Discussion 6: Methods and Exceptions in Object-Oriented Programming

Discussion Topic:
In Java, you have the opportunity to either use a predefined function/method or write a user-defined function/method. In this discussion, provide at least three criteria that would be used to develop an appropriate method of your choice and the rationale behind the selection of these criteria. Then provide an example of your method declaration and return type. Actively participate in this discussion by providing constructive feedback on the criteria, rationales, and examples posted by your peers.

My Post:

Hello class,

An example of a user-defined function/method that can be used to elaborate on the criteria utilized to develop the method and its rationale could be a method that calculates the discount for a customer based on loyalty points, seasonal promotions, and purchase history. Here are four criteria and rationales that I think should be used to develop the method:

1. Define the task – Method functionality:
   If the task involves unique business logic or specific requirements that cannot be addressed by Java predefined methods, a user-defined method is necessary. In the case of the method described above, its task is to calculate the discount for a customer based on loyalty points, seasonal promotions, and purchase history.
   Rationale: Unique business logic and specific requirements often are required to develop appropriate solutions to a need or a problem that predefined methods cannot offer. User-defined function/method ensures that the solution meets the business needs.
2. Task reusability – The method will be reused throughout the codebase:
   The task functionality will be reused in multiple parts of the application. The method can be called across different modules without duplicating code.
   Rationale: Reusability promotes modularity and reduces code duplication, ensuring consistent implementation of the discount calculation logic throughout the application.
3. Future modifications – Method maintainability:
   Over time, the task's functionality may need to change; for example, the number of loyalty points required to affect the discount calculation may vary in the future. The methods improve maintainability by encapsulating specific logic in one block of code. This makes it easier to update/change and debug the code.
   Rationale: Encapsulation of code logic and functionality within a method makes maintainability possible and easier. This makes future updates or modifications simpler and reduces the risk of introducing errors.
4. Task describing – Naming, parameters, Javadocs, and comments:
   Documentation is needed to describe the task and define the task parameters. Naming a method appropriately, choosing/declaring clear parameters, and using Javadocs and comments are crucial for code readability/understandability and defining parameters.
   Rationale**:** Well-documented code with clear naming conventions and parameter declaration and

improves code functionality, readability, and helps other developers understand the purpose and functionality of the method.

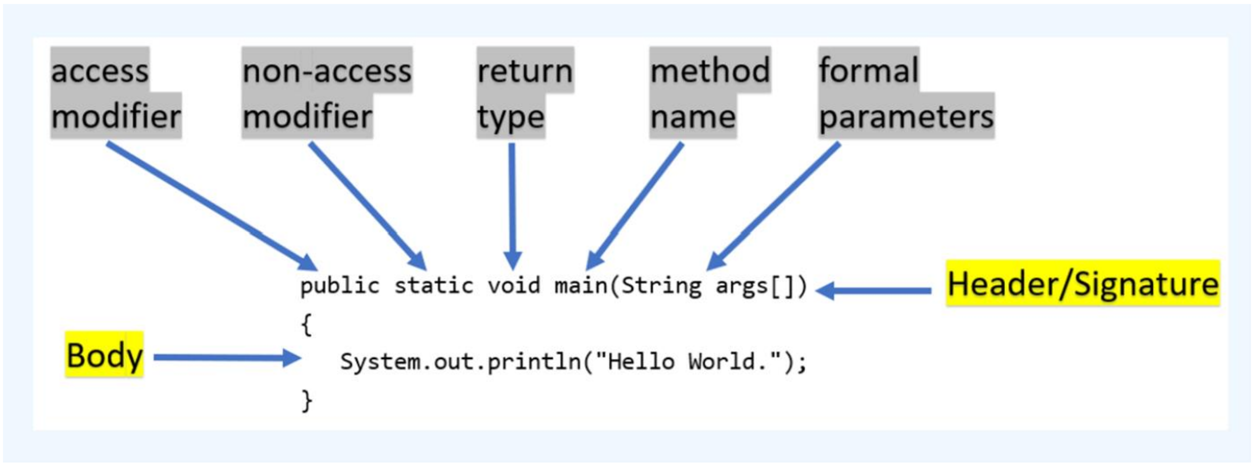
Code example:

```
public class DiscountCalculator {

    /**
     * Calculates the discount for a customer based on loyalty points,
     * seasonal promotions, and purchase history.
     *
     * @param loyaltyPoints (int) The number of loyalty points.
     * @param seasonalPromo (Double) The seasonal promotion discount percentage.
     * @param purchaseHistory (double) The customer's total purchase history amount.
     * @return (double) The calculated discount amount.
     */
    public double calculateDiscount(int loyaltyPoints, double seasonalPromo, double
purchaseHistory) {
        double baseDiscount = 0.0;

        // Add loyalty points discount
        baseDiscount += loyaltyPoints * 0.01;

        // Apply seasonal promotion
        baseDiscount += seasonalPromo;

        // Additional discount based on purchase history
        if (purchaseHistory > 1000) {
            baseDiscount += 5.0; // Additional 5% discount for high spenders
        }

        return baseDiscount;
    }
}
```

Finally, I wanted to share the following picture describing Java method vocabulary:

Figure-1

Java Method



```
public static void main(String args[])
{
   System.out.println("Hello World.");
}
```

Note: Java method vocabulary. From *Introduction to Programming with Java:* 5.1.1 Writing Static Methods [Image], by Ericson et al., 2015.Runestone Academy. https://runestone.academy/ns/books/published/csjava/Unit5-Writing-Methods/topic-5-1-writing-methods.html

-Alex

References:

Ericson, B. (2015). Introduction to programming with Java: 5.1.1 Writing static methods [Image]. Runestone Academy. https://runestone.academy/ns/books/published/csjava/Unit5-Writing-Methods/topic-5-1-writing-methods.html