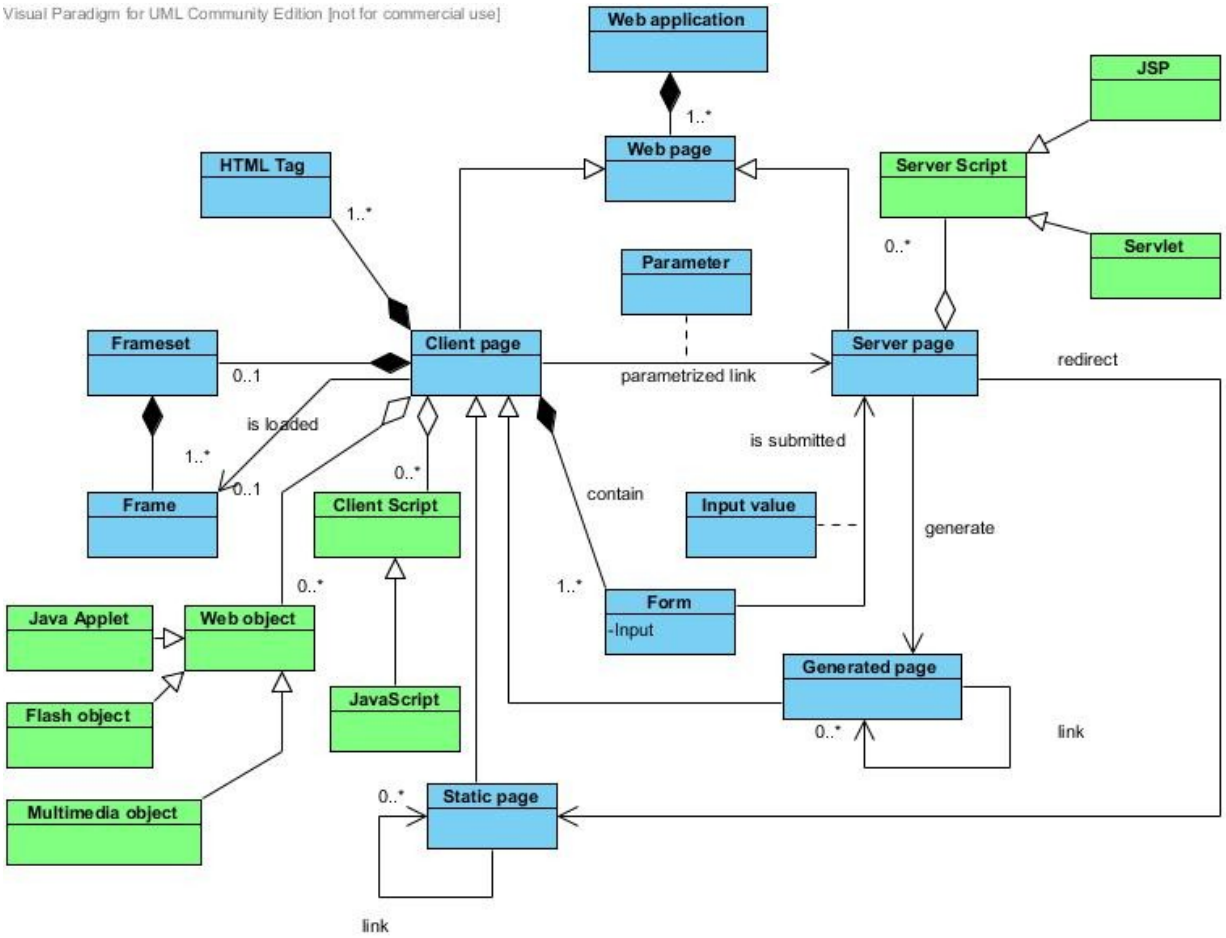
Ingegneria del Software a.a. 2012-13
Prova Scritta del 4 febbraio 2013

