# Classi E Oggetti

- Ogni oggetto è l'istanza di una classe, e ogni classe ha zero o più istanze. Le classi sono statiche, quindi per esse l'esistenza, la semantica e le relazioni sono fissate prima dell'esecuzione del programma
- La classe per ciascun oggetto è statica, ovvero una volta che l'oggetto è creato la classe è fissata. Gli oggetti sono creati e distrutti durante l'esecuzione di un'applicazione
- Le classi formano il vocabolario del dominio del problema. Gli oggetti insieme interagiscono per soddisfare i requisiti del problema
- Quante istanze ho bisogno per la classe `Pagamenti`?
  - Potrei avere una istanza per tutta l'applicazione, o una istanza per ciascun file letto
  - Se decido di avere una sola istanza, come posso vietare la creazione di più istanze



```
public class MainPagam {   // versione 0.1

   public static void main(String[] args) {

      Pagamenti p = new Pagamenti();   // p è una istanza di Pagamenti
      try {
         p.leggiFile("csvfiles", "Importi.csv");  // lettura primo file
         // p contiene tutti i valori letti dal file
      } catch (IOException e) {
      }
      System.out.println("totale: " + p.calcolaSomma());
      System.out.println("max: " + p.calcolaMassimo());
   }

}
```



```
public class MainPagam {   // versione 0.2
   public static void main(String[] args) {
      Pagamenti p = new Pagamenti();   // p e' l'unica istanza
      try {
         p.leggiFile("csvfiles", "Importi.csv");  // lettura primo file
      } catch (IOException e) {
      }
      System.out.println("file 1 totale: " + p.calcolaSomma());
      System.out.println("file 1 max: " + p.calcolaMassimo());

      try {
         p.leggiFile("csvfiles", "PagMarzo.csv");  // lettura secondo file
         // p adesso contiene tutti i valori letti da entrambi i file
      } catch (IOException e) {
      }
      System.out.println("file 1 e 2 totale: " + p.calcolaSomma());
      System.out.println("file 1 e 2 max: " + p.calcolaMassimo());
   }
}
```

- Poiché il metodo `leggiFile()` della classe `Pagamenti` non cancella i valori in lista, posso leggere più file e inserirli nella stessa lista con varie chiamate a `leggiFile()`
  - Singola responsabilità per il metodo



```
public class MainPagam {  // versione 0.3
   public static void main(String[] args) {
      Pagamenti p1 = new Pagamenti();   // prima istanza
      Pagamenti p2 = new Pagamenti();   // seconda istanza

      try {
         p1.leggiFile("csvfiles", "Importi.csv");  // lettura primo file
         p2.leggiFile("csvfiles", "PagMarzo.csv"); // lettura secondo file
      } catch (IOException e) {
      }
      System.out.println("file 1 totale: " + p1.calcolaSomma());
      System.out.println("file 1 max: " + p1.calcolaMassimo());
      System.out.println("file 2 totale: " + p2.calcolaSomma());
      System.out.println("file 2 max: " + p2.calcolaMassimo());
   }
}
```

- E' possibile usare varie istanze di `Pagamenti` per mettere valori provenienti da file diversi
- Supponiamo che l'applicazione **non** debba avere più di una istanza di `Pagamenti`, occorre il ***Design Pattern Singleton*** per vietare la creazione di più istanze

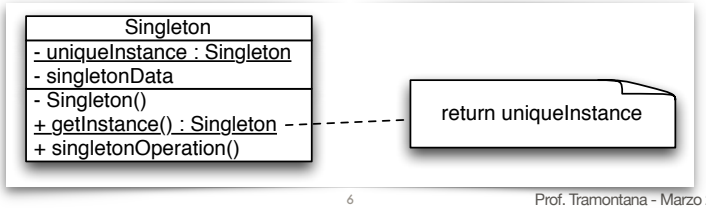
# Design Pattern Singleton

- Intento
  - Assicurare che una classe abbia una sola istanza e fornire un punto di accesso globale all'istanza
- Motivazione
  - Alcune classi dovrebbero avere esattamente una istanza in tutta l'applicazione, es. uno spooler di stampa, un file system, un window manager, una lista clienti, etc.
  - Una variabile globale rende un oggetto accessibile ma non proibisce di avere più oggetti per una classe
  - La classe stessa dovrebbe essere responsabile di tener traccia del suo unico punto di accesso



# Singleton

- Soluzione
  - La classe che deve essere un *Singleton* dovrà implementare un'operazione `getInstance()` sulla classe (ovvero, in Java è un metodo `static`) che ritorna l'unica istanza creata
  - La classe *Singleton* è responsabile per la creazione dell'istanza
  - Il costruttore della classe *Singleton* è privato, così da non permettere la creazione tramite `new` ad altre classi






