INGEGNERIA DEL SOFTWARE A.A. 2012-13
PROVA SCRITTA DEL 12 LUGLIO 2013

Esercizio 1

Si consideri la seguente classe Java:

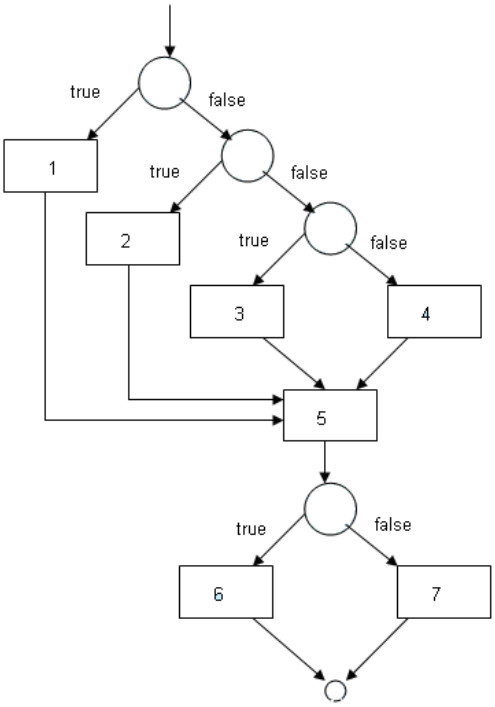
```
public class PuntoNelTempo {
	private int h; // ora
	private int m; // minuto

	public PuntoNelTempo(int ora, int minuto) {
		h = Math.abs(ora) % 24; // Math.abs ritorna il valore assoluto, % calcola il resto
		m = Math.abs(minuto) % 60;
	}

	public int intervallo (PuntoNelTempo p) {
		PuntoNelTempo p1, p2;
		if (this.h > p.h) {
			p1 = p; p2 = this; // 1
		} else if (this.h < p.h) {
			p1 = this; p2 = p; // 2
		} else if (this.m >= p.m){
			p1 = p; p2 = this; // 3
		} else {
			p1 = this; p2 = p; // 4
		}
		int int_h = p2.h - p1.h; // 5
		int int_m = p2.m - p1.m;
		if (int_m >= 0) {
			return int_h * 60 + int_m; // 6
		} else {
			return (int_h - 1) * 60 + (60 + int_m); // 7
		}
	}
}
```

a) Disegnare il control flow graph (CFG) del metodo "**intervallo**"

b) Fornire un insieme (minimo) di valori di input per i casi di test, che garantiscano la copertura di tutti i cammini del metodo intervallo (cioè viene richiesto di soddisfare il **criterio All paths coverage**)

c) Scrivere un **caso di test JUnit** in grado di testare il metodo intervallo nel caso in cui i Punti nel tempo abbiano la stessa ora

Esercizio 1 - Bozza di Soluzione
a)



b) E' possibile coprire tutti i cammini dato che il codice non contiene cicli
Un insieme di valori di input per minimizzare i casi di prova (con i relativi cammini) è il seguente:

this	**P**	**Cammino**
8:40	6:30	(1, 5, 6)
8:40	6:50	(1, 5, 7)
8:40	9:30	(2, 5, 7)
8:40	9:50	(2, 5, 6)
8:40	8:30	(3, 5, 6)
8:40	8:50	(4, 5, 6)

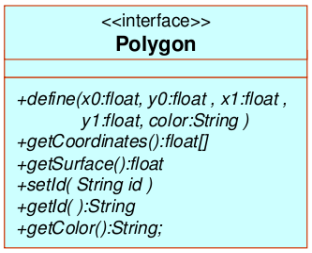
Notare i che i cammini (3, 5, 7) e (4, 5, 7) sono infeasible

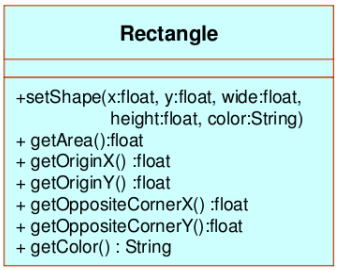
c) JUnit 4.0

```
import static org.junit.Assert.*;
import org.junit.Test;

public class PuntoNelTempoTest {

	@Test
	public void testStessaOra() {
		PuntoNelTempo p1 = new PuntoNelTempo(24,11);
		PuntoNelTempo p2 = new PuntoNelTempo(24,11);
		assertEquals(0, p1.intervallo(p2));
	}

}
```

Esercizio 2

Si vuole sviluppare un'applicazione chiamata GEO_XYZ per lavorare con oggetti geometrici. Questi oggetti saranno gestiti dall'applicazione tramite un'interfaccia particolare (Polygon), che offre un insieme di metodi che gli oggetti grafici devono implementare. A questo punto si ha a disposizione una classe (Rectangle) già implementata che si potrebbe riutilizzare, che però ha un'interfaccia diversa, e che non si vuole modificare.





