Corso di laurea in Comunicazione digitale Fondamenti di architettura e programmazione

AA 2008/09 Prof. M.A. Alberti

I Compitino - 12 novembre 2008

Cognome **Nome**

Matricola

---

1 (10 punti)

Definite una classe `Treni`, i cui oggetti modellano i treni delle linee ferroviarie. La classe deve avere:

- un costruttore pubblico
  `public Treni (int cod, String part, String arr, int posti)` che costruisce un oggetto con un codice identificativo intero `cod`, una città di provenienza `part`, una città di arrivo `arr` e un numero di posti a sedere `posti`.
- un costruttore pubblico
  `public Treni (int cod, Strin g arr, int posti)`
che costruisce un oggetto con un codice identificativo intero, una città di arrivo e un numero di posti a sedere. La città di provenienza è di default *Milano*.
- i metodi pubblici dal significato ovvio
  `String getStPartenza(), String getStArrivo(),`
  `int getPosti(), String toString()`
- i metodi
  `String getStPartenza(String st_arrivo)`
che riporta la città di partenza del treno se la città d'arrivo è la stessa del parametro e `null` altrimenti e
  `String getStArrivo(String st_partenza)`
che riporta la città d'arrivo del treno se la città di partenza è la stessa del parametro e `null` altrimenti.

La prima cosa da osservare è che i costruttori richiedono un certo numero di parametri per poter costruire l'oggetto, il che vuol dire che gli oggetti hanno dei campi che devono essere inizializzati all'atto dell'istanziazione. La struttura della classe è quindi:

```
public class Treni {
  int codice, posti;
  String partenza, arrivo;

  public Treni(injt cod, String part, String arr, int posti) {
    codice = cod;
    this.posti=posti;
    partenza = part;
    arrivo = arr;
  }

  public Treni(int cod, String arr, int posti) {
    codice = cod;
```

```
    this.posti=posti;
    partenza = "Milano";
    arrivo = arr;
  }
```

in alternativa si puo' definire questo costruttore appoggiandosi su quello precedente. Si noti come in questo caso i parametri formali del costruttore diventino i parametri attuali da passare all'operatore this() che invoca il costruttore gia' definito:

```
  public Treni(int cod, String arr, int posti) {
    this (cod, "milano", arr, posti);
  }

  String getStPartenza() {
    return partenza;
  }

  String getStPartenza(String st_arrivo) {
    If (this.arrivo.equalsIgnoreCase(st_arrivo)
      return this.partenza;
    else return null;
  }

  String getStArrivo() {
    return arrivo;
  }

  String getStArrivo(String st_partenza) {
    If (this.partenza.equalsIgnoreCase(st_partenza)
      return this.arrivo;
    else return null;
  }

  int getPosti() {
    return posti;
  }

  int getCodice() {
    return codice;
  }

  String toString() {
    return
    "treno "+codice+": "+partenza+" – "+arrivo+". Posti: "+posti);
  }
```

---

2 (3 punti)

Definite nella classe **`Treni`** il metodo pubblico **`boolean maggiore (Treni t)`** che riporta `true` se e solo se il treno corrente ha un numero di posti maggiore del treno parametro esplicito e identica provenienza e arrivo.

```
public boolean maggiore (Treni t) {
  if (this.partenza.equalsIgnoreCase(t.partenza) &&
            this.arrivo.equalsIgnoreCase(t.arrivo))
      return this.posti > t.posti;
```

```
  else return false;
}
```

3 (3 punti)

Immaginate di aver definito due variabili `Treni treno_1, treno_2;`

Inizializzatele con valori esemplificativi e scrivete un'istruzione che faccia uso dei metodi che avete definito nell'esercizio 1. per controllare se c'è coincidenza tra i due treni, dove per *coincidenza* intendiamo soltanto che un treno arrivi in una città da cui ne parte un altro, creando in tal modo un legame diretto tra la città di arrivo del primo treno e la città di partenza del secondo. L'inizializzazione delle due variabili è necessaria per non causare errori, ma l'istruzione deve essere parametrica e generale.

```
Treni treno_1 = new Treni(458, "Firenze", 350);
Treni treno_2 = new Treni(11850, "Firenze", "Pisa", 150);
if(treno_1.getStArrivo().equalsIgnoreCase(treno_2.getStPartenza())
  out.println("coincidenza " + treno_1.getStPartenza() +
              " – " + treno_2.getStArrivo());
```

