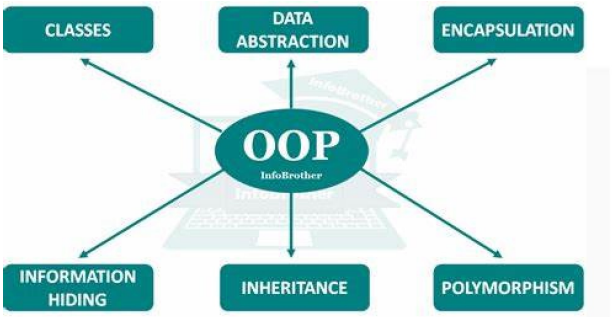
ITIS-LS "Francesco Giordani" Caserta

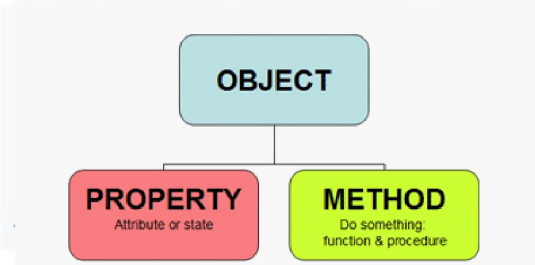
prof. Ennio Ranucci

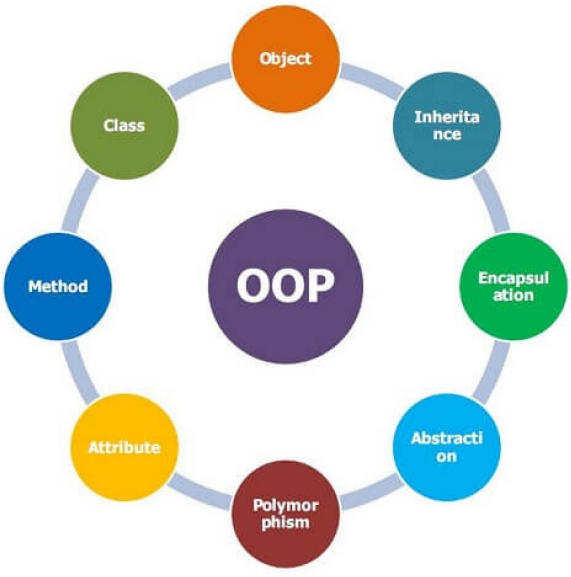
a.s. 2020-2021

OOP C++ Java

Primi esempi C++ e Java







OOP

Un programma procedurale mal si presta a realizzare il concetto di "Componente Software", ovvero di un prodotto in grado di garantire le caratteristiche di riusabilità, modificabilità e manutenibilità. Una delle cause di tale limite è da ricercare sicuramente nel fatto che esiste una evidente divisione tra i dati e le strutture di controllo che agiscono su di essi; in altre parole i moduli risultano avere un approccio orientato alla procedura piuttosto che ai dati. Con l'avvento della programmazione ad oggetti questi limiti vengono superati.
Il paradigma OOP è basato sul fatto che esiste una serie di oggetti che interagiscono vicendevolmente, scambiandosi messaggi ma mantenendo ognuno il proprio stato ed i propri dati.
Si è soliti suddividere i messaggi nelle seguenti categorie:
1) Costruttori costituiscono il momento in cui viene creato un oggetto. Essi devono essere richiamati ogni volta che si vuole creare una nuova istanza di un oggetto appartenente ad una classe e, solitamente, svolgono al loro interno funzioni di inizializzazione.
2) Distruttori come si intuisce facilmente dal nome, svolgono la funzione inversa dei costruttori
3)Accessori (Accessors) vengono utilizzati per esaminare il contenuto di una proprietà di una classe.
4)Modificatori (Mutators) rappresentano tutti i messaggi che provocano una modifica nello stato di un oggetto

Principi fondamentali OOP: Incapsulamento - Ereditarietà - Polimorfismo

L'***incapsulamento*** è uno dei quattro concetti di programmazione orientata agli oggetti, consiste nel tenere in un'unica struttura i **dati** e le **funzioni** (metodi o funzioni membro) che operano sui dati.
I vantaggi principali portati dall'incapsulamento sono: robustezza, indipendenza e l'estrema riusabilità degli oggetti creati.

L'***information hiding*** (nascondere le informazioni) è la tecnica che consente rendere invisibile l'implementazione, nel senso che si può accedere ai dati solo attraverso l'interfaccia pubblica (costituita dai metodi pubblici della classe).

