti consultare il sito: *https://luigi-v.github.io/Appunti_Universita/*

By *Vollono Luigi*

Classe Random ed Ordinamento

La classe ***Random*** della libreria Java permette di generare numeri casuali, cioè produce numeri arbitrari. Contiene i seguenti metodi:

- ***nextInt(n)*** restituisce un numero intero casuale compreso fra 0 (incluso) e n (escluso);
- ***nextDouble()*** restituisce un numero in virgola mobile casuale compreso fra 0 (incluso) e 1 (escluso).

Esempio:

Random generatore = new Random();
int d = 1+ generatore.nextInt(6);

produce un numero casuale compreso tra zero e cinque inclusi.

Esempio:

```
import java.util.Random;

public class Dice
{          public static void main(String[] args)
   {    Random generator = new Random();
        // getta il dado dieci volte

        for (int i = 1; i <= 10; i++)
        {  int d = 1 + generator.nextInt(6);
          System.out.print(d + " ");
        }
        System.out.println();
   }
}
```

https://www.tutorialspoint.com/java/util/java_util_random.htm#

È possibile ordinare un ***ArrayList*** utilizzando il metodo **sort ()** della classe **Collections.** Questo metodo accetta un oggetto list come parametro e ne ordina il contenuto in ordine crescente.

Esempio:

```
import java.util.ArrayList e java.util.Collections;

public class TestSort {

        public static void main(String[] args){

            ArrayList<String> list = new ArrayList<String>();

            list.add("ghi");

            list.add("def");

            list.add("abc");

            System.out.println("\n Non ordinato :  " + list);

            Collections.sort(list);

            System.out.println("\n Ordinato :  " + list

          }

}
```

Per ordinare liste di oggetti in Java sono a disposizione 2 interfacce:

1. **java.lang.Comparable:**

L'interfaccia ***java.lang.Comparable*** di solito si utilizza per definire ***l'ordinamento "naturale" di un oggetto*** e consta di un metodo con firma:

int compareTo(T o), che paragona l'oggetto corrente (this) con l'oggetto fornito in input al metodo (o).

Il risultato sarà:

- un intero positivo se this>o
- 0 se sono uguali
- un intero negativo se this<0

Esempio:

```
public class Persona implements Comparable<Persona> {
        public Persona(String nome,String cognome,int eta){
                this.nome=nome;
                this.cognome=cognome;
                this.eta=eta;
        }
        private String nome;
        private String cognome;
        private int eta;
        @Override
        public int compareTo(Persona o) {
                return this.cognome.compareTo(o.cognome);
        }
}
```

2. **java.util.Comparator:**

L'interfaccia ***java.util.Comparator*** si utilizza invece quando si vogliono definire ***ordinamenti alternativi*** dell'oggetto. Si ***crea*** dunque una ***classe a parte*** che implementa l'interfaccia con il seguente metodo:

public int compare(T a,T b);

Il risultato sarà:

- un intero positivo se a>b
- 0 se sono uguali
- un intero negativo se a<b

Esempio:

```
import java.util.Comparator;
public class PersonaEtaComparator implements Comparator<Persona> {
```

```
	@Override
	public int compare(Persona p1, Persona p2) {
		int retVal=0;
		if(p1.getEta()>p2.getEta()){
			retVal=1;
		}
		else if(p1.getEta()<p2.getEta()){
			retVal=-1;
		}
		return retVal;
	}
}
```

Classi e Oggetti Java - https://www.w3schools.com/java/java_classes.asp

Tutto in Java è associato a classi e oggetti, insieme ai suoi attributi e metodi. Ad esempio: nella vita reale, un'auto è un oggetto. La macchina ha **attributi** , come il peso e colore, e **metodi** , quali motrice, freni.

Una classe è come un costruttore di oggetti o un "progetto" per la creazione di oggetti.

