This is a very simple coding example, no complicated algorithms. Our intent is to collect an example of your design and code so that we can discuss your approach to fulfilling the requirements. The purpose is to generate further discussion and explore your grasp of OO concepts and knowledge of the JDK. Present your response as you see fit.

**Program requirements:**

You are working on a team developing an e-commerce application. One of your tasks is to complete the implementation of the *Order* class that an intern has started along with any other class or classes on which it depends.

The Order constructor requires an array of *OrderItems*. The business rules dictate that there are two types of order items required, service and material. There is one distinction between them, only material items are taxable. An instance of an *OrderItem*  is only required to contain an Item and a quantity.

An *Order,* once constructed, is immutable (no one should be able to change it).

The *Order* object also needs to be *Serializable* as it will be distributed across multiple VM’s.

It is critical that the method that returns the order-total returns accurately to the nearest penny.

There is an expected future requirement that Items be used as keys in a *Hashtable* so address this issue now.

Make any changes needed to the Order object to meet the requirements stated above, although you should not have to add any more public methods. This API will be used by many developers so implement all common methods.

Here is the partially implemented code:

**package com.exam;**

**/\*\***

**\* Represents a part or service that can be sold.**

**\***

**\* care should be taken to ensure that this class is immutable since it**

**\* is sent to other systems for processing that should not be able to**

**\* change it in any way.**

**\***

**\* <p>Copyright: Copyright (c) 2004</p>**

**\* <p>Company: Exams are us</p>**

**\* @author Joe Blow**

**\* @version 1.0**

**\*/**

**public class Item**

**{**

**private Integer key;**

**private String name;**

**private float price;**

**public Item(Integer key, String name, float price)**

**{**

**this.key = key;**

**this.name = name;**

**this.price = price;**

**}**

**public Integer getKey()**

**{**

**return key;**

**}**

**public String getName()**

**{**

**return name;**

**}**

**public float getPrice()**

**{**

**return price;**

**}**

**}**

**package com.exam;**

**import java.util.\*;**

**/\*\***

**\* Represents and Order that contains a collection of items.**

**\***

**\* care should be taken to ensure that this class is immutable since it**

**\* is sent to other systems for processing that should not be able**

**\* to change it in any way.**

**\***

**\* <p>Copyright: Copyright (c) 2004</p>**

**\* <p>Company: Exams are us</p>**

**\* @author Joe Blow**

**\* @version 1.0**

**\*/**

**public class Order**

**{**

**private OrderItem[] orderItems;**

**public Order(OrderItem[]orderItems)**

**{**

**this. orderItems = orderItems;**

**}**

**// Returns the total order cost after the tax has been applied**

**public float getOrderTotal(float taxRate)**

**{**

**return 0; // implement this method**

**}**

**/\*\***

**\* Returns a Collection of all items sorted by item name**

**\* (case-insensitive).**

**\***

**\* @return Collection**

**\*/**

**public Collection getItems()**

**{**

**return null; // implement this method**

**}**

**}**

package com.exam;

/\*\*

\* Represents a part or service that can be sold.

\*

\* care should be taken to ensure that this class is immutable since it

\* is sent to other systems for processing that should not be able to

\* change it in any way.

\*

\* <p>Copyright: Copyright (c) 2004</p>

\* <p>Company: Exams are us</p>

\* @author Joe Blow

\* @version 1.0

\*/

public final class Item implements Comparable<Item>

{

private Integer key;

private String name;

private float price;

public Item(Integer key, String name, float price)

{

this.key = key;

this.name = name;

this.price = price;

}

public Integer getKey()

{

return key;

}

public String getName()

{

return name;

}

public float getPrice()

{

return price;

}

public int equals(Item other) {

if (this.key != other.getKey()) {

return (this.key - other.getKey());

}

if (!this.name.equals(other.getName())) {

return (this.name.compareTo(other.getName()));

}

if (this.price != other.getPrice()) {

return (this.price - other.getPrice());

}

return 0;

}

@Override

public int compareTo(Item other) {

if (this.key != other.getKey()) {

if (this.key - other.getKey() < 0) {

return -1;

}

return 1;

}

if (!this.name.equals(other.getName())) {

return (this.name.compareTo(other.getName()));

}

if (this.price != other.getPrice()) {

if (this.price - other.getPrice() < 0) {

return -1;

}

return 1;

}

return 0;

}

}

package com.exam;

import java.util.\*;

/\*\*

\* Represents and Order that contains a collection of items.

\*

\* care should be taken to ensure that this class is immutable since it

\* is sent to other systems for processing that should not be able

\* to change it in any way.

\*

\* <p>Copyright: Copyright (c) 2004</p>

\* <p>Company: Exams are us</p>

\* @author Joe Blow

\* @version 1.0

\*/

public final class Order implements serializable

{

private static final serialID = -1L;

private OrderItem[] orderItems;

public Order(OrderItem[]orderItems)

{

this.orderItems = orderItems;

}

// Returns the total order cost after the tax has been applied

public float getOrderTotal(float taxRate)

{

BigDecimal total = new BigDecimal("0.0", 3);

for (int i = 0; i < orderItems.length(); i++) {

total.add(String.valueOf(orderItems[i].getCost(taxRate)));

}

return total.setScale(2, BigDecimal.ROUND\_UP).floatValue();

}

/\*\*

\* Returns a Collection of all items sorted by item name

\* (case-insensitive).

\*

\* @return Collection

\*/

public Collection getItems()

{

return Arrays.sort(orderItems);

}

}

package com.exam;

import java.util.\*;

/\*\*

\* Represents an Order Item

\*

\* Abstraction forces one of the subclasses to be used, enabling polymorphism for

\* faster processing.

\*

\* @author David Emler

\* @version 1.0

\*/

public abstract class OrderItem {

private Item item;

private int qt;

public OrderItem(Item item, int qt) {

this.item = item;

this.qt = qt;

}

public Item getItem() {

return item;

}

public int getQt() {

return qt;

}

abstract float getCost(float taxRate);

}

package com.exam;

import java.util.\*;

import com.exam.OrderItem;

public class ServiceItem extends OrderItem {

public ServiceItem(Item item, int qt) {

super(item, qt);

}

public float getCost(float taxRate) {

return getItem().getPrice();

}

}

package com.exam;

import java.util.\*;

import com.exam.OrderItem;

public class MaterialItem extends OrderItem {

public MaterialItem(Item item, int qt) {

super(item, qt);

}

public float getCost(float taxRate) {

return getItem().getPrice() \* taxRate;

}

}