# Metodologie di Programmazione
# Primo Esonero

docenti: Ivano Salvo, Flavio Chierichetti
Sapienza Università di Roma, 16 aprile 2014

## Parte Prima

Le soluzioni corrette a questa parte garantiscono la sufficienza. Sufficienza abbondante se si svolgono anche i punti contrassegnati da ★.

**Esercizio 1.1:** Considerare le seguenti definizioni di classi e sottoclassi:

```
class F{
   private int f;
   public F(){f=2;}
   public F(int x){ f=x; }

   public String toString(){
      return myClass()+whoAmI();}
   public String whoAmI(){return " "+f;}
   public String myClass(){return "F:";}
   public void m(){ n(); f++;}
   public void n(){ f=f*2;}
   public void print(){
      System.out.println(this);}
   }

class S extends F{
   private int s;
   public S(int x,int y){super(x); s=y;}

   public String whoAmI(){
      return "("+super.whoAmI()+", "+s+")";
      }
   public String myClass(){
      return super.myClass()+"S:";
      }
   public void n(){s=s*2;}
   }

class D extends F{
   public D(){ super();}
   public String myClass(){
     return super.myClass()+"D:";
     }
   }

public class Esonero{
  public static void main(String[] args){
      /*p0*/ F f1 = new F(); f1.print();
      /*p1*/ f1.m(); f1.print();
      /*p2*/ F f2 = new S(1,1); f2.print();
      /*p3*/ f2.m(); f2.print();
      /*p4*/ F f3 = new D(); f3.print();
      /*p5*/ f3.m(); f3.print();
      /*p6*/ S s1 = new S(); s1.print();
      /*p7*/ D d1 = new D(); d1.print();
      /*p8*/ d1.m(); d1.print();
      /*p9*/ D d2 = new S(2,2); d2.print();
     }
   }
```

Scrivete l'output prodotto dal `main`, nei punti `p0`...`p9` tenendo conto che alcune istruzioni potrebbero causare un errore in compilazione o in esecuzione (se un'istruzione causa un errore, immaginate venga poi rimossa al fine di valutare gli output delle istruzioni successive). Motivate le risposte.

**Esercizio 1.2:** Scrivere un metodo statico

`int multiplo(int m, int n)`

che termina restituendo un numero intero $k$ se $m = kn$ per qualche $k \in \mathbb{N}$ e non termina altrimenti.

Dare una versione iterativa e/o una ★ricorsiva.

**Esercizio 1.3:** Considerare il seguente metodo statico:

```
public static void m(int [] a){
    int l=0;
    int r=a.length-1;
    while (l<r){
         int h=a[l];
         a[l]=a[r];
         a[r]=h;
         l++; r--;
    }
}
```

Rispondere alle seguenti domande, motivando brevemente le risposte:

1. Mostrare due sequenze di esecuzione del metodo (prendendo una volta un vettore di lunghezza 4 e un'altra un vettore di lunghezza 5), mostrando ad ogni iterazione il contenuto del vettore `a` e i valori delle variabili `l` ed `r`.
2. cosa calcola il metodo `m`? Scrivere precondizioni e postcondizioni.
3. il ciclo `while` termina sempre? In caso di risposta affermativa, scrivere una funzione di terminazione.
4. ★Scrivere una funzione ricorsiva (proibito qualsiasi costrutto iterativo!) che calcola la stessa funzione (aiutandosi, eventualmente con funzioni ausiliarie con parametri ausiliari).

## Parte Seconda

**Esercizio 2.1:** Supponete che il vostro esecutore abbia come unica funzione aritmetica la seguente funzione (definita diciamo in una classe `MyMath`):

```
public static void boolean
       scomponi(int n, IntPair d){
/* REQ: n>0.
 * EFF: se n e' primo ritorna false
 *      altrimenti torna true e carica in d
 *      due fattori arbitrari di n
 *      tali che d.fat1*d.fat2=n
 *      & d.fat1>1 & d.fat2>1.
 */
```

1. Date la definizione della classe `IntPair`.
2. Scrivete una funzione *ricorsiva* che, usando `scomponi` calcoli il *maggiore* fattore *primo* di un numero intero positivo $n$.
3. Date una versione *iterativa* della stessa funzione. [SUGG: aiutatevi con un vettore].

Corredare possibilmente il codice di opportune asserzioni logiche.

**Esercizo 2.2:** Supponete che le uniche abilità aritmetiche del vostro esecutore siano definite da una classe `Nat` che rappresenta i numeri naturali come *oggetti immutabili*, definendo le seguenti operazioni sui numeri naturali (diamo solo il prototipo dei metodi):

```
class Nat{
   ...
   public Nat piu(Nat n){...}
   public Nat meno(Nat n)
        {/* REQ n<=this */ ...}
   public Nat per(Nat n){...}
   public Nat div(Nat n){...}
   public boolean equals(Nat n){...}
   public boolean minUg(Nat n){...}
   public String toString(){...}
   }
```

Definite la classe `Int` dei numeri interi (con segno), rappresentando gli interi come *oggetti mutabili*. Scegliere opportunamente la rappresentazione di un intero. Definire per gli interi le stesse operazioni (con tipi opportuni). previste per i naturali.