# Esempio Classe Singleton Fib

```
// Classe Singleton che tiene una lista di
// interi
public class Fib {
  // l'unica istanza e' tenuta da obj
  private static Fib obj = new Fib();
  private int[] x={1, 2, 3, 5, 8, 13, 21,
   34, 55, 89, 144};
  private int i;
  private Fib() {
     i = 3;
  }
  public static Fib getInstance() {
     return obj; // restituisce l'istanza
  }
  public int getValue() {
     if (i<11) i++;
     return x[i-1];
  }
  public void revert() {
     i = 0;
  }
}
```

```
public class TestFib {
  public static void main(String[] args) {
    // richiede una istanza di Fib
    Fib f = Fib.getInstance();
    System.out.print("f "+f.getValue());
    System.out.println(" "+f.getValue());

    // richiede una nuova istanza
    Fib f2 = Fib.getInstance();
    System.out.print("f2 "+f2.getValue());
    System.out.println(" "+f2.getValue());

    // Si ha un errore a compile-time con:
    // Fib f3 = (Fib) f2.clone();
    // Fib f4 = new Fib();
  }
}
```

Output dell'esecuzione
```
f 5 8
f2 13 21
```



# Esempio Classe Logs

```
public class Logs {                  // Classe Singleton
   private static Logs obj;          // obj tiene l'istanza
   private List<String> l;           // tiene i dati da registrare

   private Logs() {                  // il costruttore è privato
      empty();
   }
   public static Logs getInstance() {    // restituisce l'unica istanza
      if (obj == null) obj = new Logs(); // crea l'istanza se non presente
      return obj;
   }
   public void record(String s) {    // accoda il dato
      l.add(s);
   }
   public String dumpLast() {        // restituisce l'ultimo dato
      return l.getLast();
   }
   public String dumpAll() {         // restituisce tutti i dati
      String acc = "";
      for (String s : l)             // s tiene ciascun elemento in lista, ad ogni passata
         acc = acc.concat(s);
      return acc;
   }
   public void empty() {
      l = new ArrayList<String>();
   }
}
```

# Test Per Classe Logs

```
public class TestLogs {
   private Logs lg = Logs.getInstance();

   public void testSingl() {
       initLogs();
       Logs lg2 = Logs.getInstance();
       lg2.record("uno");
       lg2.record("due");
       if (lg.dumpLast().equals("due"))
           System.out.println("OK test logs singl");
       else
           System.out.println("FAILED test logs singl");
   }
   public void testLast() {
       initLogs();
       if (lg.dumpLast().equals("three "))
           System.out.println("OK test logs last");
       else
           System.out.println("FAILED test logs last");
   }
   public void testAll() {
       initLogs();
       if (lg.dumpAll().equals("one two three "))
           System.out.println("OK test logs all");
       else
           System.out.println("FAILED test logs all");
   }

   private void initLogs() {
       lg.empty();
       lg.record("one ");
       lg.record("two ");
       lg.record("three ");
   }
   public static void
     main(String[] args) {
       TestLogs tl = new TestLogs();
       tl.testSingl();
       tl.testAll();
       tl.testLast();
   }
}
```

Output dell'esecuzione

```
OK test logs singl
OK test logs all
OK test logs last
```



# Considerazioni

- Grazie al Design Pattern Singleton, la classe `Fib` non può avere più di una istanza a runtime (lo stesso per la classe `Logs`)
- Si dice che la classe `Fib` implementa (o svolge) il ruolo di Singleton
- Esercizio: trasformare la classe `Pagamenti` in Singleton
- Conseguenze: se dopo aver realizzato `Pagamenti` come un Singleton, si volessero più istanze di `Pagamenti`, anziché una sola, il codice delle classi chiamanti rimarrebbe invariato, e la sola classe da modificare è la classe `Pagamenti`. La variante che permette un numero finito di istanze si chiama Multiton
- Principio delle Conseguenze Locali: un cambiamento in qualche punto del codice non dovrebbe causare problemi in altri punti
- Una variante Multiton di `Pagamenti` potrebbe associare una istanza a ciascuna cartella, la decisione sull'istanziazione dipende quindi dall'aver già creato (o meno) un'istanza per una data cartella. Implementare come esercizio il Multiton di `Pagamenti`



# Conseguenze Del Singleton

- La classe che è un *Singleton* ha pieno controllo di come e quando i client accedono al valore della sola istanza
- Evita che esistano variabili globali che tengono la sola istanza condivisa
- Permette di controllare il numero di istanze create in un programma, facilmente ed in un solo punto
- La soluzione è più flessibile rispetto a quella di usare static per tutte le operazioni e le variabili, poiché si può cambiare facilmente il numero di istanze consentite
- L'unico frammento di codice da modificare quando si vuol variare il numero di istanze create è quello della classe che è *Singleton*, mentre usando `static` si dovrebbero modificare tutte le invocazioni