ESERCIZIO 1

Completa la seguente definizione di class MemAddress che realizza il tipo di dato indirizzo di memoria.

```
class MemAddress{
//OVERVIEW: un indirizzo di memoria ha una base ed un offset,
entrambi interi non negativi

     public MemAddress(int b, int o) throws AddressFormatException
// Costruisce un memAddress con base b ed offset o, se be ed o
sono entrambi maggiori o uguali a 0
// altrimenti lancia l'ecezione

     public MemAddress max(MemAddress a, MemAddress b)
//  Se a e b hanno la stessa base restituisce l'indirizzo con
offset maggiore. Altrimenti restituisce null

     public MemAddress shift(int n) throws AddressFormatException
//Restituisce un nuovo indirizzo con la stessa base di this e come
offset il valore ottenuto sommando
//n all'offset di this. Non modifica this.
//NB: n potrebbe essere negativo

     public int toInt()
// Restituisce il valore numerico corrispondente all'indirizzo,
ottenuto dalla somma base + offset
}
```

Implementare inoltre un metodo statico main() che usi tutti i metodi della classe MemAddress e senza mai lanciare eccezioni. In particolare il corpo del metodo deve:

- creare due indirizzi a e b,
- confrontarli ed assegnare ad una variabile c il massimo dei due
- costruire un indirizzo d ottenuto dallo shift di c di un valore intero a vostra scelta
- stampare il valore numerico di d

Nel caso si verifichino eccezioni nel corso di quest'operazioni, il main le cattura e termina stampando in output il messaggio: “operazione non ammessa”

ESERCIZIO 2

Sia data la seguente classe per rappresentare un'area di memoria cache di n posizioni

```
class Cache {
     private char[] contents;

     public Cache(int n){
          if (n <= 0) throw new ArrayOutOfBoundException();
          contents = new char[n];
     }

     public void write(int i, char c) throws
ArrayIndexOutOfBoundException
     {
          contents[i] = c;
     }

     public char read(int i) throws ArrayIndexOutOfBoundException
     {
          return contents[i];
     }
}
```

Definite una sottoclasse UndoCache di Cache con un metodo undo() che permette di annullare l'effetto delle operazioni di set sulle posizioni della cache. Il metodo undo() si comporta come l'operazione di annulla disponibile in un qualunque editor, ovvero: annulla l'effetto dell'ultima operazione di set() che non sia già stata annullata; se non ci sono set() da annullare, undo() non ha alcun effetto

```
//OPERAZIONE        STATO DELLA CACHE
b = new UndoCache(4);    _::_::_::_
b.write(1,'a');               _::a::_::_
b.write(1,'b');               _::b::_::_
b.undo();           _::a::_::_
b.write(3,'c');               _::a::_::c
b.undo();           _::a::_::_
b.undo();           _::_::_::_
b.undo();           _::_::_::_
```

ESERCIZIO 3

Vogliamo costruire un sistema di classi per gestire le operazioni di un file system. La seguente classe astratta descrive la struttura comune del file system.

```
abstract class FSItem{

     protected boolean rr, ww;
     protected FSItem(String s) { nome = s; rr = ww = true; }

     public String name() { return name; }
     public void chmod(boolean r, boolean w){ rr = r; ww = w; }
}
```

Un FSItem ha un nome, due booleani che definiscono i diritti di lettura e scrittura, ed il metodo chmod() per modificarli. Le diverse componenti del filesystem sono delle classi specificate qui di seguito.

```
class Directory extends FSItem {
     //OVWERVIEW: una directory è un FSItem che può
     //contenere un insieme di sotto-directory, file o link

     public String ls() throws IllegalOperation
     // Se this permette la lettura, restituisce una stringa
     // che contiene i nome di tutti i FSItem
     // contenuti nella directory; altrimenti lancia l'eccezione

     public void add(FSItem i)
     // Aggiunge i a this se this permette la scrittura;
     // altrimenti lancia l'eccezione
}

class File extends FSItem {
//OVERVIEW: un File è un FSItem con un contenuto di tipo String

     public void set(String s) throws IllegalOperation
     // Se l'operazione di scrittura è permessa setta
     // ad s il contenuto del file altrimenti lancia l'eccezione

     public String get()
     // Se l'operazione di lettura è permessa restituisce
     //il contenuto del file altrimenti lancia l'eccezione
}

class Link extends FSItem {
     //OVERVIEW: un Link è un FSItem con associato
     // un altro FSItem, detto il target.
     // Implementa tutte le operazioni del target, e ciascuna
     // operazione viene implementata
     // mediante l'invocazione della stessa operazione sul target
}
```

Fornire l'implementazione delle classi Directory, File e Link

ESERCIZIO 4

Considerate la seguente definizione di classe, assumendo che il tipo Element contenga un metodo double val().

```
public class DataSet {

     public void add(Element x){

          if(x == null) return;
          elements.add(x);
          total = total + x.val();
          if(elements.size() == 0 || max.val() < x.val()) max = x;

     }

     public Element max() { return max; }

     public double average() {
          int count = elements.size();
          return (count == 0) ? 0 : sum/count;
     }

     private double sum;
     private Element max;
     private List<Element> elements = new LinkedList<Element>();

}
```

Estendete la classe completando la definizione dei due metodi seguenti, e definendo gli eventuali campi necessari per la loro implementazione.

```
/*
     @pre true
     @post @nochange
     @result = l'ampiezza dell'intervallo dei valori degli elementi
nel dataset, ovvero
          la differenza tra i valori massimo e minimo
*/
public double range()

/*
     @pre d >= 0
     @post @nochange
     @result = un iteratore che permette di ottenere in sequenza
gli elementi la cui misura
          è minore del valore d
*/
public Iterator<Element> lessThen(double d)
```

