Nome_____ Cognome_____ Matricola_____

Esercizio 1

Il seguente codice Java fa parte di un metodo e compila correttamente.

```
try {
    double[] b = {1.6, 3.7, 4.3};
    c = b.length + 1 - 2 * a;
    f=null;
    if (b[c]>a) f=new JFrame(""+c);
    g = new JLabel(f.toString());
    Object h = g;
    i = (String) h;
} catch (Exception e) {
    g = new JLabel(e.toString());
} finally {
    f = new JFrame();
    // inserisci l'etichetta g nel frame f
    f.pack();
    // rendi f visibile
}
```

Si possono concludere alcune proprietà delle variabili contenute. In particolare:

1) Qual è il tipo statico di **c**? Qual è il suo tipo dinamico?
2) Supponendo che **a** valga 1, dire quali sono il tipo statico e il tipo dinamico di **h**.
3) Supponendo che **a** valga 1, 2, 3: dire il tipo statico e il tipo dinamico di **e** per ogni caso.
4) Rimpiazzare i commenti contenuti nella clausola finally con istruzioni che facciano quanto descritto.

Esercizio 2

Le classi **Pendente**, **Spilla**, **Gioiello**, possiedono un metodo **sfoggia()** definito come (rispettivamente):

```
Pendente: String sfoggia(Materiale m, int p) { return("un pendente di "+p+" grammi."); }

Spilla:   String sfoggia(Materiale m, int p) { return("una spilla di "+p+" grammi."); }

Gioiello: String sfoggia(Metallo m, double p) { return("un gioiello di "+p+" grammi."); }
```

Valgono le seguenti relazioni di ereditarietà:
Spilla è sottoclasse di **Pendente** ; **Metallo** estende **Materiale** ; **Pendente** è classe base di **Gioiello**

Infine, sono date le seguenti definizioni:

```
Materiale latta = new Metallo();        Metallo oro = new Metallo();
Pendente medaglietta = new Spilla();    Gioiello collana = new Gioiello();
Pendente catenina = collana;
```

Dire cosa succede invocando **System.out.println(x)**, dove **x** è:

1) `medaglietta.sfoggia(oro,45)`
2) `collana.sfoggia(latta,58)`
3) `catenina.sfoggia(oro,23)`
4) `collana.sfoggia((Metallo)latta,12.2)`
5) `collana.sfoggia(oro,33)`

Esercizio 3

La classe **GuardaAuto** è la classe pubblica di un file e la sua struttura è come segue. Le parti mancanti sono tutte da completare. Vanno scritti anche i package necessari ed estensioni/implementazioni di classi/interfacce.

```
public class GuardaAuto {
  // campi (da inserire). Sono tutti statici:
  //    una costante CORSA, settata a 1000.0
  //    una lista concatenata di nome sentinella
  //    un osservatore di nome x e di tipo ControllaMessaggi

  public static void main(String[] args) {
    // inizializza il thread dell'osservatore

    // definisce un BufferedReader di nome in associato a System.in
    boolean finito = false;
    String str1="";     String str2="0.0";     double d;
    while (!finito) {
      try {
      //stampa "Nome?" e leggi str1
      //stampa "Velocità?" e leggi str2
      //converti str2 in un double d
        Auto a = new Auto(str1,d,sentinella);
      //associa un thread ad a e lo fa partire
      //poi stampa "Ancora?" e leggi str1
      //se il primo carattere di str1 è "n" o "N" setta finito a true
      } catch(Exception e) { System.exit(-1); }
    }
  }
}
```

il file comprende inoltre le classi **Auto** e **ControllaMessaggi** (da scrivere).

Lo scopo della classe è di descrivere l'osservare automobili che passano per il punto 0.0 di una strada rettilinea che va dalle coordinate -CORSA a +CORSA.
Il thread del main chiede quindi all'utente di generare un'auto specificandone il nome e la velocità e cicla in modo infinito finché non si risponde di no ad "Ancora?". Gli altri thread del programma sono:

- un thread singolo definito come demone associato a **ControllaMessaggi**, che stampa un messaggio al passaggio di un'auto prendendolo da **sentinella**;
- un thread per ogni auto, che viene creato ad ogni inserimento di un input da parte dell'utente

La classe **Auto** ha 4 campi: nome, velocità, posizione, e una lista concatenata di nome **sentinella**. Il costruttore richiede 3 parametri (vedi sopra). Il valore iniziale di **posizione** è +CORSA se la velocità è negativa, -CORSA se è positiva. Potrebbe essere sensato tutelarsi contro inserimenti di velocità pari a zero o molto piccole in modulo. Ogni thread associato all'auto deve restare in esecuzione finché posizione non oltrepassa il punto opposto a quello iniziale (ovvero -CORSA se la velocità è negativa, +CORSA se è positiva). Poi: resta a dormire per 100 millisecondi, controlla se si sta per passare il punto 0.0, nel qual caso fa stampare un messaggio all'osservatore, e infine incrementa effettivamente la posizione, per poi ricominciare il ciclo. Si consideri la velocità espressa in metri al decimo di secondo, in altre parole, **posizione** viene aumentata ogni volta esattamente di **velocita**.

Il messaggio da stampare (inserendolo nella lista **sentinella**) deve essere:

```
"Ho visto "+nome+" che veniva da sinistra"
"Ho visto "+nome+" che veniva da destra"
```

Per il controllo di cui sopra (se si sta per passare il punto 0), si può usare il seguente codice:

```
if (posizione>0.0 && posizione+velocita<=0.0) ...
```

(controlla il passaggio da destra o da sinistra? Come si fa nell'altro caso? Discutere)

La classe **ControllaMessaggi** invece cicla in continuo e aspetta che le venga notificato che c'è un messaggio aggiunto alla lista **sentinella**. In tal caso lo stampa.