ASSIGNMENT 1

Individual

INSTRUCTION:

1. Students are required to submit a report **(LIGHT BLUE**).
2. This assignment contributes 10% of the overall assessment.
3. Upload a report and a zipped project directory with all codes and resources required to compile and run the code in eklas.
4. Print your report (codes & sample of output) together with cover page and all the rubrics.
   Last date of submission: 10 October 2024.

COURSE LEARNING OUTCOME:

CO2 Apply the concepts of Object-oriented Programming in solving computing problems using Java. (C3, PLO2)

QUESTION

Task:
You are required to **design a class** for a chosen object, ensuring that it adheres to the **principles of object-oriented programming (OOP)**, especially **encapsulation**. This class should include **six attributes** and **four methods**, consisting of:

- **One constructor**
- **Getter and Setter methods** for at least some attributes
- Additional methods to manipulate or interact with the object's attributes

Class Requirements:

1. **Attributes:**
   ◊ A minimum of **six attributes**, each representing a unique property of the object (for example, color, size, weight, etc.).
   ◊ Attributes must be encapsulated using **private** access modifiers, and you should provide **getter and setter methods** to access and modify the attributes safely.
2. **Methods:**
   ◊ At least **four methods**, including:
   - **A constructor** to initialize the object's attributes.
   - **Getter and Setter methods** to access and modify the encapsulated attributes.
   - At least one **other method** that provides functionality based on the object's attributes (e.g., a method to calculate something based on the object's properties).

Main Class:

- Create a **main class** to interact with your chosen object's attributes and methods.
- Your main class should demonstrate the use of:
  ◊ **Control structures**, such as:
  - **Selection (if statements)** to make decisions based on object attributes
  - **Looping (for or while loops)** to iterate or repeatedly interact with the object

    ◊ **Manipulation of attributes and methods** using getter and setter methods.

Additional Notes:

◊ Your code should be **well-documented** and **commented**, explaining each attribute and method's purpose and functionality.

◊ The evaluation of your assignment will be based on:
  - Correctness and **functionality** of your code
  - Creativity in the **choice of object**
  - The effective use of **control structures** to manipulate the object's data
  - Clarity and **quality of your code documentation**

Submission Notes:

Ensure you submit the following:

1. **Class files** for each group member's object.
2. **The main class** file that demonstrates interaction with the objects.
3. **Code comments** that clearly explain the purpose of each class, method, and control structure used.

OBJECT ORIENTED PROGRAMMING
CCS20704

Received Date	**:**
Submission Due Date:	**10 October 2024**
Lecturer	**: Dr. Jamal Abdullahi**
Weightage	**: 10%**
Semester	**: SEPTEMBER 2024**

Instruction to students:

- This is a GROUP assignment.
- Complete this cover sheet and attach it to your assignment (first page).

Student declaration:

I declare that:

- ***This assignment is my/our own work.***
- ***I/we understand what is meant by plagiarism.***
- ***My lecturer has the right to deduct my marks in case of:***
  - ***Late submission***
  - ***Any plagiarism found in my assignment.***

