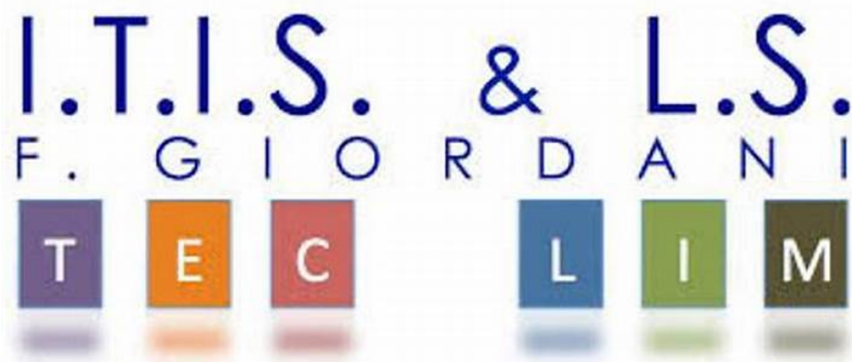
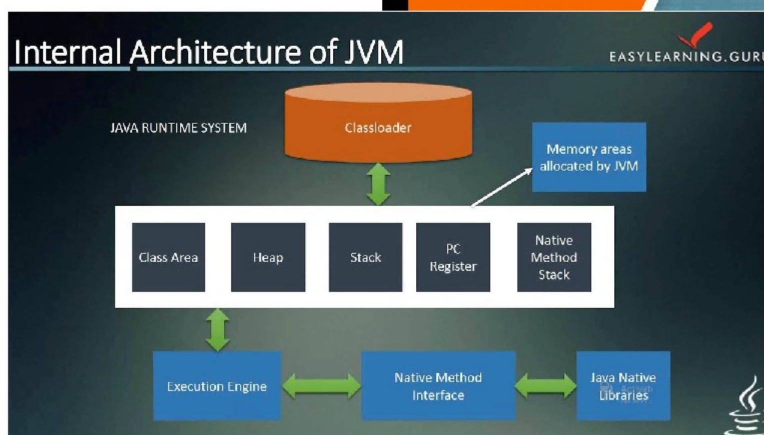


ITIS-LS "Francesco Giordani" Caserta

prof. Ennio Ranucci

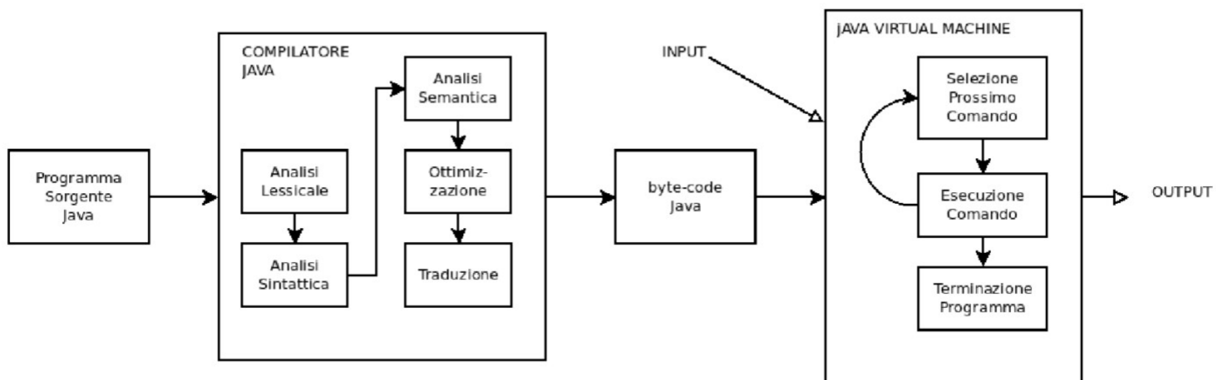
a.s. 2019-2020

Scrittura di applicazioni java "a console" e "a finestre"



Java `e un linguaggio di programmazione nato all’inizio degli anni novanta da un gruppo di lavoro della Sun Microsystems guidato da James Gosling.
Per migliorare la portabilità, il linguaggio Java si basa su un approccio che combina compilazione (in byte-code) e interpretazione (del byte-code).
Il byte-code può essere visto come l’assembly di una macchina virtuale. L’interprete del byte-code Java è detto Java Virtual Machine (JVM)

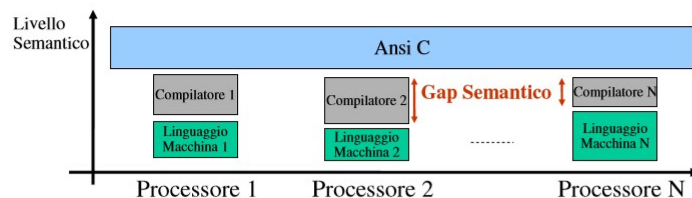
L’approccio compilazione+interpretazione schematicamente:



Java può essere usato per distribuire applicazioni su Internet.

Portabilità

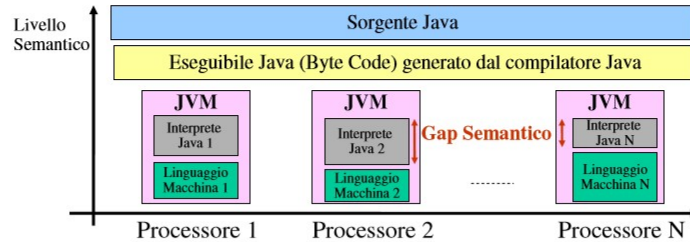
- Portabilità del codice sorgente
  - Un programma scritto in un linguaggio ad alto livello per una determinata macchina virtuale (processore, s.o.) può essere “riutilizzato” in una macchina virtuale differente?
  - Un esempio



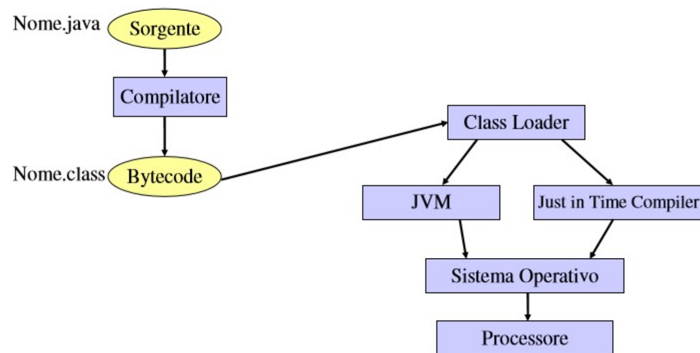
- Il codice sorgente scritto in Ansi C è portabile!
- …ma la portabilità del codice sorgente richiede la ricompilazione su ogni piattaforma
- Inoltre, cosa succede se un programma C deve invocare API del sistema operativo?
  - La portabilità è limitata a quelle macchine virtuali che supportano quelle specifiche API

Portabilità di Java

- Java garantisce la portabilità del codice eseguibile grazie alla Java Virtual Machine (JVM)
  - Write Once Run Everywhere!



Ciclo di Sviluppo



Il compilatore non produce codice macchina, ma un insieme ottimizzato di istruzioni detto **BYTECODE**.

Il sistema run-time di Java emula una macchina virtuale (***Java Virtual Machine***) che esegue il BYTECODE .

Ogni architettura per la quale la *Virtual Machine* sia implementata, può eseguire lo stesso programma Java.

L'efficienza di esecuzione di Java è superiore rispetto agli altri linguaggi interpretati (Tcl, Perl...), anche se non raggiunge quella dei linguaggi compilati.

Inoltre l'interprete Java fornisce compilatori *"just in time"* per trasformare a run-time il BYTECODE in codice macchina, guadagnando in velocità, ma perdendone la portabilità.

PARADIGMI DI PROGRAMMAZIONE

Nell'ambito della programmazione, con il termine paradigma si intende l'insieme di idee a cui ci si ispira per modellare e per risolvere i problemi .

I principali paradigmi di programmazione sono:

·Imperativo

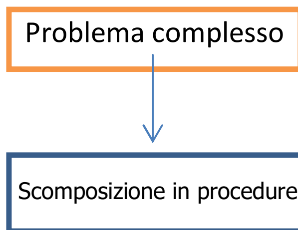
·Orientato agli oggetti

·Logico

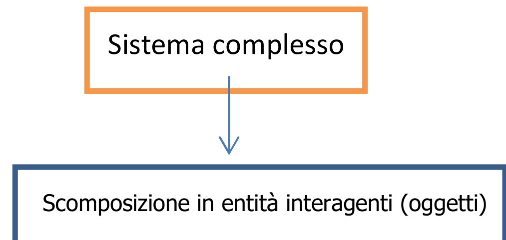
·Funzionale

OOP(Object Oriented Programming)

Programmazione strutturata



Programmazione ad oggetti



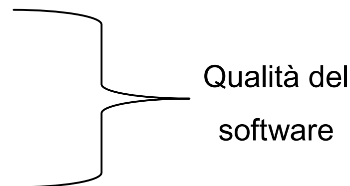
VANTAGGI DELLO STILE DI PROGRAMMAZIONE ORIENTATA AD OGGETTI

·Facilità di lettura e comprensione del codice

·Rapidità nella manutenzione

·Robustezza del programma

·Riusabilità



JAVA

Java è un linguaggio di alto livello orientato agli oggetti, creato dal *Sun Microsystem* nel 1995.

Nel 2010 la Sun è stata acquistata da *Oracle.*

Java nasce anche per poter essere eseguito senza modifiche su diversi sistemi operativi ed elaboratori con architetture hardware diverse.

La *portabilità* è la capacità di un programma di essere eseguito su piattaforme diverse senza dover essere modificato e ricompilato.

Vediamo come viene realizzata in pratica la portabilità in Java:

Quando si compila il codice sorgente in Java, il compilatore genera il codice compilato, chiamato *byte code*. Questo è un formato intermedio, indipendente dall'architettura ed è usato per trasportare il codice in modo efficiente tra le varie piattaforme hardware e software.
Questo codice non può essere eseguito da una macchina reale.
E' un codice astratto generato per una macchina astratta, detta *Java Virtual Machine* (JVM).
Per essere eseguito su una macchina reale, il codice compilato (byte code) deve essere interpretato.
Significa che ogni istruzione della macchina su cui si sta eseguendo l'applicazione in quel momento, per questi motivi si può dire che Java è un linguaggio interpretato, anche se la produzione del byte code è effettuata con un operazione di compilazione.
L'ambiente di programmazione Java include un insieme di librerie contenenti classi e metodi di varia utilità:

·Java.lang: Collezione delle classi di base.

·Java.io: Libreria per gli accessi ai file e ai flussi di input/output.

·Java.awt: Libreria contenente le classi per la gestione dei componenti grafici.

·Java.net: Librerie per le applicazioni che scambiano dati attraverso la rete.

·Java.util: Classi di utilità varie (array dinamici,gestione della data, …).

PRIMO PROGRAMMA IN JAVA

Scriviamo , utilizzando blocco note di Windows il seguente testo:

```
class ciaoMondo
{
  public static void main(String args[])
  {
    System.out.println("Ciao Mondo");
  }
}
```

Salviamolo con estensione .java: ciaoMondo.java (il nome della classe contenente il main deve essere uguale al nome del file.java)
Per compilare il programma digitiamo in ambiente prompt dei comandi: javac ciaoMondo.java
La fase di compilazione genera un file compilato che ha estensione .class e rappresenta il byte code.
Per eseguire il programma attiviamo l'interprete java usando il comando: java ciaoMondo

·Public indica che il metodo è pubblico ed è visibile.
·Void indica che non ci sono valori di ritorno.
·Static indica che il metodo è associato alla classe e non può essere richiamato dai singoli oggetti della classe.
·(Sono indicati i parametri). Il metodo main possiede come parametro un array di stringhe indicato con(args[ ]) che corrisponde ai parametri passati dalla riga di comando quando viene eseguita l'applicazione.
·L'oggetto System.out indica lo standard output, System.out.println( ) è un metodo che scrive sullo standard output i suoi argomenti e torna a capo.

ciao Mondo versione con parametri d'ingresso:

```
class ciaoMondo2
{
 public static void main(String[] args)
 {
   System.out.print ("Ciao mondo, sono il primo programma in Java ");
   System.out.println ("di "+args[0]+" "+args[1]);
 }
}
```

Eseguire con il comando: java ciaoMondo Paolo Rossi

Java è case-sensitive, ogni istruzione deve terminare con il simbolo ;

I COMMENTI

Le righe che iniziano con // sono di commento.
La seconda forma di commento è /*……*/ come commento che occupa più stringhe oppure /**…..*/

VARIABILI E COSTANTI IN JAVA

Esempio: int prezzo, double altezza=2,5 (l'assegnazione viene indicata con uguale), final double yard_metro=0,914 (le costanti sono precedute da final).

