OBJECT ORIENTED PROGRAMMING
CCS20704/DCS20504/TCS21304

Received Date :
Submission Due Date: 7 November 2024
Lecturer : Dr. Jamal Abdullahi
Weightage : 20%
Semester : SEPTEMBER 2024

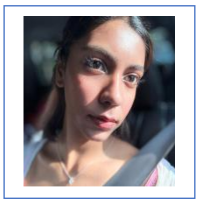
Instruction to students:

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

Student declaration:

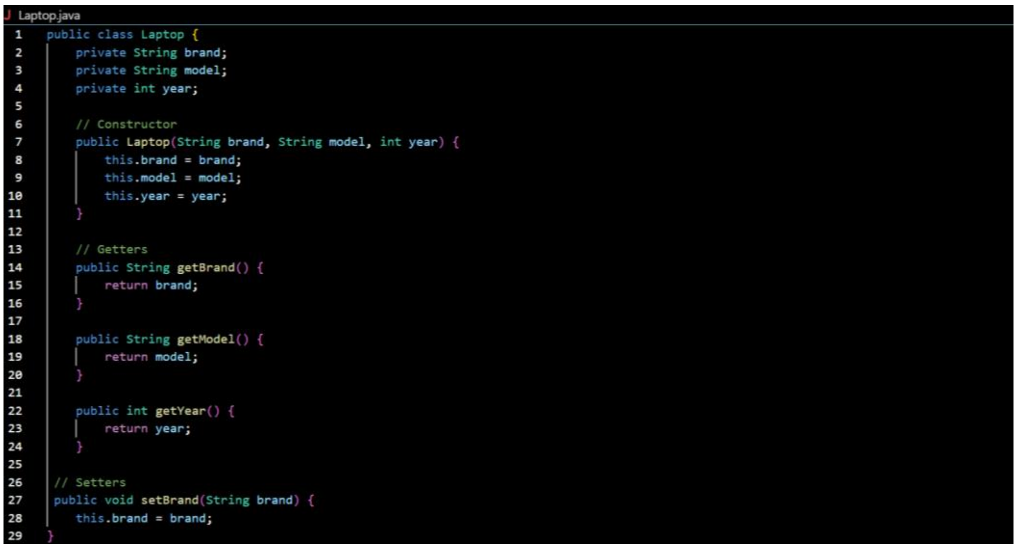
I declare that:

- ***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: SDIVYA TRISNI A/P SHANMUGALINGAM MATRICS NO: 012024021207 PROGRAMME: BCS 	**NAME: FAZIRAH BINTI MOHAMED SULAIMAN MATRICS NO: 012024021237 PROGRAMME: BCS**	**NAME:Mohamad Aqiem Danial bin Azmalli MATRICS NO: 012024020799 PROGRAMME:BCS**

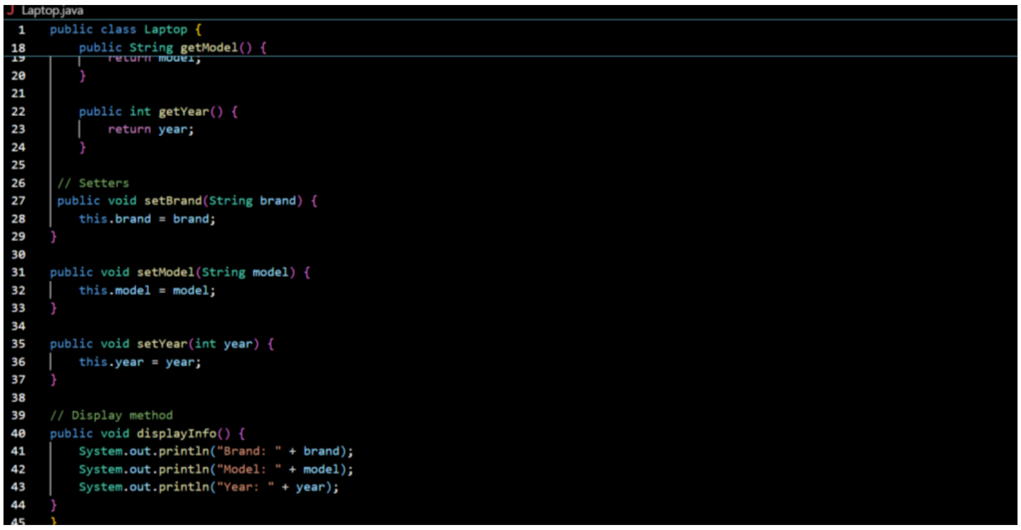
Definition of Base Class

The base class, `Laptop`, serves as the foundation of the system. It includes common attributes and methods shared by all derived classes. Key attributes include `brand`, `model`, and `year`. The `Laptop` class is designed with encapsulation to ensure data privacy, featuring private attributes and public getter and setter methods. The constructor initializes these attributes, ensuring each `Laptop` instance starts with defined values.



