Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Dipartimento di Elettronica e Informazione

Esercitazioni di Ingegneria del Software

Ereditarietà

1. Veicoli

- Siano date le classi:

```
class VeicoloAMotore {
 //@requires !acceso();
 //@ensures acceso();
 public void accendi();
 //@requires acceso();
 //@ensures !acceso();
 public void spegni();
 //@ensures (*\result true se motore acceso*)
 public boolean acceso();
```

1. Veicoli

- Dire se la seguente classe soddisfa il principio di sostituzione di Liskov rispetto alla sua superclasse:

```
class Automobile extends VeicoloAMotore {

 //@ensures (*nuova auto non accesa*)
 public Automobile() {…}

 //@also
 //@requires true
 //@ensures acceso() <==> !\old(acceso());
 public void accendi();
```


1. Veicoli

- Soluzione: il metodo accendi() in overriding potrebbe violare la regola dei metodi. La precondizione è più debole nella sottoclasse (ok "require no more"). Per quanto riguarda la postcondizione, ricordiamo che il metodo ridefinito deve comportarsi come il metodo originale *solo quando le precondizioni del metodo del padre sono soddisfatte*. Nel nostro caso, quando acceso == false, l'effetto dei due metodi è lo stesso; chi invoca accendi() su una Automobile come se fosse un VeicoloAMotore non lo farà con acceso == true. Il principio di sostituzione è soddisfatto.

1. Veicoli

- La classe AutomobileElettrica qui definita soddisfa il principio di sostituzione?

```
class AutomobileElet extends Automobile {

 //@ensures (*nuova auto accesa*)
 public AutomobileElettrica() {…}

 //@also
 //@requires !acceso();
 //@ensures acceso();
 public void accendi();
```


1. Veicoli

- Soluzione: il principio non è soddisfatto perché la precondizione è più restrittiva rispetto a quella del metodo accendi() in Automobile.

2. Archivio cd di opere musicali

- Classi di supporto per l'archivio: Opera

```
public /*@pure@*/ class Opera {
 public final String compositore;
 public final String opera;
 //@ensures c != null && o != null &&
 //@         compositore == c && opera == o;
 //@signals(NullPointerException e) c==null||o==null
 public Opera (String c, String o) { … }
 //@ensures \result == true <==>
 //@  o instanceof Opera &&
 //@  ((Opera)o).opera.equals(this.opera) &&
 //@  ((Opera)o).compositore.equals(this.compositore);
 public /*@pure@*/ boolean equals (Object o) { … }
}
```


2. Archivio cd di opere musicali

- Classi di supporto per l'archivio: Interpretazione e CD

```
public /*@pure@*/ class Interpretazione {
 public final String direttore;
 public final String orchestra;
 public final int anno;
 …
}
public /*@pure@*/ class CD {
 public final Opera op;
 public final Interpretazione interp;
 public final int giudizio;
 …
}
```

2. Archivio cd di opere musicali

- Classe principale: SimpleAdviser (archivio di cd con metodo che consiglia cd qualsiasi in base all'opera)

```
public class SimpleAdviser {
 …
 //@ requires o != null;
 //@ ensures \result.op.equals(o);
 public CD consigliaCD(Opera o) {…}
 …
}
```


2. Archivio cd di opere musicali

- La classe smartAdviser estende SimpleAdviser per consigliare CD con giudizio massimo

```
public class SmartAdviser extends SimpleAdviser {
 …
 //@also
 //@requires o != null;
 //@ensures !(\exists CD c; isIn(c);
      c.giudizio > \result.giudizio);
 public CD consigliaCD(Opera o) {…}
 …
}
```