Ingegneria del software - Esame del 10 febbraio 2004
Soluzione

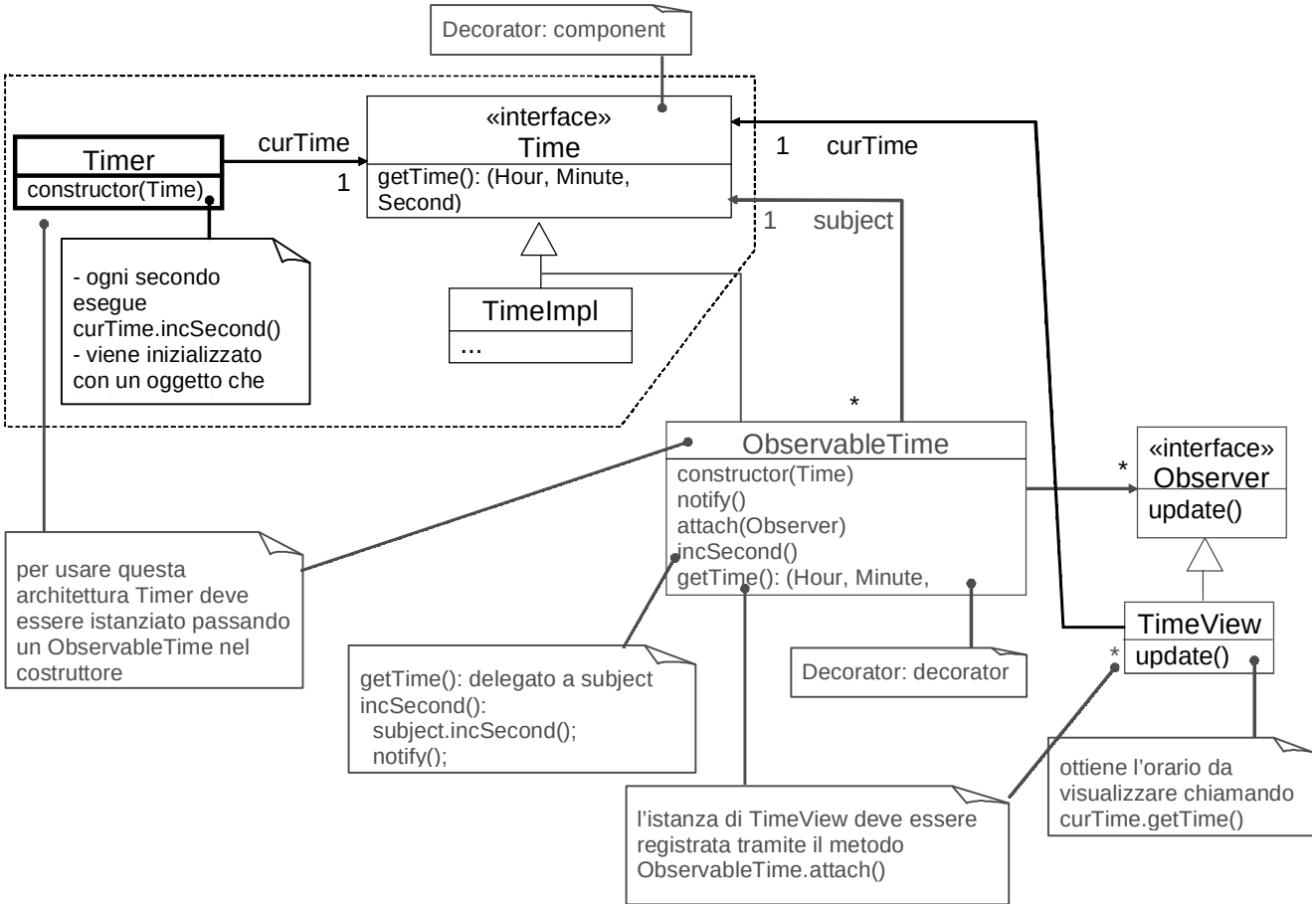
Esercizio 1 (media 6,93 decimi)

