



# Politecnico di Milano

sede di Cremona

Anno accademico 2015-2016

## Ingegneria del Software - 6 luglio 2016

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

- La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
- Al termine, consegnare solo i fogli distribuiti. Eventuali fogli di brutta, ecc. non verranno in nessun caso presi in considerazione. È possibile scrivere in matita.
- È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
- Non è possibile lasciare l'aula conservando il tema della prova in corso.
- Tempo a disposizione: 2h.

### Esercizio 1 (9 punti)

Una casa d'aste vuole informatizzare la gestione dei diversi lotti e delle aste associate. Ogni lotto è caratterizzato da un nome, una base d'asta, ovvero il prezzo minimo per vendere il lotto, un rialzo minimo e un'ultima offerta. La casa d'aste vuole anche memorizzare le offerte ricevute per ogni lotto dalla prima, che deve essere maggiore o uguale alla base d'asta, all'ultima. La classe `Lotto` mette, quindi a disposizione i seguenti metodi:

- `void offerta(int v) throws OffertaException` consente ai diversi clienti di presentare le loro offerte. La prima offerta deve essere maggiore o uguale alla base d'asta e le successive devono sempre aggiungere almeno il rialzo minimo all'offerta precedente. Il metodo solleva l'eccezione `OffertaException` se l'offerta (`v`) non rispetta i vincoli elencati in precedenza.
  - `int baseDAsta() e int rialzoMinimo()` restituiscono la base d'asta e il rialzo minimo, rispettivamente.
  - `ArrayList<Integer> offerte()` fornisce le offerte fatte fino a quel momento per il lotto.
  - `boolean chiusura()` consente di chiudere l'asta per il lotto. L'esito è positivo (`true`) se è stata presentata almeno un'offerta maggiore o uguale alla base d'asta. Sarà negativo (`false`) in caso contrario.
- Si definiscano le specifiche complete dei metodi `offerta` e `chiusura` in JML.
  - Si ipotizzi una *rappresentazione* per lo stato della classe `Lotto` e si definisca l'invariante di rappresentazione in JML.

### Esercizio 2 (5 punti)

Si progetti un semplice sistema ad oggetti per consentire ad una persona di raggiungere l'aeroporto con diversi mezzi di trasporto. Una persona potrebbe: guidare la propria macchina, prendere un taxi, uno shuttle per l'aeroporto, un autobus, o usare un servizio di limousine. In alcuni casi, si potrebbe pensare anche all'uso della metropolitana e di un elicottero. Tutti questi mezzi di trasporto potrebbero essere usati dalla stessa persona in momenti diversi e tutti consentono alla persona di raggiungere l'aeroporto. La scelta sarà influenzata dai costi, dal tempo richiesto/a disposizione e anche dal confort di viaggio richiesto.

- Si definisca un diagramma delle classi UML, magari sfruttando il design pattern *Strategy*.
- Si scriva la struttura delle classi principali in Java.

### Esercizio 3 (7 punti)

Si scriva un programma Java concorrente (classi e thread) che consenta la gestione di una stazione ferroviaria, con un numero `nBinari` di binari e un numero massimo di treni pari a `nTreni`. Chiaramente, un treno può entrare in stazione solo se esiste almeno un binario libero e lascerà la stazione dopo un tempo variabile. Per semplicità, ogni treno occupa il primo binario utile partendo dal primo.

Si modelli anche il caso in cui la stazione offre 5 binari e 20 treni, in parallelo, cercano di entrare in stazione. Ogni treno aspetterà una quantità di tempo casuale prima di cercare di occupare un binario. Si ricorda che `Math.random()` restituisce un `long` compreso tra 0 e 1.

### Esercizio 4 (7 punti)

Si consideri una classe `Roditore` che ha due sotto-classi `Ratto` e `Topo`. Inoltre, la classe `Topo` ha come sotto-classe `Topolino`. Si consideri il frammento di codice seguente e si identifichino le righe di codice che darebbero un errore in compilazione (spiegare brevemente le risposte):

```
1 Roditore ro;
2 Ratto ra = new Ratto();
3 Topo to = new Topo();
4 Topolino tn = new Topolino();

5 ro = ra;
6 to = tn;
7 tn = new Roditore();
8 ra = new Topolino();

9 Roditore[] array = new Ratto[10];
10 Roditore[] array = new Roditore[10];
11 Ratto[] array = new Topolino[10];
```

### Esercizio 5 (5 punti)

Si consideri il seguente frammento di codice Java:

```
int foo (int x, int y) {
    int z = 0;

    if ((x>0) && (y>0)) {
        z = x;
    }
    return z;
}
```

1. Si disegni il diagramma del flusso di controllo.
2. Si identifichi un insieme minimo di test per coprire:
  - (a) tutti gli statement;
  - (b) tutte le decisioni (*branch*);
  - (c) tutte le condizioni.
3. Cosa cambierebbe se l'espressione `(x>0) && (y>0)` fosse sostituita da `(z>0) || (y>0)`?



# Politecnico di Milano

sede di Cremona

Anno accademico 2015-2016

## Ingegneria del Software - 20 luglio 2016

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. Al termine, consegnare solo i fogli distribuiti. Eventuali fogli di brutta, ecc. non verranno in nessun caso presi in considerazione. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Non è possibile lasciare l'aula conservando il tema della prova in corso.
5. Tempo a disposizione: 2h.

### Esercizio 1 (10 punti)

Il sistema sanitario nazionale vuole fornire ai medici una app per la gestione delle prescrizioni ai loro pazienti. Per aiutare il medico, all'aggiunta di ogni farmaco alla storia del paziente, il sistema controlla la compatibilità del nuovo farmaco con quelli già previsti. La classe `Paziente` fornisce quindi il metodo `ArrayList<Farmaco> aggiungi(Farmaco f)` e questo:

- rimuove tutti i farmaci potenzialmente incompatibili con il nuovo aggiunto dalla storia del paziente e li restituisce, come `ArrayList`. Nel caso in cui non esistano farmaci incompatibili, il metodo restituisce una lista vuota.
- aggiunge comunque il nuovo farmaco come primo alla lista dei farmaci prescritti al paziente.
- rimuove dalla cartella del paziente tutti farmaci non più prescritti, ovvero tutti i farmaci la cui data di fine prescrizione è nel passato rispetto a oggi.

`Paziente` fornisce anche il metodo `ArrayList<Farmaco> prescrizioniAttive()`, che fornisce tutte le prescrizioni attive, ovvero associate al paziente.

La classe `Farmaco` fornisce i metodi: `boolean compatibile(Farmaco f)` per controllare la compatibilità tra `this` e `f`, e `Date finePrescrizione()` per consentire il controllo relativo alla fine della prescrizione.

1. Si definisca la specifica completa del metodo `aggiungi` in JML.
2. Si ipotizzi una *rappresentazione* per lo stato delle classi `Paziente` e `Farmaco` e si definisca tutto quanto necessario per realizzare il metodo `Iterator<Farmaco> farmaciOrdineAlfabetico()` per poter accedere ai farmaci prescritti ad un paziente in ordine alfabetico.

Si ricorda che in Java 8 la classe `Date` ha un costruttore che crea un oggetto che rappresenta la data di creazione in millisecondi a partire dal 1/1/1970. I metodi `getTime`, che restituisce il valore della data in millisecondi come sopra, `before` e `after` possono essere usati per ragionare sulle date.

## Esercizio 2 (5 punti)

Una paninoteca vuole poter offrire ai propri clienti la massima flessibilità nella creazione dei panini. Ogni panino parte da una base, ovvero dal tipo di pane: bianco, arabo o multi-cereale. A questa, i clienti possono aggiungere Crudo, Cotto, Salame, o Tonno e poi anche Pomodoro, Mozzarella, Lattuga, Insalata Iceberg o Capperi. La paninoteca vorrebbe anche avere un sistema che con il tempo possa essere espanso con l'aggiunta di nuovi ingredienti.

Si realizzi un diagramma delle classi UML che “risolva” il problema sopra descritto. L’uso di *design pattern* appropriati verrà valutato positivamente. Si scriva anche la struttura delle classi principali in Java.

## Esercizio 3 (7 punti)

Si consideri un centro termale. L’acqua della piscina ha proprietà terapeutiche e pertanto il centro è frequentato sia da normali visitatori (visitatori) sia da persone affette da patologie (pazienti). Gli ospiti accedono singolarmente alla piscina, vi permangono per una quantità di tempo arbitraria e successivamente escono dalla piscina. La piscina ha una capacità massima MAXP, che esprime il numero massimo di persone che può contenere. Il regolamento della piscina vieta la presenza contemporanea nella piscina di visitatori e pazienti.

Si sviluppi un’applicazione concorrente in Java, che rappresenti pazienti e visitatori come thread concorrenti. L’applicazione deve realizzare una politica di sincronizzazione che soddisfi i vincoli dati e che, inoltre, favorisca i pazienti, rispetto ai visitatori, nell’accesso alla piscina. Si realizzi anche un programma principale “dimostratore” che consideri almeno 50 visitatori e 20 pazienti e una capacità massima di 60 ospiti.

## Esercizio 4 (6 punti)

Si considerino le seguenti classi Java:

```
public class Animale {  
    private String nome;  
    ...  
    public void parla() {}  
    public void incontra(Animale a) {  
        System.out.println(nome + ": Ciao, " + a.nome);  
    }  
}  
  
public class Topo extends Animale {  
    public void parla() {  
        System.out.println("Squit");  
    }  
  
    public void incontra(Gatto g) {  
        System.out.println(getNome() + ": Aiutoooooo!!!!");  
        parla();  
    }  
}  
  
public class Gatto extends Animale {  
    public void parla() {  
        System.out.println("Miao");  
    }  
  
    public void incontra(Topo t) {  
        System.out.println(getNome() + ": Che buono !!!");  
        parla();  
    }  
}
```

e si risponda, motivando brevemente le risposte, alle seguenti domande:

- Dal momento che il metodo `parla` di `Animale` non fa nulla, avremmo potuto ometterlo?
- Cosa produrrebbe il seguente codice:

```

Gatto g = new Gatto("Gatto");
Topo t = new Topo("Topo");
Animale a1 = new Gatto("Gatto");
Animale a2 = new Topo("Topo");

g.incontra(t);
t.incontra(g);
a1.incontra(a2);
a2.incontra(g);
g.incontra(g);

```

### Esercizio 5 (5 punti)

Si consideri il seguente frammento di codice Java:

```

void foo(int[] a, int d1, int d2) {
    if (d1 > 0 && d2 <= d1)
        for (int i=0; i <= d2; i++) {
            if (a[i] > 0) System.out.println(a[i]);
            if (a[i] < 0) System.out.println(-a[i]);
        }
    System.out.println("Fine");
}

```

dove `a []` è un array di interi, `d1` è la lunghezza dell'array e `d2` è un numero intero positivo.

1. Si disegni il diagramma del flusso di controllo.
2. Si identifichi un insieme minimo di test per coprire tutte le istruzioni, tutte le decisioni e tutte le condizioni.
3. Quali (probabili) errori sono evidenziati dall'insieme di test definito ai punti precedenti?



# Politecnico di Milano

sede di Cremona

Anno accademico 2015-2016

## Ingegneria del Software - 7 settembre 2016

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. Al termine, consegnare solo i fogli distribuiti. Eventuali fogli di brutta, ecc. non verranno in nessun caso presi in considerazione. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Non è possibile lasciare l'aula conservando il tema della prova in corso.
5. Tempo a disposizione: 2h.

### Esercizio 1 (10 punti)

Si vuole realizzare il software di gestione di un registratore di cassa per emettere scontrini fiscali e memorizzarli in modo appropriato. Per semplicità, si ipotizzi di usare una valuta senza centesimi e quindi rappresentabile solo con numeri interi. Ogni scontrino riporta un ammontare, la data di emissione e il numero dello scontrino nel corso della giornata (chiaramente, il primo scontrino della giornata ha numero 1). Il registratore di cassa memorizza gli scontrini di un intero anno, organizzandoli per settimana (da 1 a 52) e per mese (da 1 a 12).

La classe `Registratore` fornisce, tra gli altri, i seguenti metodi:

- `void emettiScontrino(int imp)` per l'emissione di uno scontrino di importo `imp`. Chiaramente, la data è quella corrente e il numero dello scontrino viene incrementato in modo opportuno.
- `void cancellaScontrino()` per cancellare l'ultimo scontrino emesso.
- `ArrayList<Scontrino>scontrini()` per avere la lista degli scontrini emessi fino a quel momento nel corso della giornata.

La classe `Scontrino` fornisce semplicemente tre metodi `int ammontare()`, `Data data()` e `int numero()` per poter leggere i valori dei tre attributi.

Si definisca quindi:

1. Una rappresentazione appropriata per la classe `Registratore`;
2. La specifica JML dei metodi `emettiScontrino` e `cancellaScontrino`;
3. L'invariante di rappresentazione (in JML) della classe `Registratore`, considerando le scelte fatte al punto 1.

### Esercizio 2 (6 punti)

Considerando la classe `Registratore` definita all'esercizio precedente, scrivere tutto quello che serve per avere due Iteratori che consentano di accedere, in successione, a tutti gli scontrini emessi in un singolo giorno o in una settimana.

### Esercizio 3 (7 punti)

Si vuole realizzare un sistema per la gestione delle attività commerciali relative ad una società che gestisce il mercato dei lingotti d'oro. Il mercato ha una capacità massima di  $M$  lingotti e può gestire non più di  $N$  utenti contemporaneamente. Gli utenti possono essere **acquirenti** oppure **venditori**, con ruoli diversi e ovvi. Ogni utente può acquistare/vendere lingotti in numero finito, ma ovviamente non si possono comprare più lingotti di quanti disponibili in quel momento e non è possibile violare il limite di  $M$  lingotti acquisiti dalla società di gestione. Per semplicità si supponga che il prezzo di acquisto/vendita di un lingotto sia sempre uguale.

Si scriva quindi un programma Java concorrente che realizzi il sistema di controllo sopra descritto.

### Esercizio 4 (6 punti)

Una app che fornisce informazioni turistiche relative alla città di Cremona fornisce anche le previsioni meteo relative al giorno corrente e ai tre giorni successivi. La app può essere configurata usando i servizi di previsioni metereologiche di tre provider diversi (ad esempio, Google, Yahoo e il meteo). Si usi il pattern *Strategy* per realizzare un semplice diagramma delle classi UML che riassume i requisiti sopra esposti. L'intera app può essere descritta con una sola classe UML. Si scriva anche lo scheletro del codice Java che corrisponde alle classi/interfacce identificate.

### Esercizio 5 (4 punti)

Si scriva un semplice metodo Java che conta il numero di vocali presenti in una lista di stringhe utilizzando un approccio funzionale. Il punto di partenza potrebbe essere:

```
List<String> list = Arrays.asList("abc", "", "bc", "efg", "abcd", "", "jkl");
```



# Politecnico di Milano

sede di Cremona

Anno accademico 2015-2016

## Ingegneria del Software - 28 settembre 2016

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. Al termine, consegnare solo i fogli distribuiti. Eventuali fogli di brutta, ecc. non verranno in nessun caso presi in considerazione. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Non è possibile lasciare l'aula conservando il tema della prova in corso.
5. Tempo a disposizione: 2h.

### Esercizio 1 (9 punti)

Scrivere la specifica JML dei seguenti metodi statici Java:

- `int metodo1(String[] s, int p)`. Il metodo restituisce il numero di caratteri delle diverse stringhe dell'array dalla posizione `p` in poi (il `p`-esimo escluso). Ovvero, se una delle stringhe fosse Cremona e `p` valesse 3, il numero di caratteri da considerare sarebbe 4, ovvero 7, la lunghezza della parola, meno 3.
- `int metodo2(int[] a, int d1, int d2)`. Il metodo restituisce la somma degli elementi dell'array di posizione compresa tra `d1` e `d2`. Restituisce 0 nel caso `d1` e/o `d2` non fossero compatibili con l'array.
- `ArrayList<Integer> metodo3(int v1, int v2)`. Il metodo restituisce la lista di numeri compresi tra `v1` e `v2`, estremi compresi, che non sono divisibili per gli altri numeri compresi tra `v1` e `v2`.

### Esercizio 2 (6 punti)

Rispondere brevemente alle seguenti domande:

- Se un metodo ha un parametro di tipo `int v`, in JML devo sempre avere una precondizione `v != null?`
- Se in una sotto-classe definisco un costruttore senza chiamarne esplicitamente uno della super-classe, cosa succede?
- Perchè usare un pattern *Strategy* e non un *Decorator*?
- È possibile dire che “copertura delle decisioni” e “copertura delle condizioni” sono criteri equivalenti per la selezione di test? Perché?

### Esercizio 3 (7 punti)

Si scriva un programma Java concorrente per la gestione di un parcheggio. Chiaramente, le automobili (modellate come thread) potranno entrare nel parcheggio fino ad esurimento dei posti disponibili. Nel caso in cui il parcheggio fosse pieno, le automobili dovranno aspettare l'uscita di un altro mezzo. Si scriva anche un programma principale modellando un parcheggio con 20 posti auto e con 50 automobili che entrano ed escono dal parcheggio liberamente (in modo random).

### Esercizio 4 (7 punti)

Si consideri il seguente metodo Java:

```
public static int foo(int x, int y) {  
    int a[] = new int[2];  
    int r=0, i=0;  
  
    a[0] = y; a[1] = 2*y;  
  
    while (i < y%3 || i < a.length) {  
        if (a[i] < 0) a[i] = -a[i];  
        r += y/a[i];  
        i++;  
    }  
    return r;  
}
```

e si definisca:

- il diagramma del flusso di controllo;
- un insieme minimo di test per coprire tutte le istruzioni;
- un insieme minimo di test per coprire tutte le condizioni;

Inoltre, quali possibili errori sono stati inavvertitamente lasciati nel codice?

### Esercizio 5 (4 punti)

Si scrivano in Java due metodi che prendono come unico parametro in ingresso un `ArrayList<String> l` e restituiscono il numero di caratteri contenuti nelle diverse stringhe dopo aver eliminato le doppie `tt`, `ll` e `cc`. Il primo metodo deve adottare un stile di programmazione procedurale, mentre il secondo deve usare un approccio funzionale.



# Politecnico di Milano

sede di Cremona

Anno accademico 2015-2016

## Ingegneria del Software - 2 febbraio 2016

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. Al termine, consegnare solo i fogli distribuiti. Eventuali fogli di brutta, ecc. non verranno in nessun caso presi in considerazione. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Non è possibile lasciare l'aula conservando il tema della prova in corso.
5. Tempo a disposizione: 2h.

### Esercizio 1 (9 punti)

Scrivere la specifica JML dei seguenti metodi statici Java:

- `int metodo1(String[] s, int p)`. Il metodo restituisce la somma di caratteri ripetuti nelle diverse stringhe. Le ripetizioni contano solo se nella stessa stringa. Si noti anche che il numero di ripetizioni è pari al numero di volte che un carattere compare nella particolare parola meno uno, ovvero la prima volta.
- `int metodo2(int[] a, int d1, int d2)`. Il metodo restituisce la somma degli elementi dell'array di posizione compresa tra `d1` e `d2` se gli elementi medesimi sono in ordine crescente. Restituisce 0, se gli elementi non sono nell'ordine richiesto e restituisce `-1` nel caso `d1` e/o `d2` non fossero compatibili con l'array.
- `ArrayList<Integer> metodo3(int v1, int v2)`. Il metodo restituisce la lista ordinata di tutti i numeri primi compresi tra `v1` e `v2`. Restituisce una lista vuota se non ci fossero numeri primi tra `v1` e `v2` e solleva un'eccezione `ParametriException` se `v2` fosse minore o uguale a `v1`.