Laptop.java

```java
public class Laptop {
    private String brand;
    private String model;
    private int year;

    // Constructor
    public Laptop(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    // Getters
    public String getBrand() {
        return brand;
    }

    public String getModel() {
        return model;
    }

    public int getYear() {
        return year;
    }

 // Setters
 public void setBrand(String brand) {
    this.brand = brand;
}
```



Laptop.java

```java
public class Laptop {
    public String getModel() {
        return model;
    }

    public int getYear() {
        return year;
    }

 // Setters
 public void setBrand(String brand) {
    this.brand = brand;
}

public void setModel(String model) {
    this.model = model;
}

public void setYear(int year) {
    this.year = year;
}

// Display method
public void displayInfo() {
    System.out.println("Brand: " + brand);
    System.out.println("Model: " + model);
    System.out.println("Year: " + year);
}
}
```

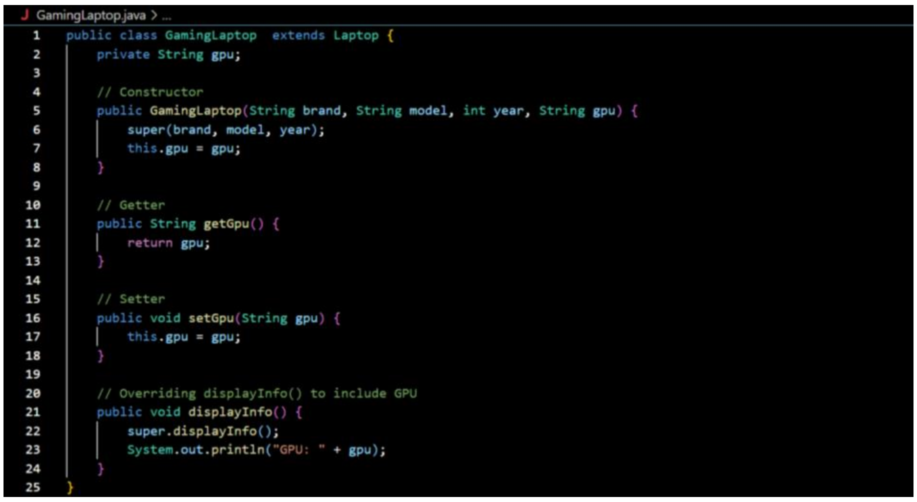
Derived Classes Implementing Inheritance

Two derived classes, GamingLaptop and BusinessLaptop, inherit from Laptop, showcasing the concept of inheritance:

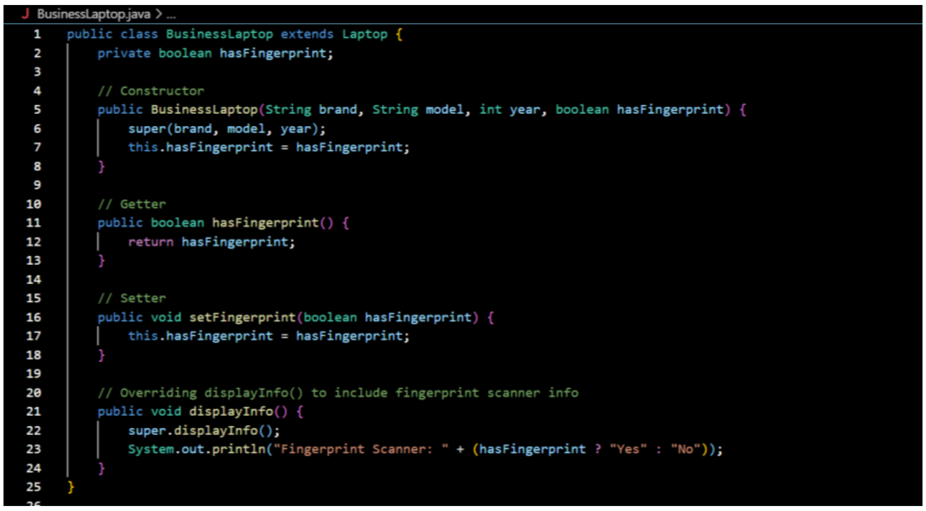
- GamingLaptop adds the unique attribute gpu, representing the specialized graphics processing unit for gaming purposes.

- BusinessLaptop includes the attribute hasFingerprint, indicating whether a fingerprint scanner is present for security.

