Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Informatica 3
Proff. Ghezzi, Lanzi, Matera e Morzenti
Appello del 19 Settembre 2005
Recupero I Parte

COGNOME E NOME (IN STAMPATELLO)

MATRICOLA

Risolvere i seguenti esercizi, scrivendo le risposte ed eventuali tracce di soluzione negli spazi disponibili.
Non consegnare altri fogli.

Spazio riservato ai docenti

Esercizio 1. Dato il seguente codice, scrivere le istruzioni SIMPLESEM relative alle istruzioni etichettate con @@.

Inoltre rispondere ai seguenti quesiti:

- dire se la dichiarazione di t_coda rappresenta un tipo di dato astratto,
- in caso di risposta affermativa, spiegare i vantaggi dei tipi di dato astratto a partire dalla dichiarazione,
- in caso di risposta negativa, implementare in C++ o Java il tipo di dato astratto equivalente.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int c[20];
  int n;
} t_coda;

t_coda coda;

void init(t_coda *coda)
{
  coda->n = 0; /* @@ */
}

void add(t_coda *coda, int x)
{
  if (coda->n<20)
     coda->c[(coda->n)++] = x; /* @@ */
}

main()
{
  init(&coda);
  coda->n=1;
  coda->c[0]=2;
  add(&coda,3);
  add(&coda,4);
}
```

Soluzione

```
set D[D[0]+3]+20,0

set D[D[0]+3]+D[D[D[0]+3]+20], D[D[0]+4]
set D[D[0]+3]+20, D[D[D[0]+3]+20] + 1
```

t_coda non rappresenta un tipo dato astratto, perchè, come evidenziato nel main la sua rappresentazione può essere manipolata direttamente senza ricorrere alle operazioni init e add.

```
public class TCoda {
   int c[];
   int n;

   public void TCoda() {
      c = new int[20];
      n = 0;
   }
   public void add(int x) {
      c[n++]=x;
   }
}
```

Esercizio 2. Si consideri il seguente programma Python.

```
class A(object):
  def f(self):
    print "Padre"

class B(A):
  def g(self):
    print "Figlio"

def g():
  print "Funzione G"

a = A()
b = B()

a.f() # @@1
b.f() # @@2

a.f=g
a.f() # @@3
b.f() # @@4

a1 = A()
b1 = B()
a1.f()      # @@5
b1.f()      # @@6
```

Illustrare cosa viene stampato nelle istruzioni etichettate con "@@" motivando la risposta.

Soluzione

```
@@1 a.f() stampa "Padre" perchè richiama il metodo f di A
@@2 b.f() stampa "Padre" perchè B eredita da A
@@3 a.f() stampa "Funzione G" perchè nell'istanza a ora il nome f punta alla
funzione globale f
@@4 b.f() stampa nuovamente Padre (b non è stata modificata)
@@5 e @@6 si comportano nuovamente come @@1 e @@2 perchè a.f=g non ha effetto sulla
classe A ma sulla singola istanza a
```

Esercizio 3.
Si definiscano in Java due tipi di thread:

- Il tipo "Conta", che stampa interi da 1 a 10, aspettando 0.5 s ad ogni passo e, prima di terminare, segnala la fine del conteggio.
- Il tipo "Attende", che alla creazione si mette in attesa di un segnale da un thread, fornito alla creazione dell'oggetto, poi termina.

Si definisca poi un programma che faccia partire due thread C e A, rispettivamente di tipo "Conta" e di tipo "Attende", in modo che A si metta in attesa di C.

Soluzione

Per semplicità si sono omessi i blocchi `try-catch`

```
class Conta extends Thread {
	public void run() {
		for (int i = 1; i <= 10; i++) {
			System.out.println(i);
			Thread.sleep(500);
		}
		synchronized (this) {
			notify();
		}
	}
}

class Attende extends Thread {
	Conta conta;

	public Attende(Conta conta) {
		this.conta = conta;
	}

	public void run() {
		System.out.println("Attende inizia e si mette in attesa");
		synchronized (conta) {
			conta.wait();
		}
		System.out.println("Attende termina");
	}
}

class Esame {
	public static void main(String args[]) {
		Object o = new Object();

		Conta conta = new Conta();
		Attende attende = new Attende(conta);

		conta.start();
		attende.start();
	}
}
```

Esercizio 4.
Quesito 1
A Si consideri il seguente frammento di codice Java:

```
void f(List l1, List l2) {
      l1.add(5)
          l2 = l1;
}

public static void main(String[] args) {
      List x = new List();
      List y = new List();

      x.add(1);
      y.add(3);

      f(x,y);

      System.out.println(x);
      System.out.println(y);
}
```

Si calcoli l'output del programma giustificando la risposta.

Soluzione

L'oggetto referenziato da `l1` viene modificato nella funzione `f` attraverso l'istruzione `add` che aggiunge un elemento in fondo alla lista per cui la prima println fornisce il seguente output:

```
[1,5]
```

L'oggetto referenziato da `l2` non viene invece modificato in alcun modo perchè l'istruzione `l2=l1` modifica la reference locale `l2` non l'oggetto esterno puntato da l2. Pertanto, il secondo output sarà:

```
[3]
```

Quesito 2
E' possibile realizzare in Java funzione `void sum(Integer op1, Integer op2, Integer res)` che restituisca in `res` il risultato della somma di `op1` e `op2`?
Nel caso sia possibile implementare la funzione, nel caso non sia possibile spiegare brevemente il problema e trovare un modo per ovviarvi (mantenedo la funzione void e con tre parametri)

Soluzione

No, perchè il tipo di dato Integer non è modificabile (non esiste alcun metodo `setValue`) e, come spiegato al punto precedente, un'istruzione `res = new Integer(...)` non avrebbe il risultato sperato perchè modificherebbe unicamente la reference locale res e non l'oggetto esterno puntato da res.

In C/C++, la soluzione sarebbe di utilizzare la doppia indirettezza, ovvero di passare come argomento alla funzione non il puntatore ad una struttura dati ma il puntatore al puntatore stesso. In Java, però questo non è possibile, perchè il linguaggio non permette di lavorare direttamente coi puntatori. Una possibile soluzione consiste nel creare una struttura dati accessoria che contenga un intero e fornire un metodo che permetta di modificare il dato contenuto.

```
class MyInteger {
    Integer value;

    Integer getValue() { return value; }
    Void setValue(Integer value) { this.value = value; }
}
```

La funzione verrebbe quindi implementata nel modo seguente

```
void f(MyInteger op1, MyInteger op2, MyInteger res) {
   int sum = op1.getValue().intValue() + op2.getValue().intValue();
   res.setValue(new Integer(res));
   return res;
}
```

In questo caso, la funzione ha successo perchè non viene modificata la reference locale ma viene invece modificato l'oggetto esterno tramite l'operazione `setValue` (esattamente come avveniva per la lista con il metodo `add` nel Quesito 1).