PROGRAMMAZIONE II (A,B) - a.a. 2019-20
Soluzione Prima Valutazione Intermedia – Novembre 2019

Domande di base

1. Cosa si intende per *dynamic dispatch* nel contesto dei linguaggi di programmazione a oggetti?
   il dynamic dispatch *è la tecnica usata per determinare il metodo giusto da applicare ad un oggetto in base suo tipo dinamico (invece che al tipo statico).*

2. Si consideri il seguente programma (scritto con una sitassi Java-like)

```
class Alpha {}

class Beta extends Alpha {}

class Gamma extends Alpha {}

class Epsilon extends Alpha{}

class Fruit extends Gamma{}

class Golf extends Beta {}

class Orange extends Fruit{}
:
\\ main
Fruit f = new Fruit();
Alpha a = f;
Gamma g = f;
a = g;
 }
```

   Definire il tipo statico e quello dinamico per ogni variable presente nel programma. *La variabile f ha tipo statico e dinamico* `Fruit`*. La variabile a ha tipo statico* `Alpha` *e dinamico* `Fruit`*. La variabile g ha tipo statico* `Gamma` *e dinamico* `Fruit`*.*

3. Supponiamo di inserire alla fine del codice del `main` le istruzioni seguenti

```
Beta b = f;
a = b;
```

   Il programma così modificato supera il controllo statico dei tipi? Motivare la risposta. *Non lo supera! Infatti la variabile b ha tipo statico* `Beta` *e f ha tipo statico* `Fruit` *ma* `Fruit` *non è sottotipo di* `Beta` *.*

Esercizio 1

Si consideri un tipo di dato astratto *RecentContacts* introdotto per contenere la lista dei contatti di una chat ordinati sulla base dell?ultima interazione (dalla piiù alla meno recente). L'interfaccia `RecenrContacts` include, tra gli altri, i seguenti metodi

- `public void add(String contact)` *il cui effetto è quello di inserire il contatto (descritto dal parametro* `contact`*) in fondo alla lista dei contatti*

- `public void remove(String contact)` *il cui effetto è quello di rrimuovere il contatto dalla lista dei contatti*
- `public void notifyInteraction(String contact)` *il cui effetto è quello di rmodificare l'ordine della lista a seguito dell'interazione (invio o recezione di un messaggio) da parte del contatto* `contact`
- `public int size ()` *restituisce il numero dei contatti presenti*
- `public String get(int pos)` *il cui effetto è quello di restituire il contatto del contatto nella lista alla posizione pos*
- `public String min ()` *restituisce il contatto aventi in interazioni meno recenti in chat*

1. Si definisca la specifica dell'interfaccia `RecentContacts`, indicando l'elemento tipico e per ogni metodo le clausole REQUIRES, MODIFY ed EFFECTS, il valore restituito e le eventuali eccezioni lanciate in dipendenza dei parametri attuali.

   Si veda il file *RecentContacts.java*

2. Si assuma di implementare la classe `MyRecentContacts` con le seguenti variabili d'istanza:

```
private int elements;
private ArrayList<String> contatti;
```

   - Si definiscano la *funzione di astrazione e l'invariante di rappresentazione.* Dire se esistono proprietà rilevanti per la classe `MyRecentContacts` che non si possono esprimere con l'invariante di rappresentazione proposto.

$$FA = [this.contatti.get(0), \ldots, [this.contatti.get(elements - 1)).$$

$$\begin{aligned} IR = {} & this.elements = this.contatti.size() \ and \ this.contatti! = null \ and \\ & \text{forall}(inti; 0 <= i < this.elements; this.contatti.get(i)! = null) \ and \\ & \text{forall}(inti; 0 <= i < this.elements; \\ & \text{forall}(intj; 0 <= j < this.elements \ and \ i! = j; \\ & !(this.contatti.get(i).equals(this.contatti.get(j))))) \end{aligned}$$

   L'invariante non cattura la proprietà della lista in cui contatti devono essere ordinati in base alle interazioni più recenti.

   - Si implementi il costruttore ed il metodo `notifyInteractions` verificando che preservino l'invariante di rappresentazione.

```
public class MyRecentContacts implements RecentContacts {

private int elements;
private ArrayList<String> contatti;

public MyRecentContacts (){
elements=0;
contatti=new  ArrayList<String>();
}



public void notifyInteraction(String contact)  throws UnknownContactException{
if contact == null throw new UnknownContactException("");
 boolean trovato=false; int i = 0;
  while (! trovato && ii < elements){
```

```
   { if  contact. equals(contatti.get (i)) then trovato=true; }  i++;}
   if ! trovato throws new UnknownContactException("");
   elements++;
 if (elements==0) then contatti.add(contact); else {contatti.add(0,contact);
boolean  res= contatti.remove(contact);}  }

}
```

- Si consideri il tipo `Contacts`, che mantiene la lista dei contatti senza un ordine preciso (le postcondizioni d

  ei modificatori non specificano un ordine preciso per gli elementi). Il tipo `Contacts` può essere sottotipo di `RecentContacts` secondo il principio di sostituzione? È possibile il contrario? Motivare le proprie risposte.
  Il tipo `Contacts` non può essere sottotipo di RecentContacts perch è non rispetterebbe il vincolo di ordinamento. Ad esempio, questo indebolirebbe le post-condizioni di `add` e `notifyInteraction`. Il tipo `RecentContacts` può invece essere sottotipo di `Contacts`, in quanto l'ordine cronologico di interazione rappresenta un caso particolare dell' assenza di ordine (ovvero di un ordinamento qualsiasi).

Esercizio 2

Si consideri la seguente specifica

```
class Cell {
  private  int n;

// @requires true
// @modifies this
// @effect post(this.n)  = p
public void set(int p) { ... }

// @requires true
// @result = pre(this.n))
// @effetct pre(this.n) = post(this.n)
  public int get() { ... }

// @equires true
// @modify this
/// @effect post(this.n) > pre(this.n)
  public void inc() { ... }
}
```

- Dire se l'implementazione del metodo `set` definita dal codice `if(p > 0) n = p;` rispetta la specifica. Motivare la risposta.
  Non la soddisfa infatti il metodo ha pre-condizione true e quindi per ogni valore del parametro deve valere la post-condizione (post(this.n) = p). Invece l'implementazione data modifica il valore solo quando positivo.

- Modificare la specifica in modo tale che la variabile di istanza della classe `Cell` sia sempre positivo.

```
class Cell {
  private  int n;

  // @effect post(this.n)  = 1
```

```
  public Cell(){}

// @requires p>0
// @modifies this
// @effect post(this.n)  = p
public void set(int p) { ... }

.....come sopra }
```