Esercizio 1

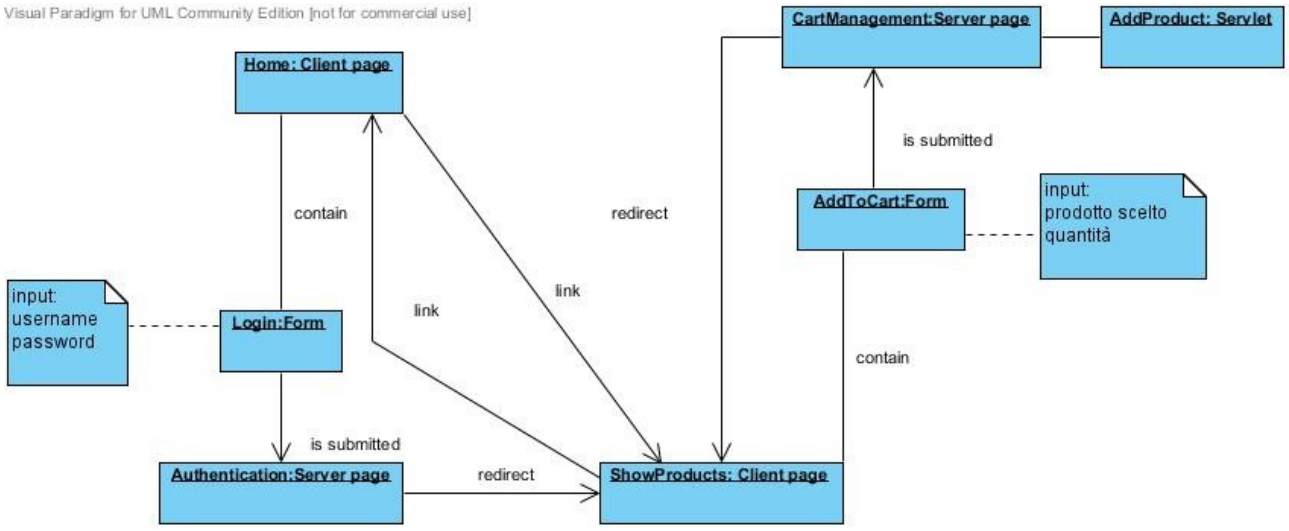
a) Rappresentare il meta-modello di una semplificata Web application con un diagramma delle classi. Una Web application è costituita da Web page che possono essere di due tipi: Client page o Server Page. Una Client page, che è costituita da HTML tag, a sua volta può essere una Static page oppure una Generated page. Una Client page può contenere una o più Form. Quando una Form è "submitted" con degli Input value a una Server page, quest'ultima genera una Generated page. Una Server Page può anche fare una "redirect" su una Static page. Esistono due tipi di hyperlink: normali che collegano le Static page con altre Static page e gli hyperlink con parametri che collegano le Client Page con le Server page. Infine una Client page può contenere un Frameset che è costituito da uno o più Frame nel quale una Client page può essere caricata.

b) Aggiungere al modello i Client script (es. JavaScript), Web object (es. Java applet, Flash object) e gli server script (es. Java server pages, Servlet)

