Polymorphism

Exercise 1

All tasks related to recipes have been completed. You are going to work on a more interesting set of tasks: calculating how much customers pay for a booking.

A booking involves a list of items. Each item has a name (String) and a price (double).

The total amount of a booking is the sum of all items. But it is not the final a customer has to pay.

Each booking has a servicing style. The servicing style can be Western or Other (note that more servicing styles can be added later). If the servicing style is Western, the customers have to pay a tip of 15% of the sum of all items. If the servicing style is Other, you can generate a random number between 0% to 10% of the total sum.

Implement the following classes: Item, Booking, WesternBooking, and OtherBooking. WesternBooking and OtherBooking are child classes of Booking. You should implement the basic calculation in Booking and provide the additional calculations in child classes.

Implement a simple program to demonstrate those classes. You can create objects directly in your code (that means you don't need to ask users to enter data).

To ensure the same signatures of all classes, implement at least the following methods in Booking class

-addItem() which accept an Item object

-bookingSum() which returns the sum of a booking (without any tip yet)

(You must override bookingSum() in WesternBooking and OtherBooking classes)

Exercise 2

Each booking can have an optional discount (there is at most one discount per booking). Discounts can be one of the following types: voucher, percent off, and special event. Note that all discounts are calculated based on the booking sum BEFORE tips. After a discount is applied, the tip is calculated based on the booking sum WITHOUT discount. (e.g., if booking sum is $10, discount is 50%, then booking sum is $5, but the tip should be calculated with $10 as the sum).

Voucher: consists of a code (String) and an amount (double). When a voucher is used, this amount is subtracted from booking sum before tip. If (amount > booking sum), the booking sum is considered zero (but you still have to pay tips).

Percent off: consists of a code (String) and a percent (integer from 1 - 100). When a percent off is used, the booking sum before tip is reduced by that percent value. For example, if booking sum is 100, a 10 percent off discount reduces the sum to 90).

Special event: consists of a code (String) and a value (integer from 1 - 100). That value can be a "percent off" or an amount reduced (similar to voucher) depends on what benefits the customer most. In any case, the minimum booking sum after discount is not less than zero. (i.e. if the booking sum is $50, and the value is 60, then 60 should be the amount reduced, not 60% off, because it benefits the customer most; however, the booking sum after discount in this case is zero, not -10).

Implement the following classes: Discount, VoucherDiscount, PercentOffDiscount, and SpecialEventDiscount. The 3 later classes are child classes of the first class.

You need to implement/override a method calculateDiscountedAmount() on each of the discount class to return the discounted amount, provided a booking sum. The signature of the calculateDiscountedAmount method is

double calculateDiscountedAmount(double sum)

You can implement a method addDiscount() on Booking class that will accept and add a discount object to a booking object. This discount object will be available to all child classes of Booking (if you don't make it private)

Use this discount object in child classes of Booking to calculate the discounted amount and update the bookingSum() accordingly with discount.

Sample code that uses the discount objects

```
Booking b = new WesternBooking();

b.addDiscount(new VoucherDiscount("GREAT", 50.0));

b.bookingSum();

------

Booking b2 = new OtherBooking();

b2.addDiscount(new PercentOffDiscount("OMG", 99));

b2.bookingSum();

------
```

Implement a simple program to demonstrate those classes. You can create objects directly in your code (that means you don't need to ask users to enter data).

Exercise 3

There is no booking in general. A booking must be of a specific type: Western or Other. How can you modify your existing classes to enforce this rule?

Hint: use abstract classes

Exercise 4

The government want to collect taxes from your business. In particular, all bookings, except SocialBooking (explained later), are taxed. The tax percent is 10% for WesternBooking and 15% for OtherBooking. The tax amount is calculated based on the total booking amount before applying discounts and tips.

SocialBooking is a special kind of booking that is used in charity events. There is no tax (but you can have discounts and tips) for special bookings. (You can freely decide what you want to calculate tips for social bookings)

Create a Taxable interface (you cannot use abstract class because multiple inheritance is not possible in Java, and your classes have a common parent, which is Booking, already). There is only one method in this interface: calculateTax(). You should implement this method appropriately in your WesternBooking and OtherBooking classes.

Now, provided a list of bookings, can you write a loop to get the total tax?

Hint: use instance of