Metodologie di Programmazione 2003 – 2004

PRIMO APPELLO: 23 Gennaio 2004

Nome: _____ Matricola: _____

Istruzioni

- Scrivete il vostro nome su tutti i fogli.
- Scrivete le soluzioni nello spazio riservato a ciascun esercizio.
- Giustificate le risposte: le risposte senza giustificazione non saranno considerate.
- Tempo a disposizione 2 ore e 30.
- No libri, appunti o altro.

LASCIATE IN BIANCO:

A1	
A2	
A3	
B1	
B2	
B3	
Totale	

Nome: _____ Matricola: _____

Esercizio A1 Considerate la seguente gerarchia di classi:

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

class A implements M {
   public M m() { return this; }
}
class B extends A {
   public void k() { }
}
class C extends A implements N {
   public void n() {}
   public void p() {}
}
```

Quale è il risultato della compilazione e della (eventuale, nel caso la compilazione non dia errori) esecuzione dei seguenti frammenti? **Motivate le risposte**

1. `N x = new C(); M y = x.m();`

2. `M x = new A(); B y = (B)x.m();`

3. `A x = new C(); ((C)x).p();`

Nome: _____ Matricola: _____

Esercizio A2 Considerate le seguenti classi.

```
class A {
  void test(double x)
  { this.foo(x); }
  void foo(double x)
  { System.out.print("A"); }
}
class B extends A {
  void foo(double x)
  { System.out.print("B-double"); }
  void foo(int x)
  { System.out.print("B-int"); }
}

A a = new A(); B b = new B();
```

1. Cosa stampa `b.test(1)` ? **Motivate la risposta**

2. Cosa stampa `b.test(1.0)` ? **Motivate la risposta**

3. Cosa stampa `b.foo(1)` ? **Motivate la risposta**

Nome: _____ Matricola: _____

Esercizio A3 Dato un tipo `T`, definite il corpo del seguente metodo:

```
int hopeYouKnowHowToIterateOnAList(List l) {
  // scorre la lista l e restituisce il numero di
  // elementi in l che hanno tipo SomeType









}
```

Nome: _____ Matricola: _____

Parte B In questo set di esercizi dovete costruire un sistema di classi per la gestione di un albergo.

L'ALBERGO La classe `Albergo` gestisce un insieme di camere, suddivise in due liste: la lista delle camere libere, e la lista delle camere occupate. Inizialmente tutte le camere sono libere.

La classe `Albergo` ha un costruttore `Albergo(int n)` che costruisce una lista di `n` camere (tutte libere) in cui le camere con numero dispari hanno tipo `Camera` mentre le camere di numero pari hanno tipo `CameraConTV` (vedi seguito). La classe fornisce inoltre due metodi:

- `Camera checkin (Cliente c) throws FullyBookedException`: restituisce la prima camera libera, se ne esiste una, la rimuove dalla lista e la inserisce nella lista delle camere occupate. Se non esiste alcuna camera libera lancia una `FullyBookedException`
- `double checkout (Cliente c, int giorni)`: rimuove la camera occupata dal cliente `c` dalla lista delle camere occupate e la inserisce nella lista delle camere libere. Restitusce il conto che il cliente deve pagare: il conto è calcolato moltiplicando il costo della camera per `giorni`.

Le CAMERE Le camere sono orgnizzate in una gerarchia che modella due tipologie di camere. Ogni `Camera` ha un numero (di tipo `int`), un cliente (il cliente *checkedIn* nella camera) ed un prezzo (di tipo `double`). La classe `Camera` ha due costruttori. Il primo prende un intero che utilizza come numero della camera, mentre inizializza il prezzo ad un valore di default. Il secondo costruttore prende un intero ed un double che utilizza per inizializzare il numero della camera e il prezzo. Al momento della costruzione la camera non ha un cliente associato.

- un metodo `double costo()` che restituisce il prezzo;
- un metodo `void entra(Cliente c):throws NotYourRoomException`: se `c` è il cliente *checkeIn* nella camera non ha effetto, altrimenti lancia `NotYourRoomException`

Alcune camere, di tipo `CameraConTV`, hanno la televisione ed un metodo:

- `void usaTV()`: alla prima invocazione, segnala che la televisione è stata usata.

`CameraConTV` è sottotipo di `Camera`. Nelle camere con TV, il metodo `costo()` restituisce il prezzo della camera sommato ad una costante `FORFAIT`, di tipo double, nel caso in cui cliente abbia usato la televisione.

I CLIENTI Ogni cliente ha un nome, di tipo `String`, ed è associato ad una camera. Il costruttore della classe `Cliente` ha un unico parametro che ne determina il nome: al momento della costruzione, pertanto, un cliente non è associato ad alcuna camera. Inoltre, i clienti hanno i seguenti metodi:

- `boolean checkIn(Albergo a)`: richiede all'albergo una camera. Se l'albergo ha camere libere, occupa la camera in questione e restitusce `true`. Altrimenti restituisce `false`. (nota: questo metodo *non* lancia `FullyBookedException`).
- `void checkOut(Albergo, ngiorni)`: richiede all'albergo il *checkOut* e stampa il valore del conto;
- `void entra() throws NotYourRoomException`: prova ad entrare in camera, se il cliente ha una camera associata;
- `boolean watchTV()`: utilizza la televisione, se la camera la fornisce. In questo caso restituisce `true`, altrimenti `false`.

In tutte le classi utilizzate i qualificatori `private` per tutti i campi, fornendo i metodi *get* e *set* se/quando necessario. Utilizzate i metodi forniti per le liste dalla libreria nel package `java.util`, ed in particolare:

```
boolean isEmpty();
boolean add(Object o);
Object remove(int index);
Object remove(Object o);
```

Nome: _____ Matricola: _____

Esercizio B1 Definite la classe `Albergo` e la classe `FullyBookedException`.

Nome: _____ Matricola: _____

Esercizio B2 Definite le classi `Camere` e `CamereConTV`, e la classe `NotYourRoomException`.

Nome: _____ Matricola: _____

Esercizio B3 Definite la class `Cliente`