NAME: yousuf majid alnaamani
MATRICS NO:012023091611
PROGRAMME: bcs

Coding:

```
import java.util.Scanner;
public class Restaurant {
 // Define instance variables to store restaurant information
 private String name;
 private String location;
 private int capacity;
 private double rating;
 private String workingHours;
 private String cuisineType;
 // Constructor to initialize the restaurant object with basic information
 public Restaurant(String name, String location, int capacity, String workingHours, String
cuisineType)
 {
 this.name = name;
 this.location = location;
 this.capacity = capacity;
 this.rating = 0.0; // Initialize the rating to 0
 this.workingHours = workingHours;


this.cuisineType = cuisineType;
 }
 // Constructor to initialize the restaurant object with detailed information, reusing the basic
constructor
 public Restaurant(String name, String location, int capacity, double rating, String workingHours,
String cuisineType) {
 this(name, location, capacity, workingHours, cuisineType); // Reuse the first constructor
 this.rating = rating;
 }
 // Getter methods to access restaurant information
 public String getName() {
 return name;
 }
 public String getLocation() {


return location;
 }
 public int getCapacity() {
 return capacity;
 }
 public double getRating() {
 return rating;
 }
```

```
public String getWorkingHours() {
return workingHours;
}
public String getCuisineType() {
return cuisineType;
}
// Setter methods to modify restaurant information
public void setName(String name) {



this.name = name;
}
public void setLocation(String location) {
this.location = location;
}
public void setCapacity(int capacity) {
this.capacity = capacity;
}
// Set the restaurant's rating if it's within the valid range (0 to 5)
public void setRating(double rating) {
if (rating >= 0 && rating <= 5) {
this.rating = rating;
} else {
System.out.println("Rating should be between 0 and 5.");

}
}
public void setWorkingHours(String workingHours) {
this.workingHours = workingHours;
}
public void setCuisineType(String cuisineType) {
this.cuisineType = cuisineType;
}
// Method to display restaurant information
public void displayInfo() {
System.out.println("Restaurant Name: " + name);
System.out.println("Location: " + location);
System.out.println("Capacity: " + capacity);
System.out.println("Rating: " + rating);
System.out.println("Working Hours: " + workingHours);
System.out.println("Cuisine Type: " + cuisineType);
}


// Method to check if the restaurant is popular (rating >= 4.0)
public boolean isPopular() {
return rating >= 4.0;
}
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
```

```
 // Create a new restaurant object with initial information
 Restaurant myRestaurant = new Restaurant("Name", "Street", 50, "12 AM - 12 PM",
"American");
 myRestaurant.displayInfo();
 double newRating = -1;
 // Get a valid rating from the user (between 0 and 5)
 while (newRating < 0 || newRating > 5) {
 System.out.print("Enter a new rating (between 0 and 5): ");
 newRating = scanner.nextDouble();
 if (newRating < 0 || newRating > 5) {
 System.out.println("Invalid rating. Please try again.");


}
 }
 // Set the new rating for the restaurant
 myRestaurant.setRating(newRating);
 // Check if the restaurant is popular based on the rating
 if (myRestaurant.isPopular()) {
 System.out.println(myRestaurant.getName() + " is a popular restaurant!");
 } else {
 System.out.println(myRestaurant.getName() + " is not very popular.");



}
 // Update restaurant name if the user chooses to do so
 System.out.print("Do you want to update the restaurant name? (yes/no):
");
 String updateNameChoice = scanner.next();
 if (updateNameChoice.equalsIgnoreCase("yes")) {
 System.out.print("Enter the new restaurant name: ");
 String newName = scanner.next();
 myRestaurant.setName(newName);
 }
 // Update restaurant location if the user chooses to do so
 System.out.print("Do you want to update the restaurant location? (yes/no): ");
 String updateLocationChoice = scanner.next();
 if (updateLocationChoice.equalsIgnoreCase("yes")) {

System.out.print("Enter the new restaurant location: ");
 String newLocation = scanner.next();
 myRestaurant.setLocation(newLocation);
 }
 // Update restaurant capacity if the user chooses to do so
 System.out.print("Do you want to update the restaurant capacity? (yes/no): ");
 String updateCapacityChoice = scanner.next();
 if (updateCapacityChoice.equalsIgnoreCase("yes")) {
 System.out.print("Enter the new restaurant capacity: ");
 int newCapacity = scanner.nextInt();
 myRestaurant.setCapacity(newCapacity);
String newWorkingHours = scanner.next();
 myRestaurant.setWorkingHours(newWorkingHours);
 }
 // Display updated restaurant information
 System.out.println("\nUpdated Restaurant Information:");
 myRestaurant.displayInfo();
 // Check if the restaurant serves a specific cuisine
 System.out.print("Enter a cuisine type to check if the restaurant serves it: ");
 String cuisineToCheck = scanner.next();
 if (myRestaurant.getCuisineType().equalsIgnoreCase(cuisineToCheck)) {
 System.out.println(myRestaurant.getName() + " serves " + cuisineToCheck + " cuisine.");
 } else {
 System.out.println(myRestaurant.getName() + " does not serve " + cuisineToCheck + "
cuisine.");
 }
 // Close the scanner
 scanner.close();
 }
 }
 // Update restaurant working hours if the user chooses to do so
 System.out.print("Do you want to update the restaurant working hours? (yes/no): ");
 String updateWorkingHoursChoice = scanner.next();
 if (updateWorkingHoursChoice.equalsIgnoreCase("yes")) {
 System.out.print("Enter the new working hours: ");
```

```
String newWorkingHours = scanner.next();
 myRestaurant.setWorkingHours(newWorkingHours);
 }
 // Display updated restaurant information
 System.out.println("\nUpdated Restaurant Information:");
 myRestaurant.displayInfo();
 // Check if the restaurant serves a specific cuisine
 System.out.print("Enter a cuisine type to check if the restaurant serves it: ");
 String cuisineToCheck = scanner.next();
 if (myRestaurant.getCuisineType().equalsIgnoreCase(cuisineToCheck)) {
 System.out.println(myRestaurant.getName() + " serves " + cuisineToCheck + " cuisine.");
 } else {
 System.out.println(myRestaurant.getName() + " does not serve " + cuisineToCheck + "
cuisine.");
 }
 // Close the scanner
 scanner.close();
 }
```

Explanation :

Setter methods such as `setLocation()` and `setRating()` allow for the modification of these attributes. The `setRating()` method includes validation to ensure the rating falls within a valid range (0 to 5).