Quella che segue è una possibile soluzione dell'esercizio. Questa specifica soluzione è basata sulla libreria JUnit ma tale caratteristica non è richiesta. Le parti più importanti sono i metodi il cui nome inizia per "`test`".

```
/*
 * SimpleJUnitTest.java
 * JUnit based test
 *
 * Created on March 2, 2004, 9:58 PM
 */

import junit.framework.*;

import java.util.LinkedList;

/**
 *
 * @author Valter Crescenzi
 */
public class LinkedListTest extends TestCase {

    private LinkedList list;
    private LinkedList listABC;
    private String[] abc;

    public LinkedListTest(java.lang.String testName) {
        super(testName);
    }

    public static Test suite() {
        TestSuite suite = new TestSuite(LinkedListTest.class);
        return suite;
    }

    public void setUp() {
        this.list = new LinkedList();
        this.listABC = new LinkedList();
        this.abc = new String[] {"a","b","c"};
        for (int i=0; i<abc.length; i++) {
            listABC.add(abc[i]);
        }
    }

    public void testInserimentoSingolo() {
        //Testa inserimento su lista vuota
        Object o = new Object();
        list.add(o);
        assertEquals("Inserimento singolo", o, list.get(0));
    }

    public void testInserimentoInCoda() {
        //Testa inserimento su lista non vuota
        Object o1 = new Object();
        list.add(o1);
```

```
        Object o2 = new Object();
        list.add(o2);
        assertEquals("Inserimento in coda", o2, list.get(1));
        assertEquals(o1, list.get(0));
    }

    public void testInserimentoTriplo() {
        Object o1 = new Object();
        list.add(o1);
        Object o2 = new Object();
        list.add(o2);
        Object o3 = new Object();
        list.add(o3);
        assertEquals("Inserimento in coda da 3", o3, list.get(2));
        assertEquals(o2, list.get(1));
        assertEquals(o1, list.get(0));
    }

    public void testRemoveTesta() {
        assertEquals("Rimozione Testa", listABC.remove(0),abc[0]);
    }
    public void testRemoveMezzo() {
        assertEquals("Rimozione Mezzo", listABC.remove(1),abc[1]);
    }
    public void testRemoveCoda() {
        assertEquals("Rimozione Coda", listABC.remove(2),abc[2]);
    }

    public void testRemoveMultiplo() {
        for(int i=0; i<abc.length; i++) {
            assertEquals("Rimozione Multipla - "+i, listABC.remove(0),abc[i]);
        }
    }

    public void testIsEmpty() {
        assertTrue("Appena creata è vuota",list.isEmpty());
    }

    public void testIsEmptyTrueAndFalse() {
        assertTrue("Appena creata è vuota",list.isEmpty());
        list.add(new Object());
        assertFalse("Un elemento e non è più vuota", list.isEmpty());
    }

    // ... ... ...

    public static void main(java.lang.String[] args) {
        junit.textui.TestRunner.run(suite());
    }

}
```

Esercizio 2 (media 6,36 decimi)



Altre soluzioni sono possibili, ad esempio si può far derivare ObservableTime da TimeImpl e fare overriding del metodo incSecond(). In questo l'applicazione del pattern decorator viene sostituito con una applicazione dell'ereditarietà.

Esercizio 3 (media 8,28 decimi)

Una possibile soluzione è la seguente.

Attori del sistema

primari: agente
secondari: magazzino.

Casi d'uso importanti

I casi d'uso ritenuti più importanti sono i seguenti.

- Inserimento ordine: l'agente inserisce l'ordine mediante il suo calcolatore portatile durante la sua attività presso il cliente
- Upload ordini: l'agente invia gli ordini alla sede centrale
- Evasione ordine: il personale del magazzino esamina gli ordini per l'evasione

Descrizione dettagliata del caso d'uso più importante: Inserimento ordine

Attori:

l'agente

Trigger:

l’agente presso il cliente intende inserire l’ordine

Stackeholders:

il **cliente** vuole che il suo ordine arrivi rapidamente ed in maniera affidabile al magazzino di Distribulibri per l’evasione

l’**agente** vuole poter inserire l’ordine con rapidità e affidabilità in modo da poter soddisfare rapidamente il cliente

il **magazzino** vuole che l’ordine contenga tutti i dati necessari per la sua evasione e che non vengano richiesti libri non in magazzino

la **sede centrale** di Distribulibri vuole che il rapporto tra agente e cliente non sia compromesso dall’introduzione del nuovo sistema ma, anzi, che la sua immagine ne tragga giovamento