Both derived classes override the `displayInfo()` method from the base class to include their distinctive attributes, demonstrating inheritance and method extension.



```java
public class GamingLaptop  extends Laptop {
    private String gpu;

    // Constructor
    public GamingLaptop(String brand, String model, int year, String gpu) {
        super(brand, model, year);
        this.gpu = gpu;
    }

    // Getter
    public String getGpu() {
        return gpu;
    }

    // Setter
    public void setGpu(String gpu) {
        this.gpu = gpu;
    }

    // Overriding displayInfo() to include GPU
    public void displayInfo() {
        super.displayInfo();
        System.out.println("GPU: " + gpu);
    }
}
```



```java
public class BusinessLaptop extends Laptop {
    private boolean hasFingerprint;

    // Constructor
    public BusinessLaptop(String brand, String model, int year, boolean hasFingerprint) {
        super(brand, model, year);
        this.hasFingerprint = hasFingerprint;
    }

    // Getter
    public boolean hasFingerprint() {
        return hasFingerprint;
    }

    // Setter
    public void setFingerprint(boolean hasFingerprint) {
        this.hasFingerprint = hasFingerprint;
    }

    // Overriding displayInfo() to include fingerprint scanner info
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Fingerprint Scanner: " + (hasFingerprint ? "Yes" : "No"));
    }
}
```

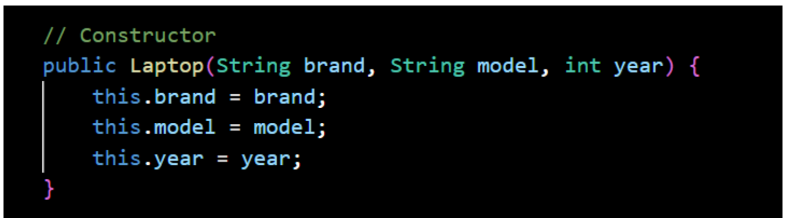
Demonstration of Polymorphism

Polymorphism is implemented through the overridden `displayInfo()` method in both derived classes. Despite using the same method signature, GamingLaptop and BusinessLaptop display their specific attributes differently:
- GamingLaptop extends `displayInfo()` to show GPU information.
- BusinessLaptop includes fingerprint scanner details in its `displayInfo()` method.

This ensures that each derived class presents its unique characteristics while adhering to a common interface.

Constructor in Base Class

The Laptop class constructor is implemented as follows:

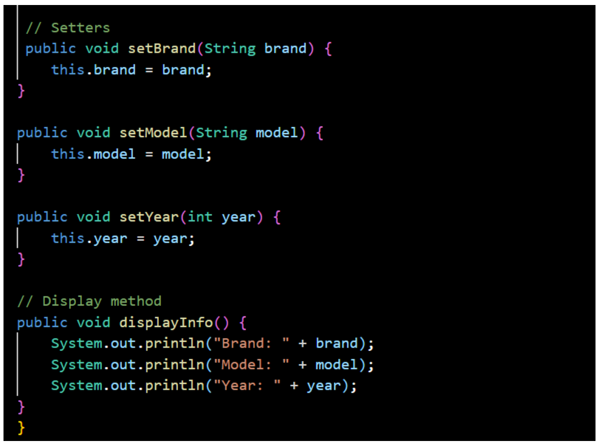


```java
// Constructor
public Laptop(String brand, String model, int year) {
    this.brand = brand;
    this.model = model;
    this.year = year;
}
```

This constructor initializes the `brand`, `model`, and `year` attributes, facilitating the creation of `Laptop` objects with pre-defined values.

<u>Setter Methods in Base Class</u>

Setter methods are included in the Laptop class to allow modifications of the private attributes. For example:

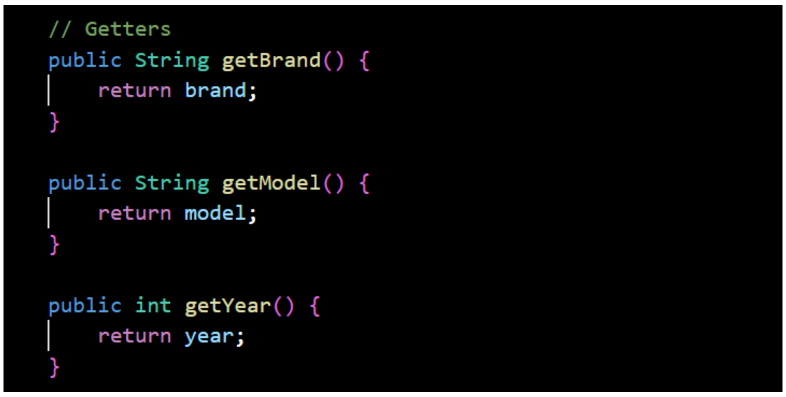


```java
 // Setters
 public void setBrand(String brand) {
    this.brand = brand;
}

public void setModel(String model) {
    this.model = model;
}

public void setYear(int year) {
    this.year = year;
}

// Display method
public void displayInfo() {
    System.out.println("Brand: " + brand);
    System.out.println("Model: " + model);
    System.out.println("Year: " + year);
}
}
```

These methods provide controlled access to modify the object's state, ensuring data integrity.

