# Ingegneria del Software a.a. 2014-15
# Prova Scritta del 15 luglio 2015

**Esercizio di sbarramento**

**COGNOME** **NOME** **MATRICOLA**

Rispondere alle seguenti domande. Per ogni domanda, solo una soluzione è corretta. L'esercizio si ritiene superato se si risponde correttamente ad **almeno 6 domande**, la valutazione è di 1 punto per ogni risposta corretta oltre le 6.

**Domanda 1**

In termini di principi OO di progettazione che cosa si può affermare della seguente classe **Car**?

```
public class Car {
    private String make = "Aston Martin";
    private int numberOfWindows = 5;
    public void drive(){
        // do some driving!
    }
    public void brake(){
        // slow down!
    }
    public void stop(){
        // STOP!!!!
    }
    public void printDocument(Document d){
        // print Document!!!!
    }
}
```

a) La classe Car è fortemente accoppiata
b) L'information hiding della classe Car è basso
c) La classe Car ha coesione alta
d) La classe Car ha coesione bassa

---

**Domanda 2**

Quale tra i seguenti *statements* non fa parte del "Agile manifesto"?
a) Gli individui e le interazioni "più che" i processi e gli strumenti
b) I requisiti software "più che" il design di un sistema
c) La collaborazione col cliente "più che" la negoziazione dei contratti
d) Rispondere al cambiamento "più che" seguire un piano

---

**Domanda 3**

In che modo possiamo esprimere i vincoli (cioè le constraint) in un diagramma delle classi UML?

a) Unicamente utilizzando OCL
b) Indicandoli tra parentesi graffe attraverso il linguaggio naturale, un linguaggio di programmazione o OCL
c) Non serve un modo esplicito per aggiungere vincoli aggiuntivi: tutti i vincoli si possono esprimere mediante associazioni, attributi e generalizzazione
d) Non esiste una notazione specifica; l'unica regola è indicarli tra parentesi tonde

---

**Domanda 4**

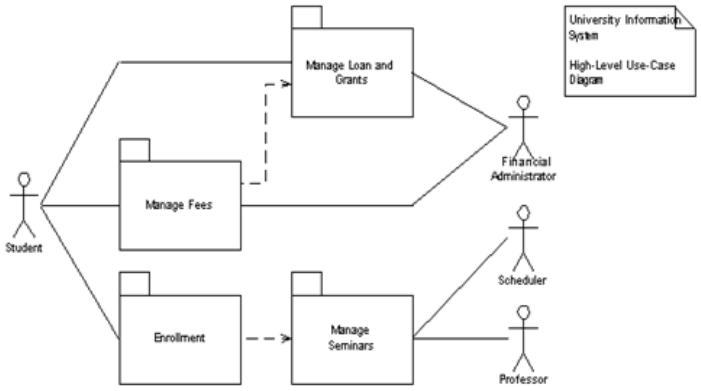
Quale è il tipo di manutenzione a cui è dedicata la percentuale più significativa dell'attività di manutenzione di un sistema software?

a) Correttiva (per "riparare" errori nel software)
b) Adattiva (per adattare il software a diversi ambienti)
c) Migliorativa (per aggiungere o modificare le funzionalità del sistema)
d) Preventiva (per prevenire problemi futuri)

---

**Domanda 5**

Nel seguente package diagram UML cosa non è normativo (o legale)?




a) La freccia tratteggiata
b) La nota posta a destra nel diagramma
c) La presenza degli attori ("omini stilizzati")
d) Il nome dei package (non deve iniziare per una lettera maiuscola)

**Domanda 6**

Quale delle seguenti frasi sul design by contract è falsa?

a) Migliora la documentazione
b) Aiuta nella scelta dell'architettura software
c) Guida il testing
d) Migliora la qualità del software prodotto

**Domanda 7**

Quale tra i seguenti è uno svantaggio dello sviluppo basato sulle componenti?

a) Aumenta la quantità di codice da sviluppare

b) Complica l'architettura software del sistema che si sta sviluppando

c) Riduce la qualità del codice prodotto (per esempio perchè l'utilizzo di componenti open-source non garantisce la correttezza delle funzionalità offerte)

d) Spesso occorre negoziare con il cliente una modifica dei requisiti. In alternativa occorre sviluppare una o più componenti "from scratch"

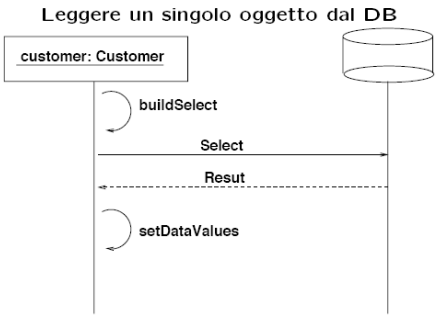
**Domanda 8**

Quale tra le seguenti affermazioni sul Software testing è vera?

a) E' solitamente possibile testare tutte le combinazioni di input/output per un sistema software. Tuttavia questa strategia non viene seguita per ragioni economiche
b) Il test esaustivo è, con sufficiente sforzo e strumenti adatti, adatto per tutto il software
c) Un test è chiamato "negative" quando si utilizzano input illegali con lo scopo di testare le eccezioni oppure sollevare dei failure
d) Lo scopo del testing è dimostrare l'assenza di difetti

**Domanda 9**

Nel contesto della persistenza cosa esemplifica il seguente sequence diagram UML?




a) L'approccio chiamato DAO

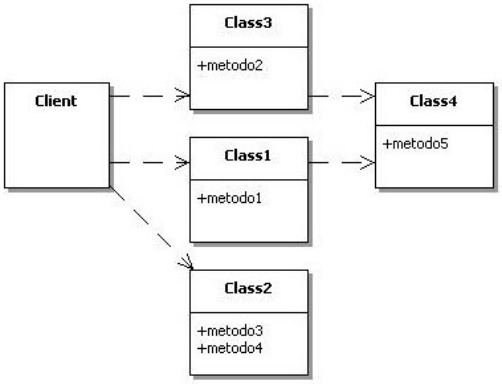
b) L'approccio denominato "Brute force"

c) L'approccio relativo ad un Persistence framework

d) L'approccio TDD

**Domanda 10**

Supponiamo di avere a che fare con una classe *Client* che per realizzare una singola operazione deve accedere ad alcune classi molto differenti tra loro (chiamando i metodi: metodo1, metodo2, metodo3 e metodo4). Durante un operazione di refactoring quale design pattern utilizzeresti per migliorare lo stato delle cose?




a) Façade
b) Adapter
c) Composite
d) Decorator