### Esercizio 2 (6 punti)

Si consideri una app per la gestione del magazzino di un piccolo supermercato. L'applicazione deve consentire le seguenti operazioni:

- I vendori devono controllare il numero di prodotti di un certo tipo che sono disponibili sugli scaffali ed aggiornano il sistema di conseguenza. Se il numero di prodotti risulta minore di una certa soglia, i vendori devono prelevare altri prodotti dal magazzino e i responsabili di magazzino devono aggiornare il numero di quelli disponibili.
- I responsabili di magazzino devono aggiornare le quantità di prodotti stoccate. Questo aggiornamento deve essere effettuato tutte le volte che i vendori prelevano prodotti e che i fornitori inviano nuovi prodotti. I responsabili del magazzino devono anche decidere quando un certo tipo di prodotto deve essere ordinato di nuovo.
- Gli addetti agli ordini ricevono le richieste dal magazzino ed usano il sistema per ottenere le informazioni relative ai fornitori e poi effettuano l'ordine.

Si definisca un diagramma delle classi per rappresentare le strutture dati e le operazioni che il sistema deve poter offrire ai diversi attori. Se si ritiene opportuno fare uso di uno o più pattern, si indichi quale e si fornisca una motivazione per il suo utilizzo.

### Esercizio 3 (7 punti)

Si scriva un programma Java concorrente per la gestione di  $n$  sale riunioni. Le sale possono essere prenotate solo nel momento del bisogno e ogni prenotazione potrà essere servita fino ad esaurimento delle sale disponibili, restituendo il numero della sala assegnata. Ogni sala deve essere poi esplicitamente “rilasciata” prima di tornare tra quelle disponibili. Nel caso in cui tutte le sale fossero occupate, le richieste inevitabili potranno aspettare la disponibilità di una sala o abortire la loro richiesta. Si scriva anche un programma principale modellando un centro congressi con 20 sale riunioni e con 40 richieste presentate al centro congressi liberamente (in modo random).

### Esercizio 4 (7 punti)

Si consideri il seguente metodo Java:

```
public static void testMethod(int x, String s) {  
    int l;  
  
    if (x <= 0 || s == null) return;  
    l = s.length();  
  
    while (x >= 1) {  
        if (x > 0 || l > 0) x = Math.abs(x) - 1;  
        else break;  
    }  
    return;  
}
```

e si risponda alle seguenti domande:

1. Si disegni il diagramma di flusso del metodo.
2. Si identifichi, se esiste, un insieme di test (minimo) per coprire tutte le istruzioni e le decisioni (branch) del metodo.
3. Si identifichi, se esiste, un insieme di test (minimo) per coprire tutte le istruzioni e le condizioni del metodo.
4. In generale, quindi non riferendosi allo specifico frammento di codice sopra, un insieme di test che copre tutte le decisioni copre anche tutte le condizioni? Nel caso non fosse vero, potremmo dire che invece un insieme di test che copre tutte le condizioni copre anche tutte le decisioni?

### Esercizio 5 (4 punti)

- Si scriva in Java un metodo che prenda come unico parametro in ingresso un `ArrayList<String>` `l` e restituisca il numero di a contenute nelle diverse parole adottando uno stile di programmazione funzionale.
- Si considerino la classe astratta `Figura` e le sue sottoclassi `Quadrato` e `Cerchio`. Si usi il pattern *Factory method* per consentire la creazione dinamica di cerchi e quadrati specificando il tipo dell'oggetto attraverso un parametro.



# Politecnico di Milano

sede di Cremona

Anno accademico 2016-2017

## Ingegneria del Software - 10 luglio 2017

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. Al termine, consegnare solo i fogli distribuiti. Eventuali fogli di brutta, ecc. non verranno in nessun caso presi in considerazione. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Non è possibile lasciare l'aula conservando il tema della prova in corso.
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Scrivere la specifica JML dei seguenti metodi statici Java:

- `int metodo1(String[] s, int p)`. Il metodo restituisce il numero di doppie (caratteri ripetuti e in sequenza) nelle diverse parole dell'array.
- `int metodo2(int[] a, int d1, int d2)`. Il metodo restituisce la differenza tra la somma degli elementi dell'array di posizione compresa tra 0 e d1 e tra d2 e la fine dell'array. Se d1 fosse maggiore della lunghezza dell'array, la prima somma comprenderebbe tutti e soli gli elementi dell'array stesso. Se d2 fosse maggiore della lunghezza dell'array, la seconda somma varrebbe 0. Se d1 e/o d2 fossero negativi, il metodo dovrebbe restituire -1.
- `ArrayList<Integer> metodo3(int v1, int v2)`. Il metodo restituisce la lista ordinata di tutte le potenze di v1 da  $v1^0$  a  $v1^{v2}$ . Il metodo solleverebbe un'eccezione `ParametroException` se v2 fosse minore di 0.
- `int metodo4(int[] a, int v)`. Il metodo restituisce 1 se l'array contiene almeno un divisore di v, -1 altrimenti.

### Esercizio 2 (7 punti)

Scrivere le classi Java, complete di rappresentazione e intestazione dei metodi principali, per realizzare un semplice programma per giocare a briscola. Non si richiede nulla di distribuito e neppure la gestione della grafica. Semplicemente, bisogna gestire tutti gli elementi per consentire una singola partita tra due giocatori. A fine partita, il programma deve scrivere a video l'esito della partita (vittoria o pareggio) ed il punteggio dei due giocatori.

È importante definire le classi, gli attributi ed i metodi principali, ma l'implementazione dei diversi metodi non è richiesta (basta la sola intestazione).

### Esercizio 3 (7 punti)

Si scriva un programma Java concorrente per la gestione di un parcheggio con  $n$  posti,  $m$  di questi ( $m < n/2$ ) sono riservati per auto a metano e 5, se disponibili (ovvero se  $n > 5$ ) per donne in stato di gravidanza. Ogni automobile può entrare nel parcheggio solo se esistono posti disponibili, in base alla tipologia del mezzo/utente, e rimanere per un tempo casuale. Se il parcheggio fosse pieno, l'automobile può aspettare o decidere di rinunciare. Chiaramente le donne incinte potrebbero anche parcheggiare in posti "normali", ma gli utenti normali no. Se una donna incinta guidasse una macchina a metano, allora potrebbe parcheggiare solo negli spazi riservati per dette autovetture.

Si scriva anche un programma principale modellando un parcheggio con 200 posti auto, di cui 50 riservati per le auto a metano e 5 per le donne incinte. Si ipotizzino anche 500 accessi, con un 40% di auto a metano e un 10% di donne incinte. Se l'ingresso fosse negato, un automobilista su due attende, mentre l'altro rinuncia. Il tempo di permanenza è casuale.

### Esercizio 4 (6 punti)

Si consideri il seguente metodo Java:

```
public static int method(int x, int y) {  
    if (x < 0) x = -x;  
    x = x+y;  
  
    while (x > 0) {  
        if (y % 3 == 0) y += 2;  
        x -= 2;  
    }  
    return x;  
}
```

e si risponda alle seguenti domande:

1. Si disegni il diagramma di flusso del metodo.
2. Si identifichi, se esiste, un insieme di test (minimo) per coprire tutte le istruzioni del metodo.
3. Si identifichi, se esiste, un insieme di test (minimo) per coprire tutte le condizioni del metodo. È uguale all'insieme precedente? Perché?
4. È vero che il metodo restituisce sempre una valore positivo? Perché?

### Esercizio 5 (5 punti)

Si consideri il codice Java seguente e si spieghi cosa verrebbe prodotto dall'esecuzione, ovvero stampato a video.

```
public class Punto2D {  
    private int x, y;  
  
    public Punto2D(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public double distanza(Punto2D p) {  
        return Math.sqrt(Math.pow(x-p.x, 2)  
                        + Math.pow(y-p.y, 2));  
    }  
}  
  
public class Punto3D extends Punto2D {  
    private int z;  
  
    public Punto3D(int x, int y, int z) {  
        super(x, y);  
        this.z = z;  
    }  
  
    public double distanza(Punto2D p) {  
        return 2*super.distanza(p);  
    }  
}  
  
public double distanza(Punto3D p) {  
    return Math.sqrt(Math.pow(super.distanza(p), 2)  
                    + Math.pow(z-p.z, 2));  
}  
  
public class Test {  
    public static void main(String[] args) {  
        Punto2D p1 = new Punto2D(0, 0);  
        Punto2D p2 = new Punto2D(3, 4);  
        Punto2D p3 = new Punto3D(3, 4, 1);  
        Punto3D p4 = new Punto3D(0, 0, 0);  
        Punto3D p5 = new Punto3D(6, 8, 5);  
  
        System.out.println(p1.distanza(p2));  
        System.out.println(p1.distanza(p3));  
        System.out.println(p2.distanza(p3));  
        System.out.println(p4.distanza(p5));  
        System.out.println(p5.distanza(p2));  
    }  
}
```



# Politecnico di Milano

sede di Cremona

Anno accademico 2016-2017

## Ingegneria del Software - 24 luglio 2017

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. Al termine, consegnare solo i fogli distribuiti. Eventuali fogli di brutta, ecc. non verranno in nessun caso presi in considerazione. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Non è possibile lasciare l'aula conservando il tema della prova in corso.
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Scrivere la specifica JML dei seguenti metodi statici Java:

- `boolean metodo1(String[] s1, String[] s2)`. Il metodo restituisce `true` se tutte le stringhe presenti in `s1` sono anche presenti in `s2`; `false`, altrimenti. `s1` non può contenere stringhe `null`.
- Cosa cambierebbe al punto precedente se `s1` e `s2` dovessero contenere esattamente le stesse stringhe e nello stesso ordine.
- `ArrayList<String> metodo3(char c, int max)`. Il metodo restituisce la lista ordinata di tutte le stringhe composte da `n` volte il carattere `c`, dove `n` varia da 1 a `max`.
- `int metodo4(int[] a)`. Il metodo restituisce 1 se l'array contiene almeno un numero che è la somma di tutti gli altri (elementi dell'array).

### Esercizio 2 (7 punti)

Definire un *class diagram* UML per realizzare un'applicazione Java per giocare a *Tris*. Non si richiede nulla di distribuito e neppure la gestione della grafica. Semplicemente, bisogna gestire tutti gli elementi per consentire una singola partita tra due giocatori. A fine partita, il programma deve scrivere a video l'esito della partita ed il vincitore. È importante definire le classi, gli attributi ed i metodi principali. Se definisce anche un *sequence diagram* che mostri il completamento di una mossa.

### Esercizio 3 (7 punti)

Si scriva un programma Java concorrente per creare un pool di thread. I thread dispari (primo, terzo, ecc) stampano i numeri compresi tra 1 ed un valore casuale compreso tra 0 e 200. I thread pari (secondo, quarto, ecc) vanno in sleep per un periodo casuale di millisecondi compreso tra 0 e 5000. Ogni thread stampa a video il proprio nome (passato al momento della creazione dal main), i numeri da stampare o il numero di millisecondi di durata della sleep, e l'avvenuta conclusione, eventualmente dopo la sleep. Il main, dopo aver creato i thread, stampa a video una stringa di avvenuta terminazione. Il numero di thread creati è un parametro da riga di comando e per generare numeri casuali, usare la classe `java.util.Random` e il metodo `nextInt(int)`.

## Esercizio 4 (5 punti)

Si consideri il seguente metodo Java:

```
public int sequenzaCrescentePiuLunga (int vett[]) {  
    int i = 0, cont=0, max = 0;  
    for (i = 1; i < vett.length; i++) {  
        if (vett[i]>vett[i-1]) cont++;  
        if (cont > max) max = cont;  
        else cont = 0;  
    }  
    return max;  
}
```

e si risponda alle seguenti domande:

1. Si disegni il diagramma di flusso del metodo.
2. Si identifichi, se esiste, un insieme di test (minimo) per coprire tutte le istruzioni del metodo.
3. Si identifichi, se esiste, un insieme di test (minimo) per coprire tutte le decisioni del metodo. È uguale all'insieme precedente? Perché?

## Esercizio 5 (6 punti)

Rispondere brevemente alle seguenti domande:

1. Quali sono le differenze principali tra una classe astratta e un'interfaccia in Java?
2. Una sotto-classe può sollevare eccezioni che la super-classe non solleva? Nel caso, sarebbe possibile il contrario?
3. Quali sono le differenze principali tra i pattern *Strategy* e *Decorator*?



# Politecnico di Milano

sede di Cremona

Anno accademico 2016-2017

## Ingegneria del Software - 14 settembre 2017

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. Al termine, consegnare solo i fogli distribuiti. Eventuali fogli di brutta, ecc. non verranno in nessun caso presi in considerazione. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Non è possibile lasciare l'aula conservando il tema della prova in corso.
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Scrivere la specifica JML dei seguenti metodi statici Java:

- `boolean metodo1(String[] s1, String[] s2)`. Il metodo restituisce `true` se `s2` contiene tutte le stringhe non `null` presenti in `s1` almeno due volte, anche non consecutive.
- Cosa cambierebbe al punto precedente se `s2` dovesse contenere le stringhe in `s1` esattamente due volte in sequenza.
- `ArrayList<Integer> metodo3(int[] a, int v)`. Il metodo restituisce la lista ordinata di tutti gli interi in `a` che sono divisori di `v`. Alla peggio, il metodo restituire comunque un array list vuoto, ovvero senza elementi.
- `boolean metodo4(int[] a)`. Il metodo restituisce `true` se l'array contiene solo numeri primi.

### Esercizio 2 (6 punti)

Rispondere brevemente alle seguenti domande:

- Come si rappresenta in UML l'ereditarietà tra classi ed interfacce e quali sono le differenze tra i due concetti.
- Come si rappresenta in UML la relazione tra classe ed interfaccia per dire che la prima implementa la seconda? Può una classe implementare più interfacce? E un'interfaccia essere implementata da più classi?
- Come si rappresentano in UML le classi astratte e gli attributi di classe (statici)?

### Esercizio 3 (8 punti)

Si consideri la seguente classe Java:

```
public class SyncStack {  
    private int[] data;  
    private int top;  
    public SyncStack(int size) {  
        data = new int[size];  
        top = 0;  
    }  
    public synchronized void push(int val) {  
        if(top>=data.length) throw new ArrayIndexOutOfBoundsException();  
        data[top] = val; top++;  
    }  
    public synchronized int pop() {  
        if(top<=0) throw new ArrayIndexOutOfBoundsException();  
        int val = data[top-1]; top--;  
        return val;  
    }  
    public int top() {  
        if(top<=0) throw new ArrayIndexOutOfBoundsException();  
        return data[top-1];  
    }  
}
```

- Le istanze della classe sono “thread safe”? Ovvero i metodi possono essere invocati da thread diversi senza che questo crei problemi? In caso negativo, come dobbiamo modificare il codice per garantire il corretto accesso alle istanze della classe da parte di thread diversi?
- Perché la classe sia “thread safe” è necessario sincronizzare il costruttore? Perché?
- Si modifichi la classe precedente in maniera che invece di restituire un’eccezione quando si invoca il metodo push su un SyncStack pieno (o i metodi pop e top su un SyncStack vuoto), il codice sospenda il chiamante fino a quando non venga rimosso (risp: aggiunto) un elemento.

### Esercizio 4 (5 punti)

Si consideri il seguente frammento di programma (a, b, c sono variabili booleane):

```
if ((a && b) || c)  
{  
    << Statements >>  
}  
else if (a || c)  
{  
    << Statements >>  
}  
else  
{  
    << Statements >>  
}
```

Si definisca, giustificando le risposte, il minimo insieme di casi di test per ciascuno dei seguenti casi:

1. vengono coperti tutti i branch;
2. vengono coperte tutte le istruzioni;
3. vengono coperti tutti i branch e tutte le condizioni.

## Esercizio 5 (6 punti)

Si consideri una generica lista di stringhe, inizializzata come

```
List<String> names = Arrays.asList("Luca", "Carlo", "Alberto", "Franco");
```

si vuole selezionare l'iniziale delle stringhe che risulta prima in ordine alfabetico. Nell'esempio, tale iniziale sarebbe A. Implementare il meccanismo necessario con una o più chiamate in stile funzionale, eventualmente utilizzando la funzione `stringOrder` indicata di seguito.

```
private static boolean stringOrder(String a, String b) {  
    if (a.toCharArray()[0] < b.toCharArray()[0])  
        return true;  
    else  
        return false;  
}
```



# Politecnico di Milano

sede di Cremona

Anno accademico 2017-2018

## Ingegneria del Software - 02 febbraio 2018

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Scrivere le intestazioni dei seguenti metodi statici Java e la loro specifica JML:

- Il metodo prende in ingresso una matrice quadrata di interi e la sua dimensione e restituisce *true* se i numeri delle diverse righe e colonne sono in ordine crescente.
- Il metodo deve inserire un intero in un array mantenendo i numeri in ordine decrescente. Il metodo deve sollevare un'eccezione se l'array è pieno o se il numero è già presente.
- Il metodo prende in ingresso un ArrayList di interi e restituisce il numero massimo di ripetizioni in sequenza dello stesso numero. Ad esempio, se la lista contenesse [1, 1, 0, 0, 1, 1, 2, 2, 2], il metodo restituirebbe 3, considerando le ripetizioni del numero 2.
- Il metodo prende in ingresso un ArrayList di interi e restituisce *true* se i numeri contenuti rappresentano correttamente una parte della serie di Fibonacci, ovvero la somma di due numeri contigui rappresenta il numero successivo della sequenza.

### Esercizio 2 (6 punti)

Scrivere l'invariante pubblico e privato della classe MazzoDiCarte (da 52 carte). I suoi metodi pubblici sono `mescola()`, `pesca()`, una carta, e `numCarte()`, che restituisce il numero di carte rimaste nel mazzo. La sua rappresentazione (**rep**), privata, è un semplice `ArrayList<Carta>`. Si ipotizzi che la classe `Carta` ridefinisca in modo opportuno il metodo `equals()`. Inoltre, considerando le informazioni a disposizione, definire la specifica JML del metodo `pesca()`. Considerando i metodi a disposizione, si riesce a dire tutto?

### Esercizio 3 (7 punti)

Un sito di commercio elettronico deve gestire i propri clienti e consentire loro di creare opportuni carrelli, scegliendo tra i prodotti a disposizione, e di pagarli con diversi metodi di pagamento (ad esempio, diverse carte di credito, PayPal, ApplePay, ecc.). La differenza fondamentale dei diversi metodi di pagamento sta nella commissione che la società deve pagare al gestore del sistema di pagamento e quindi nel guadagno effettivo derivante dalla vendita. I clienti poi sono divisi in *base* e *premium*: ai clienti del secondo gruppo è sempre applicato uno sconto del 3%. L'azienda vuole poter cambiare i sistemi di pagamento senza

che questi debbano avere alcun impatto sui prodotti venduti. Descrivere il problema precedente usando un diagramma delle classi UML ed eventualmente i *design pattern* ritenuti appropriati. Si scriva anche l'implementazione del metodo `paga` per pagare il `Carello` di prodotti scelti, secondo chiaramente le scelte fatte nel diagramma delle classi ed ipotizzando che esista un metodo `prodotti()` di `Carello` che restituisce un `Iterator<Prodotto>`.

### Esercizio 4 (6 punti)

Il centro storico di un piccolo comune è attraversato da un fiume ed un ponte collega le due parti del borgo. Le strade che arrivano al ponte sono a doppio senso di marcia, mentre il ponte consente il transito delle automobili in un solo senso di marcia per volta. Quindi, un'automobile può attraversare il ponte se non già occupato da altre automobili che procedono nel verso opposto, ma più automobili possono procedere contemporaneamente nello stesso verso. Si scriva un programma Java concorrente per risolvere il problema sopra descritto, ipotizzando anche l'arrivo random di diverse automobili nei due sensi di marcia.