New interface Available class

Si osservi che le caratteristiche del rettangolo (classe Rectangle) vengono indicate nel metodo **setShape**, che riceve le coordinate del vertice inferiore sinistro, l'altezza (height), la larghezza (wide) e il colore. Invece, l'interfaccia Polygon specifica che la definizione delle caratteristiche della Figura, avviene tramite il metodo chiamato **define**, che riceve le coordinate degli vertici opposti e il colore. Si osservi che il metodo **getCoordinates** dell'interfaccia Polygon restituisce un array contenente le coordinate dei vertici opposti (nel formato x0, y0, x1, y1). Nella classe Rectangle esistono metodi particolari per ricavare ogni singolo valore (**getOriginX**, **getOriginY**, **getOppositeCornerX** e **getOppositeCornerY**). Per quel che riguarda la superficie del rettangolo, in entrambi casi si ha a disposizione un metodo, ma con nome diverso. Si noti, anche, che Polygon aggiunge metodi non mappabili sulla classe Rectangle (**setId** e **getId**), che consentono la gestione di un identificativo tipo String per ogni Figura creata. Infine si noti che il metodo **getColor** è identico nei due casi.

La classe Rectangle è implementata in questo modo

```
public class Rectangle {
	private float x0, y0;
	private float height, width;
	private String color;
	public void setShape(float x, float y, float wide, float height, String color)  {
		x0=x; y0=y; this.height=height; width=wide; this.color=color;
	}
	public float getArea() { return height*width; }
	public float getOriginX() { return x0; }
	public float getOriginY() { return y0; }
	public float getOppositeCornerX() { return x0 + height; }
	public float getOppositeCornerY() { return y0 + width; }
	public String getColor() { return color; }
}
```

a) Quale tipo di design pattern verrà utilizzato per risolvere il problema descritto sopra durante l'implementazione del sistema GEO_XYZ e perchè?

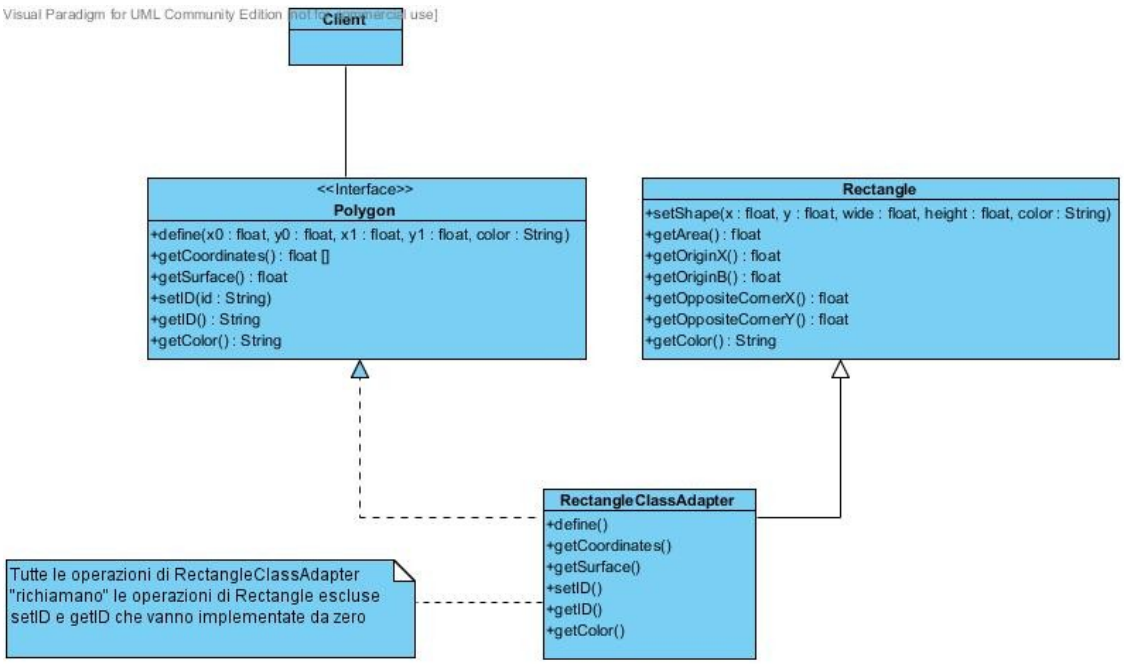
b) Instanziate il design pattern che avete deciso al punto a) creando così un class diagram che sia un utile punto di partenza per implementare la porzione di codice del sistema GEO_XYZ che gestisce gli oggetti geometrici

c) Implementare utilizzando lo pseudocodice (oppure se preferite direttamente il linguaggio Java) le classi più importanti del sistema GEO_XYZ che gestiranno gli oggetti geometrici