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: 2020/2021
Classe 4^ sez.C spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: es1
Versione: 1.0
Programmatore/i:
Sistema Operativo: Windows 10
Compilatore/Interprete: javac e java ; blueJ
Obiettivo didattico: concetto di incapsulamento e information hiding
Obiettivo del programma: Realizzare la classe "dataClass" senza incapsulamento e senza information hiding e la classe es1 che utilizza la classe "dataClass" per inserire giorno, mese, anno e poi stampare la data a video.

Codifica C++

```
#include <iostream>
using namespace std;
class dataClass
{
  public:
    int giorno;
    int mese;
    int anno;
};

int main()
{
    dataClass dataObj;
    dataObj.giorno=9;
    dataObj.mese=1;
    dataObj.anno=2021;
    cout << dataObj.giorno<<"/"<<dataObj.mese<<"/"<<dataObj.anno<< endl;
    return 0;
}
```

Codifica Java

```
public class dataClass
{
  public int giorno;
  public int mese;
  public int anno;
}

class es1
{
 public static void main(String[] args)
 {
  dataClass dataObj = new dataClass();
  dataObj.giorno=9;
  dataObj.mese=1;
  dataObj.anno=2021;
  System.out.println(dataObj.giorno+"-"+dataObj.mese+"-"+dataObj.anno);
 }
}
```

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: 2020/2021
Classe 4^ sez.C spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: es2
Versione: 1.0
Programmatore/i:
Sistema Operativo: Windows 10
Compilatore/Interprete: javac e java ; blueJ
Obiettivo didattico: concetto di incapsulamento e information hiding
Obiettivo del programma: Realizzare la classe "dataClass" con incapsulamento e con information hiding e la classe es1 che utilizza i metodi pubblici (incapsulamento) "setGiorno", "setMese", "setAnno" "getGiorno", "getMese", "getAnno" della classe "dataClass" per inserire giorno, mese, anno (attributi privati: information hiding) e poi stampare la data a video.

Codifica C++

```
#include <iostream>
using namespace std;
class dataClass
{
 // dati privati - information hiding
 private:
   int giorno;
   int mese;
   int anno;
 // metodi o funzioni membro pubblici - incapsulamento (attributi e metodi incapsulati in un'unica struttura: la classe

 public:
   void setGiorno(int giorno){this->giorno=giorno;};
   void setMese(int mese){this->mese=mese;};
   void setAnno(int anno){this->anno=anno;};
   int getGiorno(){return giorno;};
   int getMese(){return mese;};
   int getAnno(){return anno;};
};

int main()
{
   dataClass dataObj;
   /*
    information hiding: i dati non sono accessibili da funzioni che non fanno parte della classe
```

```
    dataObj.giorno=9;
    dataObj.mese=1;
   dataObj.anno=2021;
   */

   // l'accesso ai dati è consentito solo attraverso i metodi

   dataObj.setGiorno(9);
   dataObj.setMese(1);
   dataObj.setAnno(2021);
   cout << dataObj.getGiorno()<<"/"<<dataObj.getMese()<<"/"<<dataObj.getAnno()<< endl;
  return 0;
}
```

Codifica Java

```
public class dataClass
{
  private int giorno;
  private int mese;
  private int anno;
  public void setGiorno(int giorno){this.giorno=giorno;};
  public void setMese(int mese){this.mese=mese;};
  public void setAnno(int anno){this.anno=anno;};
  public int getGiorno(){return giorno;};
  public int getMese(){return mese;};
  public int getAnno(){return anno;};
}

class es2
{
 public static void main(String[] args)
 {
  dataClass dataObj = new dataClass();
  dataObj.setGiorno(9);
  dataObj.setMese(1);
  dataObj.setAnno(2021);
  System.out.println(dataObj.getGiorno()+"-"+dataObj.getMese()+"-"+dataObj.getAnno());
 }
}
```

L'***ereditarietà*** costituisce il secondo principio fondamentale della programmazione ad oggetti.
In generale, essa rappresenta un meccanismo che consente di creare nuovi oggetti che siano basati su altri già definiti.
Si definisce oggetto figlio quello che eredita tutte o parte delle proprietà e dei metodi definiti nell'oggetto padre. È semplice poter osservare esempi di ereditarietà nel mondo reale. Ad esempio, esistono al mondo centinaia di tipologie diverse di mammiferi: cani, gatti, uomini, balene e così via. Ognuna di tali tipologie di mammiferi possiede alcune caratteristiche che sono strettamente proprie (ad esempio, soltanto l'uomo è in grado di parlare) mentre esistono, d'altra parte, determinate caratteristiche che sono comuni a tutti i mammiferi (ad esempio, tutti i mammiferi hanno il sangue caldo e nutrono i loro piccoli).
ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: 2020/2021
Classe 4^ sez.C spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: es3
Versione: 1.0
Programmatore/i:
Sistema Operativo: Windows 10
Compilatore/Interprete: javac e java ; blueJ
Obiettivo didattico: concetto di ereditarietà
Obiettivo del programma: Realizzare la classe "dataOrarioClass" che deriva dalla classe "dataClass" ed aggiunge l'orario i metodi pubblici "setOra", "setMinuti", "getOra", "getMinuti", e poi stampare la data e l'orario a video.

