# Metodologie di Programmazione
# Primo Appello

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

## Parte Prima

Le soluzioni corrette a questa parte garantiscono la sufficienza.

**Esercizio 1.1: Overloading e Overriding.** Considerate le seguenti definizioni di classi e interfacce:

```
interface I{
    void m(I x);
}

class C implements I{
    public void m(I x){
       System.out.println("m(I:"+x+") in C");
       }
    public void m(C x){
       System.out.println("m(C:"+x+") in C");
       }
    public String toString(){return "C";}
}// end class C

class D extends C{
    public void m(I x){
       System.out.println("m(I:"+x+") in D");
    }
    public void m(D x){
       System.out.println("m(D:"+x+") in D");
    }
    public String toString(){return "D";}
}// end class D
```

```
public class SecondoAppello{
    public static void main(String[] args){
        I ic = new C();
        I id = new D();
        C cc = new C();
        C cd = new D();
        D dd = new D();
        /*p0*/ ic.m(ic);
        /*p1*/ ic.m(cc);
        /*p2*/ ic.m(dd);
        /*p3*/ cc.m(ic);
        /*p4*/ cc.m(cc);
        /*p5*/ cc.m(dd);
        /*p6*/ id.m(ic);
        /*p7*/ id.m(cc);
        /*p8*/ id.m(dd);
        /*p9*/ cd.m(ic);
        /*pA*/ cd.m(cc);
        /*pB*/ cd.m(dd);
        /*pC*/ dd.m(ic);
        /*pD*/ dd.m(cc);
        /*pE*/ dd.m(dd);
    }
}// fine classe SecondoAppello
```

Scrivete l'output prodotto dal `main`, nei punti `p0`...`pE` Motivate brevemente le risposte. Attenzione alla risoluzione statica dell'overloading e a quella dinamica dell'overriding.

**Esercizio 1.2: Programmazione Procedurale.** Scrivere un metodo statico:

```
int radiceIntera(int n, int k)
```

che presi in input due interi positivi $n$ e $k$ restituisca in output la *radice intera k-esima di n*. Più formalmente, il programma deve restituire in output un numero intero $r$, per cui sia verificata la disuguaglianza $r^k \leq n < (r+1)^k$. Si supponga l'esecutore in grado di calcolare le operazioni aritmetiche $+, -, \times, :$.

ESEMPI: Presi in input 11 e 2, deve essere restituito come risultato 3, in quanto $3^2 = 9 \leq 11 < 4^2 = 16$.

Presi in input 27 e 3, deve essere restituito come risultato 3, in quanto $3^3 = 27 \leq 27 < 4^3 = 81$.

Corredare il metodo di opportune precondizioni, postcondizioni e invarianti di eventuali cicli.

**Esercizio 1.3: Programmazione sui Generics ed Eccezioni.** Supponete di arricchire l'implementazione delle liste generiche viste a lezione con un metodo:

```
A ennesimo(int n)
```

che restituisce l'$n$-esimo elemento della lista ricevente. Dare una implementazione ricorsiva del metodo scrivendone il codice per le liste vuote e le liste non vuote. Sollevare eccezioni quando opportuno.

# Parte Seconda

**Esercizio 2.1: Programmazione coi Generics.** Supponete di avere un'implementazione immutabile delle liste generiche `List<A>` che vi permette di scorrere una lista usando i metodi `A head() throws EmptyListException` e `List<A> tail() throws EmptyListException` per ottenere, rispettivamente, il primo elemento e la coda della lista. Per generare nuove liste, disponete di un costruttore senza parametri che crea una lista vuota e di un metodo `void add(A a)` che aggiunge l'elemento `a` in testa alla lista che riceve l'invocazione del metodo.

Scrivere un metodo statico generico `massimiLocali` che riceve come parametro di ingresso una lista `l` e produce in output la lista contiene tutti i massimi locali di `l`. Un massimo locale è un elemento che è maggiore del precedente e del seguente (solo del seguente se è il primo, solo del precedente se è l'ultimo).

**Esercizo 2.2: Progettazione di Classi Generiche.** Supponendo di avere a disposizione un esecutore JAVA che abbia gli stack come *unico* tipo di dato (con le usuali operazioni `A top()`, `void pop()`, `void push(A a)` e `boolean isEmpty()`, definire una classe che implementa la struttura dati coda, con le usuali operazioni `void enqueue(A a)` e `A dequeue()`. [SUGG: usare 2 stack per simulare la coda, usandoli opportunamente].