TIPO DI DATO

Tipo	Dimensione	Valori
Byte	8 bit	Da -128 a 127
Short	16 bit	Da -32768 a 32767
Int	32 bit	Da -2147483648 a 2147483647
Long	64 bit	Da -2^{63} a $2^{63}-1$

TIPI A VIRGOLA MOBILE

Tipo	Dimensione	Precisione
Float	32 bit	Singola
Double	64 bit	Doppia

Precisione singola: 7 cifre decimali.
Precisione doppia: 16 cifre decimali.

TIPO DI DATO CARATTERE

Il tipo di dato carattere è definito usando 16 bit e la codifica *UNICODE.*
Java non utilizza la codifica ASCII per rappresentare caratteri.
La codifica UNICODE definisce un insieme di caratteri usando 2 byte.
I primi 128 caratteri sono equivalenti ai caratteri del codice ASCII.
Alcuni caratteri vengono rappresentati attraverso sequenze di escape: \n, \t, \\

LE STRINGHE

Non sono un tipo predefinito ma sono definite in una certa classe (string).
Le sequenze di escape possono essere usate in una stringa.

BOOLEANO

Il tipo booleano viene indicato con **Boolean** e le variabili di questo tipo possono assumere i valori **true** o **false.**

IL CASTING PER LA CONVERSIONE DEL TIPO

Quasi tutti i controlli di correttezza di tipo vengono eseguiti durante la fase di compilazione quindi di Java è fortemente tipizzato. Se si assegna un valore intero ad una variabile di tipo float, Java effettua automaticamente la conversione perché il tipo float è più grande. Questa conversione chiamata anche promozione è permessa perché solitamente non si ha perdita di dati. Quando si tenta di effettuare assegnamenti tra variabili ed espressioni con tipi che vanno in conflitto viene generato un errore, per esempio la conversione da float ad int non viene eseguita automaticamente.
Se vogliamo eseguire forzatamente l'assegnamento il programmatore deve indicare esplicitamente la sua volontà. Il casting è il meccanismo che consente al programmatore l'obbligo della conversione. Il casting si effettua anteponendo al valore da convertire. Tra parentesi tonde, il tipo di dato in cui sarà convertito.
Esempio: int num=(int) float num;

GESTIONE DELL'INPUT/OUTPUT

System.out.printl("messaggio da visualizzare");
L'oggetto System.out indica lo standard output.
Println( ) è un metodo che stampa sul video un parametro e inserisce un ritorno a capo.
Print( ) è un metodo che stampa sul video un parametro e non inserisce un ritorno a capo.

```
class stampa
{
 public static void main(String[] args)
 {
  int numIntero=5;
  float numReale=25.68f;
  boolean trovato=false;
  System.out.println("numero intero= "+ numIntero);
  System.out.println("numero reale= "+ numReale);
  System.out.println("trovato= "+ trovato);
 }
}
```

Nella dichiarazione del numero reale è stata aggiunta la lettera f dopo il numero altrimenti sarebbe stato interpretato come un double, l'aggiunta della f equivale ad un casting.
In alternativa si poteva scrivere float numReale=(float) 25.86;
Per le operazioni di input esiste un oggetto denominato System.in.
System.in viene mascherato da oggetti che forniscono maggiori funzionalità: la classe BufferedReader.

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

Con queste dichiarazioni viene definito un oggetto tastiera di classe BufferedReader con l'operatore new che crea un nuovo oggetto.
La classe BufferedReader mette a disposizione il metodo readLine() che consente di leggere una riga per volta. Una riga viene considerata terminata quando viene premuto il tasto invio. Questo metodo restituisce solo stringhe,quindi se si vogliono acquisire valori numerici, si deve effettuare la conversione tra stringhe e numeri.

Esempio:
```
String nome;
nome= tastiera.readLine();
int num;
leggiNumero=tastiera.readLine();
num=integer.valueOf(leggiNumero).intValue();
```

Gestione delle eccezioni

Quando si verifica un eccezione (ossia una situazione anomala) bisogna prevedere un blocco di istruzioni per gestire questa situazione. In java si utilizza il costrutto try{} catch{}

```
import java.io.*;

class leggiNumero

{

  public static void main(String[] argv)

 {

    InputStreamReader input=new InputStreamReader(System.in);

    BufferedReader  tastiera=new BufferedReader(input);

    String str;

    int num;

    try

    {

    str=tastiera.readLine();

    num=Integer.valueOf(str).intValue();

    System.out.println("Questo e' il numero letto: " + num);

    }

    catch(Exception e)

    {

     System.out.println("\n Numero non corretto!");

     return;

    }

 }

}
```

Operatori di confronto: == ; != ; < ; <= ; > ; >=

Operatori booleani: && indica l'operazione AND

|| indica l'operazione OR

! indica l'operazione NOT

LE STRUTTURE DI CONTROLLO

Sequenza:

La struttura di sequenza, come in C++, viene realizzata posizionando le istruzioni una di seguito all'altra e separandole col ;

Selezione:

La struttura di selezione viene realizzata con il seguente costrutto:

If(condizione)

{

Istruzioni eseguite se la condizione è vera

}

Else

{

Istruzioni eseguite se la condizione è falsa

}

Selezione multipla

Switch(espressione)

{

Case 1: istruzioni break;

case 2: istruzioni break;

…

Default: istruzioni break;

}

Iterazione:

Ciclo indefinito: cicla per vero e la condizione è in testa.

while(condizione)

{

Istruzioni

}

Ciclo indefinito: cicla per vero e la condizione è in coda.

Do

{

Istruzioni

}

While(condizione)

Ciclo definito

For (inizializzazione; condizione ; aggiornamento)

{

Istruzioni

}

ARRAY

Array monodimensionali:
In Java è necessario dichiarare un array e allocarlo e nel seguente esempio le due fasi vengono scritte contemporaneamente sulla stessa riga:

int vet[]= new int[5]
String nome[]= new String[10]

Array multidimensionali:
int mat[][]= new int[3][4]

IL METODO MAIN CON PARAMETRI

```
class ifForParametri

{

 public static void main( String args[])

 {

  System.out.println("elenco parametri: ");

  if(args.length==0){

                System.out.println("Nessun parametro");

                return;

               }
```

```
  else{
       for(int i=0;i<args.length;i++)
        {
          System.out.println(args[i]);
        }
     }
 }
}
```

Esecuzione:

Javac parametri.java
Java parametri 45 –t rossi

Risultato a video:

Elenco parametri:
45
-t
rossi

LE ECCEZIONI

Un'eccezione è una struttura anomala che si verifica durante l'esecuzione del programma, una condizione anomala può essere una divisone per 0, oppure l'utilizzo di un indice in un array maggiore della sua dimensione fisica. Il programma usando particolari costrutti può tenere sotto controllo certe parti del codice e agire in modo opportuno se viene generata un'eccezione. La gestione delle eccezioni si realizza con il costrutto try…catch. Se vengono generalizzate delle eccezioni all'interno del blocco try il controllo dell'esecuzione passa al controllo catch. Dopo la parola catch si indica tra parentesi tonde l'eccezione che può verificarsi. Ci sono alcune eccezioni predefinite:

AritmeticException: segnala errori aritmetici;

IoException: generico errore di input-output.

```
class Eccezione
{
  public static void main(String[] args)
  {
     int a;
     int divisore=0;
```

```
    try
    {
      a=15/divisore ;
    }
    catch(ArithmeticException e)
     {
          System.out.println("divisione impossibile");
         return;
     }
  }
}
```

DICHIARAZIONE E UTILIZZO DI UNA CLASSE

```
Class NomeClasse
{
  //attributi
  //metodi
}
```

NomeClasse nomeOggetto //dichiarazione di un oggetto, i nomi degli oggetti con lettere minuscole, i nomi delle classi con lettera maiuscola

nomeOggetto=new NomeClasse(); //creazione dell’istanza nome oggetto

```
class Cerchio
{
  private double Raggio;  //attributo
  public void setRaggio (double RaggioPar)  //metodo
  {
    Raggio=RaggioPar;
  }
   public double Area()  //metodo
   {
```

```
    return(Raggio*Raggio*Math.PI);
  }
}
class ProgCerchio
{
  public static void main(String args[])
  {
    Cerchio cerchioObj; //dichiarazione dell'oggetto
    cerchioObj= new Cerchio();  //creazione dell'istanza
    cerchioObj.setRaggio(0.75);
    System.out.println ("area del cerchio "+cerchioObj.Area());
  }
}
```

Un programma si compone generalmente di diverse classi. Tali classi si possono trovare fisicamente in file diversi. Viceversa un file può contenere più classi. Se un file contiene più classi, solo quella con lo stesso nome del file sarà public e visibile all'esterno del file stesso. La classe principale deve avere una visibilità public e trovarsi in un file con il suo stesso nome. E' possibile salvare ogni classe in un file distinto, purché memorizzato nella stessa cartella della classe principale. Dopo aver compilato separatamente i file sorgenti l'esecuzione del programma inizierà dalla classe principale.

GLI ATTRIBUTI

Gli attributi vengono chiamati anche variabili istanza e nella dichiarazione sono sempre preceduti dall'indicatore di visibilità:

- Public: L'attributo è accessibile a qualsiasi altra classe;
- Private: l'attributo non è accessibile alle altre classi, è nascosto all'interno di un oggetto(Information hiding)
- Protected: l'attributo è accessibile solo alle sottoclassi e alle classi appartenenti allo stesso package di quello in cui è dichiarato
- Package: se non si inserisce nessuna delle parole chiavi precedenti vuol dire che l'attributo è visibile solo dalle classe che appartengono allo stesso package.
- Static: indica un attributo legato alla classe nel senso che se vengono creati più oggetti di quella classe esiste solo una copia dell'attributo

- Final: invece se si vuol renderlo una costante. Esempio: public final int 10=10.

I METODI

Un metodo è caratterizzato da:

un nome;

un elenco di parametri;

un tipo di valore di ritorno;

un blocco di istruzioni;

un ambiente locale.

I livelli di visibilità di un metodo sono gli stessi visti per gli attributi.

COSTRUTTORI

Con l'allocazione dell'oggetto viene attivato in modo implicito un particolare metodo predefinito detto costruttore della classe. I costruttori vengono eseguiti automaticamente ogni volta che viene creato un nuovo oggetto e sono solitamente usati per l'inizializzazione dell'oggetto.

aggiungiamo alla classe Cerchio un costruttore

```
class Cerchio

{

  public double raggio;

  public Cerchio (double raggio)

  {

      this.raggio=raggio;

  }

  public void setRaggio(double raggio)

   {

      this.raggio=raggio;

   }

  public double area()

    {

      return(raggio * raggio * Math.PI);
```

```
    }


}
class ProgCerchio2
{
  public static void main(String args[])
  {
    Cerchio cerchioObj; //dichiarazione dell'oggetto
    cerchioObj= new Cerchio(0);  //creazione dell'istanza
    cerchioObj.setRaggio(0.75);
    System.out.println ("area del cerchio "+cerchioObj.area());
  }
}
```

Quindi l'allocazione sarà: cerchioObj=new Cerchio(0.41)

Una classe può avere più metodi costruttori purché abbiano diverso numero e tipo di parametri. Chiamiamo questa situazione di moltiplicità overloading dei costruttori.