### Esercizio 5 (6 punti)

Si consideri il seguente codice. Operando su un array di esattamente tre elementi, il codice deve riorganizzarli in ordine decrescente.

```
public int[] order(int v[]) {  
    int tmp;  
    if (v[0]<v[1]) {  
        tmp = v[0];  
        v[0] = tmp;  
        v[1] = tmp;  
    }  
    if (v[1]<v[2]) {  
        tmp = v[1];  
        v[1] = v[2];  
        v[2] = tmp;  
    }  
    return v;  
}
```

Si identifichi il numero minimo di casi di test in maniera da soddisfare (separatamente) ciascuno dei seguenti criteri: copertura delle istruzioni, delle decisioni e delle condizioni. Trovare anche tutti gli eventuali errori presenti nel codice. Giustificate brevemente le risposte.



# Politecnico di Milano

sede di Cremona

Anno accademico 2017-2018

## Ingegneria del Software - 23 febbraio 2018

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Scrivere le intestazioni dei seguenti metodi statici Java e la loro specifica JML:

- Il metodo prende in ingresso una matrice quadrata di interi e la sua dimensione e restituisce *true* se la somma dei numeri delle diverse righe e colonne è sempre uguale.
- Il metodo deve inserire un intero in un arrayList mettendolo sempre in prima posizione. Il metodo non deve inserire uno stesso numero due volte e deve sollevare le opportune eccezioni.
- Il metodo prende in ingresso un ArrayList di stringhe e restituisce il numero massimo di parole che iniziano con la stessa lettera.
- Il metodo prende in ingresso un ArrayList di interi e restituisce *true* se i numeri contenuti rappresentano le potenze in sequenza di un certo numero.

### Esercizio 2 (6 punti)

Una cartella della tombola contiene 15 numeri, organizzati in una matrice di tre righe e nove colonne (una per ogni decina), e non può contenere numeri ripetuti. Supponendo che gli unici metodi pubblici della classe CartellaTombola siano `estratto(int n)` per marcare un numero estratto e `numeriEstratti()` per sapere quanti numeri della particolare cartella sono stati estratti, scrivere l'invariante pubblico di `CartellaTombola`. Supponendo poi di usare un array bidimensionale di tre righe e nove colonne per memorizzare i numeri, cosa si potrebbe dire come invariante privato. Ogni cella dell'array è un oggetto di classe `NumeroTombola`, ovvero un intero, il numero specifico, ed un booleano, per tener traccia di eventuali estrazioni. Infine, considerando le informazioni a disposizione, cosa si potrebbe dire nella specifica JML del metodo `estratto(int n)`.

### Esercizio 3 (7 punti)

Un'azienda deve gestire il proprio parco clienti. Oltre a tenere traccia della solita anagrafica, l'azienda vuole poter associare ad ogni cliente un numero variabile di promozioni in base alla posizione geografica del cliente, al fatturato generato e alla tipologia di progetti acquistati. Ogni promozione comporta uno sconto sul prezzo di vendita e le diverse promozioni sono cumulabili, ovvero l'effettivo prezzo di vendita (o sconto) è calcolato tenendo conto di tutte le promozioni associate al particolare cliente. L'azienda vuole anche essere libera di aggiungere nuove tipologie di promozioni in futuro (oltre alla posizione, al fatturato ed ai tipi di prodotti) e chiede quindi una soluzione estremamente flessibile. Detto questo:

- Si descriva il problema precedente attraverso un diagramma delle classi UML utilizzando il pattern *decorator*.
- Supponendo che il diagramma precedente contenga una classe `Clienti`, ovvero la lista dei diversi clienti, si implementi un metodo `iterator()` che restituisce un `Iterator<Cliente>`, e tutto quello che il metodo richiede, per dare accesso ai diversi clienti.

### Esercizio 4 (6 punti)

Un distributore ha una sola pompa di benzina e può, quindi, servire un solo cliente per volta. Se più auto cercassero di usare la pompa contemporaneamente, solo la prima automobile potrebbe fare rifornimento, mentre le altre dovrebbero aspettare che la pompa diventi libera. Per semplicità si supponga che il tempo del rifornimento sia sempre uguale e si scriva un programma Java concorrente per risolvere il problema sopra descritto, ipotizzando anche l'arrivo random di diverse automobili. Cosa cambierebbe se il distributore avesse tre pompe?

### Esercizio 5 (6 punti)

Si consideri il codice Java seguente e si spieghi cosa verrebbe prodotto dall'esecuzione, ovvero stampato a video.

```
public class Punto2D {  
    private int x, y;  
  
    public Punto2D(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public double distanza(Punto2D p) {  
        return 1;  
    }  
}  
  
public class Punto3D extends Punto2D {  
    private int z;  
  
    public Punto3D(int x, int y, int z) {  
        super(x, y);  
        this.z = z;  
    }  
  
    public double distanza(Punto2D p) {  
        return 2;  
    }  
}  
  
public double distanza(Punto3D p) {  
    return 3;  
}  
  
public class Test {  
    public static void main(String[] args) {  
        Punto2D p1 = new Punto2D(0, 0);  
        Punto2D p2 = new Punto2D(2, 1);  
        Punto2D p3 = new Punto3D(3, 4, 2);  
        Punto3D p4 = new Punto3D(2, 1, 0);  
        Punto3D p5 = new Punto3D(6, 8, 2);  
  
        System.out.println(p1.distanza(p2));  
        System.out.println(p3.distanza(p2));  
        System.out.println(p2.distanza(p3));  
        System.out.println(p1.distanza(p5));  
        System.out.println(p4.distanza(p1));  
        System.out.println(p5.distanza(p4));  
    }  
}
```



# Politecnico di Milano

sede di Cremona

Anno accademico 2017-2018

## Ingegneria del Software - 27 giugno 2018

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Scrivere le intestazioni dei seguenti metodi statici Java e la loro specifica JML:

- Il metodo prende in ingresso una matrice quadrata di interi e la sua dimensione e restituisce *true* se la somma degli elementi delle riga e delle colonne di pari posizione (ad esempio, prima riga e prima colonna) è uguale.
- Il metodo prende in ingresso un ArrayList *as* di stringhe e restituisce un altro ArrayList che contiene tutte e sole le parole in *as* di lunghezza maggiore o uguale a 5 caratteri ed in ordine inverso.
- Il metodo prende in ingresso un ArrayList di stringhe e restituisce la lunghezza media delle stringhe contenute.
- Il metodo prende in ingresso un array di interi e restituisce *true* se non esiste nessun numero che sia divisore degli altri (o almeno di un altro).

### Esercizio 2 (6 punti)

Un Viaggio si compone di diverse Tratta e da un numero di partecipanti. Ogni Tratta è caratterizzata da un Mezzo e da un orario di partenza e arrivo previsti (definiti come Date). Ogni Viaggio deve avere almeno una tratta e non meno di 5 partecipanti. Inoltre, per ogni Tratta, si deve specificare il mezzo di trasporto e deve avere una fine successiva all'inizio. Gli unici metodi pubblici di Viaggio sono: partecipanti(), che restituisce il numero di partecipanti, e durata(), che restituisce il numero di tratte previste. Si scriva, quindi, l'invariante pubblico della classe Viaggio, una rappresentazione plausibile per Viaggio e Tratta e l'invariante privato della classe Viaggio. Si ricorda che la classe Date mette a disposizione il metodo compareTo(Date d) che restituisce 0 se le due date sono uguali, un valore minore di 0 se *this* < *d* ed uno maggiore di 0 se *this* > *d*.

### Esercizio 3 (7 punti)

Si consideri la seguente classe Java che realizza il tipo Parcheggio capace di contenere un massimo di 10 Automobili. Il valore null è usato internamente per identificare un elemento vuoto. Il metodo get(pos) restituisce il contenuto dell'elemento in posizione pos, se presente, altrimenti solleva l'eccezione EmptySlotException. Il metodo set(pos, auto) inserisce l'Automobile auto in posizione pos, sostituendo un'eventuale automobile già presente.

```

public class Parcheggio {
    private Automobile[] slot;

    public Parcheggio() {
        slot = new Automobile[10];
        for(int i=0; i<10; i++) slot[i] = null;
    }

    public Automobile get(int pos) throws EmptySlotException {
        if(slot[pos] == null) throw new EmptySlotException();
        Automobile a = slot[pos];
        slot[pos] = null;
        return a;
    }

    public void set(int pos, Automobile auto) {
        slot[pos] = auto;
    }
}

```

Si modifichi la classe `Parcheggio` in maniera che il metodo `get(pos)` sospenda il chiamante se l'elemento in posizione `pos` è vuoto, risvegliandolo quando l'elemento richiesto viene inserito. Analogamente, il metodo `set(pos, auto)` deve sospendere il chiamante se l'elemento in posizione `pos` è già pieno, in attesa che si svuoti. Si usino solo i meccanismi base della sincronizzazione di Java senza sfruttare le classi dei pacchetti `java.util.concurrent.XXX`.

### Esercizio 4 (6 punti)

Si consideri il seguente metodo statico Java:

```

static public void foo(int m, int n) {
    if (m > 10 && n < 50) System.out.println("pippo");
    else if (m > 10) System.out.println("pluto");
    if (n > 80) System.out.println("topolino!");
}

```

e (a) si disegni il diagramma di flusso del metodo e definiscano degli insiemi di test (ragionevolmente minimi) per coprire: (b) tutte le istruzioni e (c) tutte le decisioni (branch).

### Esercizio 5 (6 punti)

Si consideri il codice Java seguente e si spieghi cosa viene prodotto dall'esecuzione: stampe a video e/o eventuali errorri. In caso di errori, si continui comunque fino alla fine.

```

abstract class Operatore {
    public abstract void calcola(double a);
}

class Quadrato extends Operatore {
    public void calcola(double a) {
        System.out.println("Quadrato: " + (a*a));
    }
}

class Cubo extends Operatore {
    public void calcola(double a) {
        System.out.println("Cubo: " + (a*a*a));
    }
    public void calcola(int a) {
        System.out.println("Cubo int: " + (a*a*a));
    }
}

class DoppioCubo extends Cubo {
    public void calcola(int a) {
        System.out.println("CuboD int: " + (a*a*a)*2);
    }
}

class Test {
    public static void main(String[] args) {
        Operatore op1, op2;
        DoppioCubo op3, op4;
        op1 = new Quadrato();
        op2 = new Cubo();
        op3 = new DoppioCubo();
        op1.calcola(7.0);
        op1.calcola(3);
        op3.calcola(5);
        op4 = (DoppioCubo) op1;
        op4.calcola(3);
        op1 = op2;
        op1.calcola(3.0);
    }
}

```



# Politecnico di Milano

sede di Cremona

Anno accademico 2017-2018

## Ingegneria del Software - 20 luglio 2018

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Si scrivano le intestazioni dei seguenti metodi statici Java e la loro specifica JML:

- Il metodo prende in ingresso una matrice quadrata di interi e restituisce *true* se gli elementi sulla diagonale principale sono tutti uguali e se la somma delle elementi di ogni colonna è uguale alla somma degli elementi sulla diagonale principale.
- Il metodo prende in ingresso un ArrayList di parole e restituisce *true* se tutte le parole hanno lunghezza dispari e le parole nell'array sono ordinate dalla più corta alla più lunga.
- Il metodo prende in ingresso un ArrayList di parole e restituisce il numero di vocali presenti nelle diverse parole. Nel caso lo si ritenesse necessario, è possibile definire un metodo puro opportuno di supporto.
- Il metodo prende in ingresso un array di interi e restituisce la somma dei prodotti degli elementi specchiati nell'array, ovvero il primo con l'ultimo, il secondo con penultimo e così via. Si noti che se l'array avesse lunghezza dispari, l'ultima coppia conterebbe lo stesso elemento due volte.

### Esercizio 2 (6 punti)

Si consideri il tipo di dato astratto `PianoStudi`, che definisce l'insieme dei corsi selezionati da ogni studente di un'università, i crediti corrispondenti ed i voti eventualmente conseguiti. Ogni piano degli studi può contenere esami per un massimo di 150 crediti. Un'ipotetica implementazione (Java) fornisce anche qualche metodo: `esami()`, che restituisce gli esami scelti come un `ArrayList`, e `voto(Esame e)`, che restituisce l'eventuale voto conseguito o 0 se l'esame non fosse presente nel piano degli studi, o non esistesse un voto valido. La classe `Esame`, tra gli altri fornisce anche un metodo `crediti()`, che restituisce il numero di crediti associati all'esame. Si definisca, quindi: (a) l'invariante pubblico della classe `PianoStudi`, (b) una rappresentazione per la classe stessa e (c) la specifica JML del metodo `voto()`.

### Esercizio 3 (7 punti)

Otto amici decidono di fare una gara di velocità. Scelgono quindi di andare a correre sulla pista di go-kart che si trova a pochi chilometri di distanza dalla loro città. Il gestore della pista, per questioni di sicurezza, ha stabilito che possono correre solo 4 kart per volta e che ciascun pilota può effettuare al più 15 giri. Inoltre prima di salire sul kart il pilota deve indossare una tuta protettiva ed un casco. L'accesso agli spogliatoi per indossare la tuta è consentito a 2 persone per volta.

Scrivere un programma che simuli la gara di velocità rispettando i vincoli imposti dal gestore della pista. Ciascun pilota, rappresentato da un thread, intraprende le seguenti azioni: entra negli spogliatoi per indossare la tuta ed il casco (se non c'è posto attende), sale sul kart e entra in pista (se in pista ci sono già 4 kart attende che uno dei piloti termini i giri a disposizione), effettua 15 giri di pista, lascia la pista e torna negli spogliatoi per indossare i propri indumenti. Utilizzare il costrutto `synchronized` per gestire l'accesso concorrente alle risorse condivise.

### Esercizio 4 (6 punti)

Si consideri il seguente metodo statico Java:

```
public class Test {
    public static int foo(int[] a, int b) {
        for (int i = 0; i < a.length; i++) {
            if (a[i] < b) break;
            b++;
        }

        if (b > 5) System.out.println("B e' grande");
        System.out.println("Fine programma");
        return b;
    }

    public static void main(String[] args) {
        int a[] = {10, 2, 13};

        System.out.println(foo(a, 5));
    }
}
```

e (a) si disegni il diagramma di flusso del metodo `foo`, (b) un insieme di test (ragionevolmente minimo) per coprire tutte le decisioni (branch) e (c) si spieghi cosa produrrebbe l'esecuzione del metodo `foo` attraverso il metodo `main`.

### Esercizio 5 (6 punti)

Si consideri il codice Java seguente e si spieghi cosa viene prodotto dall'esecuzione: stampe a video e/o eventuali errori. In caso di errori, si continui comunque fino alla fine.

```
public class Punto2D {
    public double distanza(Punto2D p) {
        return 10;
    }
}

public class Punto3D extends Punto2D {
    public double distanza(Punto2D p) {
        return 30;
    }

    public double distanza(Punto3D p) {
        return 20;
    }
}

public class Test {
    public static void main(String[] args) {
        Punto2D p1 = new Punto2D();
        Punto2D p2 = new Punto2D();
        Punto2D p3 = new Punto3D();
        Punto3D p4 = new Punto3D();
        Punto3D p5 = new Punto3D();

        System.out.println(p1.distanza(p2));
        p2 = p3;
        System.out.println(p3.distanza(p2));
        p5 = (Punto3D) p2;
        System.out.println(p2.distanza(p3));
        System.out.println(p4.distanza(p1));
        System.out.println(p5.distanza(p4));
    }
}
```



# Politecnico di Milano

sede di Cremona

Anno accademico 2017-2018

## Ingegneria del Software - 3 settembre 2018

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Si scrivano le intestazioni dei seguenti metodi statici Java e la loro specifica JML:

- Il metodo prende in ingresso una matrice, di qualsiasi dimensione, di interi e restituisce *true* se ci sono (almeno) due righe, o colonne, i cui elementi hanno somma uguale. Il metodo dovrebbe terminare con un'eccezione se la matrice non esistesse, avesse problemi, o se contenesse tutti 0.
- Il metodo prende in ingresso un ArrayList di parole e restituisce *true* se le parole di posizione 1, 3, 5, ... sono palindromi di quelle di posizione 0, 2, 4, ecc. Il metodo dovrebbe restituire un'eccezione se la lunghezza dell'ArrayList avesse lunghezza dispari.
- Il metodo prende in ingresso un ArrayList di parole e restituisce un altro ArrayList di 5 interi che contiene il numero di *a*, *e*, *i*, *o*, *u* presenti nelle diverse parole e nell'ordine suddetto (sommendo le occorrenze nelle diverse parole). Il metodo deve restituire un'eccezione se l'ArrayList restituito non contenesse esattamente 5 elementi.
- Il metodo prende in ingresso un array di interi, anche negativi, e restituisce il valore più grande tra quelli ottenuti sommando tutti gli elementi dell'array, tutti meno il primo, tutti meno il primo ed il secondo, fino a considerare un array con il solo ultimo elemento.

### Esercizio 2 (6 punti)

Si consideri il tipo di dato astratto *Tessera* che definisce le pietanze degustate da un certo cliente di un ristorante, il prezzo pagato e i punti acquisiti con quel piatto. Ogni tessera può contenere pietanze per un massimo di 1000 punti o 5000 euro. Un'ipotetica implementazione (Java) fornisce anche qualche metodo: *pietanze()*, che restituisce i piatti scelti come un arrayList e *costo(Piatto p)*, che restituisce l'eventuale costo pagato per la pietanza, o 0 se il piatto non è mai stato consumato dal cliente. La classe *Pietanza*, tra gli altri fornisce anche un metodo *punti()*, che restituisce il numero di punti associati al piatto. Si definisca, quindi: (a) l'invariante pubblico della classe *Tessera*, (b) una rappresentazione per la classe stessa e (c) la specifica JML del metodo *costo()*.

### Esercizio 3 (7 punti)

Scrivere un programma multithread in Java che dato un array di 50 valori numerici ne esegue la somma in modo parallelo. Creare 5 thread e assegnare ad ogni thread una partizione dell'array (10 valori). Ogni thread esegue la somma dei propri valori e mettere il risultato in una struttura dati del thread main contenente i risultati parziali. Il thread main deve sincronizzarsi sulla terminazione (join) dei thread figli dopodiché somma i risultati parziali e li stampa su `stdout`. Si ricorda che il metodo `join` invocato su un oggetto `thread`, il thread corrente si blocca fino alla terminazione del thread associato a tale oggetto.

### Esercizio 4 (6 punti)

Si consideri il seguente metodo statico Java:

```
public class Test {
    public static void foo(int[] a, int b) {
        for (int i = 0; i < a.length; i++) {
            if (a[i] > b) break;
            b -= 20;
        }

        if (a[0] > a[1]) a[1] = a[0];
        a[0] = a[1];

        if (b > 5) System.out.println("Fine programma");
    }

    public static void main(String[] args) {
        int a[] = {10, 2, 13};

        foo(a, 50);
        for (int i : a) System.out.println(i);
    }
}
```

e (a) si disegni il diagramma di flusso del metodo `foo`, (b) un insieme di test (ragionevolmente minimo) per coprire tutte le istruzioni e tutte le decisioni (branch) e (c) si spieghi cosa produrrebbe l'esecuzione del metodo `foo` attraverso il metodo `main`.