Creazione una classe:

Per ***creare*** una classe, utilizzare la parola chiave ***class***:

```
public class MyClass {
  int x = 5;
}
```

Creare un oggetto:

Per creare un oggetto di MyClass, specificare il nome della classe, seguito dal nome dell'oggetto e utilizzare la parola chiave ***new***:

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Puoi creare più oggetti di una classe:

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj1 = new MyClass();  // Object 1
    MyClass myObj2 = new MyClass();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

Puoi anche creare un oggetto di una classe e accedervi in un'altra classe. Questo è spesso usato per una migliore organizzazione delle classi (una classe ha tutti gli attributi e i metodi, mentre l'altra classe contiene il **main**()).

```
class OtherClass {
  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Attributi della Classe: https://www.w3schools.com/java/java_class_attributes.asp

Classe chiamata " MyClass" con due attributi: x e y:

```
public class MyClass {
  int x = 5;
  int y = 3;
}
```

Un altro termine per attributi di classe è **campi** .

Accesso agli attributi:

Puoi accedere agli attributi creando un oggetto della classe e usando la sintassi **punto** (**.**):

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Modifica degli Attributi:

Impostare il valore su x su 40:

```
public class MyClass {
  int x;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

Se non si desidera la possibilità di sovrascrivere i valori esistenti, dichiarare l'attributo come ***final***:

```
public class MyClass {
  final int x = 10;

  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    myObj.x = 25; // will generate an error: cannot assign a value to afinal variable
    System.out.println(myObj.x);
  }
}
```

Oggetti Multipli:

Se si creano più oggetti di una classe, è possibile modificare i valori degli attributi in un oggetto, senza influire sui valori degli attributi nell'altro.

Modificare il valore di ***x*** 25 a ***myObj2***, e lasciare ***x*** in ***myObj1*** invariata:

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj1 = new MyClass();  // Object 1
    MyClass myObj2 = new MyClass();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

Metodi della Classe: https://www.w3schools.com/java/java_class_methods.asp

Crea un metodo denominato ***myMethod()*** in ***MyClass***:

```
public class MyClass {
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```

Per chiamare un metodo, scrivi il nome del metodo seguito da due parentesi () e un punto e virgola ;

```
public class MyClass {
  static void myMethod() {
    System.out.println("Hello World!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}
// Outputs "Hello World!"
```

Static o Public:

Nell'esempio sopra, abbiamo creato un ***static*** metodo, il che significa che è possibile accedervi senza creare un oggetto della classe, diversamente ***public***, a cui possono accedere solo gli oggetti:

```
public class MyClass {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

    MyClass myObj = new MyClass(); // Create an object of MyClass
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

Costruttore Java: https://www.w3schools.com/java/java_constructors.asp

Un costruttore in Java è un **metodo speciale** utilizzato per inizializzare gli oggetti. Il costruttore viene chiamato quando viene creato un oggetto di una classe. Può essere utilizzato per impostare i valori iniziali per gli attributi dell'oggetto:

```
// Create a MyClass class
public class MyClass {
  int x;  // Create a class attribute

  // Create a class constructor for the MyClass class
  public MyClass() {
    x = 5;  // Set the initial value for the class attribute x
  }

  public static void main(String[] args) {
    MyClass myObj = new MyClass(); // Create an object of class MyClass (This will call the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}

// Outputs 5
```

Si noti che il nome del costruttore deve **corrispondere al nome della classe** e non può avere un **tipo restituito** (come void) e viene chiamato quando viene creato l'oggetto.

Tutte le classi hanno costruttori di default: se non crei tu stesso un costruttore di classi, Java ne crea uno per te. Tuttavia, non è possibile impostare i valori iniziali per gli attributi dell'oggetto.

Parametri del costruttore:

I costruttori possono anche prendere parametri, che viene utilizzato per inizializzare gli attributi.

```
public class MyClass {
  int x;

  public MyClass(int y) {
    x = y;
  }

  public static void main(String[] args) {
    MyClass myObj = new MyClass(5);
    System.out.println(myObj.x);
  }
}

// Outputs 5
```

Eccezioni

In JAVA la gestione degli errori può essere fatta usando il meccanismo delle ***eccezioni***, che sono oggetti che possono essere creati e lanciati (***throw***) in determinate condizioni, e che possono essere catturati (***catch***) dal codice scritto appositamente per la loro gestione:

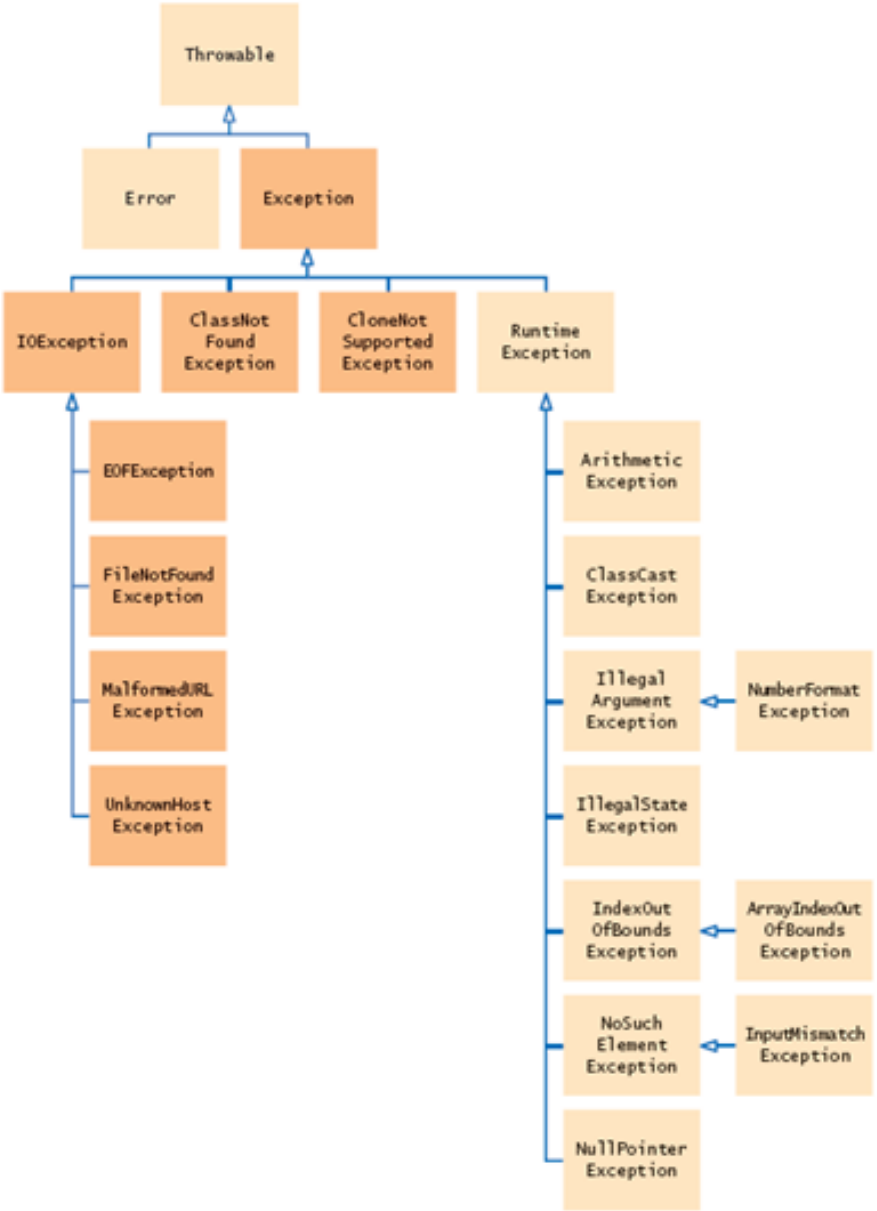
1. Le eccezioni non devono poter essere trascurate
2. Le eccezioni devono poter essere gestite da uno gestore competente, non semplicemente dal chiamante del metodo che fallisce

La superclasse ***Throwable*** ha due sottoclassi dirette, sempre in ***java.lang***

- **Error**
  Errori fatali, dovuti a condizioni accidentali
  Esaurimento delle risorse di sistema necessarie alla JVM (***OutOfMemoryError***), incompatibilità di versioni, violazione di un'asserzione (***AssertionError***), ……
  In genere i programmi non gestiscono questi errori

- **Exception**
  Tutti gli errori che non rientrano in ***Error***
  I programmi possono gestire o no questi errori a seconda dei casi

