PROVA SCRITTA

Corso di Programmazione in Rete e Laboratorio
a.a. 2002/2003 - 7 aprile 2003

1. Data la seguente classe:

```
public class CC {
private int saldo = 0;

    public void prelievo(int val) {
        saldo -= val;
    }
    public void versamento(int val) {
        saldo += val;
    }
    public void trasferimento(int val, CC altroCC) {
        prelievo(val);
        altroCC.versamento(val);
    }
    public int getSaldo() {
        return saldo;
    }
}
```

   a) Definire una classe `CCNumOperazioni` che estende la classe `CC`, introducendo un nuovo campo `numeroOperazioni` e ridefinendo le operazioni in modo che per ogni conto vengano conteggiate le operazioni effettuate sul conto stesso.
   b) Definire una classe `ProvaContoCorrente` che contiene un metodo `main` in cui si crea 10 conti correnti, 4 di tipo `CC` e 6 di tipo `CCNumOperazioni` memorizzati in un array `contiCorrentiBanca`.
   c) Siano `cc1` e `cc2` due distinti oggetti di tipo `CCNumOperazioni` con `saldo` 20 e 10, e `numeroOperazioni` 3 e 5, rispettivmente. Si rappresenti l'evoluzione dello stack di esecuzione e dello heap in memoria per la chiamata `cc1.trasferimento(5, cc2)`.

2. Modificare la classe `CC` dell'esercizio 1 in modo che i metodi `prelevamento`, `versamento` e `trasferimento` lancino una eccezione di tipo `IllegalArgumentException` nel caso che il parametro formale `val` sia inizializzato con un valore negativo o che il saldo del conto non sia sufficiente per effettuare quella data operazione. Si scrivano degli esempi che mostrano casi in cui verrebbero sollevate delle eccezioni.

3. Si consideri la seguente classe che implementa un conto bancario:

```
public class CC {
   private int saldo = 0;

   public void prelievo(int val) {
      int temp = saldo;
      temp = temp - val;
      saldo = temp;
   }
   public void versamento(int val) {
      int temp = saldo;
      temp = temp + val;
      saldo = temp;
   }
   public void trasferimento(int val, CC altroCC) {
      prelievo(val);
      altroCC.versamento(val);
   }
}
```

   a) Spiegare perché questa implementazione non è corretta in presenza di più thread che eseguono delle operazioni sullo stesso conto.
   b) Come si può modificare la definizione della classe `CC` in modo da eliminare i problemi del punto precedente?
   c) Modificare ulteriormente la definizione della classe `CC` in modo che un thread possa effettuare un prelievo solo se il saldo non è minore della cifra da prelevare (nota: il thread deve quindi essere messo in attesa per evitare soluzioni non desiderate).

4. Sia `C` una classe (bottone, finestra, ...) i cui oggetti possono generare eventi di tipo **`xxx`**. Spiegare cosa sono:

   - la classe **`xxxEvent`**
   - l'interfaccia **`xxxListener`**
   - la classe **`xxxAdapter`**
   - i metodi **`addxxxListener`** e **`removexxxListener`**

5. Con riferimento all'esercizio 3, modificare opportunamente la classe `CC` in modo da permettere la memorizzazione e la lettura da un file di oggetti della classe stessa. In particolare di definisca un metodo statico `salva` che memorizza un oggetto di tipo `CC` passato come parametro del metodo stesso ed un metodo statico `leggi` che restituisce un oggetto di tipo `CC` letto dal file. Per semplicità si assuma che i metodi `salva` e `leggi` abbiano come parametro un oggetto di tipo `java.io.ObjectOutputStream` e `java.io.ObjectInputStream` (rispettivamente) già inizializzato.