### Esercizio 5 (6 punti)

Si progetti, usando un diagramma delle classi UML, e si implementi (almeno in parte) in Java un programma per la valutazione di polinomi. Un polinomio, identificato da una lettera minuscola dell'alfabeto, è dato dalla somma di uno o più monomi. Un monomio è caratterizzato da un segno, un coefficiente ed un grado. Si supponga di avere a disposizione un certo numero di polinomi precedentemente definiti. Il programma deve leggere il nome del polinomio  $p$  ed il valore della variabile per il quale si vuole calcolare il valore di  $p$ . Ad esempio, dato il polinomio  $f : 3x^4 - 5x^3 + 2x + 1$ , l'utente dovrebbe inserire da tastiera: `f 2` ed il programma dovrebbe stampare 13, cioè  $f(2)$ .



# Politecnico di Milano

sede di Cremona

Anno accademico 2018-2019

## Ingegneria del Software - 23 gennaio 2019

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Si scrivano le intestazioni dei seguenti metodi statici Java e la loro specifica JML:

- Il metodo prende in ingresso due array di parole di lunghezza maggiore di 2 e restituisce il numero di elementi (parole) che sono presenti nel secondo array, ma non nel primo. Chiaramente, nel caso tutte le parole del secondo array fossero presenti nel primo, il metodo restituirebbe 0.
- Il metodo prende in ingresso un ArrayList `a1` di ArrayList di interi. `a1` non deve contenere buchi, ovvero elementi nulli e la lunghezza di ogni ArrayList in `a1` deve essere non inferiore alla lunghezza del precedente. Il metodo restituisce `true` se gli elementi di ogni ArrayList sono contenuti nell'ArrayList successivo, con l'ovvia eccezione dell'ultimo elemento.
- Il metodo prende in ingresso un ArrayList di parole e due interi. L'ArrayList deve avere almeno lunghezza 1 ed i due interi devono essere positivi, con il primo più piccolo del secondo. Il metodo restituisce il numero di parole la cui lunghezza è compresa tra i due valori interi.
- Il metodo prende in ingresso un array di interi, anche negativi, e restituisce `true` se tutti gli elementi sono pari e ogni numero è la somma dei precedenti. Il metodo restituisce un'eccezione se l'array non esistesse.

### Esercizio 2 (6 punti)

Si scriva una classe Java che rappresenta un array che offre accesso concorrente ad un massimo di 10 numeri interi. La classe deve offrire un metodo `get` per consentire l'accesso agli elementi in base alla loro posizione, un metodo `add` per aggiungere un elemento in coda e un metodo `remove` per eliminare l'elemento in testa, ovvero il primo elemento. Se un oggetto esterno provasse ad aggiungere un elemento quando l'array fosse pieno, dovrebbe venir sospeso fino a quando un altro oggetto/thread non rimuove un elemento. Similmente, un oggetto che provasse a togliere un elemento da un array vuoto dovrebbe essere sospeso. Si scriva il codice della classe usando solo i meccanismi base della sincronizzazione di Java senza sfruttare classi e pacchetti di `java.util.concurrent`.

### Esercizio 3 (5 punti)

Si pensi ad un modello di oggetti che rappresenta i diversi modelli proposti da una casa automobilistica. Ogni modello, ad esempio FIAT500, implementa le operazioni definite per il generico modello secondo le proprie caratteristiche. Il sistema comprende la possibilità di aggiungere optional ai diversi modelli, ad esempio, si potrebbe aggiungere la vernice metallizzata oppure il cambio automatico o gli interni in pelle. Molti optional non sono mutuamente esclusivi e le possibili combinazioni sono tantissime. Alcuni optional, poi, potrebbero cambiare le funzionalità di base offerte dal modello generico. Ad esempio, l'aggiunta del cambio automatico potrebbe cambiare la gestione della trasmissione del mezzo. Per questa ragione, sarebbe di interesse definire un modo per aggiungere dinamicamente nuove responsabilità ad oggetto specifico, eventualmente con l'ulteriore possibilità di toglierle. Si proponga una soluzione basata sul pattern *decorator* e la si rappresenti attraverso un diagramma delle classi UML.

### Esercizio 4 (8 punti)

Si considerino le seguenti dichiarazioni di tipi Java.

```
interface Product {
    public double getPrice();
}

class Drink implements Product {
    private double price;
    public Drink(double price) { this.price = price; }
    public double getPrice() { return price; }
    public boolean moreExpensive(Drink d) { return this.getPrice() > d.getPrice(); }
}

class Coca extends Drink {
    public Coca(double price) { super(price); }
    public int getCalories() { return 200; }
}

class CocaZero extends Coca {
    public CocaZero(double price) { super(price); }
    public int getCalories() { return 0; }
    public boolean moreExpensive(CocaZero c) { return false; }
}
```

Si consideri poi il seguente codice Java che utilizza le classi sopra.

```
public class Main {
    public static void main(String[] args) {
1.        Product p;
2.        Drink d1, d2;
3.        Coca c1, c2;

4.        p = new Coca(2.0);
5.        d1 = p;
6.        d2 = new Drink(1.5);
7.        c1 = new Coca(2.1);
8.        c2 = new CocaZero(2.5);

9.        System.out.println(p.getPrice());
10.       System.out.println(d1.getPrice());
11.       System.out.println(d2.getPrice());
12.       System.out.println(c1.getPrice());
13.       System.out.println(c2.getPrice());

14.       System.out.println(p.getCalories());
15.       System.out.println(c1.getCalories());
16.       System.out.println(c2.getCalories());

17.       System.out.println(c1.moreExpensive(d2));
18.       System.out.println(c1.moreExpensive(c2));
    }
}
```

```
19.     CocaZero cz = new CocaZero(2.0);
20.     System.out.println(c2.moreExpensive(cz));
}
}
```

- Si indichino quali righe del metodo main generano un errore di compilazione e perchè.
- Supponendo che tutte le righe che provocano errori in fase di compilazione siano state rimosse, illustrare l'output del programma, spiegando quali metodi vengono invocati e perchè.

### Esercizio 5 (6 punti)

Si consideri la seguente funzione:

```
public static int fun(int a, int b) {
    int temp = 0;
    if (a > b) {
        temp++;
    }
    if (a + b > 10) {
        temp += a;
    } else {
        temp += b;
    }
    return temp;
}
```

Si definisca un insieme di casi di test per ciascuna delle seguenti richieste:

1. L'insieme copre tutte le istruzioni (statement coverage);
2. L'insieme copre tutti i branch/decisioni (branch/edge coverage);
3. L'insieme copre tutti i cammini (path coverage).



# Politecnico di Milano

sede di Cremona

Anno accademico 2018-2019

## Ingegneria del Software - 13 febbraio 2019

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Si scrivano le intestazioni dei seguenti metodi statici Java e la loro specifica JML:

- Il metodo prende in ingresso due array di parole di lunghezza maggiore di 2 e restituisce *true* se gli elementi del primo array sono presenti anche nel secondo nello stesso ordine. Non è detto che gli elementi del primo array debbano essere gli unici elementi contenuti nel secondo e, pur rispettando l'ordine, non è detto che siano contigui. Ad esempio il primo ed il secondo elemento del primo array potrebbero essere in posizione 3 e 5 nel secondo, a patto chiaramente che il terzo elemento del primo non sia presente in una posizione minore di 6.
- Il metodo prende in ingresso un array (*ap*) di array di interi (*as<sub>i</sub>*). Nessun array deve contenere elementi nulli e ogni array *as<sub>i</sub>* deve avere lunghezza non inferiore a quella di *ap*. Il metodo restituisce il numero medio di elementi uguali contenuti in ogni coppia di array *as<sub>i</sub>* contigui, ovvero la media degli elementi uguali in ogni coppia di array *as<sub>i</sub>* e *as<sub>i+1</sub>*.
- Il metodo prende in ingresso un ArrayList di parole e due interi positivi. Il metodo restituisce il numero di parole palindrome la cui posizione, nell'ArrayList, è compresa tra i due valori interi. Il metodo restituisce un'eccezione se almeno uno dei due valori fosse maggiore della lunghezza dell'ArrayList.
- Il metodo prende in ingresso un array di interi, anche negativi, e restituisce *true* se tutti gli elementi, ad eccezione del primo e dell'ultimo, sono pari al doppio del precedente. L'ultimo elemento deve contenere la somma di tutti gli elementi precedenti. Il metodo restituisce un'eccezione se l'array non esistesse.

### Esercizio 2 (6 punti)

Si scriva una classe Java che rappresenta Pila di interi con al massimo 20 elementi. La classe deve offrire i soliti metodi per interagire con una pila: push per aggiungere un elemento, top per ottenere l'ultimo elemento inserito (senza cancellarlo) e pop per cancellare l'elemento in testa, ovvero l'ultimo elemento inserito (senza restituirlo). Se un oggetto esterno provasse ad aggiungere un elemento quando la pila fosse piena, dovrebbe venir sospeso fino a quando un altro oggetto/thread non rimuovesse un elemento. Similmente, anche un oggetto che provasse a chiamare top o pop su una pila vuota dovrebbe essere sospeso. Si scriva il codice della classe usando solo i meccanismi base della sincronizzazione di Java senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`.

### Esercizio 3 (5 punti)

Si supponga che esista l'interfaccia Animale (per rappresentare il generico animale) che offre il solo metodo `verso()` per ottenere il verso di un animale. Le classi che implementeranno Animale dovranno fornire un'implementazione specifica del metodo `verso()` poiché il verso di un animale è differente per ogni specie. Si scriva quindi il codice Java che, utilizzando il pattern *Factory Method* consenta di creare dinamicamente mucche, cavalli e maiali in base ad un semplice attributo (stringa) che definisce il loro tipo. Si scriva anche il codice delle classi che implementano Animale richieste ed un semplice `main` che mostri come utilizzare il pattern.

### Esercizio 4 (8 punti)

Si considerino le seguenti dichiarazioni di tipi Java.

```
public class Animale {
    private String nome;

    public Animale(String nome) {
        this.nome = nome;
    }

    public void parla() {}
    public String getNome() {return nome;}
    public void incontra(Animale a) {
        System.out.println(nome + ": Ciao, " + a.nome);
    }
}

public class Topo extends Animale {
    public Topo(String nome) {
        super(nome);
    }

    public void parla() {
        System.out.println("Squit");
    }

    public void incontra(Gatto g) {
        System.out.println(getNome() + ": Aiutoooooo!!!!");
        parla();
    }
}

public class Gatto extends Animale {
    public Gatto(String nome) {
        super(nome);
    }

    public void parla() {
        System.out.println("Miao");
    }

    public void incontra(Topo t) {
        System.out.println(getNome() + ": Che buono !!!");
        parla();
    }
}

public class Test {
    public static void main(String[] args) {
        Gatto g = new Gatto("Gatto");
        Topo t = new Topo("Topo");
    }
}
```

```

Topo tg = new Gatto("GattoTopo")

Animale a = new Animale("Animale");
Animale a1 = new Gatto("Gatto");
    Animale a2 = new Topo("Topo");

a.incontra(g);
    t.incontra(t);
    t.incontra(g);
tg.incontra(a1);
    a1.incontra(a2);
    a2.incontra(g);
    g.incontra(g);
}
}

```

e si risponda, motivando brevemente le risposte, alle seguenti domande:

- Dal momento che il metodo parla di Animale non fa nulla, avremmo potuto ometterlo?
- Quali righe del main darebbero errore?
- Togliendo le linee di codice sbagliate, cosa produrrebbe l'esecuzione del main?

### Esercizio 5 (6 punti)

Si consideri la seguente funzione:

```

public static void metodo(int a, int b) {
    int r = 0;

    if (a > b) a -= b;

    while (a > b) a--;

    if (a == 0) r = b;

    r = a + b;
    System.out.println(r);
}

```

Si definisca il *control-flow diagram* e un insieme di casi di test per ciascuna delle seguenti richieste, motivando brevemente le risposte.

1. L'insieme copre tutte le istruzioni (statement coverage);
2. L'insieme copre tutti i branch/decisioni (branch/edge coverage);
3. L'insieme copre tutti i cammini (path coverage).



# Politecnico di Milano

sede di Cremona

Anno accademico 2018-2019

## Ingegneria del Software - 19 giugno 2019

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Il metodo statico `sudoku()` prende in ingresso la solita griglia  $9 \times 9$  del Sudoku, come array di `int` bidimensionale, e restituisce un booleano: `true` se la configurazione corrente della griglia non viola le regole del gioco; `false` altrimenti.

Supponendo che il metodo venga usato per controllare solo griglie complete, si defiscano:

- gli elementi della precondizione, ovvero quanto si deve dire per essere certi che la griglia abbia la “forma” corretta e che contenga solo numeri da 1 a 9 (si ricordi l’ipotesi di griglia completa).
- gli elementi della postcondizione per dire che nessuna riga e nessuna colonna contiene numeri ripetuti.
- gli elementi della postcondizione per dire che nessuna delle 9 sotto-regioni  $3 \times 3$  contiene numeri ripetuti. Per convenienza, è possibile definire un metodo `helper` se necessario. Nel caso, è richiesta l’implementazione completa del metodo.

Si spieghi anche, a parole, in modo breve ma preciso, cosa cambierebbe se il metodo fosse usato anche per controllare griglie incomplete.

### Esercizio 2 (6 punti)

Si scriva una classe Java che rappresenta una lista (FIFO) concorrente e di 10 elementi al massimo. La classe consente ad oggetti/thread di accedere alla struttura dati in modo concorrente, senza chiaramente violare le regole di base relative agli accessi multipli. La classe offre i metodi `leggi`, per leggere il primo elemento utile (il primo inserito ed ancora presente) senza rimuoverlo, `scrivi`, per scrivere un elemento (e metterlo in ultima posizione), e `cancella` per rimuovere l’elemento in testa alla lista. Non deve chiaramente essere possibile leggere o cancellare elementi da una lista vuota o scriverne di nuovi in una lista piena. La classe deve fornire anche un metodo `vuota` che restituisce lo stato della lista: `true` se la lista è vuota; `false` altrimenti. Si scriva la struttura dati per realizzare la lista ed il codice della classe usando solo i meccanismi base della sincronizzazione di Java senza sfruttare classi e pacchetti di `java.util.concurrent`.

### Esercizio 3 (5 punti)

La catena *PastaWow* vuole offrire ai propri clienti un'esperienza completamente personalizzabile. Per questo motivo, i suoi ristoranti vendono pasta, non serve identificarne il tipo, ed un insieme di condimenti che il cliente può aggiungere a piacere: pomodoro, pesto, funghi, piselli e panna. Anche se chiaramente un pesto con la panna potrebbe sembrare un insulto, in prima approssimazione non ci sono vincoli. L'app data ai camerieri deve essere in grado di calcolare automaticamente il prezzo del piatto ordinato, partendo dal costo della pasta e aggiungendo i condimenti selezionati. Si realizzi quindi il semplice codice Java che, usando il pattern *Decorator*, realizzi l'interfaccia *Pietanza* che fornisce l'unico metodo *costo()*. Si spieghi anche, sempre a parole, in modo breve e preciso, cosa cambierebbe se si volessero aggiungere dei vincoli tra i diversi condimenti (ad esempio, si volesse vietare il suddetto pesto con la panna).

### Esercizio 4 (8 punti)

Si considerino le seguenti dichiarazioni di tipi Java.

```
public abstract class Gradi {
    int valore;

    public abstract Gradi converti();
    public abstract Gradi somma(Gradi g);
}

public class GradiCelsius extends Gradi {
    public GradiCelsius(int i) {
        valore = i;
    }

    public Gradi converti() {
        return new GradiFahrenheit(((9*this.valore)/5)+32);
    }

    public Gradi somma(Gradi g) {
        return new GradiCelsius(this.valore+g.converti().valore);
    }
}

public class GradiFahrenheit extends Gradi {
    public GradiFahrenheit(int i) {
        valore = i;
    }

    public Gradi converti() {
        return new GradiCelsius((5*(this.valore-32))/9);
    }

    public Gradi somma(Gradi g) {
        return new GradiCelsius(this.valore+g.converti().valore);
    }
}

public class Main {
    public static void main(String[] args) {
        Gradi g1 = new GradiCelsius(10);
        Gradi g2 = new GradiFahrenheit(32);
        GradiCelsius gc = new GradiCelsius(0);
        GradiFahrenheit gf;

        System.out.println(gc.converti().valore);
        gf = gc;
        System.out.println(gf.converti().valore);
        gf = new GradiFahrenheit(32);
```

```

        System.out.println(gc.somma(gf).valore);
        System.out.println(g1.converti().valore);
        System.out.println(g2.converti().valore);
        System.out.println((g1.somma(g2)).valore);
        System.out.println((g1.somma(g1)).valore);
        System.out.println((g2.somma(g1)).valore);
    }
}

```

e si risponda, motivando brevemente le risposte, alle seguenti domande:

- Quali righe del `main` darebbero errore in compilazione?
- Togliendo le linee di codice sbagliate, cosa produrrebbe l'esecuzione del `main`?
- Quali righe del `main` darebbero un risultato sbagliato?

### Esercizio 5 (6 punti)

Si consideri la seguente funzione:

```

public static int metodo(int a, int b) {
    int r = 0;
    if (a < 0) a = -a;
    a = a % 3;
    while (a > 0) {
        r = r + b;
        a = a -1;
    }
    return r;
}

```

Si definisca il *control-flow diagram* e un insieme di casi di test per ciascuna delle seguenti richieste, motivando brevemente le risposte.

1. L'insieme copre tutte le istruzioni (statement coverage);
2. L'insieme copre tutti i branch/decisioni (branch/edge coverage);
3. L'insieme copre tutti i cammini (path coverage).



# Politecnico di Milano

sede di Cremona

Anno accademico 2018-2019

## Ingegneria del Software - 18 luglio 2019

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (8 punti)

Si scrivano le pre- e post-condizioni in JML dei seguenti metodi:

1. un metodo che prende in ingresso una matrice di caratteri e controlla che gli elementi di ogni riga e di ogni colonna rispettino l'ordine alfabetico.
2. un metodo che prende in ingresso un array di interi e controlla che ogni elemento, ad eccezione del primo e dell'ultimo, sia il doppio del precedente. L'ultimo deve essere la somma di tutti gli elementi precedenti.
3. un metodo che prende in ingresso due stringhe, restituisce *true* se tutti i caratteri della seconda stringa compaiono nella prima e solleva un'eccezione se una delle due stringhe fosse *null*.
4. un metodo che prende in ingresso un ArrayList di interi e restituisce il numero massimo di ripetizioni dello stesso numero.

### Esercizio 2 (6 punti)

Si scriva una classe Java che rappresenta il portafoglio di un investitore in borsa. La dimensione è fissata al momento della creazione. Ogni elemento del portafoglio comprende il nome di una azione, il suo valore nominale e la variazione rispetto alla quotazione precedente (rendimento). Il portafoglio può essere acceduto in modo concorrente da operatori (thread) diversi per cambiare il valore nominale ed il rendimento e per calcolare il rendimento medio dei titoli in portafoglio. Si supponga anche di voler realizzare un metodo che restituisca il rendimento medio solo se in quel momento fosse positivo, o si sospenda in caso contrario (aspettando che il rendimento medio diventi positivo). Si scriva la struttura dati per realizzare il portafoglio ed il codice della classe usando solo i meccanismi base della sincronizzazione di Java senza sfruttare classi e pacchetti di `java.util.concurrent`.

