Esercizi Design Pattern

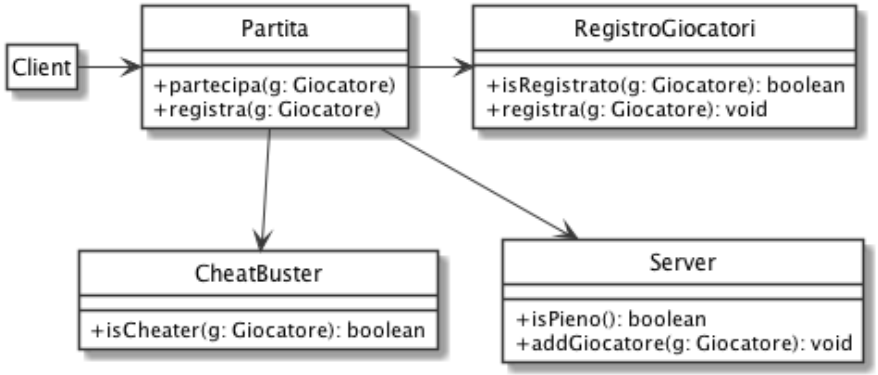
Esercizio 1

```
public class Partita {
  CheatBuster cb = new CheatBuster();
  Server s = new Server(10);
  RegistroGiocatori rg = new RegistroGiocatori();

  public void registra(Giocatore g) {
    rg.registra(g);
  }

  public void partecipa(Giocatore g) {
    boolean isRegistrato = rg.isRegistrato(g);
    boolean isCheater = cb.isCheater(g);
    boolean serverPieno = s.isPieno();

    if (isRegistrato && !isCheater && !serverPieno)
      s.addGiocatore(g);
    }
  }

  public Server getServer() {
    return s;
  }
}
```

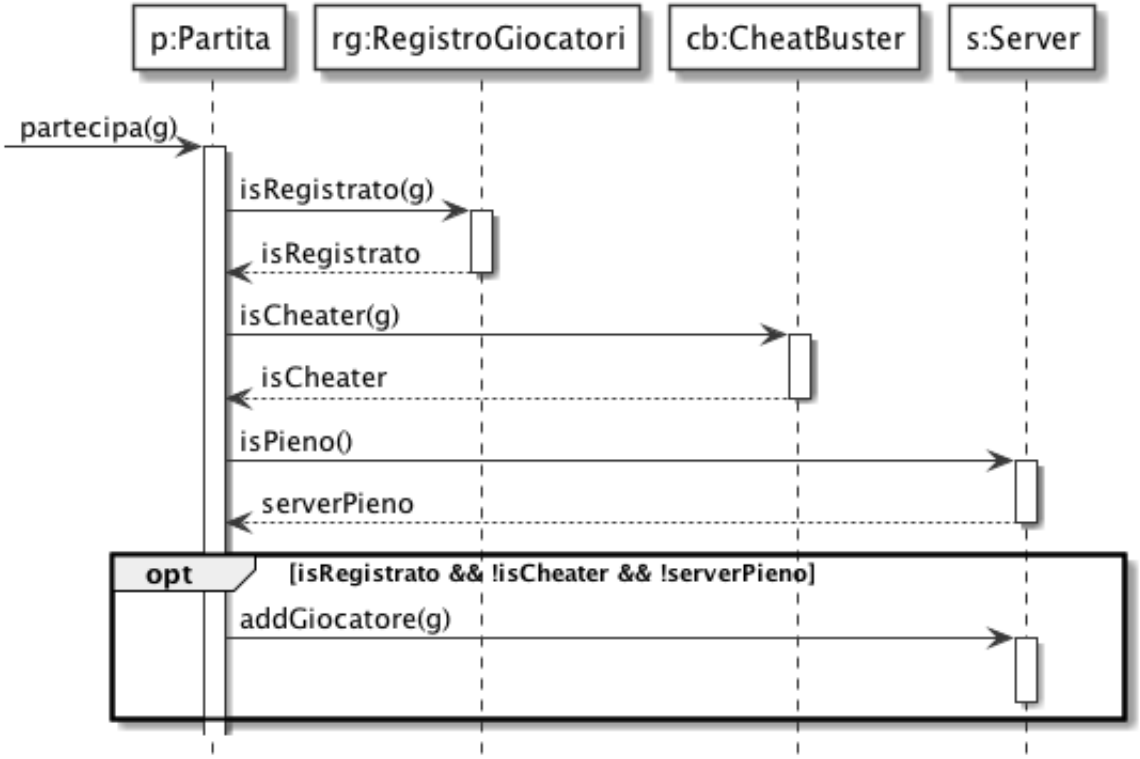
Domanda 1: qual è il design pattern implementato?
Domanda 2: ci sono problemi di implementazione del design pattern? Se sì, quali?
Domanda 3: che ruolo hanno la classe Parita, Giocatore, CheatBuster, Server e RegistroGiocatori?
Domanda 4: disegnare il diagramma UML delle classi (dopo le eventuali correzioni al codice).
Domanda 5: disegnare un diagramma UML di sequenza per il metodo partecipa().

