Programmazione a Oggetti
Metodologie di Programmazione
14 Maggio 2012

Nome: _____ Matricola: _____

Metodologia di Programmazione [ ]
Programmazione a Oggetti [ ]

Istruzioni

- Scrivete il vostro nome sul primo foglio.
- Indicare se la vostra prova è riferita all'esame MP o PO
- Scrivete le soluzioni nello spazio riservato a ciascun esercizio.
- **Gli esercizi 9 e 10 sono rivolti a coloro che non hanno svolto le esercitazioni, ovvero le hanno svolte ottenendo una valutazione insufficiente / insoddisfacente. Per chiunque scelga di svolgerli, il punteggio su questi esercizi cancella il punteggio ottenuto nelle esercitazioni.**

LASCIATE IN BIANCO:

1	2	3	4	5	6	7	8	9	10	TOT

1 Esercizio

Correggere l'errore nel seguente codice Java: [1,5pt]

```
class Base
{
      private int b;
      public Base(int b) { this.b = b; }
}

class Derived extends Base
{


}
```

Completate la definizione della classe `Derived` e del metodo `main()` nella classe `Test` in modo che l'esecuzione restituisca l'output `"AB"`. **NB: non potete utilizzare comandi di stampa.** [1,5pt]

```
abstract class Base
{
      public Base() { System.out.print("A"); }
}

class Derived extends Base
{
      int a = ... ;

      int m()
      {
        System.out.print("B");
        return 0;
      }
}

class Test
{
      public static void main(String[] args)
      {
        ...

      }
}
```

2 Esercizio

Implementare il metodo `equals` per la classe `Nil` definita qui di seguito. [2pt]

```
class Nil
{




}
```

Implementare il metodo `equals` per la realizzazione della classe `Nil` definita qui di seguito. [2pt]

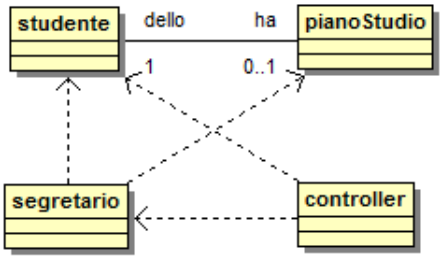
```
class Nil
{
    private static Nil singleton = new Nil();
    public Nil getSingleton(){ return singleton; }




}
```

3 Esercizio

Illustrate con un esempio, il concetto di polimorfismo in Java. (5 righe max) [3pt]

4 Esercizio

Un'applicazione desktop permette a un utente segretario la stampa di un piano di studi relativo a uno studente. L'implementazione di tale funzionalità è descritta qui di seguito mediante un diagramma UML ed il corrispondente schema di classi Java.




```
class Studente
{
   public PianoDiStudio getPiano(){...}
   ...
}

public class PianoDiStudio
{
   public String getDescription(){...}
   ...
}

public class Segretario
{
  public void printPianoStudio(Studente s)
  {
    PianoDiStudio pds = s.getPiano();
    System.out.println("Piano di Studio per "+s.getName());
    System.out.println( pds.getDescription() );
    ...
  }
}

public class Controller
{
   public void onEvent(Event e)
   {
      ...
      utente.printPianoStudio(s);
      ...
   }
}
```

Modificare il codice in modo da implementare il pattern Information Expert. [2pt]
Disegnare il nuovo diagramma delle classi. [2pt]

5 Esercizio

Considerate la seguente classe:

```
public class Giocatore
{
     public String getNome()  { return nome;    }

     private String nome;
}
```

Completate il codice della classe `TabelloneMonopoli` descritta qui di seguito, utilizzando la struttura dati che ritenete più opportuna. [3pt]

```
public class TabelloneMonopoli
{
 // inizialmente le caselle sono tutte senza proprietario
 // le caselle sono indicate da un intero da 0 a 39.

     // il giocatore g diviene proprietario della casella
     public void compra(Giocatore g, int casella)
     {
      ......................................
      ......................................

     }
     // il giocatore g vende la casella se la possiede
     // altrimenti non fa niente e ritorna false
     public boolean vendi(Giocatore g, int casella)
     {
       ......................................
       ......................................

     }
     // ritorna il giocatore proprietario della casella
     // null se la casella non ha proprietario
     public Giocatore getProprietario (int casella)
     {
       ......................................
       ......................................

     }

     private ............... tabellone =  ..................;
}
```

6 Esercizio

Vogliamo definire un sistema di classi e interfacce che rappresentino le relazioni tra le componenti di un circuito elettronico descritte qui di seguito:

- una *resistenza* è una componente che regola la quantità di corrente da cui è attraversata;
- un *condensatore* è una componente che immagazzina carica;
- un *transistor* è un amplificatore di segnale elettrico;
- una *scheda* contiene una collezione di componenti.

