Esempi di domande Design Pattern

Esempio 1

1. Dire qual è il design pattern implementato per il codice mostrato

```
public class Balancer {
   private String[] hosts = new String[]{"host1", "host2", "host3"};
   private int x;
   private static Tipo1 b = new Balancer();

   private Balancer() {
      x = 0;
   }

   public Tipo2 getHost() {
      if (x == hosts.length)
         x = 0;
      return hosts[x++];
   }

   public mod1 Tipo3 metodo1() {
      return b;
   }

}
```

Risposta: Singleton

2. Indicare i tipi appropriati per i tipi Tipo1, Tipo2, Tipo3

```
public class Balancer {
   private String[] hosts = new String[]{"host1", "host2", "host3"};
   private int x;
   private static Tipo1 b = new Balancer();

   private Balancer() {
      x = 0;
   }

   public Tipo2 getHost() {
      if (x == hosts.length)
         x = 0;
      return hosts[x++];
   }

   public mod1 Tipo3 metodo1() {
      return b;
   }

}
```

Risposta: Balancer, String, Balancer

3. Indicare qual è il modificatore mod1

```
public class Balancer {
   private String[] hosts = new String[]{"host1", "host2", "host3"};
   private int x;
   private static Tipo1 b = new Balancer();

   private Balancer() {
      x = 0;
   }

   public Tipo2 getHost() {
      if (x == hosts.length)
         x = 0;
      return hosts[x++];
   }

   public mod1 Tipo3 metodo1() {
      return b;
   }

}
```

Risposta: static

4. Dire cosa fa il metodo1 e qual è tipicamente il vero nome

```
public class Balancer {
   private String[] hosts = new String[]{"host1", "host2", "host3"};
   private int x;
   private static Tipo1 b = new Balancer();

   private Balancer() {
      x = 0;
   }

   public Tipo2 getHost() {
      if (x == hosts.length)
         x = 0;
      return hosts[x++];
   }

   public mod1 Tipo3 metodo1() {
      return b;
   }

}
```

Risposta: è il metodo che restituisce l'unica istanza presente, tipicamente è chiamato *getInstance*

5. Implementare il codice che istanzia opportunamente la classe e chiama i metodi implementati

```
public class TestBalancer {

   public static void main(String[] args) {
       Balancer b = Balancer.getInstance();
       for (int i = 0; i < 6; i++) {
          System.out.println("Call: " + b.getHost());
       }
    }
}
```

output

```
Call: host1
Call: host2
Call: host3
Call: host1
Call: host2
Call: host3
```