L'invocazione di un metodo avviene nel seguente modo:

nomeOggetto.metodo(parametri)

Nell'oop si usa il termine scambio di messaggi per indicare l'interazione tra oggetti, realizzata tramite l'invocazione dei metodi. Quando un oggetto vuole comunicare con un altro per eseguire un azione manda ad esso un messaggio. Il nome oggetto che precede il punto corrispondente al destinatario del messaggio. In ogni classe è implicitamente presente un oggetto speciale identificato dalla parola chiave THIS. Questa parola chiave costituisce un rifornimento all'oggetto medesimo.

Esempio:

```
 public cerchio(double Raggio)
{ this.Raggio=Raggio; }
```

L'uso dello stesso nome consente di non inventare nomi diversi di variabile rendendo più semplice la lettura del codice. Il parametro raggio all'interno del costruttore diventa una variabile locale che maschera l'attributo dichiarato con lo stesso nome.

In java, i parametri possono essere passati ai metodi solo per valore.

I tipi di riferimento, vengono passati per indirizzo. L'array quindi non viene duplicato e viene creata una variabile locale che contiene la copia dell'indirizzo allo stesso array. Ne segue che le modifiche all'interno del metodo, fatte usando il parametro, modificano anche l'array originale.

La dichiarazione dell'oggetto crea una variabile vuota che contiene la parola chiave **NULL.** L'allocazione dell'oggetto inserisce nella variabile il riferimento all'area di memoria contenente i valori degli attributi dell'oggetto.

L'interfaccia di una classe indica l'elenco dei metodi pubblici, cioè l'insieme delle funzionalità utilizzabili dalle istanze della classe.

Array di oggetti

Esempio: Cerchio vetCerchio[]=new Cerchio[10] la dichiarazione e l'allocazione di un array di oggetti è la stessa degli array basati sui tipi predefiniti.

L'EREDITARIETA'

```
class Cerchio
{
  private double raggio;
  public Cerchio(double raggio)
     {
        this.raggio=raggio;
     }
  public void setRaggio(double raggio)
    {
      this.raggio=raggio;
    }
 public double area()
   {
     return (raggio*raggio* Math.PI);
   }
public double circonferenza()
  {
    return(raggio*2*Math.PI);
```

```
}
}
class Cilindro extends Cerchio
{
   private double altezza;
   public Cilindro(double raggio, double altezza)
      {
         super(raggio);
         this.altezza=altezza;
      }
  public void setAltezza (double altezza)
     {
      this.altezza=altezza;
     }
public double volume()
   {
     return area()*altezza;
  }
}
class ProgCilindro
{
  public static void main(String args[])
     {
      Cilindro cilindroObj=new Cilindro(4.0,10.0);
      System.out.println("area di base= " + cilindroObj.area());
      System.out.println("volume = " + cilindroObj.volume());
     }
}
```

POLIMORFISMO

Il polimorfismo indica la possibilità per i metodi di assumere forme, cioè implementazioni, diverse all'interno della gerarchia delle classi.

Nei linguaggi ad oggetti sono presenti due tipi di polimorfismo:

- Overriding, o sovrapposizione dei metodi;
- Overloading, o sovraccarico dei metodi.

L'**overriding** di un metodo consiste nel ridefinire, nella classe derivata, un metodo ereditato, con lo scopo di modificarne il comportamento. Il nuovo metodo deve avere lo stesso nome e gli stessi parametri del metodo sovrascritto.

Esempio7:

la classe cilindro eredita dalla classe cerchio il metodo area che calcola l'area del cerchio alla base del cilindro. Questo metodo area può essere sovrascritto per calcolare l'area della superficie totale del cilindro.

Il codice del metodo sovrascritto è il seguente:

```
public double area()
  {
    double areaBase, areaLaterale;
    areaBase=super.area()*2;
    areaLaterale=circonferenza()*altezza;
    return areaBase+areaLaterale;
  }
```

```
class cerchio

{

   private double Raggio;

  public cerchio(double Raggio)

     {

         this.Raggio=Raggio;

     }

  public void setRaggio (double Raggio)

    {

       this.Raggio=Raggio;

    }
```

```
  public double area()
     {
         return(Raggio*Raggio)*Math.PI;
     }
 public double circonferenza()
     {
        return(2*Raggio*Math.PI);
     }
}
class cilindro extends cerchio
{
  private double altezza;
  public cilindro (double Raggio, double altezza)
  { super(Raggio); this.altezza=altezza; }

      public void setaltezza (double altezza)
        { this.altezza=altezza; }

      public double volume()
        { return(super.area()*altezza); }
     public double area()
        {
            double AreaBase, AreaLaterale;
            AreaBase=super.area()*2;
            AreaLaterale=circonferenza()*altezza;
            return(AreaBase+AreaLaterale);
        }
  }
```

```
class main

{

    public static void main(String args[])

     {

        cilindro cilindroObj=new cilindro(4,10);

        System.out.println("L'area del cilindro e': " +cilindroObj.area());

        System.out.println("Il volume e': " +cilindroObj.volume());

        cerchio cerchioObj=new cerchio(2);

        System.out.println("L'area del cerchio e': "+cerchioObj.area());

     } }
```

Ancora un esempio polimorfismo overriding

```
public abstract class formaGeometrica
{

  public abstract void Nascondi();

  public abstract void Mostra();

  public abstract void MuoviA(int x, int y);

}
public class punto extends formaGeometrica
{

  protected int x,y;
  public punto(int x, int y)
   {
         this.x = x;
         this.y  = y;
   }

  public void Mostra()
   {
      System.out.println("Punto visibile in "+x+" "+y);
   }

  public void Nascondi()
   {
      System.out.println("Punto nascosto in "+x+" "+y);
   }
```

```
    public void MuoviA(int NuovoX, int NuovoY)
    {
       Nascondi();
       this.x= NuovoX;this.y= NuovoY;
       Mostra();
   }

}
public class cerchio extends punto
{
  private int raggio;
  public cerchio(int x, int y, int raggio)
    {
      super(x,y);
      this.raggio = raggio;
    }
public void Mostra()
    {
       System.out.println("Cerchio visibile in "+x+" "+y);
    }

  public void Nascondi()
    {
       System.out.println("Cerchio nascosto in "+x+" "+y);
    }
 /*
    public void MuoviA(int NuovoX, int NuovoY)
    {
       Nascondi();
       this.x= NuovoX;this.y= NuovoY;
       Mostra();
   }
*/
}
public class main
{
    public static void main(String[] args)
    {
      punto puntoObj = new punto(2,4);
      cerchio cerchioObj = new cerchio(10,20,100);
      formaGeometrica figura;
      figura=puntoObj;
      //figura=cerchioObj;
      figura.Mostra();
      System.out.println();
      figura.MuoviA(20,30);
      System.out.println();
    }
}
```

OVERLOADING

L'overloading di un metodo è la possibilità di utilizzare lo stesso nome per compiere operazioni diverse. Solitamente si applica ai metodi della stessa classe che si presentano con lo stesso nome ma con un numero o tipo diverso di parametri.
Un esempio di overloading è il metodo **println**.
Esistono diversi metodi println con lo stesso nome che vengono richiamate allo stesso metodo ma con parmetri diversi.

```
System.out.println("Stringa");
System.out.println(50);
System.out.println(7,12);
```

L'overloading è usato anche per offrire più costruttori ad una classe.
Esempio7:

```
class numero
{
  private int num;
  public numero()
    {
       num=0;
    }
  public numero(int num)
    {
      this.num=num;
    }
  int getNum()
    {
       return num;
    }
  int calcola(int num1, int num2)
    {
       return num1*num2;
    }
  double calcola(double num1, double num2)
    {
       return num1+num2;
    }
}
class main
{
  public static void main(String args[])
     {
       numero numeroObj=new numero(3);
       System.out.println("valore iniziale di num: " + numeroObj.getNum());
       System.out.println("valore calcolato: " + numeroObj.calcola(2,3));
     }
}
```

La classe astratta e il polimorfismo

```
/*
 Questa classe utilizza le classi Quadrato, Rettangolo, Cerchio,
 sottoclassi della classe astratta FormaGeometrica. Esempio di   polimorfismo.
*/

/*
  Questa classe abstract e' costruita in modo che ogni sua sottoclasse
  DEBBA definire un metodo per il calcolo dell'area, un metodo per
  il calcolo del perimetro e un metodo per il disegno della figura.
*/

public abstract class formaGeometrica {

   public abstract String nome();

   public abstract double perimetro();

   public abstract double area();

   public abstract void disegna();

}
```

```
/*
 La classe definisce un rettangolo di base e altezza assegnati.
*/

public class rettangolo extends formaGeometrica{
   private double base, altezza;

   // costruisco un rettangolo assegnando un valore ala sua base e uno
   // alla sua altezza
   public rettangolo(double base, double altezza) {
          this.base = base;
          this.altezza = altezza;
   }

   // calcola il perimetro del rettangolo
   public double perimetro() {
          return 2 * (base + altezza);
   }

   // calcola l'area del rettangolo
   public double area() {
          return base * altezza;
   }

   // disegna, schematicamente, un rettangolo.
   public void disegna() {
          System.out.println("********\n*      *\n********");
   }

   // restituisce il nome della forma geometrica:
   public String nome() {
          return "rettangolo";
   }
}
/*
 La classe definisce un quadrato di lato assegnato.
*/

public class quadrato extends formaGeometrica{
   private double lato;

   // costruisco un quadrato assegnando un valore al suo lato.
   public quadrato(double lato) {
          this.lato = lato;
   }
   // calcola il perimetro del quadrato
   public double perimetro() {
          return 4 * lato;
   }
```

```
    // calcola l'area del quadrato
    public double area() {
            return lato * lato;
    }
    // disegna, schematicamente, un quadrato.
    public void disegna() {
            System.out.println("****\n*  *\n****");
    }

    // restituisce il nome della forma geometrica:
    public String nome() {
            return "quadrato";
    }
}
/*
  La classe definisce un cerchio di raggio assegnato.
*/
public class cerchio extends formaGeometrica{
    private double raggio;

    // costruisco un cerchio assegnando un valore al suo raggio
    public cerchio(double raggio) {
            this.raggio = raggio;
    }

    // calcola il perimetro del cerchio (circonferenza)
    public double perimetro() {
            return 2 * Math.PI * raggio;
    }

    // calcola l'area del cerchio
    public double area() {
            return Math.PI * raggio * raggio;
    }

    // disegna, schematicamente, un cerchio
    public void disegna() {
            System.out.println(" *  *\n*    *\n *  *");
    }

    // restituisce il nome della forma geometrica:
    public String nome() {
            return "cerchio";
    }
}


/*
```

```
  Questa classe utilizza le classi Quadrato, Rettangolo, Cerchio,
  sottoclassi della classe astratta FormaGeometrica. Esempio di   polimorfismo.
*/

public class main {

   public static void main(String[] args) {

          // disegno una figura geometrica

          cerchio cerchio1 = new cerchio(4);
          quadrato quadrato1 = new quadrato(5);
          rettangolo rettangolo1 = new rettangolo(2,5);

          formaGeometrica figura;

          // l'oggetto figura puo' riferirsi sia a un oggetto di tipo
          // Quadrato, sia ad un oggetto di tipo Rettangolo, sia ad
          // un oggetto di tipo Cerchio.

          //figura = cerchio1;
           figura = rettangolo1;
          //figura = quadrato1;

          System.out.println();
          System.out.println("La figura e' un: " + figura.nome());
          System.out.println("Il suo perimetro vale: " + figura.perimetro());
          System.out.println("La sua area vale: " + figura.area());
          System.out.println("Questo e' il suo disegno schematico: ");
          System.out.println();
          figura.disegna();
          System.out.println();
   }
}
```

