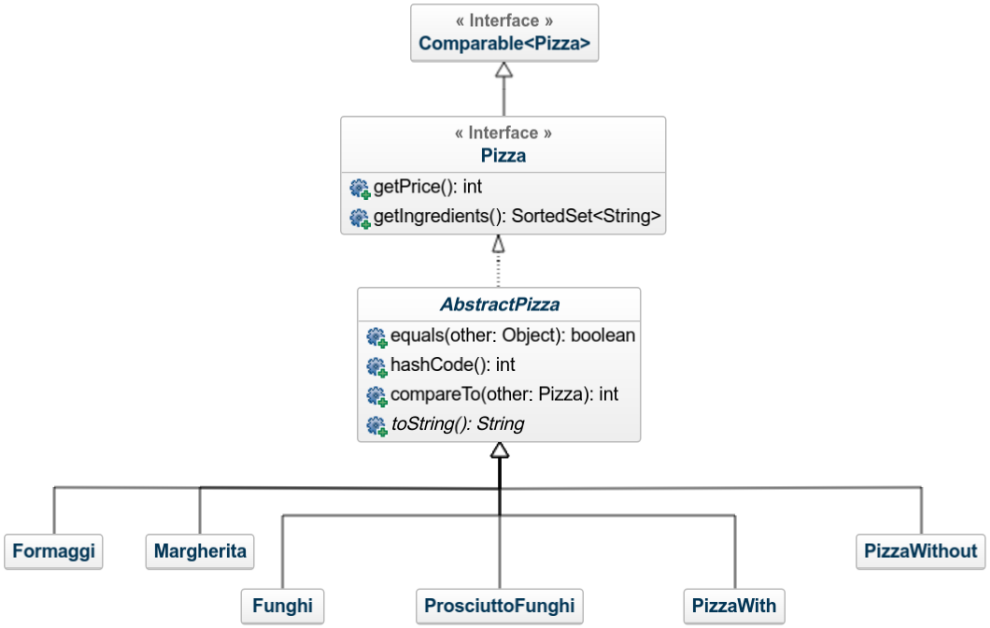
Esame di Programmazione II, 26 giugno 2017

Si consideri la seguente gerarchia di classi, che implementa le pizze di un'ipotetica pizzeria:



Si noti che `Pizza` è un'interfaccia che estende l'interfaccia `Comparable<Pizza>` della libreria Java standard. Inoltre la classe `AbstractPizza` è astratta: implementa in modo `final` i metodi `equals`, `hashCode` e `compareTo` e dichiara `toString` astratto. Due pizze sonno considerate uguali se e solo se hanno gli stessi ingredienti. Sono messe in ordine (`compareTo`) rispetto all'ordine alfabetico della concatenazione dei rispettivi ingredienti.

Esercizio 1 [**8 punti**] Si implementino l'interfaccia `Pizza` e la classe astratta `AbstractPizza`. I metodi `equals`, `hashCode` e `compareTo` devo essere consistenti fra di loro. Si eviti di implementare `hashCode` in modo costante, perché sarebbe inefficiente.

Esercizio 2 [**3 punti**] Si completi la seguente implementazione della pizza `Margherita`:

```
public final class Margherita extends AbstractPizza {
  final static Margherita INSTANCE = new Margherita(); // unico oggetto esistente
  private Margherita() {}

  .. stampandola si ottiene "Margherita", i suoi ingredienti sono "tomato" e "mozzarella", il costo 5 euro
}
```

Si faccia lo stesso anche per le classi `Formaggi` (costa 8 euro e ha come ingredienti "mozzarella", "parmesan" e "gorgonzola"), `Funghi` (7 euro, "tomato", "mozzarella" e "mushrooms") e `ProsciuttoFunghi` (8 euro, "tomato", "mozzarella", "mushrooms" e "ham"). A questo punto il menù della pizzeria è pronto:

```
public class Menu {
  public final static Pizza MARGHERITA = Margherita.INSTANCE;
  public final static Pizza FUNGHI = Funghi.INSTANCE;
  public final static Pizza FORMAGGI = Formaggi.INSTANCE;
  public final static Pizza PROSCIUTTO_FUNGHI = ProsciuttoFunghi.INSTANCE;
}
```