Precondizioni:

l’agente è presso il cliente, il cliente ha a disposizione il catalgo per indicare i libri all’agente

Postcondizioni:

tutti i libri che il cliente intende ordinare sono stati registrati nel calcolatore portatile con le loro quantità

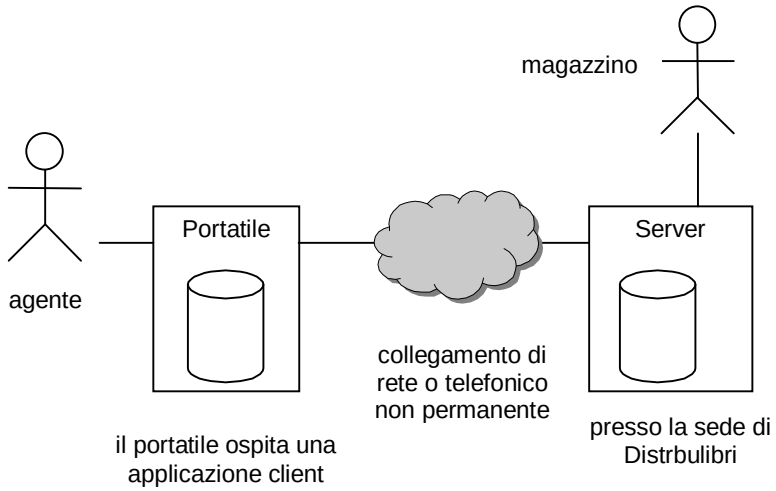
Scenario principale:

1. L’agente specifica la libreria per chi si sta inserendo l’ordine.
   I seguenti passi vengono ripetuti per tutto il libri che il cliente vuole ordinare
2. il cliente comunica il libro che intende ordinare mediante titolo, parte del titolo, autore o codice
3. l’agente fa la ricerca inserendo le informazioni, anche parziali, che ha ricevuto
4. il sistema risponde con uno o più elementi
5. l’agente e il cliente risolvono l’eventuali ambiguità della ricerca scegliendo dalla lista di titoli proposta
6. l’agente conferma e la lista del’ordine viene aggiornata come pure il totale dell’importo dell’ordine

Scenari secondari:

- Un libro può non risultatare nel DB in tal caso il passo 3 ritorna zero elementi e si deve modificare la ricerca (ricominciare da 2).
- Un libro può essere presente nel DB ma non in magazzino. In tal caso si passa al prossimo libro.

Architettura



Va deciso con quale protocollo il portatile debba scambiare informazioni con il server. Probabilmente sia Portatile che Server conterranno un DB. Non è chiaro se l'uso di un application server (esempio Tomcat) sia utile. Infatti, un application server è normalmente usato per l'applicazione web-based ma nel nostro caso sul portatile avremo con alta probabilità una applicazione standard che gestirà il DB locale. Potrebbe essere invece utile scegliere un linguaggio di programmazione che abbia un supporto per la serializzazione di oggetti (es. Java).

Lista dei rischi

Rischio	probabilità	impatto	mitigazione
Il sistema sul portatile è troppo macchinoso da usare per gli agenti	alta	alto	• convincere la Distribulibri a mettere a disposizione il tempo dei suoi agenti • seguire un agente nel suo lavoro
Il protocollo di comunicazione non risulta affidabile e si perdono ordini o arrivano duplicati	media	alto	• test accurato sul protocollo • progettazione con spirito "fault tollerant"
Troppo poco tempo per il testing sul campo	alta	alto	• prevedere una fase di test sul campo prima del collaudo vero e proprio
Basse competenze nel settore librario	alta	medio	• stretto contatto con il personale di Distribulibri

Ulteriori casi d'uso

- Invio ordini (attore principale: l'agente)
- Evasione ordini (attore principale: magazzino)