Le **classi astratte e le interfacce** sono due concetti fondamentali della programmazione orientata agli oggetti (OOP, Object Oriented Programming)

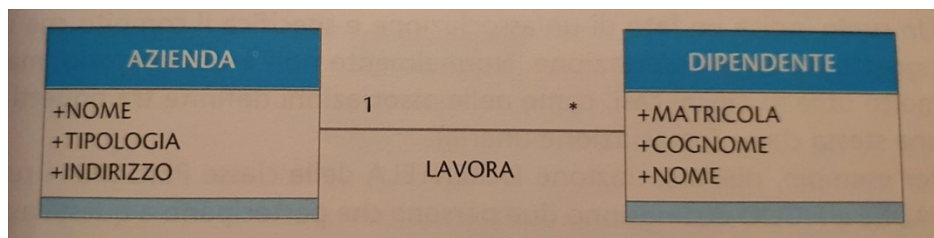
1. Una classe astratta è una classe che non può essere instanziata e che serve solo per essere derivata, definendo al suo interno metodi e proprietà tutti o in parte anch'essi astratti. Una classe astratta che implementa solo ed esclusivamente metodi e proprietà astratte viene detta classe astratta pura. Le classi astratte permettono d'implementare il concetto di polimorfismo.
   Una classe astratta può contenere metodi completi o incompleti.
2. Le interfacce sono molto simili alle classi astratte, in quanto anch'essa definisce metodi e proprietà astratte. Nelle interfacce non troveremo l'implementazione di alcun metodo o proprietà, come per le classi astratte pure, si dice che le interfacce stipulino un contratto con la classe derivata che le implementa.
   Le interfacce possono contenere solo la firma di un metodo ma nessun corpo.

a) La classe astratta ha il costruttore, ma l'interfaccia no.
b) Le classi astratte possono avere implementazioni per alcuni dei suoi membri (metodi), ma l'interfaccia non può avere l'implementazione per nessuno dei suoi membri.
c) Le classi astratte potrebbero avere sottoclassi.
d) Le interfacce devono avere sottoclassi.
e) Solo un'interfaccia può estendere un'altra interfaccia, qualsiasi classe può estendere una classe astratta.
f) Tutte le variabili nelle interfacce sono definitive per impostazione predefinita
g) Una classe può implementare diverse interfacce. Una classe può estendere solo una classe astratta.
h) Il modificatore di accessibilità (pubblico / privato / interno) è consentito per la classe astratta. L'interfaccia non consente il modificatore di accessibilità

```
public interface figuraPiana
{
 public double Perimetro();
 public double Area();
}
public class rettangolo implements figura Piana
{
  private double base; private double altezza;
  public rettangolo(double b, double a){ base=b; altezza=a;}
  public double Perimetro(){ return (base * 2)+(altezza * 2);}
  public double Area(){ return base * altezza;}
 }
public class quadrato implements figura Piana
{
  private double lato;
  public quadrato(double l){ lato=l;   }
  public double Perimetro(){ return (lato+lato)*2;}
  public double Area(){ return lato*lato;}
}
public class main
{
 public static void main(String[] args)
  {
    rettangolo rettangoloObj = new rettangolo(2,4);
    quadrato quadratoObj = new quadrato(10);
    figuraPiana figura;
    //figura=rettangoloObj;
    figura=quadratoObj;
    System.out.println("Area:  " + figura.Area());
    System.out.println("Perimetro:  " + figura.Perimetro());
 }
}
```

Associazione

L'Associazione è una relazione tra classi: una classe A è associata ad una classe B se un oggetto della classe A è in grado di inviare dei messaggi ad un oggetto di classe B oppure se un oggetto di classe A può creare, ricevere o restituire oggetti di classe B.
Una classe è associata ad un'altra se è possibile "navigare" da oggetti della prima classe ad oggetti della seconda classe seguendo semplicemente un riferimento ad un oggetto.
Ad esempio, dato un oggetto di tipo Dipendente, è possibile giungere ad oggetti di tipo Azienda accedendo semplicemente alla variabile istanza azienda definita all'interno della classe Dipendente.

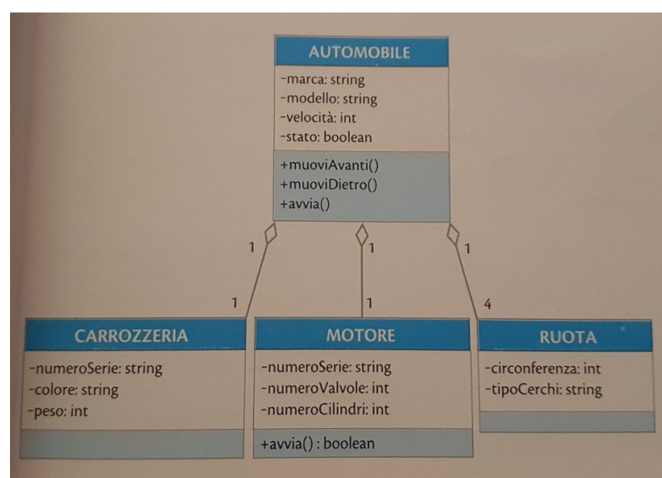


Aggregazione

La relazione di tipo Aggregazione si basa, invece, sul seguente concetto: Un oggetto di classe A contiene un oggetto di classe B se B è una proprietà (attributo) di A.
In sostanza, l'aggregazione è una forma di associazione più forte: una classe ne aggrega un'altra se esiste tra le due classi una relazione di tipo "intero-parte".
Ad esempio la classe Azienda aggrega la classe Dipendente perché una ditta (che costituisce l'"intero") è composta da dipendenti (che costituiscono la "parte").

Composizione

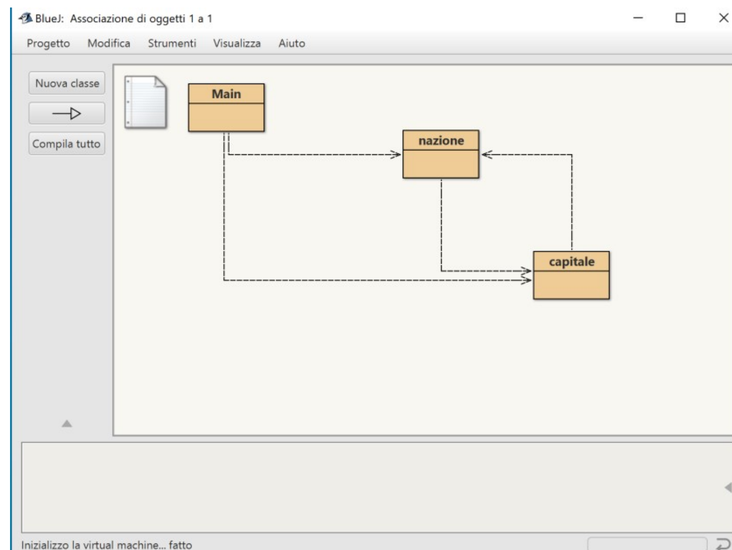
La Composizione è una forma di aggregazione ancora più forte che indica che una "parte" può appartenere ad un "intero" in un certo istante di tempo.
Ad esempio, una ruota può far parte di una sola automobile in un certo istante, mentre, al contrario, una persona potrebbe lavorare contemporaneamente per due ditte.



Specializzazione

La relazione di tipo Specializzazione si basa sul concetto di ereditarietà. Un oggetto di classe A deriva da un oggetto di classe B se A è in grado di compiere tutte le azioni che l'oggetto B è in grado di compiere. Inoltre l'oggetto di classe A è in grado di eseguire anche azioni che l'oggetto B non può compiere.

ESEMPIO ASSOCIAZIONE 1:1



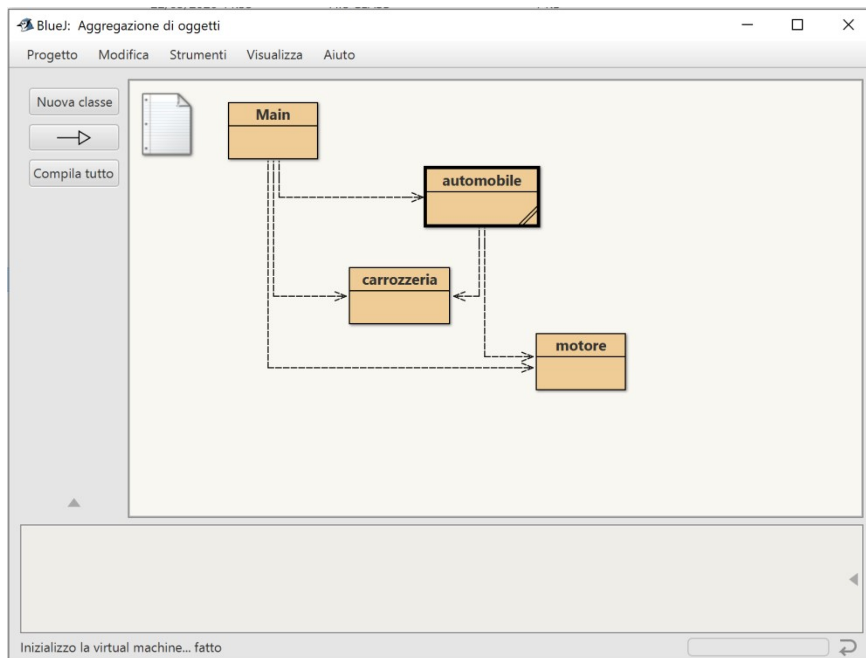

```
/**

* La classe nazione incorpora la classe capitale

*/

public class nazione

{

   private String nomeNazione;

   private capitale  capitaleAssociata;

   public nazione(String nomeNazione)

   {

     this.nomeNazione=nomeNazione;

   }

   public String getNomeNazione()

   {
```

```
      return nomeNazione;
    }
    public void setNomeNazione(String nomeNazione)
    {
       this.nomeNazione=nomeNazione;
    }
    public void setCapitale(capitale capitaleAssociata)
    {
      this.capitaleAssociata=capitaleAssociata;
    }
    public capitale getCapitale()
    {
      return capitaleAssociata;
    }
}
public class capitale
{
    private String nomeCapitale;
    private nazione nazioneAssociata;
    public capitale(String nomeCapitale)
    {
       this.nomeCapitale=nomeCapitale;
    }
    public String getNomeCapitale()
    {
      return nomeCapitale;
```

```
    }
    public void setNomeCapitale(String nomeCapitale)
    {
      this.nomeCapitale=nomeCapitale;
    }
    public void setNazione(nazione nazioneAssociata)
    {
      this.nazioneAssociata=nazioneAssociata;
    }
    public nazione getNazione()
    {
     return nazioneAssociata;
    }
}
class Main
{
  public static void main(String args[])
  {
    String s="";
    String s1="";
    nazione nazioneObj;
    nazioneObj= new nazione("Italia");
    System.out.println ("nome della Nazione: "+nazioneObj.getNomeNazione());
    capitale capitaleObj;
    capitaleObj= new capitale("Roma");
    System.out.println ("nome della Capitale: "+capitaleObj.getNomeCapitale());
```

```
    //creaiamo l'associazione 1 a 1
    nazioneObj.setCapitale(capitaleObj);
    capitaleObj.setNazione(nazioneObj);
    s=s+"Nazione: "+nazioneObj.getNomeNazione()+" ";
    s=s+"Capitale: "+nazioneObj.getCapitale().getNomeCapitale();
    System.out.println ("dati "+s);
    nazione nazioneObj2;
    nazioneObj2= new nazione("Spagna");
    capitale capitaleObj2;
    capitaleObj2= new capitale("Madrid");
    nazioneObj2.setCapitale(capitaleObj2);
    capitaleObj2.setNazione(nazioneObj2);
    s1=s1+"Nazione: "+nazioneObj2.getNomeNazione()+" ";
    s1=s1+"Capitale: "+nazioneObj2.getCapitale().getNomeCapitale();
    System.out.println ("dati "+s1);
   }
}
```

