## Linguaggi di Programmazione (III Anno), Giugno 1998 - OOP in Java

### Esercizio OOP.1

```
class persona {
private persone f;   /* figli */
public persona(persona padre, persona madre)
  { f=nil;
    padre.f=new persone(this,padre.f);
    madre.f=new persone(this,madre.f); }
public persone figli() { return(f); }
}
class persone {
private persona x; /* primo */
private persone X; /* altri */
public persone(persona primo, persone altri) { x=primo; X=altri; }
public persona primo() {return(x); }
public persone altri() {return(X); }
}
```

### Esercizio OOP.1

```
class Segmento implements SegmentoInt {
private float lx;
public Segmento(float x) { lx=x; }
public float lughezza(); { return(lx); }
}
class Rettangolo extends Segmento implements RettangoloInt {
private float ly;
public Rettangolo(float x, float y) { super(x); ly=y; }
public float area(); { return(lunghezza() * ly); }
}
class Parallelepipedo extends Rettangolo implements ParallelepipedoInt {
private float lz;
public Parallelepipedo(float x, float y, float y) { super(x,y); lz=z; }
public float volume(); { return(area() * lz); }
}
```

In alternativa si potrebbe dichiarare $lx$, $ly$ e $lz$ protected.

## Linguaggi di Programmazione (III Anno), Giugno 1998 - FP

### Esercizio FP.1

1. $\lambda y, x.y\ (y\ x)$ ha tipo piu' generale $(X \to X) \to X \to X$
2. $\lambda f.cons(nil, f\ nil)$ ha tipo piu' generale $(X\ list \to Y\ list\ list) \to Y\ list\ list$

dove $cons : \forall X.X \to X\ list \to X\ list$ e $nil : \forall X.X\ list$.

### Esercizio FP.2

```
fun insert x nil = [x]
  | insert x (y::l) = if x=y then (y::l) else y::(insert x l);
fun delete x nil = nil
  | delete x (y::l) = if x=y then l else y::(delete x l);
fun union nil l2 = l2
  | union (x::l1) l2 = insert x (union l1 l2);

datatype 'X Term = var of 'X | app of 'X Term * 'X Term | lam of 'X * 'X Term;
fun FV (var x) = [x]
  | FV (lam(x,t)) = delete x (FV t)
  | FV (app(t1,t2)) = union (FV t1) (FV t2);
```

### Esercizio FP.3

```
datatype 'X Btree = leaf of 'X | node of 'X * 'X Btree * 'X Btree;
fun paths (leaf x) = [[x]]
  | paths (node(x,t1,t2)) = let fun f l = x::l
    in (map f (paths t1)) @ (map f (paths t2))
    end;
```