Esercizio 3 [**10 punti**] La classe `PizzaWith` rappresenta la modifica di un'altra pizza, con l'aggiunta di un singolo ingrediente. Tale ingrediente non deve già essere fra quelli della pizza a cui lo si sta aggiungendo, perché in tal caso non avrebbe senso aggiungerlo. Aggiungendo l'ingrediente, il prezzo della pizza aumenta e la sua stampa è espansa con l'ingrediente aggiunto. Si completi l'implementazione seguente:

```
public class PizzaWith extends AbstractPizza {
  ...
  // costruisce la pizza ottenuta da base aggiungendo l'ingrediente indicato, che costa
```

```
  // il prezzo extra indicato. Se l'ingrediente era gia' fra quelli della pizza base,
  // deve lanciare un'eccezione di classe IllegalPizzaModificationException
  public PizzaWith(Pizza base, String addedIngredient, int extraPrice) { ... }

  @Override public String toString() {
    ... ritorna il toString() di base concatenato con "with" e l'ingrediente aggiunto
  }

  @Override public SortedSet<String> getIngredients() { ... ha gli ingredienti di base piu' quello aggiunto }
  @Override public int getPrice() { ... ha il prezzo di base piu' il prezzo extra da aggiungere }
}
```

Si faccia la stessa cosa per la classe `PizzaWithout`, che rappresenta una pizza ottenuta eliminando un ingrediente da un'altra pizza. Questa volta si ottiene una `IllegalPizzaModificationException` se la pizza di partenza non conteneva l'ingrediente che si vuole togliere.

Esercizio 4 [**2 punti**] Si implementi l'eccezione `IllegalPizzaModificationException`.

Esercizio 5 [**5 punti**] Si implementi un ordine della pizzeria, che contiene la sequenza delle pizze che si ordina:

```
public class Order {
  ...
  public Order(Pizza... pizzas) { ...prende nota delle pizzas ordinate }

  @Override public String toString() {
    ... ritorna la stampa delle pizze ordinate con i prezzi di ciascuna
    e in fondo il prezzo totale dell'ordine (si veda esempio in basso)
  }

  public int getPrice() { ... ritorna il prezzo totale dell'ordine }
}
```

Se tutto è corretto, l'esecuzione del programma:

```
public class Main {
  public static void main(String[] args) {
    Pizza p1 = Menu.FUNGHI; Pizza p2 = Menu.MARGHERITA; Pizza p3 = Menu.PROSCIUTTO_FUNGHI;
    Pizza p4 = new PizzaWithout(p1, "mushrooms", 2); // p4 e' p1 con la rimozione dei funghi
    Pizza p5 = new PizzaWith(p4, "mushrooms", 2); // p5 e' p4 con l'aggiunta dei funghi

    Set<Pizza> allFiveSet = new HashSet<>();
    allFiveSet.add(p1); allFiveSet.add(p2); allFiveSet.add(p3); allFiveSet.add(p4); allFiveSet.add(p5);

    List<Pizza> allFiveList = new LinkedList<>();
    allFiveList.add(p1); allFiveList.add(p2); allFiveList.add(p3); allFiveList.add(p4); allFiveList.add(p5);

    Pizza[] allFiveArray = new Pizza[] { p1, p2, p3, p4, p5 };

    System.out.println("allFiveSet.size() = " + allFiveSet.size());
    System.out.println("allFiveList.size() = " + allFiveList.size());
    System.out.println("allFiveArray.length = " + allFiveArray.length);

    Order order = new Order(allFiveArray);
    System.out.println(order);
  }
}
```

dovrà stampare:

```
allFiveSet.size() = COSA STAMPA?
allFiveList.size() = COSA STAMPA?
allFiveArray.length = COSA STAMPA?
----------------------
Funghi (7 euros)
Margherita (5 euros)
Prosciutto & Funghi (8 euros)
Funghi without mushrooms (5 euros)
Funghi without mushrooms with mushrooms (7 euros)
----------------------
Total price: 32 euros
```

Esercizio 6 [**3 punti**] Quali numeri vengono stampati dove sopra è indicato `COSA STAMPA?`

È possibile definire campi, metodi, costruttori e classi aggiuntive, ma solo `private`.