Dato il seguente diagramma UML che rappresenta una possibile soluzione del problema, scrivete il corrispondente codice Java che rappresenti le relazioni del diagramma. [3pt]

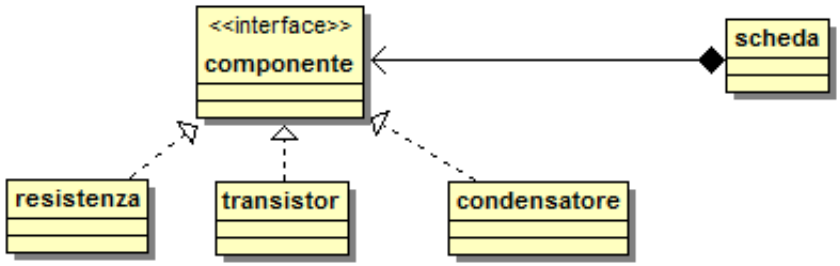


Figure 1: Diagramma delle classi

7 Esercizio

Si implementi il seguente diagramma delle classi in modo che:

- l'invocazione del metodo `impostaSuoneria` imposti come suoneria la suoneria passata come parametro.
- l'invocazione del metodo `onKeyPressed` comporti l'alternanza tra `radio` e `allarme` come impostazione della suoneria.
- l'invocazione del metodo `onTimer` comporti l'attivazione della suoneria (invocazione del metodo `suona` della suoneria impostata).
- la `radio` quando attivata stampi nello standard output `"musica"`.
- l'`allarme` quando attivato stampi nello standard output `"bee bee"`.
- il metodo `alterna` di `suoneria` per ogni sua implementazione cambi l'impostazione della sveglia passata come parametro, impostando l'altra suoneria (se chiamato su `allarme` imposta `radio`, viceversa se chiamato su `radio` imposta `allarme`).
- alla creazione la sveglia sia impostata su `allarme`.

[3pt]

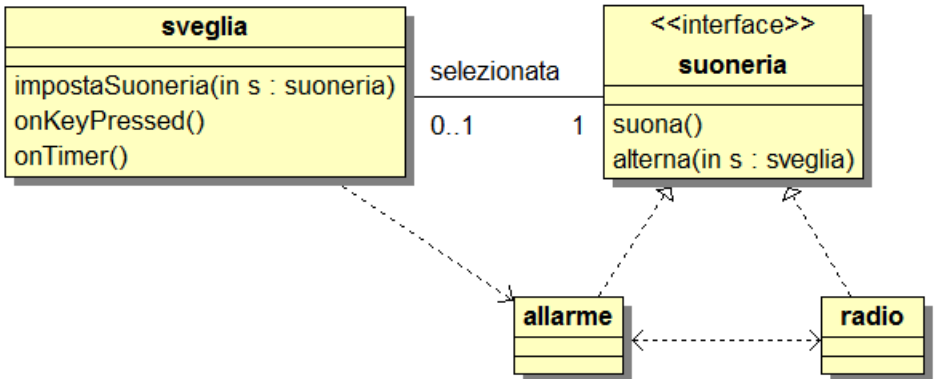


Figure 2: Diagramma delle classi

8 Esercizio

Data l'interfaccia lista:

```
interface Lista<T> {

    boolean isEmpty();
    T getFirst();
    Lista<T> getTail();

}
```

Implementare l'interfaccia Lista usando il pattern NullObject.

[3pt]

9 Esercizio

Con riferimento all'interfaccia `Lista` dell'Esercizio 9, implementate il metodo `public Iterator<T> iterator()` in modo che restituisca un iteratore che visita tutti gli elementi della lista nell'ordine naturale. [3pt]

10 Esercizio

Data la definizione alternativa di Lista:

```
interface ListaM<T>{

    T getFirst();
    void addFirst(T t);
    T removeFirst();

}
```

Mostrare perché non è possibile applicare il pattern NullObject. [3pt]