Programmazione in rete e laboratorio

Scritto - 6 settembre 2005

1) Data la seguente classe:

```
public class Eccezioni
{   public static int m(int i)
    {   try
        {   if (i <= 0) throw new Ecc1();
            if (i <= 1) throw new Ecc2();
            if (i <= 2) throw new Ecc3();
        }
        catch(Ecc1 e) {i = 10;}
        catch(Ecc3 e) {i = 30;}
        i = i + 10;
        return i;
    }

    public static void main(String[] args)
    {   int i = ...; int k;
        try
        {   k = m(i);
        }
        catch(Ecc1 e) {k = 100;}
        catch(Ecc2 e) {k = 200;}
        System.out.println(k);
    }
}
```

si considerino le esecuzioni in cui alla variabile **`i`** del main vengono assegnati i valori: 0, 1, 2 e 3, descrivendo i passi principali dell'esecuzione, dicendo in particolare se il programma termina con errore o regolarmente, e, in quest'ultimo caso, che valore stampa. (**Ecc1**, **Ecc2** e **Ecc3** sono tre tipi di eccezioni.)

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

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

   public void prelievo(int val) {
      saldo -= val;
   }
   public void versamento(int val) {
      saldo += val;
   }
   public int getSaldo() {
      return saldo;
   }
}
```

a) Spiegare perché questa implementazione non è corretta in presenza di più *thread* che eseguono delle operazioni sullo stesso conto.
b) Modificare la definizione della classe **ContoBancario** in modo da eliminare i problemi del punto precedente, facendo anche in modo che un *thread* possa effettuare un prelievo solo se il saldo non è minore della cifra da prelevare (altrimenti il *thread* dovrà attendere che il saldo sia sufficiente).

3. Si consideri la seguente classe che definisce una finestra contenente una JLabel:

```
public class Finestra extends JFrame
{  private JLabel display;
   private ContoBancario cb;

   public Finestra(ContoBancario conto) {
      cb = conto;
      display = new JLabel();
      getContentPane().add(display);
   }
}
```

Completare la definizione di questa classe **Finestra** e della classe **ContoBancario** dell'esercizio precedente, in modo che il campo *display* contenga sempre il valore aggiornato del saldo del conto *cb*, utilizzando la metodologia *observer-observable*.
Si ricorda che **Observer** è una *interfaccia* che contiene il metodo **update(Observable ob, Object extra_arg)**, mentre **Observable** è una *classe astratta* che fornisce i metodi **addObserver (Observer o)**, **setChanged()** e **notifyObservers()**.

4. Supponendo che un *thread* t sia in possesso del *lock* di un oggetto e che ci siano altri thread in attesa di acquisire il lock ed altri ancora in *wait*, spiegare che cosa avviene quando il *thread* t esegue una delle seguenti azioni:

a) esegue *notify*
b) esegue *notifyAll*
c) si mette in *wait*
d) esegue la *sleep*