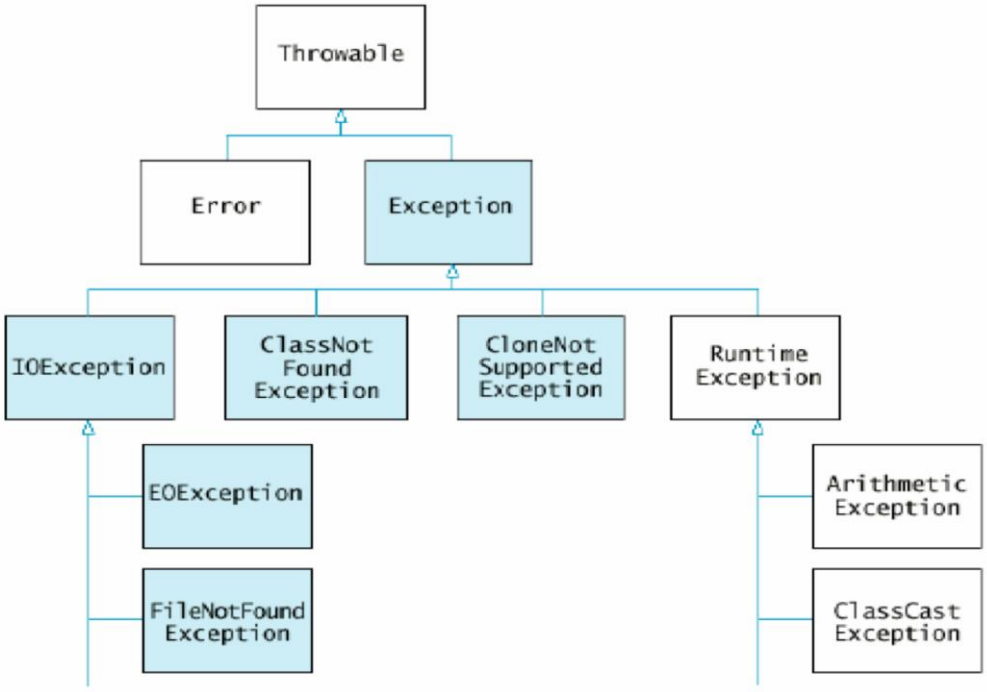
Tutte le classi che rappresentano eccezioni sono sottoclassi della classe **Exception**



Categorie di Eccezioni

- **eccezioni non controllate**: (Tutte le sottoclassi di **RunTimeException**)
  l dovute a circostanze che il programmatore **può evitare**, correggendo il programma.
  Esempio:
  NullPointerException: uso di un riferimento null.
  IndexOutofBoundException: accesso ad elementi esterni ai limiti di un array.
  Non è obbligatorio scrivere un codice per gestire questo tipo di eccezione.
  Il programmatore può prevenire queste anomalie, correggendo il codice.

- **eccezioni controllate**: (Tutte le sottoclassi di **IOException**)
  dovute a circostanze esterne che il programmatore **non può evitare,** il compilatore vuole sapere cosa fare nel caso si verifichi l'eccezione.
  Esempio:
  EOFException: terminazione inaspettata del flusso di dati in ingresso.
  Può essere provocata da eventi esterni: errore del disco, interruzione del collegamento di rete.
  Il gestore dell'eccezione si occupa del problema.



Per lanciare un`eccezione, usiamo la parola chiave **throw** (lancia), seguita da un oggetto di tipo eccezione:

throw exceptionObject;

Syntax `throw exceptionObject;`

Example

```
if (amount > balance)
{
   throw new IllegalArgumentException("Amount exceeds balance");
}
balance = balance - amount;
```

Most exception objects can be constructed with an error message.

A new exception object is constructed, then thrown.

This line is not executed when the exception is thrown.

Le eccezioni di runtime (**RuntimeException**) possono essere utilizzate per segnalare problemi dovuti ad input errati.

Syntax

```
try
{
   statement
   statement
   . . .
}
catch (ExceptionClass exceptionObject)
{
   statement
   statement
   . . .
}
```

Example

```
try
{
   Scanner in = new Scanner(new File("input.txt"));
   String input = in.next();
   process(input);
}
catch (IOException exception)
{
   System.out.println("Could not open input file");
}
```

This constructor can throw a FileNotFoundException.