Risposta 1: Façade.
Risposta 2: il Façade non dovrebbe esporre le classi interne al sottosistema: il metodo getServer() andrebbe eliminato.
Risposta 3: Parita ha il ruolo di Façade; Giocatore è una classe di supporto esterna al sottosistema; CheatBuster, Server e RegistroGiocatori sono classi del sottosistema nascosto dal Façade.

Risposta 4



Risposta 5



Esercizio 2

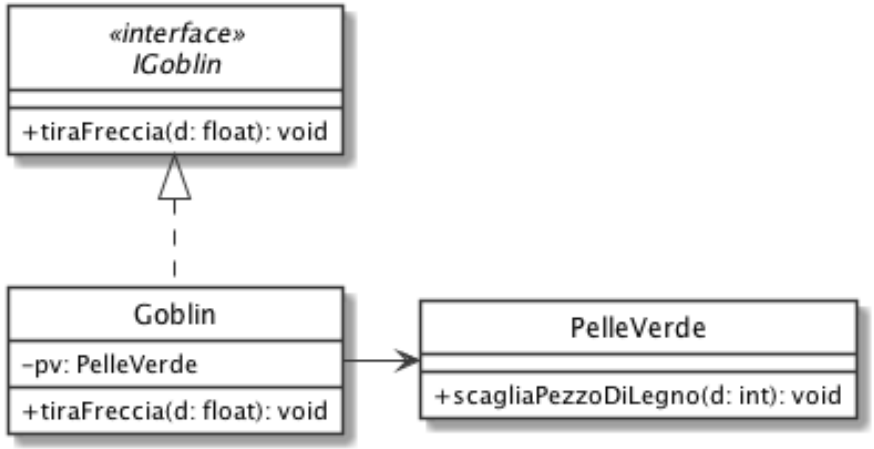
```java
public class Goblin implements IGoblin {
  PelleVerde pv = new PelleVerde();

  public void tiraFreccia(float d) {
    int d2 = Math.round(d * 10);
    pv.scagliaPezzoDiLegno(d2);
  }
}

public class PelleVerde {
  public void scagliaPezzoDiLegno(int d) {
    System.out.println("- danno: " + d);
  }
}
```

Domanda 1: qual è il design pattern implementato? Quale variante nota?
Domanda 2: che ruolo hanno la classe Goblin, IGoblin e PelleVerde?
Domanda 3: scrivere il codice dell'interfaccia IGoblin.
Domanda 4: disegnare il diagramma UML delle classi.
Domanda 5: disegnare un diagramma UML di sequenza per il metodo partecipa().
Domanda 6: scrivere il codice di un client che utilizzi opportunamente le classi del design pattern.
Domanda 7: modificare il codice in modo da ottenere Lazy Initialization.
Domanda 8: modificare il codice in modo da ottenere un'altra variante nota (indicandone il nome).

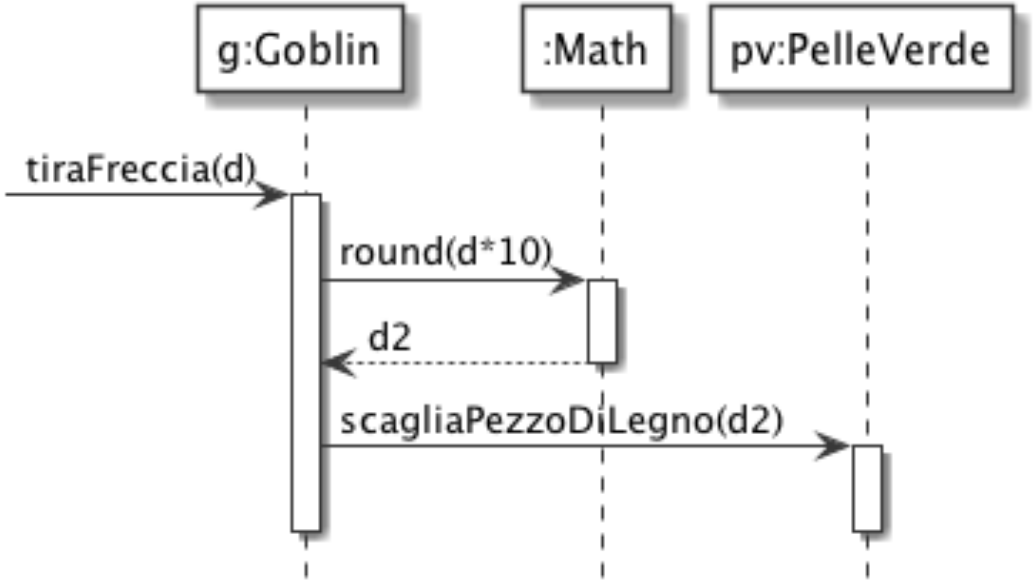
Risposta 1: Adapter, variante Object Adapter;
Risposta 2: Goblin ha il ruolo di Adapter; IGoblin ha il ruolo di Target, PelleVerde ha il ruolo di Adaptee;
Risposta 3:

```
public interface IGoblin {
  public void tiraFreccia(float d);
}
```