### Esercizio 3 (6 punti)

Un sistema di geolocalizzazione offre un’interfaccia standard che comprende il solo metodo `Location getLocation()`. Il sistema però può fornire l’effettiva posizione usando tre modalità diverse: l’antenna GPS, gli access point WiFi e sensori di prossimità specifici. La selezione avviene in base alle informazioni fornite da una classe `Contesto`, che fornisce tre metodi booleani `hasXXX()`, dove XXX può essere GPS, WiFi, o Sens. Solo uno dei tre metodi può restituire `true` per il medesimo contesto. Mediante l’uso di appositi design pattern, progettare (diagramma delle classi UML) una soluzione che:

- Nasconde agli utilizzatori la scelta della specifica implementazione di `Location getLocation()` e semplifichi l’evoluzione del sistema nel caso nuove implementazioni dell’interfaccia diventino disponibili in futuro.
- Fornisca informazioni relative alla posizione solamente in base allo stato di privacy dell’utente (per semplicità si supponga di avere solo due stati *on/off*). Se l’utente è nello stato *off*, il sistema fornisce la posizione corretta; se è nello stato *on*, il sistema restituisce sempre una posizione fittizia di default.

### Esercizio 4 (6 punti)

1. Si considerino le classi `RecentContacts` e `Contracts`. La prima modella la lista dei contatti di una chat ordinati sulla base dell’ultima interazione (dalla più alla meno recente). La seconda mantiene la lista dei contatti senza un ordine preciso. `Contracts` può essere sottotipo di `RecentContacts` secondo il principio di sostituibilità di Liskov? È possibile il contrario?
2. Si consideri la classe `Connect4` che modella una partita al noto gioco *Forza 4*. Una pedina viene depositata invocando il metodo `put` ed indicando la colonna in cui si vuole depositare la pedina (nell’intervallo 1-7) e l’identificativo del giocatore. Si consideri poi la classe `Space4` che modella una variante di `Connect4` seconda la quale `put` posiziona la pedina del giocatore indicato in una posizione libera casuale (della colonna indicata). `Space4` può essere un sottotipo di `Connect4` secondo il principio di sostituibilità di Liskov? È possibile il contrario?
3. Si considerino le classi `Sequenza` e `SequenzaOrdinata` (di parole). Il metodo `inserisci` della prima classe aggiunge gli elementi alla sequenza secondo l’ordine d’arrivo. Il metodo della seconda, al contrario, dispone gli elementi in base all’ordine alfabetico. `SequenzaOrdinata` può essere un sottotipo di `Sequenza` secondo il principio di sostituibilità di Liskov? È possibile il contrario?

Motivare brevemente le risposte.

### Esercizio 5 (7 punti)

Si consideri la seguente funzione:

```
public static int metodo(int a, int b) {  
    int r = 3;  
  
    if (a < 0) a = -a/2;  
  
    for (int i = 0; i < a; i++) {  
        if (b > a) r = r*3;  
        r = r+4;  
    }  
  
    return r;  
}
```

Si definisca il *control-flow diagram* e un insieme di casi di test per ciascuna delle seguenti richieste, motivando brevemente le risposte.

1. L’insieme che copre tutte le istruzioni (statement coverage);
2. L’insieme che copre tutti i branch/decisioni (branch/edge coverage);
3. L’insieme che copre tutti i cammini.



# Politecnico di Milano

sede di Cremona

Anno accademico 2018-2019

## Ingegneria del Software - 12 settembre 2019

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (9 punti)

Si scrivano le pre- e post-condizioni in JML per i seguenti metodi statici:

- `boolean diagonali(int [][] a)` che riceve come parametro una matrice quadrata di interi e restituisce `true` se le somme dei valori sulle due diagonali sono uguali, `false` altrimenti.
- `boolean maggiori(ArrayList<Integer> a, int p)` che riceve come parametri un `ArrayList` di interi e una posizione. Il metodo restituisce `true` se `p` non è maggiore della lunghezza di `a` e se tutti gli elementi in posizione minore di `p` sono minori di tutti quelli in posizione maggiore di `p`; restituisce `false` altrimenti.
- `boolean riquadri(int [][] a, int p)` che riceve come parametri una matrice quadrata di interi e un intero. Se `p` è minore della dimensione della matrice, la coppia di indici  $(p, p)$  identifica un elemento  $e$  sulla diagonale della matrice e divide la diagonale medesima in due parti  $d_1$  e  $d_2$  (e escluso). Il metodo restituisce `true` se, e solo se, la somma degli elementi della sotto-matrice la cui diagonale è  $d_1$  è uguale alla somma degli elementi della sotto-matrice la cui diagonale è  $d_2$ .

### Esercizio 2 (6 punti)

Si scriva una classe Java che rappresenta una catena di montaggio robotizzata che può gestire un numero fisso, definito in fase di *deployment* del sistema, di artefatti. La catena di montaggio accetta nuovi elementi se non è piena e consente di prelevare l'artefatto finito dopo un tempo di lavorazione variabile. L'utente deve poter sempre conoscere il numero di elementi in lavorazione nella catena di montaggio e l'artefatto in lavorazione da più tempo. Si scrivano le classi Java per realizzare la catena di montaggio ipotizzando che gli artefatti siano thread indipendenti e la classe `CatenaMontaggio` debba sincronizzarne l'accesso al sistema di lavorazione e debba consentire all'utente di avere accesso a dati "corretti". Si risolva il problema usando solo i meccanismi base della sincronizzazione di Java senza sfruttare classi e pacchetti di `java.util.concurrent`.

### Esercizio 3 (6 punti)

Si progetti il diagramma delle classi UML per realizzare una semplice applicazione che opera su array di interi. Oltre alle solite operazioni di somma degli elementi, ricerca del valore minimo, medio e massimo e di calcolo della mediana, l'applicazione deve consentire la visualizzazione degli array in modi diversi. Ad esempio, sarebbe possibile usare il modo compatto (3, 45, -8, 56), oppure quello standard ( $a[0] = 3, a[1] = 45, a[2] = -8, a[3] = 56$ ). Si vorrebbe anche poter aggiungere ulteriori formati di stampa personalizzati senza stravolgere le classi esistenti. Come si potrebbe realizzare la struttura delle classi per essere certi di avere un solo motore di calcolo e per poter variare le opzioni di stampa in modo semplice? Quali *design pattern* utilizzereste?

### Esercizio 4 (6 punti)

Il *Guardaroba* di una persona contiene abiti indossabili sempre (istanze di *Abito*) e abiti stagionali (*AbitoStagionale*), adatti solo per una o più stagioni:

1. Volendo definire un'opportuna gerarchia di abiti, e supponendo che *boolean indossabile(Stagione s)* sia definito per ogni abito, sarebbe possibile definire una gerarchia di ereditarietà che rispetti il principio di sostituibilità di Liskov?
2. Volendo definire un iteratore *abitiEstivi()* per la classe *Guardaroba* che restituisca i soli abiti adatti alla stagione estiva, quali classi definireste?
3. L'uso di una classe astratta semplificherebbe la realizzazione dell'iteratore?

Motivare brevemente le risposte.

### Esercizio 5 (6 punti)

Si consideri il seguente metodo Java:

```
public static int sum(List<Integer> list, boolean discardNeg) {  
    int result = 0;  
    for (Integer i : list) {  
        if (i > 0) result += i;  
        else if (!discardNeg) result += i;  
    }  
    return result;  
}
```

Si definisca il *control-flow diagram* e un insieme di casi di test che copra tutte le istruzioni (statement coverage) e tutti i branch/decisioni (branch/edge coverage);



# Politecnico di Milano

sede di Cremona

Anno accademico 2018-2019

## Ingegneria del Software - 14 gennaio 2020

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici e della sezione comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (9 punti)

Si pensi a due classi Java che implementano i concetti di `MazzoDiCarte` e `Carta`, considerando un mazzo da 54 carte ed i soliti 4 semi: 13 carte per seme più due jolly. Ogni carta è definita dal seme e dal valore (da 1 a 10, fante, cavallo e re); ogni mazzo contiene 52 carte ed ha un colore. A fronte di queste informazioni:

- Si definisca una rappresentazione (plausibile) per le due classi in Java.
- Si definiscano gli invarianti privati delle due classi in JML.
- Si definisca la specifica JML del metodo `mescola` della classe `Mazzo`, che mescola "virtualmente" il mazzo di carte, con il vincolo che almeno il 50% delle carte del mazzo sia in una posizione diversa. Si definiscano eventuali metodi *pure*, se necessario.

### Esercizio 2 (6 punti)

Un gruppo di 1500 persone vorrebbe comprare i biglietti per il concerto di LP. Al botteghino sono a disposizione solo 500 biglietti e vi sono solo 5 persone che li vendono. Questo significa che solo 5 persone alla volta potranno comprare i biglietti e solo se ve ne fossero ancora a disposizione. Ogni persona impegna un tempo variabile (casuale) per comprare un biglietto e può comprare fino ad un massimo di 3 biglietti. Ogni persona, quindi, prova a comprare i biglietti richiesti, aspetta, se tutti i vendori sono occupati, e, appena possibile, conclude l'acquisto, se i biglietti rimasti sono sufficienti. Se i biglietti disponibili fossero in numero inferiore a quelli richiesti, la persona comprerebbe comunque tutti i biglietti disponibili. Se non ci fossero più biglietti disponibili, la persona semplicemente lascerebbe il sistema. Si implementi il sistema usando solo i meccanismi base della sincronizzazione di Java senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`.

### Esercizio 3 (6 punti)

Un semplice programma di disegno offre all'utente alcuni strumenti di base: matita, gomma, pennello, per svolgere le diverse funzioni di disegno. Il programma deve trasformare gli eventi del mouse in modo opportuno in funzione dello strumento selezionato dall'utente. Chiaramente, se l'utente seleziona la matita, eventuali click del mouse disegnano una linea, mentre se l'utente avesse selezionato la gomma, i medesimi click cancellerebbero zone dello schermo. Il progettista dell'applicazione

vorrebbe: (a) poter aggiungere facilmente ulteriori strumenti di disegno ed (b) associare i comportamenti corretti alle azioni del mouse in base allo strumento selezionato. Come si potrebbe realizzare la struttura delle classi (diagramma delle classi UML) per avere le funzionalità richieste? Quali *design pattern* utilizzereste?

### Esercizio 4 (6 punti)

Si consideri il seguente frammento di codice Java.

```
public class A {
    public void g(A x) {System.out.println("called on instance of A");}
}

public class B extends A{
    public void g(A x) {System.out.println("called 1 on instance of B");}
    public void g(B x) {System.out.println("called 2 on instance of B");}
}

public class Test {
    public static void f(A x) {System.out.println((x instanceof A)+" A");}
    public static void f(B x) {System.out.println((x instanceof B)+" B");}

    public static void main (String s[]) {
        A a=new A();
        A ab=new B();
        B b=new B();

        f(a);
        f(ab);
        f(b);
        f((A)b);

        b.g(a);
        ab.g(ab);
        b.g(b);
    }
}
```

Cosa stampa il programma? Motivare brevemente le risposte.

### Esercizio 5 (6 punti)

Si consideri il seguente metodo Java:

```
public static boolean foo(List<Integer> list, boolean par) {
    int sum = 0;
    int sumEven = 0;
    boolean result;

    for (Integer i: list) {
        if (i%2 == 0) sumEven += i;
        sum += i;
    }

    result = par ? sumEven == sum : sumEven != sum;
    return result;
}
```

Si definisca il *control-flow diagram* e un insieme di casi di test che copra tutte le istruzioni (statement coverage) e tutti i branch/decisioni (branch/edge coverage);



# Politecnico di Milano

sede di Cremona

Anno accademico 2019-2020

## Ingegneria del Software - 04 febbraio 2020

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. La mancata indicazione dei dati anagrafici comporta l'annullamento del compito.
2. È possibile scrivere in matita. Non è possibile lasciare l'aula conservando il tema della prova in corso.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 2h.

### Esercizio 1 (9 punti)

Un comune virtuoso vuole facilitare la raccolta differenziata e ragiona sui concetti di Contenitore e Spazzatura. Ogni contenitore può accogliere solamente spazzatura di un certo tipo (carta, vetro, metallo, indifferenziata) fino al massimo delle sua capienza. Ogni elemento Spazzatura deve essere classificato in base ad una delle quattro tipologie precedenti ed ha un volume. L'idea è che ogni cittadino venga dotato di 4 contenitori intelligenti. A fronte di queste informazioni:

- Si definisca una rappresentazione (plausibile) per le due classi in Java.
- Si definiscano gli invarianti privati delle due classi in JML.
- Si definisca la specifica JML del metodo `gettaVia` della classe `Contenitore`, che consente all'utente di buttare della spazzatura di tipo opportuno nel contenitore preposto. Il metodo non deve consentire di buttare un elemento nel contenitore sbagliato e neppure di buttare un elemento nel contenitore giusto se il suo volume supera la capienza rimasta.

### Esercizio 2 (6 punti)

Una pasticceria ha deciso di aprire uno store online per vendere i propri panettoni. Ogni cliente può comprare al massimo 3 panettoni per volta, ma il sistema, sperimentale, non è in grado di servire più di 5 richieste in parallelo. Inoltre, la pasticceria mette in vendita una quantità limitata di panettoni, come primo esperimento. Il sistema, quindi, può consentire all'utente di acquistare i panettoni richiesti, può non accettare l'ordine perché la quantità richiesta è eccessiva, ma può anche rifiutare l'ordine perché i panettoni rimasti non sono sufficienti. Se i panettoni rimasti sono meno di quelli richiesti, il sistema deve rifiutare l'intera transizione. Si esemplifichi il funzionamento del sistema ipotizzando 50 panettoni in vendita per 30 possibili clienti. L'arrivo dei clienti ed il tempo d'acquisto può essere modellato con un ritardo casuale. Si implementi il sistema usando solo i meccanismi base della sincronizzazione di Java senza sfruttare classi e pacchetti di `java.util.concurrent`.

### Esercizio 3 (6 punti)

Un'azienda che vende giacche invernali deve gestire la configurazione dei propri prodotti. Ogni giacca, oltre ad avere un tessuto ed un colore, può avere tasche normali o esterne, tasche interne e tasche doppie. Ogni giacca può avere il cappuccio, il collo alla coreana, oppure il collo normale. Infine, la chiusura può essere attraverso bottoni, di diverse tipologie, o zip. L'azienda

vuole realizzare un'applicazione per gestire tutte le configurations attuali e poter anche, magari, aggiungere ulteriori elementi in futuro. Come si potrebbe realizzare la struttura delle classi (diagramma delle classi UML) per avere le funzionalità richieste? Quali *design pattern* utilizzereste?

### Esercizio 4 (6 punti)

Si consideri il seguente metodo statico Java.

```
public static int foo(Integer v, List<Integer> values) {  
    int sum = 0;  
  
    if (values == null || v == null) return 0;  
  
    for (Integer x : values) {  
        if (x%3 == 0) sum += v;  
        else sum += x%3;  
    }  
    return sum;  
}
```

e si riscriva la stessa funzione in stile funzionale.

### Esercizio 5 (6 punti)

Si consideri il seguente metodo Java:

```
1- static int test(int x, int y) {  
2-     if (x>=0 && y>0)  
3-         do {  
4-             if (x%2 ==0)  
5-                 y++;  
6-             else  
7-                 y--;  
8-             x--;  
9-         } while (x!=1);  
10-    if (y == 0 || x < 1)  
11-        y = x+3;  
12-    return y;  
13- }
```

Si definisca il *control-flow diagram* e un insieme di dati di test minimo che copre tutte le istruzioni (statement coverage) e tutti i branch/decisioni (branch/edge coverage). Si identifichi anche un test (specifico) per eseguire, se possibile, il cammino 2, 4, 5, 8, 9, 10, e 12.



# Politecnico di Milano

sede di Cremona

Anno accademico 2019-2020

## Ingegneria del Software - 12 giugno 2020

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti. È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello**
5. Tempo a disposizione: 90 min.

### Esercizio 1 (13 punti)

#### Parte A (9 punti)

Si scrivano le pre- e post-condizioni in JML per i seguenti metodi statici:

- Il metodo `m1` prende in ingresso un `arrayList` di `Stringhe` e restituisce un altro `arrayList` di `Stringhe`. Il risultato deve contenere solo il sottoinsieme delle stringhe in input di lunghezza pari e che non contengano più di 2 a. Per semplicità si supponga che le stringhe contengano solo caratteri minuscoli.
- Il metodo `m2` prende in ingresso un array di caratteri e restituisce `true` se l'array contiene almeno un carattere ripetuto, `false` altrimenti.
- Il metodo `m3` prende in ingresso un `arralist` di interi e un numero intero `n`. Il metodo aggiunge l'elemento all'`arrayList` se l'`arrayList` stesso non contiene già `n` o un numero divisibile per `n`.

#### Parte B (4 punti)

Si definisca una semplice classe `Rettangolo`. Ogni rettangolo è caratterizzato da un'altezza e da una larghezza e non può mai essere degenere, ovvero avere lati lunghi 0 o diventare un quadrato.

Si usi JML per:

- Definire l'invariante privato della classe `Rettangolo`.
- Ipotizzando che esista un costruttore, che non deve essere definito, che consente di creare rettangoli di altezza e larghezza dati, definire le pre- e post-condizioni di un metodo `cambiaLarghezza` che consente di cambiare la sola larghezza del rettangolo.

### Esercizio 2 (8 punti)

Un gruppo di 100 turisti si recano al Louvre per vedere la Gioconda. Per motivi di sicurezza possono entrare nel museo solo 10 persone alla volta. I turisti attendono un tempo casuale prima di decidere di entrare. Altrettanto casualmente, alcuni di loro poi guardano la Gioconda per 1 minuto, altri per 2 minuti ed altri ancora stufi, appena tocca a loro, lasciano la sala immediatamente. Simulare la suddetta situazione in Java, utilizzando solo i meccanismi base della sincronizzazione di Java senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`.

### Esercizio 3 (4 punti)

Si consideri il seguente metodo statico Java.

```
public static int method(int v, List<String> values) {  
    int chars = 0;  
  
    if (values == null || v == 0) return 0;  
  
    for (String s : values) {  
        if (s.length() >= 3)  
            if (s.charAt(0) != 'A') chars += s.length();  
            else chars += 1;  
    }  
    return chars;  
}
```

e si riscriva la stessa funzione in stile funzionale.

### Esercizio 4 (8 punti)

Si consideri il seguente metodo Java:

```
static void foo(String s, int v) {  
    int l, p = 0;  
  
    if (s == null) return;  
    l = s.length();  
    if (l == 0) return;  
  
    if (v > l) v = v%l;  
  
    while (p < v) {  
        if (s.charAt(p) == 'a') {  
            System.out.println("trovato");  
            return;  
        }  
        p++;  
    }  
    System.out.println("non trovato");  
}
```

Si definisca il *control-flow diagram* e un insieme di dati di test minimo che copra tutte le istruzioni (statement coverage) e tutti i branch/decisioni (branch/edge coverage). Si identifichi anche un test (specifico) per eseguire, se possibile, il ciclo while 2 volte e: (a) senza trovare la lettera a e (b) trovandola.



# Politecnico di Milano

sede di Cremona

Anno accademico 2019-2020

## Ingegneria del Software - 7 luglio 2020

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver condiviso lo schermo). È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
5. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 90 min.

### Esercizio 1 (9 punti)

Si scrivano le pre- e post-condizioni in JML per i seguenti metodi statici:

- Il metodo `m1` prende in ingresso un `ArrayList<Integer> ai` di interi, anche ripetuti, e un intero `n` e restituisce un altro `ArrayList<Integer>` di interi. Il risultato deve contenere solo gli interi in `ai` ripetuti almeno `n` volte. Ogni intero nell'`ArrayList` restituito deve comparire una volta sola.
- Il metodo `m2` prende in ingresso un array di caratteri e restituisce `true` se i caratteri nell'array rispettano l'ordine alfabetico, `false` altrimenti.
- Il metodo `m3` prende in ingresso un array di stringhe ed un'ulteriore stringa. Il metodo non restituisce nulla e aggiunge la stringa all'array solo nel caso in cui nell'array non esista già una stringa che inizi con lo stesso carattere.