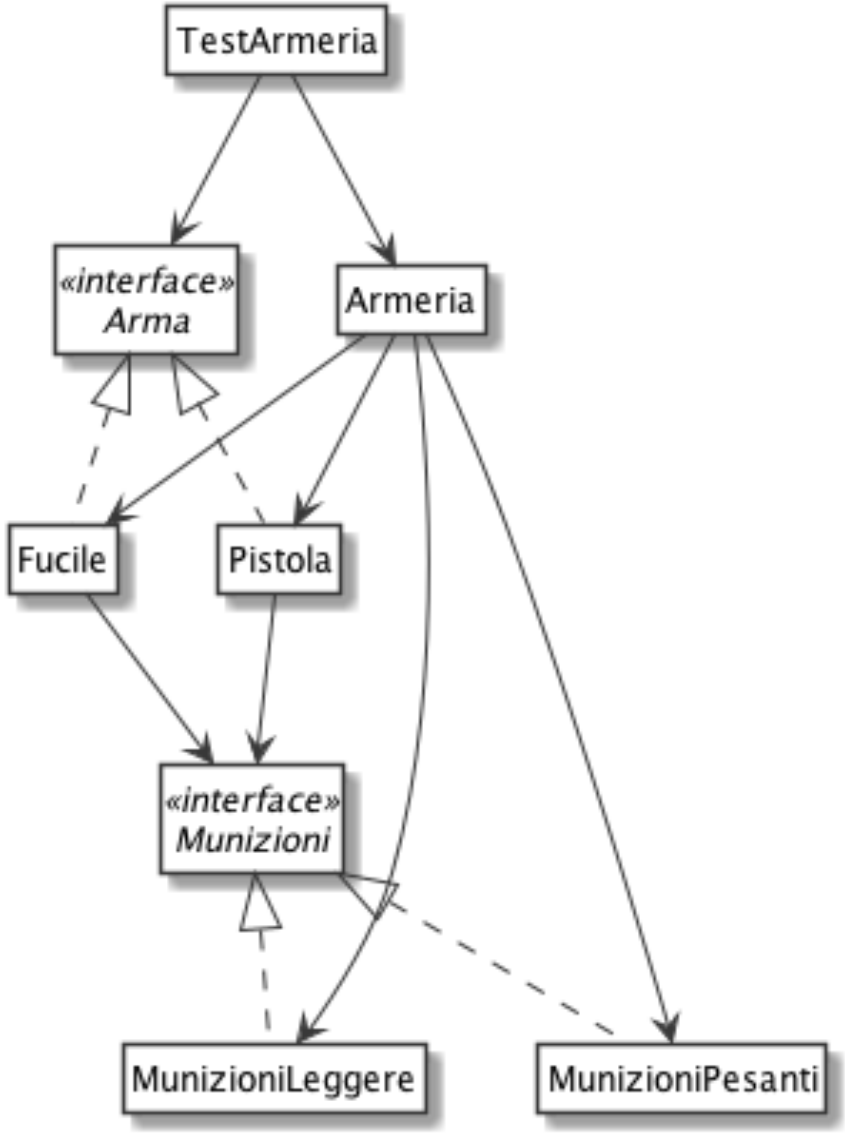
Esempio 2

1. Disegnare il diagramma UML delle classi per il codice mostrato

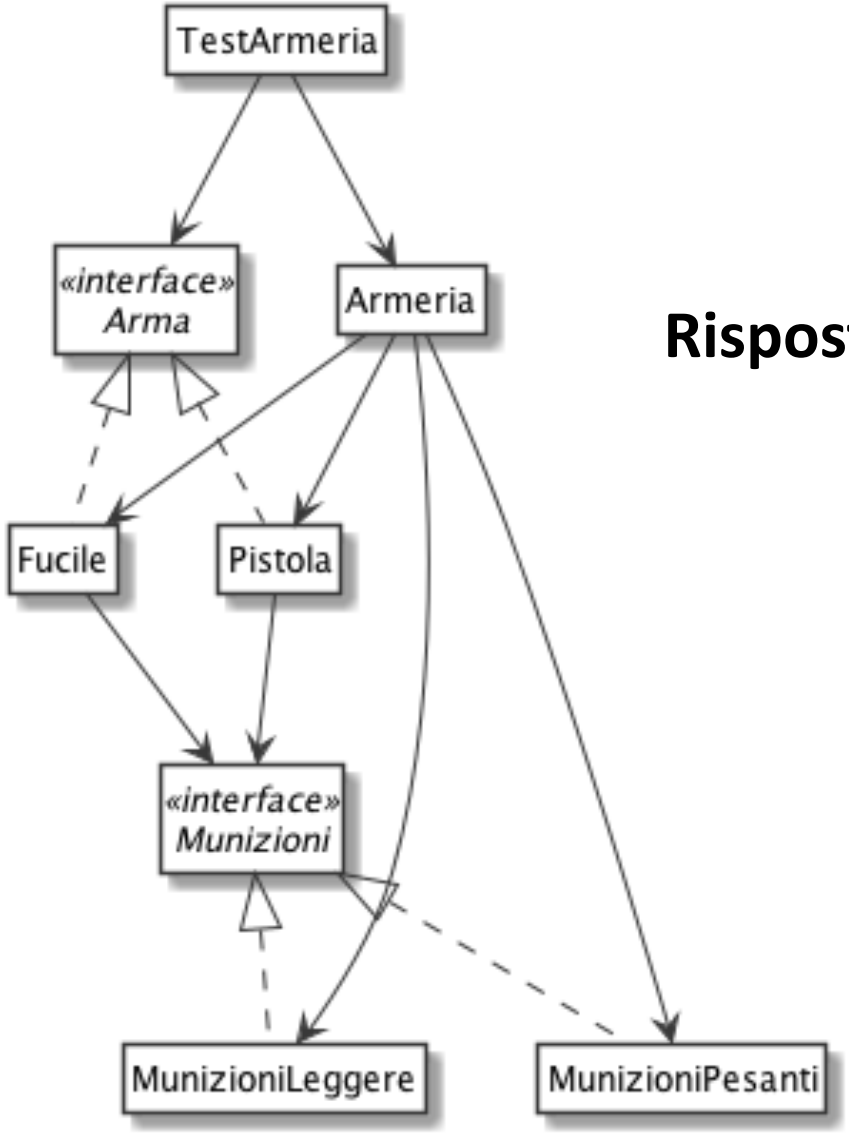
```
public interface Arma {
   public String getTipo();
   public int getDannoArea();
   public int getDannoMirato();
}

public interface Munizioni {
   public int getMoltiplicatoreDanno();
}

public class Fucile implements Arma {
   private Munizioni m;

   public Fucile(Munizioni m) {
      this.m = m;
   }
   public String getTipo() {
      return "Fucile";
   }
   public int getDannoArea() {
      return 100 * m.getMoltiplicatoreDanno();
   }
   public int getDannoMirato() {
      return 20 * m.getMoltiplicatoreDanno();
   }
}
```

```
public class MunizioniPesanti implements Munizioni {
   public int getMoltiplicatoreDanno() {
      return 5;
   }
}

public class Armeria {
   public static Arma getFucilePesante() {
      return new Fucile(new MunizioniPesanti());
   }

   public static Arma getPistolaOrdinanza() {
      return new Pistola(new MunizioniLeggere());
   }
}
```

Diagramma UML delle classi



2. Dire qual è il design pattern implementato



Risposta: Factory Method

3. Dire il ruolo svolto dalle interfacce Arma e Munizioni

Risposta: Arma svolge il ruolo di Product; Munizioni è un'interfaccia da cui dipende Fucile. Munizioni permette di realizzare Depencency Injection

