THIS IS JUST A PRACTICE QUIZ USED IN LAB. Do not submit to submit server. It is just for practice on your local machine.

Specifications

For this practice quiz you will implement methods for a class named **MailPackage** that represents a mail package. You will find the class in the **sysImplementation** package. A description of each method is provided below. A driver (that you can ignore if you know what to implement) and associated output is provided at the end. Regarding the code you need to implement:

- Do not use **package** as a variable name (it is a reserved word in Java).
- You don't need to add comments to your code, but you must have good variable names, indentation and you should avoid code duplication.
- You must provide an implementation for every method, otherwise your code will not work in the submit server. If you don't know what to do for a method, you can add the following statement:

throw new UnsupportedOperationException("Not Implemented");

1. The class defines the following private instance variables:

    a. **recipient** - a string representing the person receiving the package.
    b. **numberOfItems** - an integer representing the number of items in the package.
    c. **description** - a string describing the package's contents.

2. You can define additional instance variables.
3. You can define constants (static final), but do not define static variables that can be modified.
4. **Constructor #1** - Takes a recipient, a number of items, and a description as parameters. It initializes the corresponding instance variables. If recipient is null, an IllegalArgumentException will be thrown (any message is fine) and further initialization will not take place.
5. **Constructor #2** - Takes a recipient and a description as parameters. It assumes the number of items is one. If recipient is null, an IllegalArgumentException will be thrown (any message is fine) and further initialization will not take place.
6. **Copy Constructor -** Define a copy constructor for the class.
7. **addItem -** Takes a string as a parameter representing the description of an item. The method will append the parameter string to the **description** instance variable. The number of items will be adjusted accordingly. If the parameter is null, no item will be added. The method returns a reference to the current object.
8. **getRecipient** - Get method for recipient.
9. **getNumberOfItems -** Get method for number of items.
10. **getDescription -** Get method for description.
11. **getPackageCost –** Returns an integer value representing the package cost. The cost of the package is based on the number of items. Each item costs $2. For example, if a package has 3 items, the cost will be $6.
12. **equals -** Define an equals method for the class. Two packages are considered the same if they have the same recipient and the same number of items. The package's description is ignored.
13. **merge –** This method has a MailPackage as a parameter, and it will merge two packages if they have the same recipient. To merge two packages, the number of items in the current object will be increased by the number of items in the parameter object. In addition, the description associated with the parameter object will be appended to the description of the current object. If the method tries to merge to itself, no processing will take place. If the method tries to merge to a package that has a different recipient, an IllegalArgumentException (with any message) will be thrown. The method returns a reference to the current object.
14. **compareTo –** This method will allow us to compare two MailPackage objects. It has a MailPackage as parameter and returns an integer. You can use the String class compareTo method during the implementation of this method. The MailPackage compareTo method will return:
    a. A negative value if the number of items of the current object is less than the parameter.
    b. A positive value if the number of items of the current object is greater than the parameter.
    c. If the current object and the parameter have the same number of items, the method will return:
        i. A negative value if the recipient of the current object precedes the recipient of the parameter in alphabetical order.
        ii. A positive value if the recipient of the current object follows the recipient of the parameter in alphabetical order.
        iii. 0 otherwise.

Driver / Expected Output (Feel free to ignore)

Driver

```
public static void main(String[] args) {
   String answer = "";

   MailPackage jacks = new MailPackage("Jack", 2, "Book, Pencil and Misc");
   MailPackage kellys = new MailPackage("Kelly", "Computer");
   MailPackage kellysCopy = new MailPackage(kellys);
   MailPackage toms = new MailPackage("Tom", 2, "Ipad and PC");
   MailPackage lauras = new MailPackage("Laura", 4, "Pad, Pen and Misc");
   MailPackage alices = new MailPackage("Alice", 2, "Mac, Donut");
   MailPackage alices2 = new MailPackage("Alice", 3, "Phone, Cup, Pen");

   answer += getInfo(jacks);
   answer += getInfo(kellys);
   answer += getInfo(kellysCopy);
   answer += "Jack's package cost: " + jacks.getPackageCost();
   answer += "\nJack's and Tom's equal?: " + jacks.equals(toms);
   answer += "\nComparison #1: " + (toms.compareTo(lauras) < 0);
   answer += "\nComparison #2: " + (toms.compareTo(alices) > 0);
   alices.merge(alices2);
   answer += "\n\nMerged:\n" + getInfo(alices);

   System.out.println(answer);
}

private static String getInfo(MailPackage mailPackage) {
   String answer = "";

   answer += "Recipient: " + mailPackage.getRecipient();
   answer += "\nNumberOfItems: " + mailPackage.getNumberOfItems();
   answer += "\nDescription: " + mailPackage.getDescription() + "\n\n";

   return answer;
}
```

Output

```
Recipient: Jack
NumberOfItems: 2
Description: Book, Pencil and Misc

Recipient: Kelly
NumberOfItems: 1
Description: Computer

Recipient: Kelly
NumberOfItems: 1
Description: Computer

Jack's package cost: 4
Jack's and Tom's equal?: false
Comparison #1: true
Comparison #2: true

Merged:
Recipient: Alice
NumberOfItems: 5
Description: Mac, DonutPhone, Cup, Pen
```