PROGRAMMAZIONE II
Seconda Valutazione Intermedia 29 Maggio 2013

Es1. Si consideri la specifica parziale di una gerarchia di classi:

```
class Set {
  private int size \\numero degli elementi
  :
  public void add(Object o){
  //Require: true
  //Effect: this.contains(o) = true
  }

  public boolean contains(Object o){
  }
}

class BSet extends Set{
  private int size \\numero degli elementi
  private int capacity \\numero max elementi
  :
  public void add(Object o){
  //Require: this.size < this.capacity
  //Effect: this.contains(o) = true
  }

  public boolean contains(Object o){
  }
}
```

La specifica della gerarchia di classi soddisfa il principio di sostituzione? Si motivi la risposta tramite un esempio di uso.

Es2 Consideriamo la dichiarazione di classe:

```
public class Interval {
  private final Date start, stop;
  private final long duration;

  public Interval(Date start, Date stop) {
  this.start = start;  this.stop = stop;
  duration = stop.getTime()-start.getTime();
  }
  public Date getStart() { return start; }
  public Date getStop() { return stop; }
  public long getDuration() { return duration; }
 }
```

Supponiamo che la classe `Date` sia una API di sistema utilizzate per descrivere oggetti che simulano il passare del tempo (con gli opportuni metodi).

1. Definire l'invariante di rappresentazione della classe `Interval`.
2. Mostrare, tramite un esempio, che la classe `Interval` espone la rappresentazione.
3. Modificare la definizione della classe `Interval` in modo da evitare il problema dell'esposizione della rappresentazione.

Es3 Consideriamo il tipo di dato astratto `pSet` per rappresentare un contenitore di oggetti di tipo `P`. Supponiamo che `P` sia un tipo predefinito in cui il contenuto informativo dell'oggetto e' identificato univocamente dal valore di una chiave (di tipo string). Il tipo di dato astratto `pSet` deve fornire *almeno* due metodi pubblici con l'ovvio significato:

- `public void add(P p)`,
- `public P get(string s)`.

Nel progettare il tipo di dato astratto `pSet` si deve tenere conto dei seguenti vincoli

- Gli oggetti `P` contenuti in `pSet` non sono ripetuti (tutte le chiavi devono essere distinte),
- Il tipo di dato astratto `pSet` deve essere strutturato in modo tale da favorire le operazioni di recupero degli oggetti usati piu' frequentemente.

1. Definire la specifica del tipo di dato astratto `pSet`.
2. Definire le strutture di implementazione del tipo di dato astratto `pSet` individuando la funzione di astrazione e l'invariante di rappresentazione
3. Si fornisca limplementazione del metodo `add` e si dimostri che l'implementazione soddisfa linvariante di rappresentazione e le specifiche

Traccia della soluzione

Es1 La specifica parziale viola la regola dei metodi. In particolare la regola della precondizione

Set.effect $\longrightarrow$ BSet.effect

In generale questa implicazione e' falsa: non vale visto che viene rafforzata la proprieta' in BSet.

Es2

- L'invariante di rappresentazione deve catturare le relazioni tra i valori delle variabili di instanza duration, start, e stop. Una possibile definizione risulta:

  $$c.start! = null \& c.stop ! null \& c.stop.getTime() - c.start.getTime() > 0$$

  questa definizione implica che $duration.getTime() > 0$.

- E' facile per un cliente ottenere, tramite l'invocazione dei metodi pubblici, un riferimento all'oggetto Date. Il riferimento pu' essere utilizzato per modificare i valori dell'oggetto Date riferito (ad esempio il valore di start.getTime() senza modificare il valore della durata).

```
import java.util.*;
public class Interval {
  private final Date start, stop;
  private final long duration;

  public Interval(Date start, Date stop) {
  this.start = start;  this.stop = stop;
  duration = stop.getTime()-start.getTime();
  }
  public Date getStart() { return start; }
  public Date getStop() { return stop; }
  public long getDuration() { return duration; }
    public void PrintStart() { System.out.println(start.toString()); }
 }

import java.io.*;
import java.util.*;
class Msin {

  public static void main(String[] args) {
    long i = 10;
    long t = 15;
    Date Di = new Date(i);
```

```
    Date Dt = new Date(t);
    Interval I = new Interval(Di, Dt);
    I.PrintStart();
    Date D = null;
    D = I.getStart();
    D.setTime(100000005);
    I.PrintStart();
}
}

\item Una soluzione difensiva risulta:

\begin{verbatim}
public class Interval {
//...
public Interval(Date start, Date stop) {
this.start = new Date(start.getTime());
this.stop = new Date(stop.getTime());
duration = stop.getTime() - start.getTime();
}
public Date getStart() { return new Date(start.getTime()); }
public Date getStop() { return new Date(stop.getTime()); }
}
```

Es3

1. Specifica di `public void add(P p)`

```
public void add(P p) throws DuplicateItemException
//Require: p != null
//Effect: if this.IsIn(p) then throw DuplicateItemException else
// Modify(This): this 0 this + P with usage = 0
```

2. Specifica di `public P get(String s)`

```
public P get (String s) throws Empty_pSetException
//Require: s != null
//Effect: if this= null then throw Empty_pSetException
//Effect: if this.contains(s) = Q then Q else null
// Modify(This): Q.usage ++
```

3. Strutture di implementazione

```
public class Info {
P item;
int usage}
//Assunzione: metodi osservatori tradizionali per oggetti
// della classe Info e della classe P

//Implementazione
Vector<Info> els
//Invariante di rappresentazione
I(c) = c,els !=null &
forall i, 0<= i < s.els.size: c.els[i]: Info e' un oggetto della classe Info &
forall i, j, 0<= i < s.els.size, 0<= j < s.els.size, i != j:
         c.els[i].getKey() != c.els[j].getKey() &
forall i, j, 0<= i < s.els.size, 0<= j < s.els.size, i <= j:
         c.els[i].getUsage() >=  c.els[j].getUsage()

//Implementazione di add
basta utilizzare il metodo di vecctor add dopo i controlli di rito

//Implementazione alternativa con liste: si doveva modificare la classe
//Info con un campo next.
```