ESEMPIO AGGREGAZIONE




```
/**

* La classe carrozzeria è incorporata nella classe automobile (aggregazione)

*/

public class carrozzeria
{
   private String colore;
   public carrozzeria(String colore)
   {
     this.colore=colore;
   }
   public String getColore()
   {
     return colore;
   }
   public String descrCarrozzeria()
   {
      String s="";
      s=colore;
      return s;
   }

}
```

```
/**
* La classe motore è incorporata nella classe automobile (aggregazione)
*/
public class motore
{
   private int numCilindri;
   public motore(int numCilindri)
   {
     this.numCilindri=numCilindri;
   }
   public int getNumCilindri()
   {
     return numCilindri;
   }
   public String descrMotore()
   {
      String s="";
      s=" "+numCilindri;
      return s;
   }
 }
```

```
/**
* La classe automobile incorpora le classi carrozzeria e motore (aggregazione)
*/
public class automobile
{
   private String marca;
   private carrozzeria  carrozzeriaAggregata;
   private motore  motoreAggregato;
   public automobile(String marca, carrozzeria carrozzeriaAggregata, motore motoreAggregato)
   {
     this.marca=marca;
     this.carrozzeriaAggregata=carrozzeriaAggregata;
     this.motoreAggregato=motoreAggregato;
   }
   public String getMarca()
   {
     return marca;
   }
   public String descrAuto()
   {
      String s="";
      s=marca+" "+carrozzeriaAggregata.descrCarrozzeria()+" "+motoreAggregato.descrMotore();
      return s;
   }
 }
```

```
class Main
{
 public static void main(String args[])
 {
   motore motoreObj;
   motoreObj= new motore(4);
   carrozzeria carrozzeriaObj;
   carrozzeriaObj= new carrozzeria("Rossa");
   automobile automobileObj;
   automobileObj=new automobile("Fiat",carrozzeriaObj,motoreObj);
   System.out.println ("descrizione automobile "+automobileObj.descrAuto());
  }
}
```

LE LIBRERIE

In java le librerie sono costituite da un insieme di classi già compilate. Per riferirsi alle librerie java usa il termine package. I package sono raggruppamenti che contengono più classi correlate. Alcuni esempi di package in JDK sono:
Java.applet, raggruppa le classi per gestire le applet;
Java.awt, raggruppa le classi usate per creare interfacce grafiche (pulsanti, menù, finestre);
java.io, raggruppa le classe per la gestione dei file e dell' I/O;
Java.lang, raggruppa le classi di base per il linguaggio tra cui math,string;
java.net, raggruppa le classi per le applicazioni di rete;
java.util, raggruppa classi di varia utilità per la gestione della data,calendario,vettori.

LE STRINGHE

Le stringhe non sono un tipo di dato predefinito in java. Sono definite usando una classe chiamata String che è inclusa nel package **java.lang**. Per usare le stringhe in java si deve usare un oggetto di classe String.
String nome=new String("Elena");
Oppure String nome="Elena";

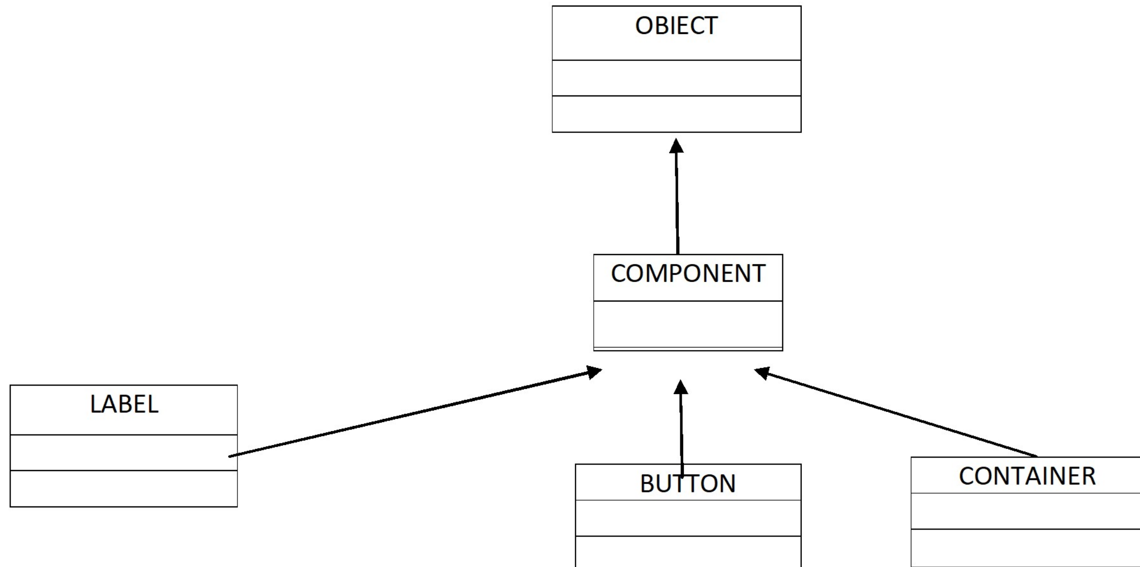
L'operatore di concatenazione tra stringhe è +.
I metodi principali della classe String sono:

- **Lenght**, restituisce un numero corrispondente alla lunghezza della stringa.
- **charAT**(int), restituisce il carattere che si trova nella posizione indicata dal parametro.
- **toLowerCase, to LapperCase**, convertono tutti i caratteri.
- **Equals**, controlla se la stringa che riceve come parametro è uguale a quella a cui viene applicato, oppure si può usare invece di equals ==.

PROGRAMMAZIONE GUIDATA DAGLI EVENTI

I programmi eseguiti da un elaboratore possono essere divisi in due categorie: programmi che eseguono loro compiti senza richiedere intervento di utenti, e altri che hanno bisogno di interagire con gli utilizzatori. Per esempio un programma che si preoccupa di tenere aggiornato un orologio non ha bisogno di input. Al contrario un programma che esegue calcoli su certi valori inseriti da un utente ha bisogno di interagire con lo stesso. Le interfacce possono essere a carattere o grafiche. Le interfacce a caratteri possono visualizzare solo caratteri e l'interazione avviene inserendo i comandi da tastiera. Le interfacce grafiche sono composte da immagini, icone, finestre, bottoni. Le interfacce grafiche si indicano con il termine **GUI** (grapichal user interface). La OOP per gestire le interfacce grafiche usa la programmazione guidata dagli eventi, cioè ad ogni interazione dell'utente viene associato un evento che deve essere gestito richiamando un azione appropiata senza rispettare tempi prefissati. La generazione di un evento è causata da un azione effettuata dall'utente. Una volta generato un evento si deve provvedere alla sua gestione che viene demandata ad un gestore di eventi. Un gestore, detto anche **handler**, è la parte dell'applicazione software che si preoccupa di dare una risposta in base al tipo di evento ricevuto.
La parte grafica di java è contenuta nel package **java.awt** e **java.swing.**
l'awt (Abstract window toolkit) è il sistema utilizzato da java per definire un insieme di elementi grafici indipendenti della piattaforma su cui verranno visualizzati.
Swing rappresenta l'evoluzione di awt in quanto aggiunge nuovi componenti metodi e funzionalità. Una componente è un oggetto con una rappresentazione grafica e la sua caratteristica principale è quella di offrire un interazione con l'utente. Esempi di componenti sono:

i pulsanti, i menù, le caselle di testo, le barre di scorrimento e cosi via. Un contenitore è un oggetto che può contenere le componenti, il compito del contenitore è quello di posizionare e di mensionare le componenti all'interno del contenitore stesso. Un contenitore comunemente usato è la finestra. Le classi di java che realizzano le componenti e i contenitori sono organizzate in una gerarchia delle componenti che ha come padre la classe component.



I pannelli hanno lo scopo di organizzare i componenti nella finestra.

CREARE UNA FINESTRE GRAFICA COMPOSTA DA UN' ETICHETTA E UN PULSANTE

Programma con libreria awt:

```
import java.awt.*;
class FinestraAwt
{
 public static void main (String args[])
 {
 Frame frmF=new Frame();
Panel pnlP=new Panel();
Label lblL=new Label("Etichetta");
Button btnB=new Button("Bottone");
pnlP.add(lblL);
pnlP.add(btnB);
frmF.add(pnlP);
frmF.setSize(300,200);
frmF.setVisible(true);
}
}
```

Programma con libreria swing:

```
import javax.swing.*;
```

```
import java.awt.*;
class FinestraSwing
{
 public static void main (String args[])
 {
 JFrame frmF=new JFrame();
JPanel pnlP=new JPanel();
JLabel lblL=new JLabel("Etichetta");
JButton btnB=new JButton("Bottone");
pnlP.add(lblL);
pnlP.add(btnB);
Container cntC=frmF.getContentPane();
cntC.add(pnlP);
frmF.setSize(300,200);
frmF.setVisible(true);
}
}
```

In Swing, l'aggiunta al contenitore principale viene eseguito tramite un ulteriore contenitore intermedio (Container) che rappresenta il *contentpane,* pannello del contenuto della finestra *JFrame.* Il metodo **getContentPane** rende disponibile questo contenitore. Per chiudere la finestra, occorre gestire l'evento di chiusura associando ad esso un'istruzione che causa la fine del programma. Non avendo previsto nel programma la gestione di questa operazione, per chiudere la finestra bisogna premere la combinazione Ctrl+C dal *Prompt dei comandi*.

ETICHETTE

L'etichetta è rappresentata dalla classe **JLabel** e può contenere solo una riga di testo. L'etichetta può essere modificata solo da programma ma non dall'utente.

Un'etichetta può essere definita attraverso tre diversi costruttori:

- **JLabel()**
  costruttore senza parametri che crea un'etichetta vuota.
  Esempio:

```
import java.awt.*;
import javax.swing.*;
class JLabel1
{
public static void main(String args[])
{
 JFrame frmF=new JFrame();
 JLabel lblL=new JLabel();
 frmF.add(lblL);
 frmF.setSize(200,200);
 frmF.setVisible(true);
}
```

```
}
```

- **JLabel(String)**
  crea un’etichetta con la stringa passata come parametro e l’allinea a sinistra.
  Esempio:
  ```
  import java.awt.*;
  import javax.swing.*;
  class JLabel2
  {
   public static void main(String args[])
   {
    JFrame frmF=new JFrame();
    JLabel lblL=new JLabel("Etichetta");
    frmF.add(lblL);
    frmF.setSize(200,200);
    frmF.setVisible(true);
   }
  }
  ```
- **JLabel(String, int)**
  crea un’etichetta contenente come testo la stringa passata come parametro e il parametro intero indicherà l’allineamento della stringa
  Esempio:
  ```
  import java.awt.*;
  import javax.swing.*;
  class JLabel3
  {
   public static void main(String args[])
   {
    JFrame frmF=new JFrame();
    JLabel lblL=new JLabel("Etichetta", JLabel.CENTER);
    frmF.add(lblL);
    frmF.setSize(200,200);
    frmF.setVisible(true);
   }
  }
  ```

Le costanti statiche definite all’interno della classe sono: *JLabel.LEFT, JLabel.RIGHT, JLabel.CENTER.*

I metodi della JLabel sono:

- **setText(String)**
  consente di modificare il testo.
- **setForeground(Color)**
  modifica il colore del testo
- **setBackground(Color)**
  modifica il colore dello sfondo.

Esempio:

lblNome.setText(“Ciao”);

lblNome.setForeground(Color.red);

La classe **Color** presenta le costanti statiche che possono essere usate per il colore (black, blue, green, magenta, orange, pink, red, white, yellow, cyan, gray).