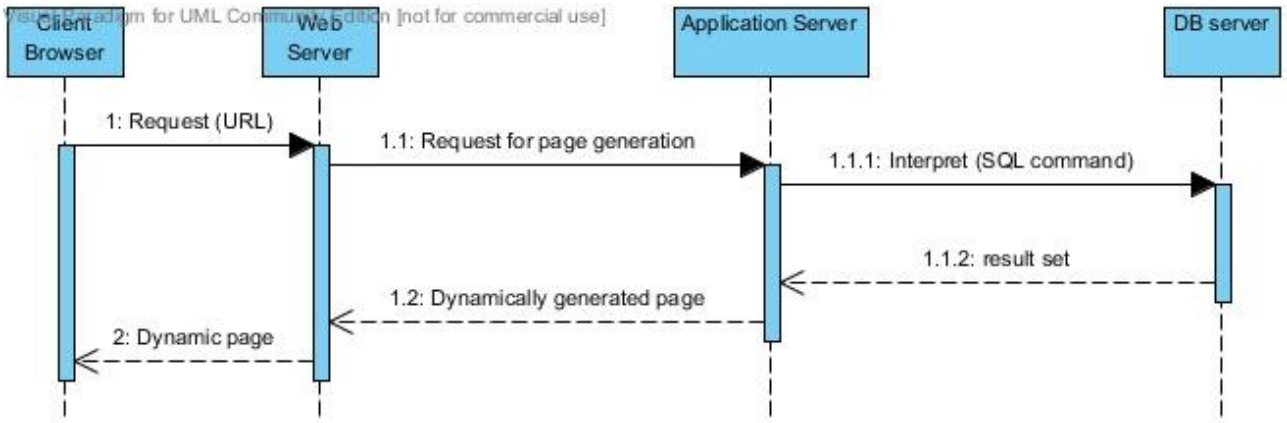
a) in azzurro e b) in verde



c) Instanziare il precedente meta-modello con una Web application di e-commerce avente la Client page "Home" che contiene la Form "Login" con gli usuali Input (username, password). La Form "Login" sottomette i valori alla Server Page "Autentication" che fa una redirect ad un altra Client Page chiamata "ShowProducts". La "Home" è collegata con un link normale con "ShowProducts" che contiene una Form "AddToCart" con due input: prodotto scelto e quantità. La Form "AddToCart" trasmette i valori alla Server Page "CartManagement" che tramite una Servlet (AddProduct) aggiunge il prodotto scelto nel carello e fa una redirect alla pagina "ShowProduct". Da "ShowProduct" è possibile tornare a "Home".



d) (facoltativo) Rappresentare mediante un sequence diagram l'interazione tra Browser, Web server, Application server e DB server quando ad un Web server viene richiesta una pagina Web dinamica (cioè il cui codice HTML è generato dinamicamente da un programma che gira sull'Application Server) che necessita di dati presenti nel DB.



Esercizio 2

Si supponga di dover implementare un sistema "Hospital" nel quale esistono tre attori: Patient, Watcher e Psychologist che si comportano nel seguente modo:

- **Patient**. Riceve un valore intero compreso tra 1 e 10 inclusi. In modo casuale (Math.random()) Patient può decidere di tenere il numero oppure rifiutarlo. Nel caso il numero sia tenuto, Patient setta un suo attributo a quel numero (cioè esegue un cambiamento) e risponde: "I like it". Viceversa, Patient risponde "I am not interested in this number".
- **Watcher**. E' interessato a capire quanti cambiamenti sono stati fatti da Patient. Sostanzialmente Watcher conta il numero di volte che il Patient risponde "I like it"
- **Phychologist**. Cerca di interpretaare il comportamento del Patient. Se il Patient non cambia mai allora risponde "The patient doesn't like to change", se il Patient cambia più spesso numero quando gli vengono proposti numeri piccoli (cioè <= 5) allora risponde "The patient likes small numbers", se invece il Patient cambia più spesso numero quando gli vengono proposti numeri grandi (cioè > 5) allora risponde "The patient likes big numbers". Infine se il Patient cambia in modo uguale indipendentemente dalla grandezza del numero che gli è stato presentato risponde "The patient likes both small and big numbers"

Supponiamo che a Patient vengano mandati 10 valori interi a caso compresi tra 1 e 10 inclusi

Esempio di esecuzione del sistema Hospital:

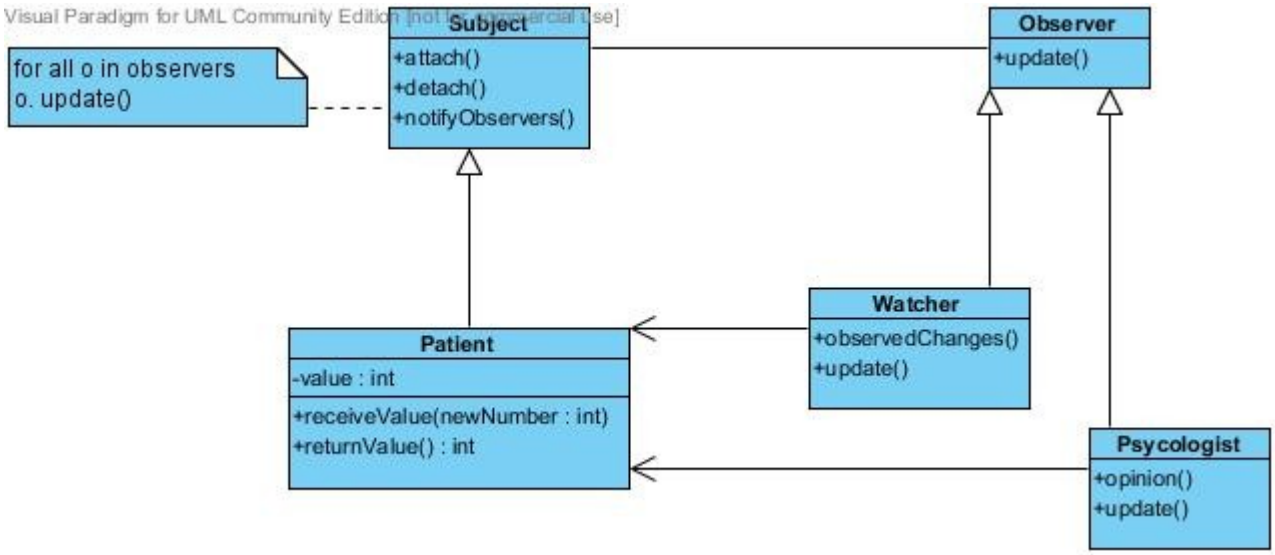
Main: do you like number 3?
Patient: I like it
Main: do you like number 6?
Patient: I like it
Main: do you like number 1?
Patient: I like it
Main: do you like number 8?
Patient: I am not interested in this number
Main: do you like number 3?
Patient: I like it
Main: do you like number 9?
Patient: I am not interested in this number
Main: do you like number 7?
Patient: I am not interested in this number
Main: do you like number 2?
Patient I like it
Main: do you like number 1?
Patient: I like it
Main: do you like number 3?
Patient: I am not interested in this number