Esercizio 2 - Bozza di Soluzione

a) Il design pattern da utilizzare è l'adapter (direi che vanno bene entrambe le alternative: Object adapter e Class adapter) perchè è l'unico che converte un interfaccia di una classe in un altra, che è proprio quello che ci serve in questo caso.

b)

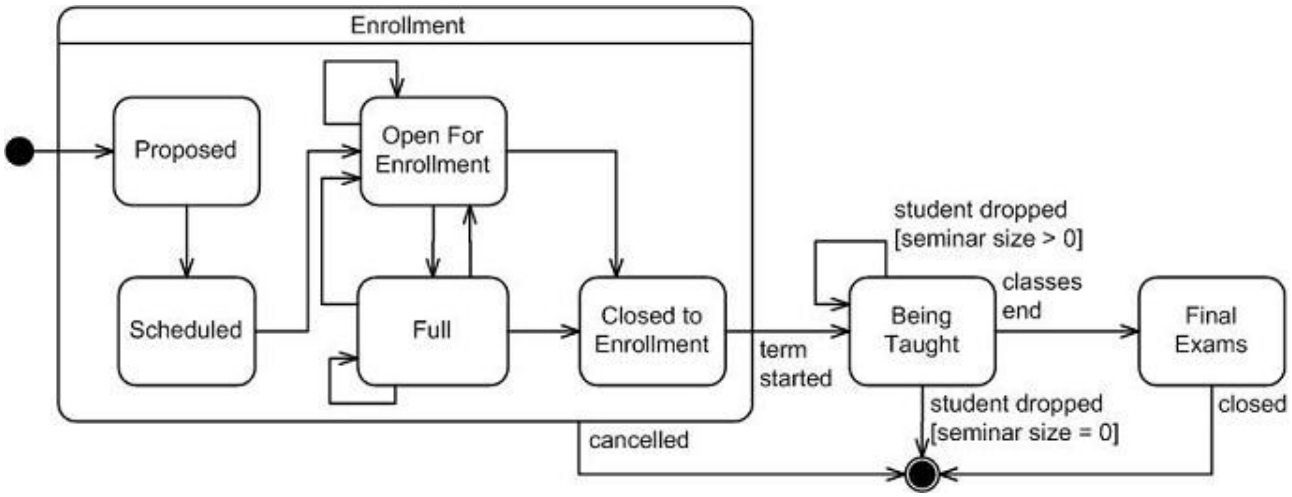


c)

```
public class RectangleClassAdapter extends Rectangle implements Polygon{
        private String name = "NO NAME";

        public void define( float x0, float y0, float x1, float y1,
                        String color ) {
                float a = x1 - x0;
                float l = y1 - y0;
                setShape( x0, y0, a, l, color );
        }

        public float getSurface() {
                return getArea();
        }

        public float[] getCoordinates() {
                float aux[] = new float[4];
                aux[0] = getOriginX();
                aux[1] = getOriginY();
                aux[2] = getOppositeCornerX();
                aux[3] = getOppositeCornerY();
                return aux;
        }

        public void setId( String id ) {
                name = id;
        }

        public String getId( ) {
                return name;
        }
}
```

Esercizio 3

Il seguente diagramma rappresenta il "ciclo di vita" di un seminario per studenti in cui è previsto un esame finale. E' composto dagli stati (alcuni composti/compositi) di: iscrizione al seminario (**Enrollment**), svolgimento del seminario da parte del docente (**Being Taught**) ed esame finale (**Final Exams**).



a) Considerando il diagramma, in quanti modi e perchè un docente termina il seminario durante lo svolgimento?

b) Disegnare un diagramma equivalente a quello mostrato (ovvero che esprime lo stesso comportamento senza perdere informazione) senza fare uso degli stati composti/compositi

c) Completare il diagramma costruito al punto b) con delle transizioni sensate (ove mancanti) mantenendo tutti gli stati del diagramma ed avendo la possibilità di cancellare o aggiungere una o più frecce. **Suggerimento (da seguire)**: quando il seminario è "open for enrollment" e finchè ci sono sedie disponibili allora è possibile aggiungere uno studente al seminario. Quando non ci sono più sedie disponibili il seminario diventa "Full". A questo punto se ci sono ancora richieste gli studenti vengono inseriti in una lista di attesa. Nel momento in cui uno studente già iscritto al seminario si ritira allora il primo della lista di attesa viene iscritto. E' possibile che il seminario ritorni nello stato "open for enrollment" se uno studente si ritira dal seminario e la lista di attesa è vuota.

Esercizio 3 - Bozza di Soluzione

a) Esistono due possibilità. Il seminario finiscse in modo regolare (classes end) e tutti gli studenti abbandonano l'aula (seminar size = 0)

b) E' sufficiente cancellare lo stato composito **Enrollment** ed aggiungere per ogni sottostato di **Enrollment** una transizione **cancelled** che finisce nello stato finale

c)