4. Dire il ruolo svolto dalle classi Fucile, MunizioniPesanti e Armeria

Risposta: Fucile svolge il ruolo di ConcreteProduct; MunizioniPesanti implementa Munizioni e si può iniettare dentro Fucile; Armeria svolge il ruolo di ConcreteCreator.

5. Implementare il codice delle classi Pistola e MunizioniLeggere

```
public class Pistola implements Arma {
   private Munizioni m;
   public Pistola(Munizioni m) {
      this.m = m;
   }
   public String getTipo() {
      return "Pistola";
   }
   public int getDannoArea() {
      return 20 * m.getMoltiplicatoreDanno();
   }
   public int getDannoMirato() {
      return 50 * m.getMoltiplicatoreDanno();
   }
}

public class MunizioniLeggere implements Munizioni {
   public int getMoltiplicatoreDanno() {
      return 2;
   }
}
```

6. Implementare il codice che istanzia le classi opportune e chiama i metodi definiti in Arma

```
public class TestArmeria {
    public static void main(String[] args) {
        Arma a = Armeria.getFucilePesante();
        Arma b = Armeria.getPistolaOrdinanza();

        System.out.println("Danno ad area " + a.getTipo() + ": " + a.getDannoArea());
        System.out.println("Danno mirato " + b.getTipo() + ": " + b.getDannoMirato());
    }
}
```