Codifica C++

```
#include <iostream>
using namespace std;
class dataClass
{
   private:
   int giorno;
   int mese;
   int anno;
  public:
   dataClass(){giorno=0;mese=0;anno=0;};
   void setGiorno(int giorno){this->giorno=giorno;};
   void setMese(int mese){this->mese=mese;};
   void setAnno(int anno){this->anno=anno;};
   int getGiorno(){return giorno;};
   int getMese(){return mese;};
   int getAnno(){return anno;};
};
```

```
class dataOrarioClass: public dataClass
{
 private:
    int sec;
 public:
    dataOrarioClass():dataClass()
    {sec=15248;};

    void setOra(int ora){sec=ora*3600;};
    void setMinuti(int minuti){sec=sec+minuti*60;};
    int getOra();
    int getMinuti();
};

int dataOrarioClass::getOra()
{
    return int(sec/3600);
}
int dataOrarioClass::getMinuti()
{
    return int(sec%3600/60);
}
int main()
{
    dataOrarioClass dataOrarioObj;
    dataOrarioObj.setGiorno(9);
    dataOrarioObj.setMese(1);
    dataOrarioObj.setAnno(2021);
    //dataOrarioObj.setOra(12);
    //dataOrarioObj.setMinuti(10);
   cout<<dataOrarioObj.getGiorno()<<"/"<<dataOrarioObj.getMese()<<
"/"<<dataOrarioObj.getAnno()<< endl;
   cout << dataOrarioObj.getOra()<<":"<<dataOrarioObj.getMinuti()<< endl;
   return 0;
}
```

Codifica Java

```
public class dataClass
{
  private int giorno;
  private int mese;
  private int anno;
  public void setGiorno(int giorno){this.giorno=giorno;};
  public void setMese(int mese){this.mese=mese;};
  public void setAnno(int anno){this.anno=anno;};
  public int getGiorno(){return giorno;};
  public int getMese(){return mese;};
  public int getAnno(){return anno;};
}
class dataOrarioClass extends dataClass
{
 private int sec;
 public  dataOrarioClass()
    {
       super();
       sec=15248;
    };
    void setOra(int ora){sec=ora*3600;};
    void setMinuti(int minuti){sec=sec+minuti*60;};
    int getOra()  { return (int)sec/3600; }
    int getMinuti()  {  return (int)sec%3600/60;  }

}
class es3
{
 public static void main(String[] args)
 {
  dataOrarioClass dataOrarioObj = new dataOrarioClass();
  dataOrarioObj.setGiorno(9);
  dataOrarioObj.setMese(1);
  dataOrarioObj.setAnno(2021);
  System.out.println(dataOrarioObj.getGiorno()+

  "-"+dataOrarioObj.getMese()+"-"+dataOrarioObj.getAnno());
  System.out.println(dataOrarioObj.getOra()+":"+dataOrarioObj.getMinuti());
 }
}
```

Il terzo elemento fondamentale della programmazione ad Oggetti è il ***polimorfismo***.
Letteralmente, la parola polimorfismo indica la possibilità per uno stesso oggetto di assumere più forme. La possibilità che le classi derivate implementino in modo differente i metodi e le proprietà dei propri antenati rende possibile che gli oggetti appartenenti a delle sottoclassi di una stessa classe rispondano diversamente alle stesse istruzioni. Ad esempio in una gerarchia in cui le classi Cane e Gatto discendono dalla SuperClasse Mammifero potremmo avere il metodo mangia() che restituisce la stringa "osso" se eseguito sulla classe Cane e "pesce" se eseguito sulla classe Gatto. I metodi che vengono ridefiniti in una sottoclasse sono detti “polimorfi”, in quanto lo stesso metodo si comporta diversamente a seconda del tipo di oggetto su cui è invocato.

Nei linguaggi ad oggetti sono presenti due tipi di polimorfismo:
- Overriding, o sovrapposizione dei metodi;
- Overloading, o sovraccarico dei metodi.