4 (2 punti)

Esprimere in linguaggio Java la seguente condizione, usando gli operatori di relazione e quelli logici:
il numero **n** deve essere **maggiore** di **3 ma non** di **8**

```
(n > 3 && n <= 8)
```

Esprimere in linguaggio Java la negazione della condizione precedente senza introdurre l'operatore di negazione (applicate la legge di De Morgan).

```
(n <= 3 || n > 8)
```

5 (2 punti)

Esprimete in Java l'affermazione "se ho i soldi e non piove oppure mi invitano allora vado alla partita", introducendo tre variabili booleane `a, b, c` che esprimano rispettivamente le condizioni "avere i soldi", "piovere" e "essere invitati", mentre l'evento "andare alla partita", sarà rappresentato da una istruzione di stampa di un messaggio opportuno.

```
boolean a, b, c;
if ( a && !b || c) out.println("vado alla partita");
```

Dire per quali valori di `a, b, c` si verifica l'evento (tabella di verità intera)

a	b	c	!b	a&&!b	a&&!b||c
F	F	F	T	F	F
F	F	T	T	F	T
F	T	F	F	F	F
F	T	T	F	F	T
T	F	F	T	T	T
T	F	T	T	T	T
T	T	F	F	F	F

T	T	T	F	T	T

---

6 (3 punti)

Date la stringa: `String riga = new String("Ehi fu, siccome immobile");`

calcolare il valore delle espressioni:

`riga.length()` **24**

```
riga.substring(4, 13).length() + riga.indexOf('m');
                  9                    13                  22
riga.substring(0,4)+
     "Ehi "
    riga.substring(4,13).toUpperCase().replace('S','R')
                "FU, RICCO"      ⇒    "Ehi FU, RICCO"
```

`riga` vale: **"Ehi fu, siccome immobile"**

---

7 (.5 punto)

Dire qual'è la caratteristica di Java che consente di definire due costruttori diversi, come ad esempio nella classe **`Treni`**:

overloading o sovraccarico

Dire come si distinguono i due costruttori:

dai parametri (numero, ordine, tipo), ovvero dalla segnatura o firma

---

8 (.5 punto)

Dire qual'è la caratteristica di Java che consente di specializzare un metodo, come il metodo `toString()`, per una data classe:

sovrascrittura

---

9 (4 punti)

Date le variabili: `int a = 2, b = 4;` calcolare i valori delle due variabili, dopo aver eseguito separatamente i due blocchi di istruzioni:

`a = a++ + --b;` `b += a – --b;`	`a:` **5** `b:` **6**

Infatti dopo la prima istruzione: **`a = 5 b = 3`**

Dopo la seconda che non modifica la variabile `a`:

`b = b + a - --b = 3 + 5 – 2 = 6`

E ancora dopo l'inizializzazione a = 2, b = 4 :

`a = ++a + b--;` `b += a - b--;`	`a:` **7** `b:` **7**

Infatti dopo la prima istruzione: **`a = 7 b = 3`**

Dopo la seconda che non modifica la variabile `a`:

`b = b + a – b-- = 3 + 7 - 3`

---

10 (5 punti)

Usando la classe **`Dado`** introdotta a lezione si scriva il frammento di programma che utilizza due dadi, che vanno dichiarati e istanziati, e un ciclo, in cui si lanciano i dadi fino all'ottenimento di un 12. Una variabile intera conta il numero di iterazioni.

```
Dado dado1 = new Dado(), dado2 = new Dado();
int tot, lanci = 0;

do {
  tot = dado1.lancio() + dado2.lancio();
  lanci++;
  out.println("dado1: "+dado1.leggi()+", dado2: "+dado2.leggi());
} while (tot != 12);
out.println(lanci);
```

oppure:

```
tot = dado1.leggi() + dado2.leggi();
lanci =  1;
while (tot != 12) {
  out.println("dado1: "+dado1.leggi()+", dado2: "+dado2.leggi());
  tot = dado1.lancio() + dado2.lancio();
  lanci++;
}
out.println(lanci);
```

oppure:

```
tot = dado1.leggi() + dado2.leggi();
for (lanci = 1; tot < 12; lanci++) {
   out.println("dado1: "+dado1.leggi()+", dado2: "+dado2.leggi());
   tot = dado1.lancio() + dado2.lancio();
}
```