La classe JLabel inoltre eredita tutti i metodi della classe **JComponent** e della classe **Object** . Infatti i metodi *setForegroud()* e *setBackground()* sono ereditati dalla classe JComponent, mentre il metodo *setText()* è un metodo proprio della classe JLabel.

PULSANTI

I pulsanti sono rappresentati dalla classe **JButton**. Solitamente contengono una stringa e sono usati per invocare un’azione quando vengono premuti dall’utente. La stringa che appare nel pulsante indica quindi un’azione da compiere ed è quindi solitamente un verbo. Il costruttore può essere generico oppure contiene come parametri la stringa da visualizzare.

Esempio:

JButton btnOk= new JButton(“OK”);

E` possibile disattivare il pulsante con il metodo *setEnabled(boolean)*.

Esempio:

btnOk.setEnabled(false);

Esempio:

```
import javax.swing.*;
import java.awt.*;
class SetEnabled
{
 public static void main(String arg[])
 {
  JFrame frmF=new JFrame();
  JButton btnB=new JButton("ok");
  frmF.add(btnB);
  frmF.setSize(100,100);
  frmF.setVisible(true);
  btnB.setEnabled(false);
 }
}
```

CASELLE DI TESTO

Sono realizzate dalla classe **JTextField**. La casella di testo è composta da una sola riga che può essere usata per l’input o l’output di stringhe.

I quattro costruttori di questa classe consentono di impostare la stringa da visualizzare e il numero di colonne della casella:

- **JTextField()**
- **JTextField(String)**
- **JTextField(int)**
- **JTextField(String, int)**

La classe JTextField eredita tre metodi molto utili dalla classe **JTextComponent**:

- **setText(String)**
- **getText()**
- **setEditable(boolean)**

Esempio:

txtPrezzo.setEditable(false);

Esempio 2:

```
import javax.swing.*;
import java.awt.*;
class CasellaDiTesto
{
 public static void main(String arg[])
 {
  JFrame frmF=new JFrame();
  JPanel pnlP=new JPanel();
  JTextField txtSenzaPar=new JTextField();
  JTextField txtConPar=new JTextField("Ciao 2");
  JTextField txtIntPar=new JTextField(3);
  JTextField txtStringInt=new JTextField("Ciao 3",8);
  txtSenzaPar.setText("Ciao");
  txtConPar.setEditable(false);
  txtIntPar.getText();
  pnlP.add(txtSenzaPar);
  pnlP.add(txtConPar);
  pnlP.add(txtIntPar);
  pnlP.add(txtStringInt);
  frmF.add(pnlP);
  frmF.setSize(200,150);
  frmF.setVisible(true);
 }
}
```

AREE DI TESTO

La classe **JTextArea** consente di creare un'area di testo composta da più righe.

I costruttori sono simili a quelli delle caselle di testo con l'aggiunta di un parametro che indica il numero di righe:

- **JTextArea()**
- **JTextArea(int, int)**
- **JTextArea(String)**
- **JTextArea(String, int, int, int)**

Il terzo int è una costante interna che specifica la visualizzazione delle barre di scorrimento:

SCROLLBARS_BOTH, SCROLLBARS_VERTICAL, SCROLLBARS_ONLY, SCROLLBARS_HORIZONTAL_ONLY, SCROLLBARS_NONE.

L'area di testo può usare gli stessi metodi della casella di testo.

- **setText(String)**
  cancella il testo precedente e aggiunge il nuovo testo
- **getText()**
  legge tutto il contenuto del testo
- **append(String)**
  per inserire un nuovo testo alla fine dell'area senza cancellare il contenuto.

  Esempio:

```
import javax.swing.*;
import java.awt.*;
class AreaDiTesto
{
 public static void main(String arg[])
 {
  JFrame frmF=new JFrame();
  JPanel pnlP=new JPanel();
  JTextArea txtSenzaPar=new JTextArea();
  JTextArea txtIntPar=new JTextArea(4,5);
  JTextArea txtStringPar=new JTextArea("Ciao 2");
  JTextArea txtStringInt=new JTextArea("Ciaooo",8,5);
  txtSenzaPar.setText("Ciao");
  txtIntPar.getText();
  txtStringPar.append("!");
  pnlP.add(txtSenzaPar);
```

```
   pnlP.add(txtIntPar);
   pnlP.add(txtStringPar);
   pnlP.add(txtStringInt);
   frmF.add(pnlP);
   frmF.setSize(200,150);
   frmF.setVisible(true);
  }
 }
```

LE CASELLE COMBINATE (COMBO BOX)

Le caselle combinate raggruppano un elenco di voci e consentono, tramite un menu a tendina, la scelta di una singola voce. Questo elemento grafico è rappresentato dalla classe **JComboBox**.

La costruzione di una *combo box* avviene attraverso due fasi:

1. creazione del oggetto
2. aggiunta delle voci

la creazione avviene attraverso un costruttore senza parametri. Le voci vengono aggiunte mediante il metodo *addItems(String)*.

Esempio:

JComboBox cboColori= new JComboBox();

cboColori.addItems("rosso");

cboColori.addItems("verde");

LE CASELLE DI CONTROLLO (CHECK BOX)

La casella di controllo è un elemento grafico che gestisce solo due stati, corrispondenti ai valori booleani true e false, se la casella la il segno di spunta si associa il valore true.

Questo elemento grafico viene implementato con la classe **JCheckBox** che mette a disposizione I pulsanti a due stati, se l'utente fa click sulla casella ne causa il cambio di stato. Ogni casella di control è fornita di un'etichetta che viene usata per spiegare all'utente il significato della casella.

Esempio:

JCheckBox ckbScelta= new JCheckBox("prima scelta");

Il costruttore è **JCheckBox(String, boolean).**

La classe GridLayout permette di utilizzare una griglia.

Ecco un esempio completo di tutti i componenti:

```
import java.awt.*;
import javax.swing.*;
class Componenti
 {
   public static void main(String[] args)
     {
       JFrame frmF = new JFrame("Ecco i prncipali componenti della GUI");
       JPanel pnlP = new JPanel();
       JLabel lblEtichetta = new JLabel("Questa è un'etichetta");
       JButton btnPulsante = new JButton("Questo è un pulsante");
       JTextField txtCasellaTesto = new JTextField("inserire qui il testo", 30);
       JTextArea txtAreaTesto = new JTextArea("inserire in questa area il testo", 4,30);
       JCheckBox chkCheck = new JCheckBox("scelta già spuntata",true);
       ButtonGroup radioGroup = new ButtonGroup();
       JRadioButton chkC1 = new JRadioButton("1 scelta spuntata",true);
       JRadioButton chkC2 = new JRadioButton("2 scelta non spuntata",false);
       radioGroup.add(chkC1);
       radioGroup.add(chkC2);
       Choice cboC = new Choice();
       cboC.addItem("prima scelta");
       cboC.addItem("seconda scelta");
       pnlP.add(lblEtichetta);
       pnlP.add(btnPulsante);
       pnlP.add(txtCasellaTesto);
       pnlP.add(txtAreaTesto);
       pnlP.add(chkCheck);
       pnlP.add(chkC1);
       pnlP.add(chkC2);
       pnlP.add(cboC);
       Container cntC = frmF.getContentPane();
       cntC.add(pnlP);
       frmF.setSize(600,500);
       frmF.setLocation(200,200);
       frmF.setVisible(true);
     }
 }
```

Gestione degli eventi

Il software deve riconoscere quando l'utente compie le azioni e predisporre le operazioni da eseguire in corrispondenza delle azioni.
Il primo è un compito affidato al sistema che gestisce gli eventi.
Il secondo è affidato al programmatore che scrive il codice da eseguire per ogni possibile azione dell'utente.
In java gli eventi vengono gestiti solo se è stato registrato un gestore.
Gli eventi sono generati a partire da un oggetto chiamato origine, tutte le componenti sono possibili oggetti di origine.
Uno o più ascoltatori possono essere registrati nell'oggetto origine.
Un ascoltatore è un oggetto.
Per realizzare quindi la gestione degli eventi è necessario:
1)Creare uno o più ascoltatori in base agli eventi che si vogliono gestire;
2)Registrare l'ascoltatore in un oggetto origine ;
3)Gestire l'evento eseguendo il metodo associato;

La gestione degli eventi

Ogni oggetto grafico predisposto ad essere sollecitato in qualche modo dall'utente genera degli *eventi* che vengono inoltrati ad appositi *oggetti ascoltatori* che reagiscono agli eventi secondo quanto codificato dal programmatore.

La gestione degli eventi di Java ha il vantaggio di separare la sorgente degli eventi dal comportamento ad essi associato: un componente non sa cosa avverrà alla sua sollecitazione ma si limita a notificare ai propri ascoltatori che l'evento che essi attendevano è avvenuto e questi provvedono a produrre l'effetto desiderato.

Esempio pulsanteEvento:

```
import java.awt.*;
import java.awt.event.*;
public class PulsanteConEvento
 {
    TextField txtT;
    public PulsanteConEvento()
    {
        Frame frmF = new Frame("Pulsante con evento");
        txtT = new TextField(30);
        Button btnPulsante = new Button("Saluta");
        frmF.setLayout(new FlowLayout());  //mette in colonna le componenti
        btnPulsante.addActionListener(new ascoltaPulsante());
        frmF.add(txtT);
        frmF.add(btnPulsante);
        frmF.pack();
        frmF.setSize(600,500);
        frmF.setLocation(200,200);
        frmF.setVisible(true);
    }
private class ascoltaPulsante implements ActionListener
{
 public void actionPerformed(ActionEvent e)
 {
  txtT.setText("Buongiorno ragazzi");
  }
 } // fine classe ascoltaPulsante
} // fine classe PulsanteConEvento
```

Esempio PulsanteEvento2
(due pulsanti due ascoltatori)

```
import java.awt.*;

import java.awt.event.*;


class PulsantiConEvento

{

 TextField txtT;

  public PulsantiConEvento()

  {

    Frame frmF = new Frame("Pulsanti con Evento");

    txtT = new TextField(30);

    Button btnB1 = new Button("Saluta");

    Button btnB2 = new Button("Chiudi applicazione");

    frmF.setLayout(new FlowLayout());

    btnB1.addActionListener(new ascoltaPulsante1());

    btnB2.addActionListener(new ascoltaPulsante2());

    frmF.add(txtT);

    frmF.add(btnB1);

    frmF.add(btnB2);

    frmF.setSize(300,300);

    //frmF.pack();

    frmF.setVisible(true);

  }

  private class ascoltaPulsante1 implements ActionListener

  {

    public void actionPerformed(ActionEvent e)

    {

      txtT.setText("Ciao ragazzi");

    }
```

```
  }
  private class ascoltaPulsante2 implements ActionListener
  {
    public void actionPerformed(ActionEvent e)
    {
      System.exit(0);
    }
  }
}
public class mainPulsanti
{
  public static void main(String args[])
  {
    PulsantiConEvento PulsantiConEventoObj = new PulsantiConEvento();
  }
}
```

Esempio contatore:
(due pulsanti un ascoltatore)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
public class IncDec extends JFrame
            implements ActionListener {
   JButton pulsanteInc;
   JButton pulsanteDec;
   JLabel eticIncDec;
   JPanel pannello;
   int conta;
```