### Esercizio 2 (4 punti)

- Si definisca una semplice classe `Porta`. Ogni porta è caratterizzata da un'altezza, da una larghezza e non può mai essere degenere, ovvero avere una dimensione pari a 0. Ogni porta poi ha anche la capacità di aprirsi: se è chiusa si apre e se è aperta può essere chiusa. Se si prova ad aprire una porta aperta, questa resta aperta e se si prova a chiudere una porta chiusa, questa resta chiusa.
- Ipotizzando che esista la classe `PortaMagica`, che richiede una parola magica per potersi aprire (chiudere), dovrebbe essere una super o sotto classe di `Porta`? Perché?
- Si usi JML per definire le pre- e post-condizioni del metodo `apri(String pm)` di `PortaMagica`, coerentemente con la definizione delle classi data in precedenza.

### Esercizio 3 (8 punti)

Il supermercato SM si è appena dotato di un sistema di casse intelligente. Le persone in attesa formano un'unica fila, mentre il sistema individua di volta in volta la cassa, tra quelle disponibili, che può servire il cliente. Si modelli il suddetto sistema in Java, utilizzando solo i meccanismi base della sincronizzazione di Java senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`. Chiaramente, il tempo speso da ogni utente per le operazioni di pagamento varia di volta in volta: ogni cassa viene prima occupata dal cliente e poi liberata a pagamento avvenuto. Si esemplifichi anche il comportamento del sistema supponendo di avere 30 clienti in attesa e 10 casse a disposizione.

### Esercizio 4 (4 punti)

Data una lista di stringhe `List<String> list`, si definisca un metodo in stile funzionale che conti e restituisca il numero di consonanti contenute nelle stringhe.

### Esercizio 5 (8 punti)

Si consideri il seguente metodo Java che prende in ingresso un array di interi e ne restituisce la media se l'array contiene almeno 4 elementi, oppure -1 se l'array fosse troppo piccolo.

```
private static int media(Integer[] a) {
    int somma = 0;

    if (a.length < 3) return -1;

    for (int i: a)
        somma += i;

    return somma/a.length;
}
```

Si definisca:

- Il *control-flow diagram* del codice proposto;
- Le percentuali di copertura delle istruzione dei seguenti test:  $\{1,2,3\}/2$ ,  $\{1,1\}/-1$ ,  $\{1,2,3,4,5\}/3$ ,  $\{-1,-2,-3,-4,-1\}/-2.2$  e  $\{\}/-1$ . Il formato è: {valori in input}/risultato.
- La correttezza dei test proposti. Ci sono test che darebbero errore? Perché?



## Ingegneria del Software - 11 settembre 2020

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver condiviso lo schermo). È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
5. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 2 h.

### Esercizio 1 (5 punti)

Il metodo statico `controlla` riceve in ingresso un array `nums` e un intero `n` positivo e restituisce un valore booleano. L'array `nums`, di lunghezza non precisata, contiene dei numeri interi, per ipotesi tutti diversi tra loro. Il metodo restituisce `true` se ognuno dei primi `n` numeri dell'array è più piccolo di ognuno dei numeri che occupano le posizioni da `n` alla dimensione dell'array; restituisce `false` negli altri casi. L'array non viene modificato.

- Si fornisca la specifica JML del metodo. Si rammenta che la specifica deve considerare i casi in cui l'array sia nullo o di lunghezza minore di `n` e quello in cui gli elementi non siano tutti distinti fra loro.
- Si trasformi la specifica JML del punto precedente sostituendo una a scelta delle precondizioni nel lancio di opportune eccezioni.

### Esercizio 2 (6 punti)

Si consideri la classe `NumeroBinario` che rappresenta numeri interi, positivi e senza segno in binario (ad esempio,  $1110_2$  corrisponde al numero  $14_{10}$ ). Si supponga che la classe fornisca:

- il metodo `int dimensione()`, per conoscere il numero di cifre del `NumeroBinario` considerato;
- il costruttore `NumeroBinario(int n)`, per creare un `NumeroBinario` di `n` bit inizializzati a 0;
- il metodo `void cambiaBit(int pos, int val)`, per assegnare al bit di posizione `pos` il valore `val`;
- due iteratori per scorrere le cifre binarie dalla meno significativa alla più significativa e dalla più significativa alla meno significativa: `Iterator<Integer> destraSinistra()` e `Iterator<Integer> sinistraDestra()`, rispettivamente.

Utilizzando solamente i metodi elencati sopra, si realizzino i seguenti metodi:

- `somma` per sommare due numeri binari e ottenere un terzo numero binario. Si consideri l'eventuale riporto, prevedendo quindi un numero congruo di cifre per il `NumeroBinario` somma.
- `simmetrico` per sapere se la sequenza di bit è simmetrica (palindroma)

### Esercizio 3 (6 punti)

Si considerino in Java le classi `Film` e `FilmMuto` e si discuta se `Film` debba essere sottoclasse di `FilmMuto` o viceversa nei seguenti casi alternativi:

- `Film` ha due metodi `startPlayingSound()` e `startPlayingVideo()` mentre `FilmMuto` ha solo il metodo `startPlayingVideo()`.
- entrambe le classi hanno un solo metodo `startPlaying()` e inoltre esistono due metodi osservatori `hasVideo()` e `hasAudio()`. Il metodo `startPlaying()` ha precondizione `true` per entrambe le classi. Per la classe `Film` ha post-condizione `hasVideo() && hasAudio()`, mentre per la classe `FilmMuto` ha post-condizione `hasVideo()`.
- come sopra, ma la post-condizione di `startPlaying()` di `FilmMuto`, è `hasVideo() && !hasAudio()`.

### Esercizio 4 (5 punti)

Data una lista di stringhe `List<String> list`, si definisca un metodo in stile funzionale che restituisca il numero di stringhe di lunghezza pari che iniziano per `a`.

### Esercizio 5 (5 punti)

Un corso di studi è costituito da insegnamenti, studenti e informazioni relative. Esistono diverse tipologie di corsi di studio; in particolare i corsi di laurea e quelli di laurea specialistica. Uno studente può essere iscritto a un corso di laurea o a un corso di laurea specialistica, di cui può frequentare alcuni insegnamenti. Gli insegnamenti possono essere comuni a più corsi di laurea o laurea specialistica, ma ogni studente si può iscrivere a un solo corso di laurea o di laurea specialistica. Un corso di laurea specialistica è esattamente come un corso di laurea, ma ha una fase iniziale in cui si decide se ammettere o meno uno studente in base alla sua carriera. Si modelli la specifica fornita con un diagramma delle classi UML, ponendo particolare attenzione ai metodi e attributi principali delle diverse classi e alle relazioni esistenti fra esse.

### Esercizio 6 (6 punti)

Si consideri il seguente frammento di codice Java:

```
static String tParallelogramma(float l1, float l2, float l3, float l4, float d1, float d2) {  
    if (l1==l3) && (l2==l4)  
        if (l1==l2) && (d1==d2) return "quadrato";  
        else if (d1 != d2) return "rombo";  
        else return "rettangolo";  
    else return "altroParallelogramma";  
}
```

è facile vedere che l'implementazione è scorretta, perché non riconosce il caso `altroParallelogramma`. Si deve quindi:

1. definire il diagramma di flusso del metodo;
2. fornire un insieme di casi di test (indicando anche il risultato atteso) che soddisfi il criterio della copertura delle decisioni (branch) ma che NON permetta di evidenziare il difetto sopra citato.
3. spiegare, fornendo anche opportune esemplificazioni, perché utilizzando un insieme di dati di test che soddisfi il criterio di copertura delle condizioni il difetto viene rivelato.



# Politecnico di Milano

sede di Cremona

Anno accademico 2019-2020

## Ingegneria del Software - 22 gennaio 2021

Cognome:

LAUREANDO

Nome:

Matricola:

--	--	--	--	--	--

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver condiviso lo schermo). È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
5. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 90 min.

### Esercizio 1 (5 punti)

Il metodo statico `metodo` riceve in ingresso un `ArrayList` `nums` e un intero `n` positivo e restituisce un altro valore intero. L'array `nums`, di lunghezza non precisata, contiene dei numeri interi, per ipotesi tutti diversi tra loro, e `n` rappresenta la posizione di un elemento dell'`ArrayList` e lo divide in due parti. Se la somma `s` degli elementi delle due parti è uguale, ovvero degli elementi di posizione da `0` a `n - 1` e da `n + 1` alla dimensione dell'array, il metodo restituisce `s`; 0 altrimenti. L'array non viene modificato.

- Si fornisca la specifica JML del metodo. Si rammenta che la specifica deve considerare i casi in cui l'`ArrayList` sia nullo o di lunghezza minore di `n` e quello in cui gli elementi non siano tutti distinti fra loro.
- Si trasformi la specifica JML del punto precedente sostituendo una a scelta delle precondizioni nel lancio di opportune eccezioni.

### Esercizio 2 (9 punti)

Si consideri la classe `Numerobinario` che rappresenta numeri interi, positivi e senza segno in binario (ad esempio,  $1110_2$  corrisponde al numero  $14_{10}$ ). Si supponga che la classe fornisca: (a) il metodo `int dimensione()`, per conoscere il numero di cifre del `Numerobinario` considerato; (b) il costruttore `Numerobinario(int n)`, per creare un `Numerobinario` di `n` bit inizializzati a 0; (c) il metodo `void cambiaBit(int pos, int val)`, per assegnare al bit di posizione `pos` il valore `val`; (d) due iteratori per scorrere le cifre binarie dalla meno significativa alla più significativa e dalla più significativa alla meno significativa: `Iterator<Integer> destraSinistra()` e `Iterator<Integer> sinistraDestra()`, rispettivamente.

- Si realizzi il metodo `Numerobinario specchio()`, per invertire l'ordine di memorizzazione della sequenza di bit, usando i soli metodi elencati sopra;
- Supponendo che la classe usi un `ArrayList` per memorizzare la sequenza di cifre binarie e un `int` per memorizzare il numero, si implementi il costruttore che prende in ingresso un numero intero e memorizza le cifre binarie a partire da quella meno significativa;
- Si scriva l'invariante di rappresentazione come invariante privato in JML.

### Esercizio 3 (8 punti)

Uno stabilimento balneare offre un servizio di noleggio di  $N$  postazioni in riva al mare, ciascuna costituita da 2 sdraio ed 1 ombrellone. I clienti si recano allo stabilimento in gruppi di 1, 2, 3 o 4 persone e possono accedere alla spiaggia attraverso un singolo varco. Ogni gruppo può accedere alla spiaggia soltanto se vi sono postazioni disponibili sufficienti (una persona richiede comunque una postazione e tre persone comunque due). Si realizzi il sistema concorrente in Java usando solo i meccanismi base della sincronizzazione, senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`.

### Esercizio 4 (4 punti)

Data una lista di stringhe `List<String> list`, si definisca un metodo in stile funzionale che restituisca le stringhe che iniziano con una lettera maiuscola e di lunghezza maggiore di 4.

### Esercizio 5 (7 punti)

Si consideri il seguente frammento di codice Java:

```
static void metodo(int n) {  
1. int j;  
2. if (n>0) {  
3.     j = 2*n;  
4.     while (j>0) {  
5.         j--;  
6.     }  
7.     System.out.println(j);  
8. };
```

Si definisca:

- Il diagramma di controllo di flusso;
- Un insieme di casi di test che coprano tutte le istruzioni;
- Un insieme di casi di test che coprono tutti i branch;
- Considerando il cammino: 1 2 3 4 5 6 4 7 (che attraversa il ciclo 1 sola volta), un valore specifico dell'input che ne causa l'attraversamento;
- Considerando il cammino 1 2 3 4 5 6 4 5 6 4 7 (che corrisponde ad eseguire il ciclo due volte), un valore specifico dell'input che ne causa l'attraversamento.



# Politecnico di Milano

sede di Cremona

Anno accademico 2020-2021

## Ingegneria del Software - 16 febbraio 2021

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver condiviso lo schermo). È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
5. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 90 min.

### Esercizio 1 (5 punti)

Si consideri il seguente metodo statico: `public static String maxSottostringa(String[] s)`. Il metodo riceve come parametro un array di stringhe, che contiene almeno una stringa, e in cui tutte le stringhe sono non vuote. Il metodo restituisce la sottostringa massimale comune, ossia la più grande sottostringa consecutiva comune a tutti gli elementi dell'array. Qualora non vi sia nessuna stringa comune (nemmeno di un singolo carattere) il metodo restituisce la stringa vuota `" "`. Se vi sono più sottostringhe massimali della stessa lunghezza, il metodo ne restituisce una qualunque. Ad esempio, se `s = [acca accbb ccaba]`, il metodo restituisce la stringa `cc`.

- Specificare il metodo in JML. Non è necessario che la specifica sia eseguibile. Per comodità si ricorda che `String` offre il metodo `String substring(int beginIndex, int endIndex)`, che restituisce una nuova stringa che è una sottostringa della stringa; il significato dei due parametri è evidente.
- Il metodo definito al punto precedente definisce una funzione parziale o totale? Perché? Se la funzione non è totale, come si potrebbe modificare la specifica rendendo la funzione totale?

### Esercizio 2 (9 punti)

Il tipo di dato astratto `Tombola` fornisce un'astrazione per una partita a tombola. Un giocatore estrae un numero usando il metodo `estrai()`. Questo metodo restituisce ogni volta un numero (diverso a caso) tra 1 e 90. Se l'estrazione di un numero provoca una tombola, il metodo `estrai` deve generare un'eccezione `ExceptionTombola` e chiudere la partita. A ogni partita a tombola è associato un certo insieme di cartelle dei vari giocatori partecipanti; ogni cartella è rappresentata da un esemplare del tipo di dato astratto `Cartella`. `Tombola` fornisce i metodi `puri` `tombola`, `cartelle` e `numeriEstratti`:

```
//@ requires true;
//@ ensures (*restituisce true se i numeri di c sono stati estratti tutti*) ;
public boolean /*@pure@*/ tombola(Cartella c)

//@ requires true;
```

```

//@ ensures (*restituisce le cartelle attive*) ;
public arrayList<Cartella> /*@pure@*/ cartelle()

//@ requires true;
//@ ensures (*restituisce la sequenza dei numeri estratti fino a quell'istante,
//@ in ordine cronologico di estrazione (quelli estratti per primi occupano
//@ le prime posizioni dell'arrayList)*) ;
public arrayList<Integer> /*@pure@*/ numeriEstratti()

```

- Si scriva la postcondizione del metodo estrai, che restituisce il nuovo estratto, evidenziando che: (a) il numero estratto è un numero non estratto in precedenza, compreso tra 1 e 90, (b) la sequenza dei numeri già estratti resta immutata, con l'unica aggiunta del numero estratto e (c) il numero estratto non causa una tombola.
- Sapendo che Cartella fornisce il metodo puro per conoscere i numeri di una cartella:

```

//@ requires true ;
//@ ensures (*restituisce i numeri in this*) ;
public arrayList<Integer> /*@pure@*/ numeri()

```

Si scriva l'invariante pubblico per la classe Tombola per imporre che una tombola non può avere due cartelle contenenti gli stessi numeri e che le cartelle di ogni esemplare di Tombola devono coprire tutti i numeri da 1 a 90, ovvero ogni numero deve essere presente in almeno una cartella.

### Esercizio 3 (8 punti)

Un kartodromo ha ideato una semplice applicazione per gestire il traffico in pista e comunicare con i piloti. Il commissario di gara deve definire il numero massimo *nm* di piloti ammessi in pista. Se durante la sessione, non si raggiunge il numero massimo, tutti i piloti possono entrare liberamente sul circuito (semaforo verde a box) e uscirne poi. Se si raggiunge il numero massimo, ad ogni giro, il pilota più lento (fornito al sistema da un metodo opportuno che non si deve modellare), deve lasciare la pista al giro successivo, facendo quindi diventare verde il semaforo per il primo pilota in attesa di entrare in pista. Si realizzi il sistema concorrente in Java usando solo i meccanismi base della sincronizzazione, senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`.

### Esercizio 4 (4 punti)

Scrivere una sola classe Java a piacere, senza ipotizzare gerarchie di ereditarietà e senza usare classi ben note, che contenga:

- un caso di *overloading* tra metodi
- un caso di *overriding* tra metodi

Motivare brevemente le scelte fatte e gli esempi presentati.

### Esercizio 5 (7 punti)

Si consideri il seguente frammento di codice Java:

```

1- static int test(int x) {
2-     int y = x%4;
3-     System.out.println(y);
4-     while (x>0 && y<4) {
5-         if (y==2)
6-             return y;
7-         y--;
8-     }
9-     return 0;
10- }

```

Si definisca:

- Il diagramma di controllo di flusso;
- Un insieme di casi di test che coprano tutte le istruzioni;

- Un insieme di casi di test che coprono tutte le decisioni (branch);
- Un insieme di casi di test che coprono tutte le condizioni;
- In base eventualmente anche ai casi di test definiti, individuare la presenza di un possibile errore di programmazione.



# Politecnico di Milano

sede di Cremona

Anno accademico 2020-2021

## Ingegneria del Software - 24 giugno 2021

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver condiviso lo schermo). È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
5. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 2 ore.

### Esercizio 1 (8 punti)

Usando Java e JML si definiscano l'intestazione e la specifica di:

- un metodo che prende in ingresso tre stringhe e restituisce *true*, se la terza stringa contiene almeno i caratteri delle prime due; *false* altrimenti. Cosa cambierebbe se le prime due stringhe avessero dei caratteri ripetuti e si volesse tener conto anche del numero di ripetizioni? Cosa cambierebbe se anziché dire “almeno”, volessimo dire che la terza stringa deve contenere “solamente” i caratteri delle prime due?
- un metodo che prende in ingresso due array di interi e controlla che: (i) la lunghezza del primo array sia sempre minore di quella del secondo, (ii) le ripetizioni nel primo array siano di cardinalità minore che nel secondo, ovvero se nel primo array ci sono tre 45, nel secondo devono essercene almeno 4 e (iii) la differenza tra la somma dei numeri del secondo array e quella del primo è sempre positiva.

### Esercizio 2 (9 punti)

Si utilizzi un diagramma delle classi UML per modellare le griglie del Sudoku: ogni griglia ha dimensione  $9 \times 9$ , suddivisa in 9 sotto-griglie  $3 \times 3$ . Si ricorda che una colonna, o una riga, (di nove elementi) deve contenere i numeri da 1 a 9 senza ripetizioni e lo stesso deve valere per le 9 sotto-griglie. Oltre alle classi, il diagramma UML deve contenere anche tutti gli attributi ed i metodi necessari ad una formulazione completa del problema.

Considerando quanto definito, si usi poi JML per definire l'invariante pubblico della classe *Sudoku* che semplicemente impone il rispetto dei vincoli: ogni griglia, in qualsiasi momento della propria evoluzione, non può mai violare le regole suddette. È chiaramente possibile definire metodi *helper* senza snaturare il modello creato.

### Esercizio 3 (4 punti)

Si definisca in Java un metodo (statico) che prende in ingresso una lista di interi ed una lista di predicati. La funzione deve applicare i predicati alla lista per filtrarla e restituire poi la somma dei numeri rimanenti. Ad esempio, i predicati potrebbero dire che il risultato deve essere la somma dei soli valori non negativi, minori di 100 e multipli di 7.