ESERCIZIO 5

Considerate la seguente gerarchia di tipi:

```
interface M { M m(); }
interface N { }

class A implements M {
     public M m() { return new B(); }
}

class B extends A {
     public M m() { return new A(); }
}

class C extends A implements N {
     public M m() { return this; }
}
```

Indicare il tipo statico ed il tipo dinamico per ciascuna (sotto) espressione nei seguenti frammeni di codice.
Determinare il risultato della compilazione e, nel caso non dai errori, dell'esecuzione.

```
1. N x = new C(); M y = X.m();
2. M x = new A(); B y = (B)x.m();
3. M x = new B(); B y = (B)x.m();
```

1 ESERCIZIO [2pt]

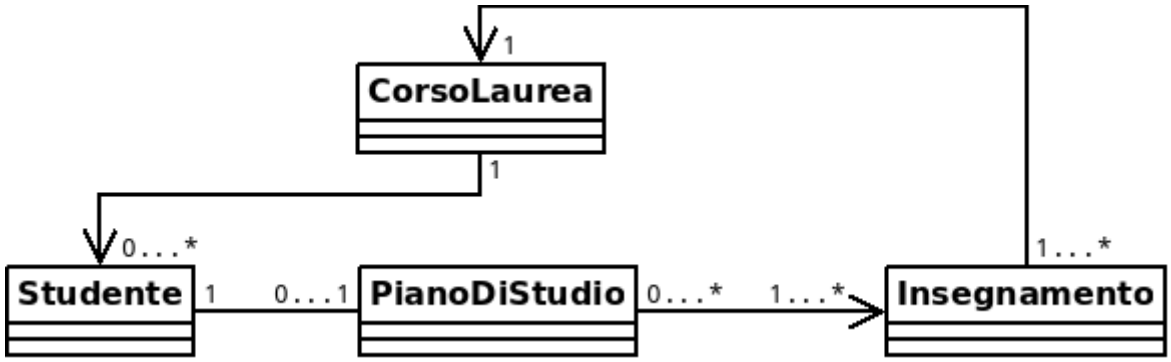
Date le seguenti definizioni di classe:

```
public class Farmacia
{
     ArrayList<Utente> utenti;
     Map<Integer, Prodotto> prodotti;
     Utente responsabile;
     Set<Scontrino> scontrini;
}

public class Scontrino
{
     ArrayList<Prodotto> prodotti;
}

public class Utente
{
     Farmacia farmacia;
}

public class Prodotto
{
}
```

Disegnare il diagramma delle classi facendo attenzione alla molteplicità e al verso delle associazioni.

2 ESERCIZIO [2pt]

Dato il seguente diagramma UML, scrivete il codice Java per definire le associazioni tra le classi prestando attenzione alla molteplicità e verso delle associazioni stesse.



3 ESERCIZIO [2pt]

Data la seguente definizione di classe

```
public class Farmacia
{
     public static String nome;
     public static Set<Prodotto> prodotti;
     public static boolean creaProdotto(Integer codice,
                                        String nome, double prezzo)
     { /* …. */ }
}
```

Applicare il pattern Singleton

4 ESERCIZIO [2pt]

Due applicazioni utilizzano il pattern MVC per poter riusare lo stesso Modello senza modifiche. Il Modello ha almeno una classe che deve informare la View del cambiamento del proprio stato interno. Illustrare come bisogna procedere e disegnare il relativo diagramma UML evidenziando le dipendenze.

5 ESERCIZIO [2pt]

Dato il seguente codice, disegnare il relativo diagramma degli oggetti creati dal main

```
public class Albero
{
     ArrayList<Albero> figli = new ArrayList<Albero>();
     Albero padre;
     String descrizione;

     public Albero(String d)
     {
          descrizione = d;
     }

     public static void main(String[] args)
     {
          Albero radice = new Albero("radice");
          Albero primoFiglio = new Albero("primo figlio");
          Albero secondoFiglio = new Albero("secondo figlio");
          radice.figli.add(secondoFiglio);
          radici.figli.add(primoFiglio);
          Albero nipote = new Albero("nipote");
          secondoFiglio.figli.add(nipote);
     }
}
```

6 ESERCIZIO [2pt]

Un'applicazione per generare numeri random utilizza una libreria A a pagamento. Successivamente decide di usare una libreria B gratuita. Indicare qual'è la maniera più conveniente per sostituire la libreria A con la libreria B e scrivere il relativo codice. (Si ricorda che il codice delle funzioni di entrambe le librerie non è conosciuto e non può essere modificato!)

Liberia A:

```
public class RandomGenerator {
     public RandomGenerator(int maxNum){ /* … */ }
     public int nextValue()
     { /* … */ } //Resituisce un numero random tra 0 e maxum
}
```

Libreria B:

```
public class Random {
     public Random() { /* … */ }
     public double random()
     { /* … */ } // Restituisce un numero random tra [0,1[
}
```

7 ESERCIZIO [2pt]

1) Che design pattern è stato usato in questo codice?
2) Disegnate il diagramma delle classi nel caso fosse stato utilizzato il polimorfismo.

```
public class Inserzione {
     String titolo;
     String testo;
     Categoria categoria;
     /* … */
}

public class Categoria {
     String descrizione;
     boolean aPagamento = false;
     public Categoria(String d)
     {
          descrizione = d;
     }
     /* … */
}

public class Giornale{
     public static void main(String[] args)
     {
          Categoria auto = new Categoria(“auto”);
          Cateogia sportETempoLibero = new Categoria(“sport”);
          Categoria elettronica= new Categoria(“elettronica”);
     }
}
```

ESERCIZIO 8 [2pt]

Dato il seguente diagramma UML, disegnate e implementate il design pattern State per la classe Utente.