```
   public IncDec() {
          conta = 0;
          pannello = new JPanel();
          pulsanteInc = new JButton("+");
          pulsanteDec = new JButton("-");
          eticIncDec = new JLabel("Contatore = 0");
          pannello.add(pulsanteInc);
          pannello.add(pulsanteDec);
          pannello.add(eticIncDec);
          setContentPane(pannello);
          pulsanteInc.addActionListener(this);
          pulsanteDec.addActionListener(this);
          setTitle("Incrementa Decrementa");
          pack();
          setSize(300,100);
      setVisible(true);
   }
   public void actionPerformed(ActionEvent e) {
          if (e.getActionCommand().equals("+"))
              conta++;
          if (e.getActionCommand().equals("-"))
          conta--;
          eticIncDec.setText("Contatore = " + conta);
   }
}
class mainContatore
{
public static void main(String[] args)
  {
      IncDec frame = new IncDec();
   }
}
```

Esercizio SommaGUI:

Dati due numeri interi fare la somma utilizzando caselle di testo e pulsanti (due pulsanti due ascoltatori)

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

import java.awt.Font;

public class sommaGui

{

 JFrame frmSommaGui;

 JButton Somma, Resetta;

 JTextField txtNum1, txtNum2, txtNum3;

 JPanel pnlPannello;

 public sommaGui ()

  {

   JFrame frmSommaGui= new JFrame ("somma di due TextField");

   frmSommaGui.setSize(800,800);

   //frmSommaGui.setLayout(new FlowLayout());

   //frmSommaGui.pack();

   frmSommaGui.setLayout(null);

   frmSommaGui.setLocation(200,200);

   frmSommaGui.setVisible(true);

   txtNum1= new JTextField(10);

   txtNum2 = new JTextField(10);

   txtNum3 = new JTextField(15);

   JButton btnSomma= new JButton("Somma");

   JButton btnResetta= new JButton("Resetta");

   JPanel pnlPannello = new JPanel();
```

```
  frmSommaGui.add(txtNum1);
  frmSommaGui.add(txtNum2);
  frmSommaGui.add(btnSomma);
  frmSommaGui.add(btnResetta);
  frmSommaGui.add(txtNum3);
  btnSomma.addActionListener(new ascoltaSomma());
  btnResetta.addActionListener(new ascoltaResetta());
  //setBounds(posizioneX, posizioneY, larghezza, altezza);
  txtNum1.setBounds(80,30,120,40);
  btnSomma.setBounds(210,30,220,40);
  btnResetta.setBounds(450,30,220,40);
  txtNum2.setBounds(80,70,120,40);
  txtNum3.setBounds(80,110,120,40);
  Font f = new Font("Helvetica", Font.PLAIN, 28);
  //Font.BOLD
  txtNum1.setFont(f);
  txtNum2.setFont(f);
  txtNum3.setFont(f);
  txtNum3.setForeground(Color.red);
  btnSomma.setFont(f);
  btnResetta.setFont(f);
 }
private class ascoltaSomma implements ActionListener
 {
  public void actionPerformed(ActionEvent e)
   {
    int num1=Integer.parseInt(txtNum1.getText());
```

```
      int num2=Integer.parseInt(txtNum2.getText());
      int num3=num1+num2;
      txtNum3.setText(Integer.toString(num3));
    }
  }
 private class ascoltaResetta implements ActionListener
 {
   public void actionPerformed(ActionEvent e)
    {
      txtNum1.setText("");
      txtNum2.setText("");
      txtNum3.setText("");
    }
  }
}
```

Layout Management

FlowLayout

```
import javax.swing.*;
import java.awt.*;
public class LayoutManagement1 extends JFrame
{
JButton uno=new JButton("Uno");
JButton due=new JButton("Due");
JButton tre=new JButton("Tre");
JButton quattro=new JButton("Quattro");
JButton cinque = new JButton("Cinque");
public LayoutManagement1()
{
super("FlowLayout");
Container c = this.getContentPane();
c.setLayout(new FlowLayout());
c.add(uno);
c.add(due);
c.add(tre);
c.add(quattro);
c.add(cinque);
setSize(800,800);
setVisible(true);
}
}
```

GridLayout

```
import javax.swing.*;
import java.awt.*;
public class LayoutManagement2 extends JFrame
{
JButton uno=new JButton("Uno");
JButton due=new JButton("Due");
JButton tre=new JButton("Tre");
JButton quattro=new JButton("Quattro");
JButton cinque = new JButton("Cinque");
public LayoutManagement2()
{
super("FlowLayout");
Container ctnC = this.getContentPane();
ctnC.setLayout(new GridLayout(2,3,10,10));
ctnC.add(uno);
ctnC.add(due);
ctnC.add(tre);
ctnC.add(quattro);
ctnC.add(cinque);
setSize(800,800);
setVisible(true);
}
}
```

```
import javax.swing.*;
import java.awt.*;
public class LayoutManagement3 extends JFrame
{
public LayoutManagement3()
{
super("GridLayout");
Container ctnC = this.getContentPane();
ctnC.setLayout(new GridLayout(4,4,10,10));
for(int i = 0; i<15; i++)
{
  JButton btnB=new JButton(String.valueOf(i));
  btnB.setFont(new Font("Dialog", Font.PLAIN, 38));
  ctnC.add(btnB);
}
setSize(800,800);
setVisible(true);
}
}
```

```
import javax.swing.*;
import java.awt.*;
public class LayoutManagement4 extends JFrame
{
JButton nord = new JButton("Nord");
JButton centro = new JButton("Centro");
JButton ovest=new JButton("Ovest");
JButton est=new JButton("Est");
JButton sud=new JButton("Sud");
public LayoutManagement4()
{
super("BorderLayout");
Container ctnC = this.getContentPane();
ctnC.setLayout(new BorderLayout());
ctnC.add(nord,BorderLayout.NORTH);
ctnC.add(centro,BorderLayout.CENTER);
ctnC.add(ovest,BorderLayout.WEST);
ctnC.add(est,BorderLayout.EAST);
ctnC.add(sud,BorderLayout.SOUTH);
setSize(800,800);
setVisible(true);
}
}
```

GRAFICA BI-DIMENSIONALE E ANIMAZIONE IN JAVA

La classe disegna Cerchio

```
import java.awt.*;
class cerchioClass extends Canvas
{
 private int raggio;
 private int x;
 private int y;
 public void paint(Graphics g)
 {
   g.setColor(Color.red);
   g.fillOval(x - raggio, y - raggio, 2*raggio, 2*raggio);
   g.setColor(Color.blue);
   g.setFont(new Font("Dialog", Font.PLAIN, 40));
   g.drawString("Area:"+area(), 50, 40);
 }

 public void setRaggio (int raggio)
  {
    this.raggio=raggio;
  }
  public void setX (int x)
  {
    this.x=x;
  }
  public void setY (int y)
  {
```

```
    this.y=y;
  }
  public double area()
   {
     return(raggio*raggio*Math.PI);
   }
}
import java.awt.*;
class disegnoCerchio
{
  public static void main(String args[])
  {
    Frame f = new Frame("Disegno del Cerchio");
    cerchioClass cerchioObj = new cerchioClass();
    cerchioObj.setRaggio(65);
    cerchioObj.setX(200);
    cerchioObj.setY(200);
    System.out.println ("area del cerchio "+cerchioObj.area());
    f.setSize(900,600);
    f.setLocation(400,400);
    f.add(cerchioObj);
    f.setVisible(true);

  }
  }
```

La classe muoviCerchio

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.Timer;


/*

Muove un cerchio da sinistra a destra, quando cozza la parete

destra il moto si inverte ecc.

*/


class MuoviCerchio extends Panel implements ActionListener

{


  int raggio;         /* raggio del cerchio */

  int millisec;       /* ogni quanti millisecondi si muove */

  Timer timer;        /* per regolare l'animazione */

  boolean versoDestra; /* da che parte va */

  int x_min;          /* ascissa minima corrente del cerchio */


  public MuoviCerchio (int raggio_cerchio, int velocita)

  {

    raggio = raggio_cerchio;

    millisec = velocita;

    x_min = 0;  /* cerchio appoggiato a parete sinistra */

    versoDestra = true; /* direzione di moto */

    timer = new Timer(millisec,this);

    timer.start();
  }
```

```
/* azione da compiere quando scatta il timer, sposta il cerchio
   e poi invoca il ridisegno */
public void actionPerformed(ActionEvent e)
{
  Dimension dimFinestra = getSize();
  if (versoDestra)
  {
    if (x_min>(dimFinestra.width-2*raggio)) /* cozza parete destra */
    {  versoDestra = false;
       x_min--;
    }
    else x_min++;
  }
  else
  {
    if (x_min<=0) /* cozza parete sinistra */
    {  versoDestra = true;
       x_min++;
    }
    else x_min--;
  }
  repaint();
}

public void paint(Graphics areaDisegno)
{
  super.paint(areaDisegno);
```

```
    Dimension dimFinestra = getSize();
    areaDisegno.setColor( Color.blue );
    areaDisegno.fillArc(x_min, (dimFinestra.height/2-raggio), 2*raggio, 2*raggio, 0, 360);
    /*FillArc(int x, int y, int width, int height, int startAngle, int arcAngle) */
  }
  public static void main(String[] args)
  {
     Frame frmAreaDisegno = new Frame("MuoviCerchio");
     MuoviCerchio animazioneCerchio = new MuoviCerchio(50,10);
     animazioneCerchio.setBackground(Color.white);
     frmAreaDisegno.add(animazioneCerchio);
     frmAreaDisegno.setSize(new Dimension(1500,800));
     frmAreaDisegno.setVisible(true);
  }
}
```

Promemoria:
Le reazioni agli eventi generati dall'utente sono gestite da interfacce EventListener. Per potere gestire gli eventi bisogna:
1) importare il pacchetto: **import java.awt.event.*;**
2) usare l'istruzione implements per dichiarare che si userano le interfacce del pacchetto:
class MuoviCerchio extends Panel implements ActionListener
3) Dopo avere implementato l'interfaccia bisogna configurare il componente:
pulsante.addActionListener(this);
In questo modo il componente pulsante è associato agli eventi tramite l'interfaccia ActionListener nella classe this, cioè la classe attuale;
4) Adesso bisogna scrivere la reazione agli eventi. Per ActionListener abbiamo il metodo actionPerformed():
public void actionPerformed(ActionEvent evt) {…}

ActionListener è un'interfaccia usata con i bottoni (pulsanti) e altri componenti. Altri esempi di interfacce: MouseListener, ItemListener,...
public abstract class MouseAdapter extends Object implements MouseListener, MouseWheelListener, MouseMotionListener
Methods of interface MouseListener:
void mouseClicked(MouseEvent e)
void mouseDragged(MouseEvent e)
void mouseEntered(MouseEvent e)
void mouseExited(MouseEvent e)
void mouseMoved(MouseEvent e)
void mousePressed(MouseEvent e)
void mouseReleased(MouseEvent e)

public abstract class **Component** extends Object implements ImageObserver, MenuContainer, Serializable
addActionListener e addMouseListener sono metodi di Component (e quindi di tutte le sue sottoclassi: pulsanti, …)

```
import java.awt.*;

import java.awt.event.*;

import java.awt.geom.*;


/*

Permette di disegnare e cancellare cerchi.

Usa i pulsanti radio per scegliere l'operazione.

Per inserire clicca sul punto che sara' il centro.

Per cancellare clicca sul cerchio.