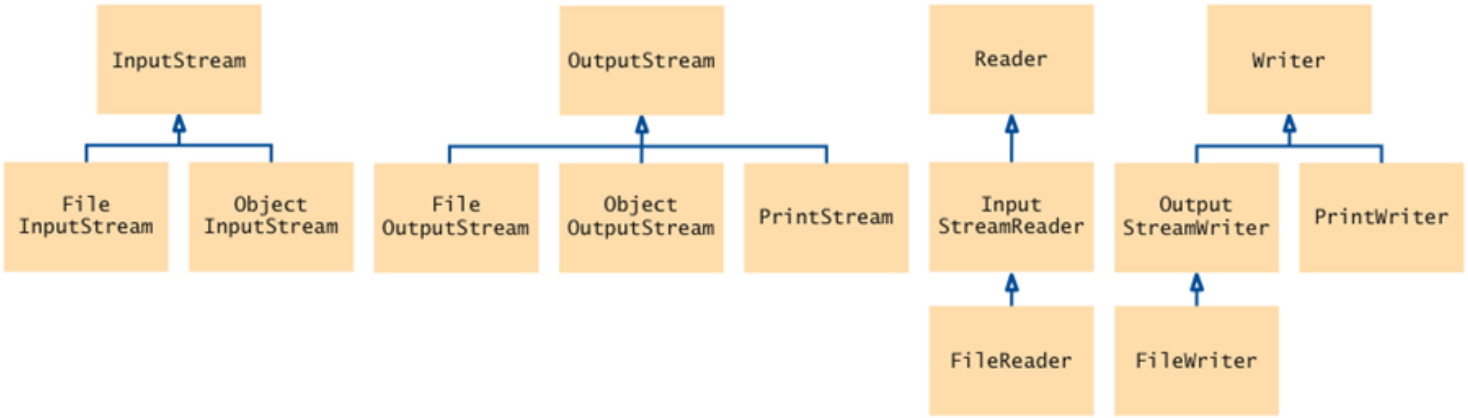
This is the exception that was thrown.

When an IOException is thrown, execution resumes here.

Additional catch clauses can appear here.

A FileNotFoundException is a special case of an IOException.

File e Flussi



In Java input e output sono definiti in termini di flussi (stream): Sequenze ordinate di dati.

- Per dati binari, usare la classe **InputStream, OutputStream.**
- Per caratteri, usare la classe **Reader, Writer.**

Costruttori che ricevono come parametro il nome di un file/directory o un oggetto File possono lanciare una ***FileNotFoundException*** (sottoclasse di ***IOException***).

Flussi di Oggetti:

Consentono di operare su interi oggetti:

- Per scrivere un oggetto non dobbiamo prima decomporlo
- Per leggere un oggetto non dobbiamo leggere i dati separatamente e poi ricomporre l'oggetto

Flussi in *scrittura*: Classe ***ObjectOutputStream***

Flussi in *lettura*: Classe ***ObjectInputStream***

L'oggetto da inserire nel flusso deve essere serializzabile altrimenti viene sollevata la **NotSerializableException**

- Appartenere a una classe che implementa l'interfaccia **Serializable**
- **Serializable** non ha metodi

```
MyClass mc = new MyClass(…);
ObjectOutputStream out =
     new ObjectOutputStream(new FileOutputStream("mc.dat"));
out.writeObject(mc); //MyClass implementa Serializable
```

Legge un ***Object*** da file e restituisce un riferimento a tale ***Object***

- L'output necessita di un cast
- Può lanciare un'eccezione controllata di tipo ***ClassNotFoundException***

```
ObjectInputStream in =
    new ObjectInputStream(new FileInputStream("mc.dat"));
MyClass mc = (MyClass) in.readObject();
```

Esempio:

```
import java.io.*;

public class ObjectInputStreamExample {

    public static class Person implements Serializable {
        public String name = null;
        public int age  =   0;
    }

    public static void main(String[] args) throws IOException, ClassNotFoundException {

        ObjectOutputStream objectOutputStream = new ObjectOutputStream(new FileOutputStream("person.txt"));

        Person person = new Person();
        person.name = "Jakob Jenkov";
        person.age  = 40;

        objectOutputStream.writeObject(person);
        objectOutputStream.close();

        ObjectInputStream objectInputStream = new ObjectInputStream(new FileInputStream("person.txt"));

        Person personRead = (Person) objectInputStream.readObject();

        objectInputStream.close();

        System.out.println(personRead.name);
        System.out.println(personRead.age);
    }
}
```