Risposta 4



Risposta 5



Risposta 6:

```
// il risultato atteso è la stampa di: “danno: 52”
public void client() {
  IGoblin g = new Goblin();
  g.tiraFreccia(5.23);
}
```

Risposta 7:

```
public class Goblin implements IGoblin {
  PelleVerde pv = null;

  public void tiraFreccia(float d) {
    if (pv == null) pv = new PelleVerde();
    int d2 = Math.round(d * 10);
    pv.scagliaPezzoDiLegno(d2);
  }
}
```

Risposta 7: un’altra variante nota è il Class Adapter

```
public class Goblin implements IGoblin extends PelleVerde {
  public void tiraFreccia(float d) {
    int d2 = Math.round(d * 10);
    this.scagliaPezzoDiLegno(d2);
  }
}
```

Esercizio 3

```
public class AssistenteVocale extends T1 {
  List<ComandoVocale> storicoComandi = new ArrayList<>();
  public void registra(ComandoVocale cv) {
    storicoComandi.add(cv);
    this.m1();
  }
  T2 getUltimoComando() {
    return storicoComandi.get(storicoComandi.size()-1);
  }
}
public abstract class T1 {
  private List<T2> l = new ArrayList<>();
  public void m1() {
    for (T2 x : l)
      x.m2();
  }
}
public class SmartLight implements T2 {
  private AssistenteVocale assistente;
  public void m2() {
    ComandoVocale cv = assistente.getUltimoComando();
    elaboraComando(cv);
  }
}
```

Domanda 1: qual è il design pattern implementato? Quale variante nota?
Domanda 2: indicare i tipi appropriati per T1, T2
Domanda 3: indicare i nomi appropriati per i metodi m1 e m2
Domanda 4: che ruolo hanno le classi AssistenteVocale e SmartLight?
Domanda 5: completare il codice della classe T1 (se necessario) e scrivere il codice dell'interfaccia T2

Risposta 1: Observer, variante pull;
Risposta 2: Subject e Observer
Risposta 3: tipicamente notify() e update() (nota: in java notify() è già definito in Object, quindi bisogna usare un nome o una firma diversa, es. notifyObservers())
Risposta 4: ConcreteSubject e ConcreteObserver
Domanda 5:

```
public abstract class Subject {
  private List<Observer> obList = new ArrayList<>();
  public void attach(Observer ob) {
    if (! obList.contains(ob)) {
      obList.add(ob);
    }
  }
  public void detach(Observer ob) {
    if (obList.contains(ob)) {
      obList.remove(ob);
    }
  }
  public void notifyObservers() {
    for (Observer ob : obList)
      ob.update();
  }
}
public interface Observer {
  public void update();
}
```

```
public class AssistenteVocale extends T1 {
  List<ComandoVocale> storicoComandi = new ArrayList<>();
  public void registra(ComandoVocale cv) {
    storicoComandi.add(cv);
    this.m1();
  }
  T2 getUltimoComando() {
    return storicoComandi.get(storicoComandi.size()-1);
  }
}
public abstract class T1 {
  private List<T2> l = new ArrayList<>();
  public void m1() {
    for (T2 x : l)
      x.m2();
  }
}
public class SmartLight implements T2 {
  private AssistenteVocale assistente;
  public void m2() {
    ComandoVocale cv = assistente.getUltimoComando();
    elaboraComando(cv);
  }
}
```

Domanda 6: cosa bisogna modificare per poter ottenere un'altra variante nota?

Domanda 7: modificare il codice usando i tipi Publisher, Subscriber, SubmissionPublisher e Subscription di Java 9