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);
```

L’overloading è usato anche per offrire più costruttori ad una classe.

ITIS-LS “Francesco Giordani” Caserta
Anno scolastico: 2020/2021
Classe 4^ sez.C spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell’esercizio: es4
Versione: 1.0
Programmatore/i:
Sistema Operativo: Windows 10
Compilatore/Interprete: javac e java ; blueJ
Obiettivo didattico: concetto di polimorfismo-overloading
Obiettivo del programma: Realizzare la classe “numeroClass” che utilizza 3 costruttori diversi e la funzione “calcola” che moltiplica o somma i parametri d’ingresso a seconda che siano interi o reali.

Codifica C++

```
#include <iostream>
using namespace std;
class numeroClass
{
  private:
    int num;
  public:
    numeroClass(){ num=0;};
    numeroClass(int num){this->num=num;};
    numeroClass(int num1, int num2){this->num=num1*num2;};
    int getNum(){return num;};
    int calcola(int num1, int num2){return num1*num2;};
    double calcola(double num1, double num2) {return num1+num2;};
};
int main()
{
  numeroClass numeroObj(4,5);
  cout<<"valore iniziale di num: "<<numeroObj.getNum()<<endl;
  float n1=3.42;  float n2=5.31;
  cout<<"valore calcolato: "<<numeroObj.calcola(n1,n2);
  return 0;
}
/* test
1) costruttore senza parametri e calcola(2,3)
2) numeroObj(4) e calcola(2.1,3.2)
3) numeroObj(4,5) e float n1=3.42 n2=5.31 calcola(n1,n2)
*/
```

Codifica Java

```
class numeroClass
{
  private int num;
  public numeroClass() { num=0;}
  public numeroClass(int num){ this.num=num;}
  public numeroClass(int num1, int num2){this.num=num1*num2;}
  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[])
     {
       numeroClass numeroObj=new numeroClass(4,5);
       float n1=3.42f;   float n2=5.31f;
       System.out.println("valore iniziale di num: " + numeroObj.getNum());
       System.out.println("valore calcolato: " + numeroObj.calcola(n1,n2));
     }
}
```

OVERRIDING

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.

Codifica C++

```
#include <iostream>
using namespace std;
class forma Geometrica
{
 public:
   void Nascondi();
   void Mostra();
   void MuoviA(int x, int y);
};
class puntoClass: public forma Geometrica
{
 protected:
     int x,y;
 public:
   puntoClass(int x, int y)
    {
          this->x = x;
          this->y = y;
    }
   virtual void Mostra()
    {
     cout<<"Punto visibile in "<<x<<" "<<y<<endl;
    }
   virtual void Nascondi()
    {
     cout<<"Punto nascosto in "<<x<<" "<<y<<endl;
    }
   void MuoviA(int NuovoX, int NuovoY)
    {
     Nascondi();
     this->x= NuovoX;this->y= NuovoY;
     Mostra();
    }
};
class cerchioClass: public puntoClass
{
 private:
   int raggio;
 public:
   cerchioClass(int x, int y, int raggio):puntoClass(x,y)
    {
```

```
      this->raggio = raggio;
     }
   void Mostra()
    {
     cout<<"Cerchio visibile in "<<x<<" "<<y<<endl;
    }
   void Nascondi()
    {
     cout<<"Cerchio nascosto in "<<x<<" "<<y<<endl;
    }
/*
   void MuoviA(int NuovoX, int NuovoY)
   {
      Nascondi();
      this->x= NuovoX;this->y= NuovoY;
      Mostra();
  }
*/
};
int main()
 {
   cerchioClass cerchioObj(10,20,100);
   cerchioObj.Mostra();
   cout<<endl;
   cerchioObj.MuoviA(20,30);
   cout<<endl;
 }
```

Codifica Java

```
public abstract class formaGeometrica
{
  public abstract void Nascondi();
  public abstract void Mostra();
  public abstract void MuoviA(int x, int y);
}

public class puntoClass extends formaGeometrica
{
  protected int x,y;
  public puntoClass(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 cerchioClass extends puntoClass
{
  private int raggio;
  public cerchioClass(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 es5
{
   public static void main(String[] args)
   {
     puntoClass puntoObj = new puntoClass(2,4);
     cerchioClass cerchioObj = new cerchioClass(10,20,100);
     formaGeometrica figura;
     //figura=puntoObj;
     figura=cerchioObj;
     figura.Mostra();
```

```
        System.out.println();
        figura.MuoviA(20,30);
        System.out.println();
    }
}
```