```java
/**
 * Prodotto svolge il ruole di Component per il design pattern Composite.
 *
 * I metodi add() e remove() in Prodotto forniscono trasparenza nell'uso di
 * Leaf o Composite, tuttavia non si ha sicurezza.
 */

public interface Prodotto {
   public void print();

   public float getPrezzo();

   public int getPeso();

   public void add(Prodotto p);

   public void remove(Prodotto p);
}



/**
 * Libro svolge il ruolo di Leaf per il design pattern Composite
 */

public class Libro implements Prodotto {
   private String titolo;
   private float prezzo;
   private int peso;

   public Libro(String titol, float prez, int pes) {
      titolo = titol;
      prezzo = prez;
      peso = pes;
   }

   @Override
   public void print() {
      System.out.println("Libro: " + titolo + "\tPrezzo: " + getPrezzo());
   }

   @Override
   public float getPrezzo() {
      return prezzo;
   }

   @Override
   public int getPeso() {
      return peso;
   }
```

```
    @Override
    public void add(Prodotto p) {
    }

    @Override
    public void remove(Prodotto p) {
    }
}
```

```java
/**
 * EBook svolge il ruolo di Leaf per il design pattern Composite
 */

public class EBook implements Prodotto {
   private String titolo;
   private float prezzo;

   public EBook(String titol, float prez) {
      titolo = titol;
      prezzo = prez;
   }

   @Override
   public void print() {
      System.out.println("EBook: " + titolo + "\tPrezzo: " + getPrezzo());
   }

   @Override
   public float getPrezzo() {
      return (prezzo * (1 - percSconto() / 100));
   }

   @Override
   public int getPeso() {
      return 0;
   }

   private float percSconto() {
      return 15;
   }

   @Override
   public void add(Prodotto p) {
   }

   @Override
   public void remove(Prodotto p) {
   }
}



import java.util.ArrayList;
import java.util.List;

/**
 * Carrello svolge il ruolo di Composite per il design pattern Composite
 **/
```

```
public class Carrello implements Prodotto {
   private List< Prodotto > nestedElem = new ArrayList< >();

   @Override
   public void print() {
      System.out.println("Carrello ------- ------- ------- ------- ---");
      for (Prodotto res : nestedElem)
         res.print();
      System.out.println("---------------- ------- ------- ------- ---");
      System.out.println("---------------- ------- Prezzo totale: " + getPrezzo(
   }

   @Override
   public float getPrezzo() {
      return nestedElem.stream().map(e -> e.getPrezzo()).reduce(0f, Float::sum);
   }

   @Override
   public int getPeso() {
      return nestedElem.stream().map(e -> e.getPeso()).reduce(0, Integer::sum);
   }

   @Override
   public void add(Prodotto p) {
      System.out.println("Carrello: add()");
      nestedElem.add(p);
   }

   @Override
   public void remove(Prodotto p) {
      nestedElem.remove(p);
   }

}



/**
 * Creator svolge il ruolo di ConcreteCreator per il design pattern Factory Meth
 * fornisce vari metodi factory per istanziare Leaf o Composite
 **/

public class Creator {
   private static final String[] libri = { "Gamma Design Pattern", "Fowler Refac
   private static final float[] prezzi = { 38.5f, 40.0f, 28.5f, 22 };
   private static final int[] pesi = { 650, 700, 300, 200 };

   private static int i = -1;

   public static Prodotto getCarrello() {
      System.out.println("Creator: istanzio Carrello");
      return new Carrello();
```

```
    }

    public static Prodotto getLibro() {
        System.out.println("Creator: istanzio Libro");
        aggiornaIndice();
        return new Libro(libri[i], prezzi[i], pesi[i]);
    }

    public static Prodotto getEbook() {
        System.out.println("Creator: istanzio EBookk");
        aggiornaIndice();
        return new EBook(libri[i], prezzi[i]);
    }

    private static void aggiornaIndice() {
        if (i < libri.length - 1)
            i++;
        else
            i = 0;
    }
}



/**
 * L'applicazione che usa il Design Pattern Composite consiste dell'interfaccia
 * Prodotto (Component), e delle classi Libro e EBook (Leaf), della classe
 * Creator che istanzia oggetti, della classe Carrello (Composite) per
 * raggruppare istanze, e del client MainCart che chiama operazioni su Leaf e
 * Composite.
 */

public class MainCart {
    private static final Prodotto cart = Creator.getCarrello();
    private static final Prodotto lib = Creator.getLibro();
    private static final Prodotto eb = Creator.getEbook();

    public static void main(final String[] args) {
        System.out.println("\nChiama print su lib");
        lib.print();
        System.out.println("Peso di lib " + lib.getPeso());

        System.out.println("\nInserimento di Libri in Carrello");
        cart.add(lib);
        cart.add(eb);

        System.out.println("\n");
        cart.print();
        System.out.println("\nPeso del Carrello " + cart.getPeso());
    }
}
```

```
Output dell'esecuzione

Creator: istanzio Carrello
Creator: istanzio Libro
Creator: istanzio EBookk

Chiama print su lib
Libro: Gamma Design Pattern     Prezzo: 38.5
Peso di lib 650

Inserimento di Libri in Carrello
Carrello: add()
Carrello: add()


Carrello ------- ------- ------- ------- ---
Libro: Gamma Design Pattern     Prezzo: 38.5
EBook: Fowler Refactoring       Prezzo: 34.0
---------------- ------- ------- ------- ---
---------------- ------- Prezzo totale: 72.5

Peso del Carrello 650
```