## Esercizio 4 (4 punti)

Si modelli il sistema di smistamento bagagli di un aeroporto. Ogni aeroporto è dotato di un numero  $n$  di nastri per la consegna bagagli e ogni nastro può gestire al più tre voli e  $b$  bagagli contemporaneamente. Questo significa che il sistema deve indirizzare i diversi bagagli al nastro di pertinenza e deve controllare che questo non contenga mai troppi bagagli, per evitare che il sistema si fermi. Si applichino poi le classi definite ad un aeroporto con 5 nastri per la consegna dei bagagli, 12 voli in contemporanea e 500 bagagli da smaltire, supponendo che ogni nastro non possa gestire più di 20 bagagli contemporaneamente. Si realizzi il sistema concorrente in Java usando solo i meccanismi base della sincronizzazione, senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`.

## Esercizio 5 (8 punti)

Si consideri il seguente frammento di codice Java:

```
1- public static int test(int x, int[] a) {  
2-     int count;  
3-     if (x < 1 || a == null) return 0;  
4-     count = x;  
5-     while (count > 1) {  
6-         if (count > 0) count %= 3;  
7-         else return count;  
8-     }  
9-     return count;  
 }
```

Si definisca:

- Il diagramma di controllo di flusso;
- Un insieme di casi di test che coprano tutte le istruzioni;
- Un insieme di casi di test che coprono tutte le decisioni (branch);
- Un insieme di casi di test che coprono tutte le condizioni;



## Ingegneria del Software - 15 luglio 2021

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver condiviso lo schermo). È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
5. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 2 ore.

### Esercizio 1 (8 punti)

Si consideri la seguente classe Java `MusicLibrary` per la gestione di una collezione di brani musicali. Ogni brano è rappresentato dalla classe immutabile `Song` che contiene alcune informazioni tra cui l'artista o gruppo musicale (`Artist`) che esegue il brano e la data (`Date`) in cui il brano è stato registrato, come riportato di seguito.

```
public class MusicLibrary {  
    // Ritorna l'insieme di artisti che eseguono almeno un brano nella collezione.  
    public /*@ pure */ Set<Artist> getArtists();  
  
    // Ritorna la lista di brani eseguiti dall'artista.  
    // Lancia una UnknownArtistException se la collezione non contiene brani di artist.  
    public /*@ pure */ List<Song> getByArtist(Artist artist) throws UnknownArtistException;  
  
    // Ritorna l'insieme dei brani eseguiti in una certa data.  
    // Lancia una UnknownDateException se nessun brano e' stato eseguito in quella data.  
    public /*@ pure */ Set<Song> getByDate(Date date) throws UnknownDateException;  
  
    // Aggiunge il brano alla collezione.  
    public void addSong(Song s);  
}  
  
public /*@ pure */ class Song {  
    // Ritorna l'artista che esegue il brano  
    public Artist getArtist();  
  
    // Ritorna la data in cui e' stato eseguito il brano  
    public Date getDate();
```

}

### Domanda a)

Si specifichi in JML il metodo `getByDate()`.

### Domanda b)

Si consideri un'implementazione che utilizza un `Set` per contenere le canzoni, come mostrato di seguito.

```
public class MusicLibrary {  
    private final Set<Song> songs;  
    ...  
}
```

Per tale implementazione si definisca l'invariante di rappresentazione.

## Esercizio 2 (9 punti)

Si consideri un'applicazione per un negozio (reale o virtuale), da realizzare in Java. Una parte dell'applicazione gestisce il calcolo del prezzo finale di vendita del "carrello" dell'acquirente. Il pagamento viene poi gestito da un'altra parte dell'applicazione. A tale scopo, il progetto del sistema prevede una classe `Sale` con due metodi di calcolo (prima dello sconto e dopo lo sconto), oltre ad ulteriori dati e operazioni che qui ignoriamo.

```
public class Sale {  
    ...  
    public Money totaleNonScontato() { ... }  
    public Money totaleScontato() { .... }  
    ...  
}
```

Il calcolo del totale deve essere effettuato in base al costo dei singoli prodotti moltiplicato per loro quantità, considerando però anche la politica di sconto. Il negozio infatti prevede una serie di iniziative promozionali, non sempre uguali, per cui ad esempio ci potrebbe essere uno sconto, di percentuale variabile, il martedì oppure uno sconto del 5% la sera dopo le 23, oppure un'offerta speciale del 15% il mercoledì per gli anziani, ecc. ecc. Tali politiche non sono interamente previste o prevedibili nella specifica del sistema. Occorre quindi progettare un sistema software che sia in grado di gestire facilmente politiche diverse e che renda possibile introdurre facilmente nuove politiche di sconto, o variazioni delle politiche esistenti.

Si utilizzi un design pattern opportuno per la progettazione del sistema. Si trateggino in Java le classi necessarie e le loro relazioni, utilizzando opportuni frammenti di codice o diagrammi UML per definire, illustrare ed esemplificare il sistema così progettato e il suo funzionamento.

## Esercizio 3 (5 punti)

Si considerino i seguenti metodi Java e se ne completino le parti omesse, *utilizzando esclusivamente i concetti e i costrutti della programmazione funzionale*.

### Domanda a)

Il metodo seguente prende come argomento una lista di persone (`Person`) e restituisce una lista delle persone con più di 60 anni.

Si ipotizzi che la classe `Person` abbia il metodo `getAge()` che restituisce l'età della persona.

```
public static List<Person> older(List<Person> people) {  
    List<Person> elder = ... ;  
    return elder;
```

### Domanda b)

Il metodo seguente prende come argomento una lista di nomi e stampa il nome più lungo, se la lista non è vuota, altrimenti non stampa nulla.

```

public static void longest(List<String> names) {
    final Optional<String> theLongest = ... ;
    theLongest.ifPresent(.....);
}

```

### Esercizio 4 (5 punti)

Si consideri il seguente programma Java

```

public class Esame implements Runnable {
    private transient int x;
    private volatile int y;

    public static void main(String [] args) {
        Esame that = new Esame();
        (new Thread(that)).start(); /* Linea 7 */
        (new Thread(that)).start(); /* Linea 8 */
    }
    public synchronized void run() /* Linea 10 */ {
        for (;;) /* Line 11 */ {
            x++;
            y++;
            System.out.println("x=" + x + ",y=" + y);
        }
    }
}

```

Quali delle seguenti risposte sono corrette? Si fornisca anche una breve spiegazione per la risposta fornita.

1. Si genera un errore di compilazione in corrispondenza di Linea 11;
2. Si genera un errore di compilazione a causa di errori alle Linee 7 e 8;
3. Il programma stampa valori per x e y che potrebbero non essere sempre uguali sulla stessa riga di stampa (per esempio, può stampare  $x = 1, y = 2$ );
4. Il programma stampa valori per x e y che sono sempre gli stessi sulla stessa riga di stampa (per esempio, stampa  $x = 1, y = 1$  su una riga e  $x = 2, y = 2$  sulla successiva);
5. Il programma stampa i valori di x in ordine crescente, ovvero  
 $x = 1, \dots$   
 $x = 2, \dots$   
 $x = 3, \dots$

### Esercizio 5 (6 punti)

Il seguente metodo Java prende in ingresso un array di al più 5 interi e restituisce l'elemento più grande in valore assoluto, oppure 0 se l'array fosse vuoto oppure -1 se troppo grande.

```

public int max(int[] numbers) {
    int i, max_value = 0;
    if (numbers.length > 5) return -1;
    for (i = 0; i < numbers.length; i++) {
        if (numbers[i] < 0)
            max_value = Math.max(max_value, Math.abs(numbers[i]));
        else max_value = Math.max(max_value, numbers[i]);
    }
    return max_value;
}

```

- Disegnare il diagramma del flusso di controllo.

- Il metodo è collaudato con i casi di test seguenti (input; valore restituito):

- T1: {0,0,0,0,0}; 0;
- T2: {1,2,3,4,5}; 5;
- T3: {-1,-2,-3,-4,-5}; 5;
- T4: {1,2,3,4,5,6}; -1;
- T5: {-10,10,3,5,-6}; 10;
- T6: {}; -1;

Definire le percentuali di copertura delle istruzioni (*statement*) e delle decisioni (*branch*) ottenute eseguendo ogni test separatamente.

- Quale test darebbe errore?



# Politecnico di Milano

sede di Cremona

Anno accademico 2020-2021

## Ingegneria del Software - 9 settembre 2021

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver condiviso lo schermo). È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
5. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 2 ore.

### Esercizio 1 (8 punti)

Si scriva la specifica JML dei seguenti metodi statici Java:

- int metodo1(String[] s). Il metodo restituisce il numero di vocali nelle stringhe. Si supponga che esista un metodo statico *helper* vocale che restituisce *true* se il carattere passato come parametro è una vocale, *false* altrimenti.
- int metodo2(int[] a, int d1, int d2). Il metodo restituisce la media degli elementi dell'array di posizione compresa tra d1 e d2, estremi compresi. Restituisce 0 nel caso d1 e/o d2 non siano compatibili con l'array.
- ArrayList<Integer> metodo3(List<Integer> l1, List<Integer> l2). Il metodo restituisce la lista di numeri *dispari* presenti in l1 e in l2, senza ripetizioni.

### Esercizio 2 (9 punti)

Si consideri la struttura tipica di un *file system*. Le directory sono organizzate gerarchicamente: ogni directory può contenere altre directory, file, oppure link. Un link è un riferimento a un file fisicamente memorizzato in un'altra directory; in questo modo il file riferito diventa virtualmente parte anche della directory che contiene il link. Una directory ha un nome; ogni file è caratterizzato da un nome, una dimensione e un tipo. Un link ha un nome e un tipo.

Ogni elemento (directory, file o link) è associato con un insieme di diritti d'accesso: lettura, scrittura e esecuzione. Questi sono concessi al proprietario di una risorsa (un singolo utente), a gruppi di utenti o a tutti gli utenti.

- Si modelli il problema descritto con un diagramma delle classi UML, evidenziando i metodi e gli attributi principali delle diverse classi.
- In funzione delle classi identificate nel diagramma precedente, si scriva il corpo del metodo *stampaNomeCompleto* per scrivere a video il nome completo di un file partendo dalla radice del file system.
- Si scriva l'invariante privato della classe *Directory* per dire che un suo oggetto non può mai contenere più di 200 file e deve esistere almeno un utente in grado di leggere il contenuto della directory.

Se si ritiene che la descrizione informale contenga ambiguità, si descriva a parole come il diagramma UML proposto risolve tale ambiguità.

### Esercizio 3 (6 punti)

Pensando di giocare a tombola, si scriva in Java un iteratore che permetta di estrarre, in una sequenza casuale, tutti i numeri compresi tra 1 e 90. Nell'implementare l'iteratore si suggerisce di far uso dei seguenti dati:

```
private boolean[] tabellone;
private int numEstratti;
```

Si definisca sia il metodo `nuovaSequenza` (della classe `Tombola`) che restituisce l'iteratore, sia la classe che definisce l'iteratore (ossia il generatore di numeri interi). L'iteratore deve: (a) fornire ogni numero una sola volta, (b) marcare il numero estratto nell'array di booleani e (c) incrementare il contatore `numEstratti`. Si ipotizzi di poter disporre di un metodo statico `numero` che restituisce un numero casuale tra 1 e 90.

### Esercizio 4 (6 punti)

Si scriva un programma Java che consenta la gestione di una catena di montaggio, con un numero `nCorsie` di corsie e un numero massimo di pezzi da lavorare a `nPezzi`. Chiaramente, un pezzo può essere lavorato solo se esiste almeno una corsia libera e lascerà la postazione dopo un tempo variabile. Per semplicità, ogni pezzo occupa la prima corsia utile partendo dalla prima.

Si modelli poi il caso in cui la catena di montaggio offre 7 corsie e 30 pezzi, in parallelo, cercano di acquisire una postazione di lavoro. Ogni pezzo aspetterà una quantità di tempo casuale e poi cercherà di occupare una postazione. Si ricorda che `Math.random()` restituisce un *long* maggiore o uguale a 0 e minore di 1. Si realizzi il sistema concorrente in Java usando solo i meccanismi base della sincronizzazione, senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`.

### Esercizio 5 (4 punti)

Si consideri il seguente metodo Java:

```
public static int test(int x, int y) {
    int z = x;

    while (z > 0) {
        if (z > y || x == 1) break;
        else x = x % z;
        --z;
    }
    return z;
}
```

1. Si disegni il diagramma del flusso di controllo;
2. Si identifichi, se esiste, un insieme di test (minimo) per coprire tutte le istruzioni e le decisioni (branch) del metodo.
3. Si identifichi, se esiste, un insieme di test (minimo) per coprire tutte le istruzioni e le condizioni del metodo.
4. Si spieghi il comportamento del metodo per  $x = 4$  e  $y = 4$ .



# Politecnico di Milano

sede di Cremona

Anno accademico 2021-2022

## Ingegneria del Software - 24 gennaio 2022

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver disattivato la connessione a Internet). È proibito l'uso di ogni dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
5. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 2 ore.

### Esercizio 1 (4 punti)

Si ipotizzi la classe Java Cassetto, per semplicità caratterizzata dalla sola dimensione in centimetri cubi del cassetto. La classe fornisce due metodi pubblici: `spazioDisponibile()`, che restituisce il numero di centimetri non occupati nel cassetto, e `oggetti()`, che restituisce la lista degli elementi nel cassetto. La classe Elemento fornisce un metodo pubblico `grandezza` che restituisce la dimensione —in centimetri cubi— dell'elemento.

Si definisca l'intestazione di un metodo pubblico `aggiungi` che consente di aggiungere un elemento al cassetto e la specifica JML del metodo medesimo. Chiaramente la specifica deve gestire in modo appropriato l'eventuale problema che l'elemento che si vuole aggiungere potrebbe essere troppo grande rispetto alla spazio disponibile.

### Esercizio 2 (6 punti)

Si consideri il seguente frammento di codice, si evidenzino eventuali errori di compilazione e, supponendo di correggere eventuali errori riscontrati (magari commentando la riga di codice), si spieghi quali metodi vengono invocati in ciascun caso e si scriva il risultato calcolato (valore stampato a video).

```
public abstract class Point {  
    public abstract double distance (Point p);  
}  
  
public class Point1D extends Point {  
    private double c1;  
    public Point1D (double c1) {this.c1 = c1;}  
    public double getC1() {return c1;}  
    public double distance (Point p) {return Math.abs((Point1D) p).getC1()-c1;}  
}  
  
public class Point2D extends Point1D {  
    private double c2;
```

```

public Point2D (double c1, double c2) {super(c1); this.c2 = c2;}
public double getC2() {return c2;}

public double distance (Point p) {
    return Math.sqrt(Math.pow((Point2D) p).getC1()-this.getC1(), 2) +
           Math.pow((Point2D) p).getC2()-this.getC2(), 2));
}

public double distance (Point2D p) {
    return Math.sqrt(Math.pow(p.getC1()-this.getC1(), 2) +
                   Math.pow(p.getC2()-this.getC2(), 2));
}

01- Point p = new Point();
02- Point p1 = new Point1D(0.0);
03- Point p2 = new Point1D(1.0);
04- Point p3 = new Point2D(0.0, 1.0);
05- Point2D p4 = new Point2D(1.0, 0.0);
06- double x;

07- x = p1.distance(p1);
08- System.out.println(x);

09- x = p4.distance(p3);
10- System.out.println(x);

11- x = p1.distance(p2);
12- System.out.println(x);

13- x = p3.distance(p1);
14- System.out.println(x);

15- p3 = p4;
16- x = p2.distance(p3);
17- System.out.println(x);

18- x = p4.distance(p);
19- System.out.println(x);

```

### Esercizio 3 (5 punti)

Si supponga di volere modellare in UML il comportamento di un Cronometro con i (soliti) due pulsanti *start* e *stop* e con il solito comportamento. Il cronometro parte premendo il pulsante *start* e si ferma con il pulsante *stop*. A questo punto, il cronometro mostra un primo tempo intermedio e riprende se si preme *start* nuovamente. Al contrario, premendo *stop*, il tempo mostrato diventerebbe finale ed il cronometro ritornerebbe a zero.

Si modelli il sistema con un diagramma delle classi, sfruttando se necessario opportuni *design pattern*. Il diagramma deve contenere le classi ed i metodi necessari. Per ogni metodo definito, si deve anche spiegare a parole, e brevemente, il suo comportamento.

### Esercizio 4 (3 punti)

Adottando uno stile di programmazione funzionale, si definisca in Java una funzione statica che prende una lista di parole ed un carattere come parametri e restituisce il numero di parole che contengono il carattere. Il carattere passato come parametro può essere maiuscolo o minuscolo e lo stesso vale per i caratteri delle parole nella lista: la funzione deve contare comunque il numero di parole ignorando eventuali differenze di rappresentazione.

### Esercizio 5 (8 punti)

Fuorando l'affare, una piccola software house creare un semplice sistema software per organizzare la gestione dei tamponi COVID nelle farmacie. Il sistema deve adattarsi alla capacità delle diverse farmacie, ovvero il numero di pazienti che possono essere gestiti contemporaneamente deve essere configurabile, e deve contentire/impedire ai pazienti di accedere al luogo dove vengono effettuati i tamponi non appena si raggiunga la capienza massima. Ogni paziente, dopo essersi registrato, aspetto che

non deve essere affrontato, prova ad entrare nella sala dei tamponi. Se il display è verde, il paziente può entrare, gli verrà fatto il tampone e dovrà poi uscire dalla stanza ed aspettare l'esito in un'area apposita. Se il display fosse rosso, il paziente deve aspettare che diventi verde, ovvero che esca almeno un paziente. Ovviamente, il colore del display dipende dal numero di pazienti nella sala tamponi: verde se inferiore al massimo stabilito, rosso altrimenti. Ricevuto l'esito, dopo un tempo variabile, l'utente lascia il sistema. Non ci interessa registrare l'esito del tampone. Si istanzi poi il sistema considerando una farmacia che ha 5 postazioni nella sala tamponi e che deve gestire 20 pazienti che arrivano quasi contemporaneamente.

Si realizzi il sistema concorrente in Java usando solo i meccanismi base della sincronizzazione, senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`.

### Esercizio 6 (7 punti)

Si consideri il seguente frammento di codice Java:

```
static void foo(int x, int y) {  
    if (x > 0) {  
        while (x != 0 && y > 0) {  
            y = y - x;  
            x = x - 2;  
            if (x < 0) x = -x;  
        }  
    }  
}
```

Si definisca:

- Il diagramma di controllo di flusso;
- Un insieme di casi di test che coprano tutte le istruzioni;
- Un insieme di casi di test che eseguono il ciclo while esattamente tre volte;
- Un insieme di casi di test che coprono tutte le condizioni;



# Politecnico di Milano

sede di Cremona

Anno accademico 2021-2022

## Ingegneria del Software - 11 febbraio 2022

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver disattivato la connessione a Internet). È proibito l'uso di qualsiasi ambiente di sviluppo e di qualsiasi dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
- 5. Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 2 ore.

### Esercizio 1 (4 punti)

Si pensi ad una semplice macchina da caffè che usa un sistema a cialde. Si ipotizzi che abbia un contenitore (di cialde) caratterizzato solamente dal numero di cialde usate che può contenere. La MacchinaCaffe consentirà all'utente di preparare un caffè, attraverso il metodo `preparaCaffe` solo se il contenitore non è pieno. Nel caso, la macchina dovrebbe generare un'eccezione (`ContenitorePienoEcc`) per segnalare la cosa all'utente. La classe `MacchinaCaffe` fornisce un metodo osservatore `numeroCialde` che restituisce il numero di cialde contenute. L'utente può anche vuotare il contenitore (in automatico) attraverso il metodo `svuotaCialde`, che può essere invocato in ogni momento e se il contenitore fosse già vuoto, non farebbe nulla.