Watcher: The Patient has changed number 6 times
Psychologist: The patient likes small numbers

a) Quale tipo di design pattern ritenete sia utile per implementare il sistema Hospital e perchè?

Design pattern Observer, perchè esistono due partecipanti (Watcher e Psychologist) interessati a quello che fa un altro partecipante (Patient). Esiste una dipendenza tra Patient e Watcher e Psychologist; quando Patient cambia stato, Watcher e Psychologist che dipendono da lui sono avvertiti.

b) Instanziate il design pattern che avete deciso al punto a) creando così un class diagram che sia un utile punto di partenza per l'implementazione del sistema Hospital



c) Implementare utilizzando lo pseudocodice (oppure se preferite direttamente il linguaggio Java) le classi più importanti che costituiranno il sistema Hospital

```
public class Patient extends Subject {
        private int value = 0;

        public void receiveValue( int newNumber ) {
                if (Math.random() < .5) {
                        System.out.println( "Subject : I like it, I've changed my
                                + "internal value." );
                        value = newNumber;

                } else
                        System.out.println( "Subject : I have a number "+ value +
                                " now, and I not interested in the number "
                                + newNumber + "." );
                        this.notifyObservers();
                }

        public int returnValue() {
                return value;
        }
}
```

Chiamata sempre!

```
public class Watcher extend Observer {
        private int changes = 0; oldnumber =0

        public void update {}
                If (p.getNumber() <> oldnumber)
                { changes++; oldnumber =p.getNumber();}
        }

        public int observedChanges() {
                return changes;
        }
}
```

```
public class Psychologist extend Observer {
        private int countLower, countHigher = 0; oldnumber =0

        public void update {}
                If (p.getNumber() <> oldnumber){ value = p.getNumber()
                if( value <= 5 )
                        countLower++;
                else
                        countHigher++;
                oldnumber = value; }
        }
        public String opinion() {
                float media;
                if( (countLower + countHigher ) == 0 )
                        return( "The Subject doesn't like changes.");
                else
                        if( countLower > countHigher )
                                return( "The Subject likes little numbers.");
                        else if ( countLower < countHigher )
                                return( "The Subject likes big numbers.");
                        else
                                return( "The Subject likes little numbers and big numbers.")
        }
}
```

```
public class ObserverExample {
        public static void main (String[] args) {
                Patient s = new Patient () ;
                Watcher o = new Watcher();
                Psychologist p = new Psychologist();
                s.addObserver( o );
                s.addObserver( p );
                for(int i=1;i<=10;i++){
                        System.out.println( "Main : Do you like the number " + i +"?" );
                        s.receiveValue( i );
                }

        System.out.println( "The Subject has changed " +
                o.observedChanges()
                        + " times the internal value.");
        System.out.println("The Psychologist opinion is:" + p.opinion() );
        }
}
```