- Displaying Restaurant Information:** The `displayInfo()` method presents the restaurant's information in a structured manner, printing out details such as the restaurant's name, location, capacity, rating, working hours, and cuisine type, making it easy for users to view the restaurant's profile.

- Assessing Popularity:** The `isPopular()` method evaluates whether the restaurant is considered popular based on its rating. If the rating is greater than or equal to 4.0, the method returns true, indicating popularity; otherwise, it returns false.

- Main Method:** The main method serves as the program's entry point. It starts by initializing a `Scanner` object to collect user input, then creates an instance of the `Restaurant` class named `myRestaurant` with some default values and displays the initial restaurant information using the `displayInfo()` method.

The program engages the user in several interactions: it first prompts the user to input a new rating within the valid range [0, 5] and updates the restaurant's rating accordingly. It checks if the restaurant is popular and provides a message based on the rating.

Next, the program asks the user if they want to update various attributes of the restaurant, including its name, location, capacity, and working hours. If the user chooses to update these attributes, the program collects new values through user input and uses the corresponding setter methods to apply the changes.

After these updates, the program displays the updated restaurant information, ensuring users can see the effects of their modifications. Finally, it prompts the user to enter a cuisine type to check if the restaurant serves it, displaying an appropriate message.

To wrap up, this code represents a basic yet functional restaurant management system, showcasing object-oriented programming principles, user interaction through a console interface, and the ability to create, modify, and evaluate restaurant objects.

RUBRIC FOR ASSIGNMNET 1

Criteria	Excellent (4)	Good (3)	Satisfactory (2)	Needs Improvement (1)	Marks
Object Design (20%)	Object has at least 6 attributes, well-named, and appropriately typed. Object has at least 4 well-defined and well-documented methods. Proper use of constructor, setters, and getters with clear functionality and documentation.	Object has at least 5 attributes, mostly well-named and appropriately typed. Object has at least 4 well-defined and documented methods. Adequate use of constructor, setters, and getters.	Object has less than 5 attributes, some not well-named or typed. Object has less than 4 methods or lacking documentation. Limited or unclear use of constructor, setters, and getters.	Object design is incomplete or missing significant attributes or methods. Poor or missing use of constructor, setters, and getters.	20
Encapsulation (20%)	Excellent use of encapsulation with clear visibility modifiers and access control. All data is properly encapsulated with no direct access to attributes. Proper use of accessors and mutators.	Good use of encapsulation with mostly clear visibility modifiers and access control. Most data is encapsulated, with minimal direct access to attributes. Adequate use of accessors and mutators.	Some issues with encapsulation and access control, but it generally prevents unauthorized access. Limited use of accessors and mutators.	Poor or no use of encapsulation, allowing direct access to attributes. Lack of accessors and mutators.	15
Main Class Implementation (20%)	Effective and creative use of control structures to manipulate the object's attributes and methods Code is well-organized, easy to read, and free from errors. Exceptional demonstration of programming skills.	Good use of control structures to manipulate the object, with some creativity. Code is mostly organized, readable, and has minor errors. Demonstrates solid programming skills.	Adequate use of control structures, but the manipulation lacks creativity or completeness. Code organization and readability need improvement. Basic demonstration of programming skills.	Incomplete or minimal use of control structures, leading to limited object manipulation. Poor code organization, readability, and numerous errors. Lack of programming skills demonstrated.	15

Creativity and Object Choice (20%)	Highly creative and unique choice of object, showing deep understanding of OOP principles. Innovative use of attributes and methods that go beyond the basic requirements. Demonstrates exceptional creativity.	Creative choice of object, demonstrating an understanding of OOP principles. Some innovative use of attributes and methods. Shows creativity.	Object choice is somewhat conventional, with limited creativity. Attributes and methods are basic and meet minimum requirements. Limited creativity.	Lack of creativity in object choice or generic selection. Attributes and methods are basic and lack creativity. No demonstration of creative thinking.	10
Documentation and Comments (10%)	Code is extensively documented with clear explanations for each attribute, method, and control structure. Comments are used effectively to provide insights into the code's functionality. Excellent documentation.	Code is adequately documented with explanations for most attributes, methods, and control structures. Some comments are used to clarify code functionality. Good documentation.	Limited documentation with explanations for only a few attributes, methods, or control structures. Minimal use of comments. Basic documentation.	Code is poorly or not documented, with little or no explanation of attributes, methods, or control structures. Lack of comments. Inadequate documentation.	5
REMARKS use the getters in the display methods and try to use setters and getters in the main method as many times as possible also incorporate more comments to explain the lines of code. Improve code creativity as well.					10
TOTAL MARKS					75

END OF QUESTION