Si definisca l'intestazione dei metodi pubblici `preparaCaffe` e `svuotaCialde` e la loro specifica JML. Chiaramente la specifica deve gestire in modo appropriato gli eventuali problemi.

### Esercizio 2 (6 punti)

Si consideri il seguente frammento di codice, si evidenzino eventuali errori di compilazione e, supponendo di correggere eventuali errori riscontrati commentando la riga di codice, si spieghi quali metodi vengono invocati in ciascun caso. Si scriva anche il risultato calcolato (valore stampato a video).

```
class Persona {  
    protected String nome;  
    public Persona(String nome) { this.nome = nome; }  
    public void guida(Auto a) {  
        System.out.println(nome+" guida "+a.modello());  
    }  
}  
  
class Autista extends Persona {  
    public Autista(String nome) { super(nome); }  
    public void guida(Camion c) {
```

```

        System.out.println("Autista "+nome+ " guida "+c.modello());
    }
    public void guida(Camion c, Autista secondo) {
        System.out.println("Autista "+nome+ " guida "+c.modello()+" con "+secondo.nome);
    }
}

class AutistaCamion extends Autista {
    public AutistaCamion(String nome) { super(nome); }
    public void guida(Auto a) {
        System.out.println("Autista camion "+nome+ " guida "+a.modello());
    }
}

class Auto {
    public String modello() { return "auto AA"; }
}

class Camion extends Auto {
    public String modello() { return "camion CC"; }
}

public class Main {
    static public void main(String[] argc) {
        Persona p = new Persona("Filippo");
        Autista au1 = new AutistaCamion("Maria");
        Auto a = new Auto();
        Auto ac = new Camion();
        Camion c = new Camion();

        p.guid(a);
        p = new Autista("Luigi");
        AutistaCamion au2 = new AutistaCamion("Gianni");
        au2.guid(c);
        p.guid(p, au1);
        au1.guid(c, au2);
        p.guid(c);
        au2.guid((Camion) a, au1);
    }
}

```

### Esercizio 3 (5 punti)

Si supponga di voler modellare in UML un semplice social network. Dopo essersi registrato, un utente può postare quanti messaggi vuole relativi ad argomenti particolari, stabiliti dal gestore del sistema. Ogni utente registrato può anche leggere i messaggi degli altri utenti relativi agli argomenti che ha deciso di seguire. Ovviamente, il sistema deve notificare l'utente quando qualcuno dovesse postare un messaggio di suo interesse, ovvero un messaggio relativo ad uno degli argomenti selezionati. La selezione degli argomenti di interesse —che si seguono— deve essere dinamica ed ogni utente può continuare a cambiare la propria lista di interessi. Si modelli il sistema con un diagramma delle classi, sfruttando se necessario opportuni *design pattern*. Il diagramma deve contenere le classi ed i metodi necessari. Per ogni metodo principale definito, si deve anche spiegare a parole, e brevemente, il suo comportamento.

### Esercizio 4 (5 punti)

Dopo aver abbozzato la classe Java corrispondente al concetto di Argomento del social network dell'esercizio precedente, si adotti uno stile di programmazione funzionale per definire due metodi che restituiscono: (a) tutti i messaggi generati da un certo utente in un certo giorno e (b) il numero di messaggi generati da un certo utente.

### Esercizio 5 (6 punti)

Una comunità remota deve approvvigionarsi d'acqua attraverso un serbatoio. Il serbatoio dispone di 10 rubinetti ed ogni abitante può riempire fino a 5 contenitori da un litro. La capienza del serbatoio è di 1000 litri. Chiaramente, un abitante può riempire i propri contenitori se c'è un rubinetto disponibile e se il serbatoio non è vuoto. Potrebbe anche succedere che l'acqua rimasta

non sia sufficiente per riempire tutti i contenitori dell’utente. Il comune riempie il serbatoio quando è vuoto e lo riempie sempre fino alla capacità massima. Si realizzi il sistema concorrente in Java usando solo i meccanismi base della sincronizzazione, senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`. Si esemplifichi poi il funzionamento del sistema considerando 500 abitanti che si recano al serbatoio con un numero variabile (ma sempre minore o uguale a 5) di contenitori e che il comune riempia il serbatoio quando richiesto.

## Esercizio 6 (7 punti)

Si consideri il seguente frammento di codice Java:

```
00 int cerca(int[] a, int v) {  
01     int r = -1, i = 0;  
02     boolean t = false;  
03  
04     if (a == null)  
05         return r;  
06  
07     while(i < a.length && t == false) {  
08         if (a[i] == v) {  
09             t = true;  
10             r = i;  
11         }  
12         else  
13             i++;  
14     }  
15     return r;  
16 }
```

Si definisca:

- Il diagramma di controllo di flusso;
- Un insieme di casi di test che coprano tutte le istruzioni;
- Un insieme di casi di test che eseguano le decisioni;
- Un insieme di casi di test che coprano tutte le condizioni;



# Politecnico di Milano

sede di Cremona

Anno accademico 2021-2022

## Ingegneria del Software - 24 giugno 2022

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver disattivato la connessione a Internet). È proibito l'uso di qualsiasi ambiente di sviluppo e di qualsiasi dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
- 5. Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 2 ore.

### Esercizio 1 (4 punti)

Si definisca l'intestazione dei due metodi statici seguenti e la loro specifica JML.

- Il metodo prende in ingresso un array di stringhe  $a$ , una stringa  $s$  ed un intero  $n$  e aggiunge  $s$  all'array se la stringa ha lunghezza pari (dispari) e  $a$  non contiene già  $n$  stringhe di lunghezza pari (dispari).
- Il metodo prende in ingresso due array di caratteri  $x$  e  $y$  e restituisce, in un array di interi, l'elenco, in ordine qualunque, di tutte e sole le posizioni in cui gli elementi dei due array sono uguali.

La specifica deve gestire in modo appropriato gli eventuali problemi con i parametri in ingresso.

### Esercizio 2 (8 punti)

Si consideri il seguente frammento di codice:

```
public abstract class Point {  
    public abstract double distance (Point p);  
}  
  
public class Point1D extends Point {  
    private double c1;  
  
    public Point1D (double c1) {this.c1 = c1;}  
    public double getc1() {return c1;}  
    public double distance (Point p) {return Math.abs((Point1D) p).getc1()-c1;}  
}  
  
public class Point2D extends Point1D {  
    private double c2;
```

```

public Point2D (double c1, double c2) {super(c1); this.c2 = c2;}
public double getC2() {return c2;}
public double distance (Point p) {
    return Math.sqrt(Math.pow((Point2D) p).getc1()-this.getc1(), 2) +
           Math.pow((Point2D) p).getc2()-this.getc2(), 2));
}
public double distance (Point2D p) {
    return Math.sqrt(Math.pow(p.getc1()-this.getc1(), 2) +
                   Math.pow(p.getc2()-this.getc2(), 2));
}
}

```

- Quale dei due metodi `distance` della classe `Point2D` rappresenta un caso di *overloading* e quale invece rappresenta un caso di *overriding*?
- Quale dei seguenti assegnamenti è corretto? Perché?

```

Point p;
Point1D p1;
Point2D p2;

p = p1; // 1
p = p2; // 2
p1 = p; // 3
p1 = p2; // 4
p2 = p1; // 5

```

- Si aggiunga il seguente frammento e si pieghi sinteticamente a parole l'effetto di ciascun assegnamento, indicando quali metodi vengono invocati in ciascun caso e scrivendo il risultato calcolato (valore stampato a video).

```

Point p1 = new Point1D(0.0);
Point p2 = new Point1D(1.0);
Point p3 = new Point2D(0.0, 1.0);
Point p4 = new Point2D(1.0, 0.0);
double x;

x = p1.distance(p2); // 1
System.out.println(x);

x = p3.distance(p4); // 2
System.out.println(x);

x = p1.distance(p3); // 3
System.out.println(x);

x = p1.distance(p1); // 4
System.out.println(x);

p1=p3; // 5
x = p2.distance(p3);
System.out.println(x);

x = p4.distance(p1); // 6
System.out.println(x);

```

### Esercizio 3 (5 punti)

Si realizzi un diagramma delle classi che consenta di gestire parole sia rappresentate attraverso testo “semplice”, sia con rappresentazioni in corsivo, grassetto e sottolineato. La soluzione proposta, basandosi magari sull’uso di un design pattern opportuno, deve considerare tutte le possibili combinazioni, ovvero un testo potrebbe essere rappresentato in grassetto sottolineato e un altro in corsivo e grassetto.

### Esercizio 4 (3 punti)

Spiegare brevemente, meglio se attraverso un esempio (non già visto a lezione o in altri temi d’esame), il principio di progettazione ad oggetti *open/closed*, evidenzandone i punti di forza.

## Esercizio 5 (6 punti)

Un produttore genera in modo casuale (anche con ripetizioni) numeri da 1 a 10 (inclusi) e li memorizza in un buffer che può contenere un solo numero alla volta. Due processi consumatori concorrenti tentano di acquisire tali numeri soltanto dopo la loro produzione. Uno dei due consumatori tenta di acquisire solo numeri da 1 a 5, l'altro solo numeri che vanno da 6 a 10. Si realizzi il sistema concorrente in Java usando solo i meccanismi base della sincronizzazione, senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`. Si esemplifichi poi il funzionamento del sistema considerando 20 numeri.

## Esercizio 6 (7 punti)

Si consideri il seguente frammento di codice Java:

```
00 public static void foo(int[] a) {  
01     int i, j, n;  
02     for (i = 1; i < a.length; i++) {  
03         n = a[i];  
04         j = i;  
05         while (j > 0 && a[j - 1] > n) {  
06             a[j] = a[j - 1];  
07             j -= 2;  
08         }  
09         a[j] = n + 1;  
10     }  
11 }
```

Si definisca:

- Il diagramma di controllo di flusso;
- Un insieme di casi di test che coprano tutte le istruzioni;
- Un insieme di casi di test che esercitino le decisioni;
- Un caso di test che copra il ciclo `while` esattamente 2 volte;



# Politecnico di Milano

sede di Cremona

Anno accademico 2021-2022

## Ingegneria del Software - 22 luglio 2022

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver disattivato la connessione a Internet). È proibito l'uso di qualsiasi ambiente di sviluppo e di qualsiasi dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
5. **Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 2 ore.

### Esercizio 1 (6 punti)

Si definisca l'intestazione dei due metodi statici seguenti e la loro specifica JML.

- Il metodo prende in ingresso una stringa `s` e un numero `n` e restituisce la lunghezza della stringa se `n` è 0, il numero di lettere maiuscole se `n` è 1 e il numero di minuscole se `n` è 2. Il metodo restituisce 0 per qualsiasi altro valore di `n`.
- Il metodo prende in ingresso due array di stringhe `as` e `bs` e restituisce il numero (intero) di stringhe che compaiono nei due array anche in posizioni diverse. Si faccia prima l'ipotesi che gli array non contengano stringhe ripetute e si spieghi poi cosa cambierebbe nel caso ci fossero ripetizioni.

La specifica deve gestire in modo appropriato gli eventuali problemi con i parametri in ingresso.

### Esercizio 2 (8 punti)

Si considerino le seguenti classi Java, in cui viene omessa l'ovvia implementazione dei metodi `getMembers()`, `getSize()` e `getName()`.

```
public class Group {  
    public List<Element> getMembers() { ... }  
}  
  
class Element {  
    public String getName() { ... }  
    public int getSize() { ... }  
}
```

Si implementino i seguenti metodi facendo uso esclusivamente di costrutti della programmazione funzionale:

1. Il metodo `void printType(Group g)` deve stampare la dimensione media degli elementi del gruppo `g`. Se il gruppo è vuoto, il metodo non deve stampare nulla.
2. Il metodo `List<String> namesInGroups(List<Group> gs)` restituisce la lista con il nome di tutti gli elementi presenti all'interno dei gruppi in `gs` con una dimensione maggiore di 10.

### Esercizio 3 (4 punti)

Si realizzi un diagramma delle classi che consenta di gestire le modalità di notifica del vostro smartphone. La soluzione proposta, basandosi magari sull'uso di un design pattern opportuno, deve considerare che se il telefono è in modalità *silenzioso*, la notifica deve apparire sullo schermo e non fare altro, se la modalità è *aereo*, la notifica non deve essere visualizzata, se il telefono è in uso *esterno*, la notifica deve essere associata ad un opportuno segnale acustico, e se il telefono è configurato per situazioni rumorose, la notifica deve anche far lampeggiare la luce del telefono.

### Esercizio 4 (3 punti)

Spiegare brevemente, meglio se attraverso un esempio (non già visto a lezione o in altri temi d'esame), il principio di sostituzione di *Liskov*, evidenziandone i punti di forza.

### Esercizio 5 (6 punti)

La catena di montaggio di un'azienda comprende quattro robot: due producono pezzi di tipo A e B e due usano/prelevano questi pezzi. I due robot producono pezzi di tipo A o B in modo casuale e anche i due robot che devono prelevarli possono semplicemente prendere il primo pezzo disponibile. Il nastro trasportatore può contenere non più di tre pezzi (di tipo qualsiasi). Si realizzi il sistema concorrente in Java usando solo i meccanismi base della sincronizzazione, senza sfruttare classi e pacchetti di `java.util.concurrent.XXX`. Si esemplifichi poi il funzionamento del sistema considerando i 4 robot ed il nastro trasportatore.

### Esercizio 6 (6 punti)

Si consideri il seguente frammento di codice Java:

```
public static int f(int n) {
    int a = 0;

    if (n <= 0 || n > 4) return 0;
    if (n%2 != 0) n = n + 1;

    while (n > 1) {
        a = a + n;
        n = n - 2;
    }
    return a;
}
```

Si definisca:

- Il diagramma di controllo di flusso;
- Un insieme di casi di test che coprano tutte le istruzioni;
- Un insieme di casi di test che esercitino le decisioni (*branch*);
- Un caso di test che copra tutti i cammini (*path*);



# Politecnico di Milano

sede di Cremona

Anno accademico 2021-2022

## Ingegneria del Software - 29 agosto 2022

Cognome:	LAUREANDO <input type="checkbox"/>						
Nome:	Matricola:						
<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>							

### Istruzioni

1. È necessario scrivere il proprio nome, cognome e numero di matricola su ogni foglio.
2. È possibile scrivere in matita.
3. È possibile consultare liberamente libri, manuali o appunti (anche dal proprio computer, ma solo dopo aver disattivato la connessione a Internet). È proibito l'uso di qualsiasi ambiente di sviluppo e di qualsiasi dispositivo elettronico (quali calcolatrici tascabili, telefoni cellulari, ecc.).
4. Eventuali laureandi sono pregati di scriverlo sul primo foglio.
- 5. Coloro che prendessero meno di 10 punti dovranno saltare il prossimo appello.**
6. Tempo a disposizione: 2 ore.

### Esercizio 1 (6 punti)

Si definisca l'intestazione dei due metodi statici seguenti e la loro specifica JML:

- Il metodo prende in ingresso una stringa `s` e restituisce il numero di caratteri non ripetuti nella stringa. Il metodo non considera le differenze tra caratteri maiuscoli e minuscoli.
- Il metodo prende in ingresso un array list di stringhe `as` ed una stringa `s` e aggiunge `s` alla lista se non è già presente e rispettando la lunghezza delle stringhe (dalla più corta alla più lunga).

La specifica deve gestire in modo appropriato gli eventuali problemi con i parametri in ingresso.

### Esercizio 2 (9 punti)

Si considerino le seguenti classi Java:

```
abstract class Date {  
    public abstract String dateDistance(Date d);  
}  
  
class DayDate extends Date {  
    int day;  
    public String dateDistance(Date p) {return dateDistance((DayDate) p);}  
    public String dateDistance(DayDate p) {return (day-p.day) + " days";}  
    public DayDate(int d) {day=d;}  
}  
  
class DayMonthDate extends DayDate {  
    int month;  
    DayMonthDate(int d, int m) {super(d); month = m;}  
    public String dateDistance(Date p) {return dateDistance((DayMonthDate) p);}  
    public String dateDistance(DayMonthDate p) {  
        return super.dateDistance(p) + ":" + (month - p.month) + " months";  
    }  
}
```

```

    }
}

class DayMonthYearDate extends DayMonthDate {
    int year;
    DayMonthYearDate(int d, int m, int y) { super(d, m); year = y; }
    public String dateDistance(Date p) { return dateDistance((DayMonthYearDate) p); }
    public String dateDistance(DayMonthYearDate p) {
        return super.dateDistance(p) + ":" + (year - p.year) + " years";
    }
}

```

e il seguente frammento di codice le cui righe sono state numerate per semplicità:

```

DayDate d1 = new DayDate(22);
01- DayDate d2 = new DayMonthDate(26, 3);
02- DayMonthDate d3 = new DayMonthDate(14, 9);
03- Date d4 = new DayMonthDate(12, 5);
04- DayMonthYearDate d5 = new DayMonthYearDate(18, 4, 2021);
05- Date d6 = new Date(8);
06- System.out.println(d1.dateDistance(d2));
07- System.out.println(d1.dateDistance(d3));
08- System.out.println(d1.dateDistance(d6));
09- System.out.println(d2.dateDistance(d3));
10- System.out.println(d3.dateDistance(d2));
11- System.out.println(d3.dateDistance(d5));
12- System.out.println(d4.dateDistance(d5));
13- System.out.println(d5.dateDistance(d3));
14- System.out.println(d5.dateDistance(d6));

```

Si scriva:

- Il numero di riga delle istruzioni (se ne esistono) che generano un errore in compilazione o un'eccezione a run time, chiarendo se il problema è a compile time o a run time, e spiegandone brevemente le ragioni.
- Dopo aver rimosso le linee identificate al punto precedente, cosa stampa il programma (indicando il numero di riga che produce ogni stampa).

### Esercizio 3 (5 punti)

Utilizzando un opportuno design pattern, si scriva il codice Java, o un diagramma delle classi UML dettagliato, che consente di controllare un contatore attraverso tre bottoni 1, 2 e +. Premendo 1, il sistema resetta il contatore e predisponde un incremento di un'unità per ogni pressione di +. Similmente, premendo 2, il sistema resetta il contatore e predisponde un incremento di due unità per ogni pressione di +.

### Esercizio 4 (7 punti)

Un sistema di controllo di un parcheggio consente l'entrata delle auto solo se i posti non sono tutti occupati. Se un'auto non riesce a parcheggiare, deve aspettare almeno 5 minuti prima di poter riprovare. Ogni auto può restare parcheggiata per non più di 20 minuti. Si realizzi il sistema concorrente in Java usando solo i meccanismi base della sincronizzazione. Si esemplifichi poi il funzionamento del sistema considerando un parcheggio con 5 posti e 10 auto che vogliono parcheggiare.

### Esercizio 5 (6 punti)

Si consideri il seguente frammento di codice Java:

```

public static void method(int x) {
    if (x < 3) {
        return;
    }
    int m = x - 2;
    while (x > 0) {
        x = x % m;
        if (m == 1 || x >= 0) {
            x = x - 1;
        }
    }
    return;
}

```

Si definisca:

- Il diagramma di controllo di flusso;
- Un insieme di casi di test che coprano tutte le istruzioni;
- Un insieme di casi di test che esercitino le decisioni (*branch*);
- Un caso di test che esegua il ciclo *while* una sola volta.