*/


public class disegnaCerchio extends Panel

{


  /* radioPulsanti a due stati, mutuamente esclusivi */

  Checkbox chkInserimento;

  Checkbox chkCancellazione;


  /* raggio di un cerchio */

  int raggio;


  /* massimo numero di cerchi ammessi */

  static final int MAX_NUM = 30;


  /* array di cerchi presenti e loro numero */

  Ellipse2D[] circle_list;

  int circle_num;
```

```
 public disegnaCerchio()
  {   /* crea i due radioPulsanti in un gruppo in modo che siano esclusivi */
   CheckboxGroup chkGruppo = new CheckboxGroup();
   chkInserimento = new Checkbox ("Inserimento pallini rossi",chkGruppo,true);
   chkCancellazione = new Checkbox ("Cancellazione pallini rossi",chkGruppo,false);
   /* stabilisce situazione iniziale per i cerchi */
   circle_list = new Ellipse2D[MAX_NUM];
   circle_num = 0;
   raggio = 10;
   /* rende il pannello sensibile al click del mouse in modo che
      reagisca guardando quale dei due radioPulsanti è stato selezionato e
      invocando inserimento o cancellazione secondo il caso
*/
   addMouseListener( new MouseAdapter()
   {   public void mouseClicked(MouseEvent e)
       {
        if ( chkInserimento.getState()==true )
             insertCircle(e.getX(),e.getY());
        else deleteCircle(e.getX(),e.getY());
       }
   });
}

 /* Disegna i cerchi in rosso */
 public void paint(Graphics areaGrafico)
 {
```

```
  int i;
  super.paint(areaGrafico);
  areaGrafico.setColor(Color.red);
  Graphics2D areaGraficoObj = (Graphics2D)areaGrafico;
  for (i=0; i<circle_num; i++)  areaGraficoObj.fill(circle_list[i]);
}
/* inserisce nuovo cerchio alla posizione indicata e ridisegna */
public void insertCircle(int x, int y)
{
  if (circle_num==MAX_NUM-1)
    System.out.println("Troppi cerchi, non posso inserire");
  else
  {  Ellipse2D ellipseObj = new Ellipse2D.Float(x-raggio,y-raggio,2*raggio,2*raggio);
    circle_list[circle_num++] = ellipseObj;
  }
  repaint();
}
/* Cancella cerchio su cui cade (x,y), se esiste, e ridisegna */
public void deleteCircle(int x, int y)
{
  int i, j;
  boolean deleted = false;
  for (i=0; (i<circle_num)&&(!deleted); i++)
  {  if ( circle_list[i].contains(x,y) )
    {  /* cancella cerchio dall'array spostando tutti i seguenti */
      for (j=i+1; j<circle_num; j++)
      {  circle_list[j-1] = circle_list[j];
```

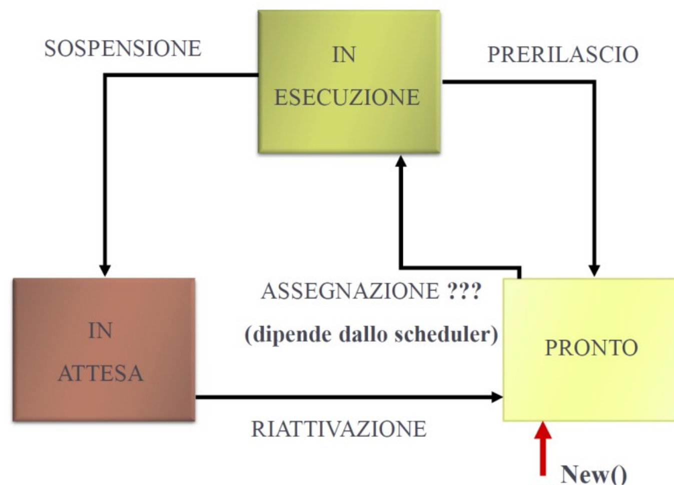
```
        }
        circle_num--;
        deleted = true;
      }
  }
  if (deleted) repaint();
}

/* crea frame che ospita il pannello e i suoi due pulsantiRadio */
public static Frame createFrame(int dimX, int dimY)
{
  Frame frmFinestra = new Frame("Disegna Cerchio");
  frmFinestra.setLayout(new BorderLayout());
  /* mette al centro il pannello grafico */
  disegnaCerchio disegnaCerchioObj = new disegnaCerchio();
  disegnaCerchioObj.setBackground(Color.white);
  frmFinestra.add(BorderLayout.CENTER,disegnaCerchioObj);
  /* mette in alto il pannello coi due bottoni */
  Panel p = new Panel();
  p.setLayout( new FlowLayout() );
  disegnaCerchioObj.chkInserimento.setBounds( 10, 10, 60, 10 );
  p.add(disegnaCerchioObj.chkInserimento);
  p.add(disegnaCerchioObj.chkCancellazione);
  frmFinestra.add(BorderLayout.NORTH,p);
  /* stabilisce dimensioni */
  frmFinestra.setSize(dimX,dimY);
  return frmFinestra;
```

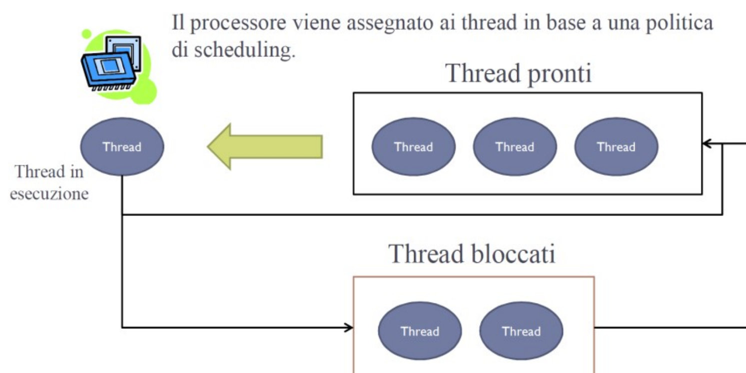
```
 }

  public static void main(String s[])
 {
   Frame frmFinestra = disegnaCerchio.createFrame(1500,800);
   frmFinestra.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e)
          {System.exit(0);}
       });
   frmFinestra.show();
 }
}
```

Cos'è un Thread?

Si può avere la necessità di suddividere un programma in sottoattività separate da eseguire indipendentemente l'una dall'altra.
Queste sottoattività prendono il nome di *thread*: esse vengono progettate come entità separate.
Col termine processo si intende un programma in esecuzione sul sistema;
Col termine thread si intende una sotto-attività all'interno di un processo.
Per operazioni complesse, bisognerà sincronizzare i vari thread per farli cooperare e raggiungere il risultato finale.
Utilizzare il modello dei thread consente di ripartire il tempo di CPU fra tutti i thread in esecuzione. Ciascun thread ha la "sensazione" di avere per se tutta la CPU che in realtà viene ripartita in maniera automatica fra tutti i thread. L'uso dei thread può in qualche caso ridurre le prestazioni dei programmi, ma in generale i vantaggi in fase di progettazione, uso delle risorse e tempi di risposta rendono il loro uso da preferirsi. I sistemi che consentono la gestione nei programmi di più thread si dicono multithreading.
Un thread può trovarsi nei seguenti stati:

1. Nuovo: il thread è stato creato con un'operazione di new() ma non è ancora stato mandato in esecuzione.
2. Eseguibile: il thread è pronto ad essere eseguito immediatamente appena gli viene assegnata la CPU.
3. Morto: il thread ha terminato la sua esecuzione ritornando dal metodo run.
4. Bloccato: il thread non può essere eseguito, per esempio potrebbe essere in attesa di avere a disposizione una risorsa.





}

Esercizio Macchine (utilizzo dei Thread)

```
public class Macchine extends Thread
{
	public Macchine(String nome_thread)
	{
		super(nome_thread);
	}

	public void run()
	{
		for (int i=0;i<7;i++)
		{
			System.out.println(getName());
		}
	}

	public static void main(String[] str)
	{
		Macchine auto1=new Macchine("Macchina 1");
		auto1.setPriority(10);
		Macchine auto2=new Macchine("Macchina 2");
		auto2.setPriority(7);
		Macchine auto3=new Macchine("Macchina 3");
		auto3.setPriority(1);
		auto1.start();
		auto2.start();
		auto3.start();
	}
}
```

Esercizio ThreadApp(utilizzo dei Thread, pallino animato)

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Event;
import java.awt.*;
import javax.swing.*;

public class ThreadApp
{
	public static void main(String args[])
	{
		JFrame f = new JFrame("Disegno animato");
		f.setSize(780,150);
		f.setLocation(1,1);
		TPallino t1 = new TPallino(Color.red,100);
		f.getContentPane().add(t1);
		f.setVisible(true);
		Thread runner1 = new Thread(t1);
		runner1.start();
	}
}

class TPallino extends JComponent implements Runnable
{
	int xpos;
	int raggio;
	int vel=100;
	Color colore;

	public TPallino(Color c,int r)
```

```
	{
		xpos=0;
		raggio=r;
		colore=c;
	}
	public void run()
	{
		while(true)
		{
			for (xpos=5+raggio/4;xpos<=705;xpos+=4)
			{
				repaint();
				try{Thread.sleep(100);}
				catch(InterruptedException e){}
			}
		for (xpos=705;xpos>5;xpos-=4)
		{
			repaint();
			try{Thread.sleep(100);}
			catch(InterruptedException e){}
		}
		}
	}
	public void paint(Graphics g)
	{
		g.setColor(colore);
		g.fillOval(xpos,5,raggio/2,raggio/2);
	}
}
```

Esercizio MainProvaThreadBottoni(utilizzo dei Thread, pallino animato con bottoni)

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class ProvaThreadBottoni
{
	Thread runner;
	public ProvaThreadBottoni()
	{
		JFrame F1=new JFrame("Controllo");
		JButton INIZIA=new JButton("INIZIA");
		JButton STOP=new JButton("STOP");
		JButton RIAVVIA=new JButton("RIAVVIA");
		F1.setSize(300,200);
		F1.setLocation(50,50);
		F1.getContentPane().setLayout(new FlowLayout());
		INIZIA.addActionListener(new ascoltaBottone());
		STOP.addActionListener(new ascoltaBottone());
		RIAVVIA.addActionListener(new ascoltaBottone());
		F1.getContentPane().add(INIZIA);
		F1.getContentPane().add(STOP);
		F1.getContentPane().add(RIAVVIA);
		F1.setVisible(true);
		JFrame F2=new JFrame("Disegno");
		F2.setSize(300,200);
		F2.setLocation(1,1);
		TPallino t= new TPallino();
		F2.getContentPane().add(t);
```

```
			F2.setVisible(true);
			runner=new Thread(t);
		}
		private class ascoltaBottone implements ActionListener
		{
			public void actionPerformed(ActionEvent e)
			{
				String bottone=e.getActionCommand();
				if(bottone.equals("INIZIA"));
				if(!runner.isAlive())
				runner.start();
				if(bottone.equals("STOP"));
				runner.suspend();
				if(bottone.equals("RIAVVIA"));
				runner.resume();
			}
		}
}



class TPallino extends JComponent implements Runnable
{
		int xpos;

		public void run()
		{
			while(true)
```

```
			{
				for (xpos=5;xpos<=150;xpos+=4)
				{
					repaint();
					try{Thread.sleep(100);}
					catch(InterruptedException e){}
				}
				for (xpos=150;xpos>5;xpos-=4)
				{
					repaint();
					try{Thread.sleep(100);}
					catch(InterruptedException e){}
				}
			}
		}
		public void paint(Graphics g)
		{
			g.setColor(Color.white);
			g.fillRect(0,0,100,100);
			g.setColor(Color.blue);
			g.fillOval(xpos,5,90,90);
		}
}
public class MainProvaThreadBottoni
{
		public static void main(String args[])
		{
			ProvaThreadBottoni kk=new ProvaThreadBottoni();
		}
}
```

Le applet

Un **applet** è un applicazione java che viene eseguita all'interno di un browser.
Le applet presentano alcune caratteristiche che le differenziano dalle altre applicazioni:

- Un'applet non ha il metodo main;
- Un'applet è un oggetto grafico, contenitore di *awt* e *swing* con le caratteristiche di un pannello;
- Un'applet non può leggere né modificare file nel sistema client;
- Un'applet non può modificare file né verificare quali programmi sono installati sul disco;
- Un'applet non può eseguire programmi sul sistema client;
- Un'applet non può utilizzare librerie del sistema client.

Esempio 1 (creare un'applet) :

CiaoMondo.java

```
import java.applet.*;
import javax.swing.*;
import java.awt.*;
public class CiaoMondo extends JApplet
{
	public void init()
	{
		Container sfondo=getContentPane();
		sfondo.setLayout(new FlowLayout());
		JLabel miaLabel=new JLabel("Ciao, sono la vostra prima applet!");
		sfondo.add(miaLabel);
	}
}
```

CiaoMondo.html

```
<html>
<head></head>
<body>
	Questa pagina web contiene un'applet java
	<applet code="CiaoMondo.class" width=500 height=300>
	</applet>
</body>
</html>
```