Esercizio 3

Dato il seguente codice Java e considerando di avere giù implementato la classe Order con i seguenti attributi: `private String description; private double price; private int quantity;` e relativi metodi getter rispondere alle seguenti 4 domande (una facoltativa).

```
public class OrderReceipt {
	public List<Order> orders;

	public OrderReceipt(List<Order> orders) {
		this.orders = new ArrayList<Order>(orders);
	}

	public String printReceipt() {
		StringBuilder output = new StringBuilder();

		// print headers
		output.append("======Printing Orders======\n");

		// prints orders
		double salesTax = 0.0;
		double totalAmount = 0.0;
		for (Order order : orders) {
			output.append(order.getDescription());
			output.append('\t');
			output.append(order.getPrice());
			output.append('\t');
			output.append(order.getQuantity());
			output.append('\t');
			output.append(order.getPrice()*order.getQuantity());
			output.append('\n');

			// calculate sales tax @ rate of 10%
			salesTax = salesTax + (order.getPrice()*order.getQuantity()*.10);

			// calculate total amount of order
			totalAmount = totalAmount + (order.getPrice()* order.getQuantity()) +
					(order.getPrice() * order.getQuantity()*.10);
		}

		// prints the state tax
		output.append("Sales Tax").append('\t').append(salesTax);
		output.append('\n');

		// print total amount
		output.append("Total Amount").append('\t').append(totalAmount);
		return output.toString();
	}
}
```

a) Quali casi di test ritenete sia oppurtuno considerare per testare la classe `OrderReceipt`? Spiegare le scelte e fare degli esempi

Bisogna considerare che printReceipt() lavora su una lista per cui ci serviranno almeno tre casi di test: lista vuota, lista con un elemento e lista con più di un elemento. Bisognerà verificare che gli elementi sono stampati nel riepilogo. Inoltre il metodo printReceipt() fa due calcoli: l'iva e l'ammontare totale. Quindi dobbiamo controllare con un caso di test entrambi i calcoli.

b) Scrivere dei casi di test JUnit per testare l'operazione printReceipt() in accordo a quanto scritto al punto a)
Possibile soluzione:

```
public class OrderReceiptTest {

	@Test
	public void printReceiptPrintsHeader() throws Exception {
		List<Order> orders = new ArrayList<Order>();
		OrderReceipt orderReceipt = new OrderReceipt(orders);
		String output = orderReceipt.printReceipt();
		System.out.println(output);
		assertTrue(output.startsWith("======Printing Orders======"));
	}

	@Test
```

```
    public void printReceiptPrintsOrderDetails() {
            List<Order> orders = new ArrayList<Order>();
            orders.add(new Order("Milk", 10.0, 2));
            OrderReceipt orderReceipt = new OrderReceipt(orders);
            String output = orderReceipt.printReceipt();
            System.out.println(output);
            assertTrue(output.contains("Milk\t10.0\t2\t20.0"));
    }

    @Test
    public void printReceiptPrintsTotalSalesTax() {
            List<Order> orders = new ArrayList<Order>();
            orders.add(new Order("Milk", 10.0, 2));
            orders.add(new Order("Chocolate", 20.0, 1));
            OrderReceipt orderReceipt = new OrderReceipt(orders);
            String output = orderReceipt.printReceipt();
            System.out.println(output);
            assertTrue(output.contains("Sales Tax\t4.0"));
    }

    @Test
    public void printReceiptPrintsTotalAmount() {
            List<Order> orders = new ArrayList<Order>();
            orders.add(new Order("Milk", 10.0, 2));
            orders.add(new Order("Chocolate", 20.0, 1));
            OrderReceipt orderReceipt = new OrderReceipt(orders);
            String output = orderReceipt.printReceipt();
            System.out.println(output);
            assertTrue(output.contains("Total Amount\t44.0"));
    }
}
```

c) Il codice presenta alcuni code smells. Quali e dove? Applicare i refactoring che ritenete più opportuni per migliorare il codice

Il codice ha diversi problemi. Il metodo printReceipt() è lungo, poco coeso (calcola due cose e crea la stringa di stampa) e ripete 4 volte il calcolo: order.getPrice()*order.getQuantity(). Inoltre è inutile calcolare l'IVA per ogni ordine visto che è sempre il 10%. Sarebbe molto meglio spostare il calcolo fuori dal ciclo e farlo solo una volta. I refactoring che si possono applicare sono extract Method, replace Temp with Query e anche move Method per spostare la stampa dell'ordine nella classe Ordine.

d) (facoltativo) Quale è la complessità ciclomatica di printReceipt()? Spiegare in modo conciso come è stato svolto il calcolo

La complessità ciclomatica è 2. Si contano i "punti decisionali e le istruzioni iterative" e si aumenta di uno. Nel caso di printReceipt() c'è solo un for.