<u>Getter Methods in Base Class</u>

Getter methods in the Laptop class retrieve the values of private attributes, promoting data encapsulation. An example getter is:



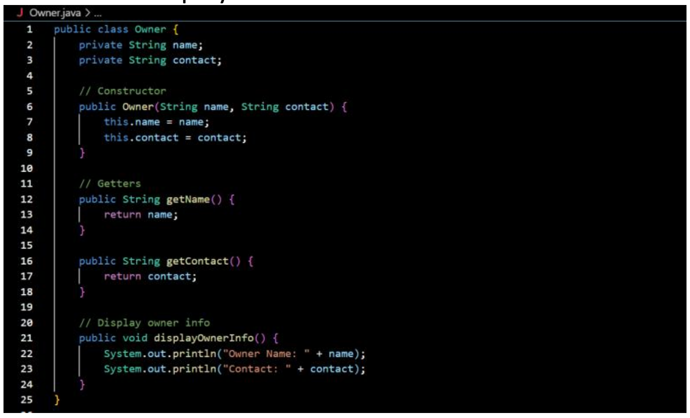
```java
// Getters
public String getBrand() {
    return brand;
}

public String getModel() {
    return model;
}

public int getYear() {
    return year;
}
```

These methods allow read-only access to the attributes.

<u>Additional Classes in the System</u>

Three supporting classes enhance the collaborative aspect of the application:

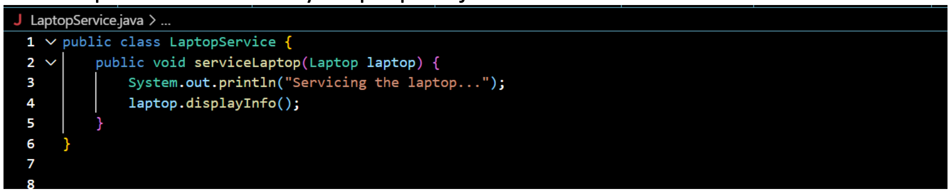
- Owner: Represents the owner of a laptop, with attributes `name` and `contact`. The class includes methods to display owner details.



Owner.java > ...

```java
public class Owner {
    private String name;
    private String contact;

    // Constructor
    public Owner(String name, String contact) {
        this.name = name;
        this.contact = contact;
    }

    // Getters
    public String getName() {
        return name;
    }

    public String getContact() {
        return contact;
    }

    // Display owner info
    public void displayOwnerInfo() {
        System.out.println("Owner Name: " + name);
        System.out.println("Contact: " + contact);
    }
}
```

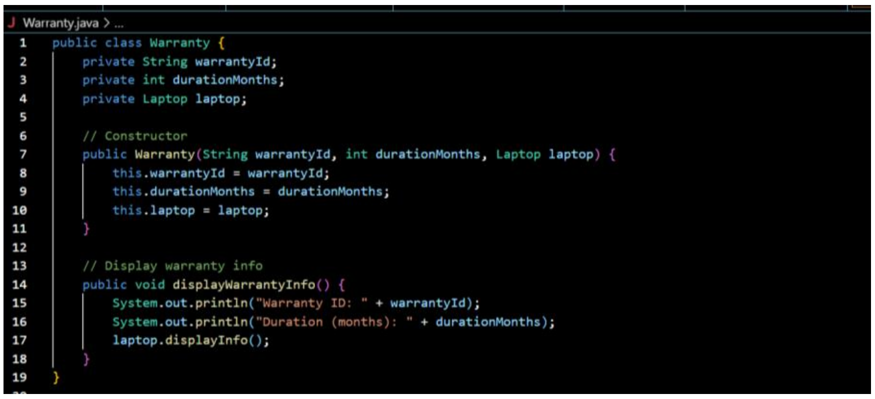
- LaptopService: A utility class that provides a method `serviceLaptop(Laptop laptop)` to demonstrate service procedures for any `Laptop` object.



LaptopService.java > ...

```java
public class LaptopService {
    public void serviceLaptop(Laptop laptop) {
        System.out.println("Servicing the laptop...");
        laptop.displayInfo();
    }
}
```

- Warranty`: Contains warranty information such as `warrantyId`, `durationMonths`, and a reference to the `Laptop` object it covers. The `displayWarrantyInfo()` method outputs warranty and associated laptop details.

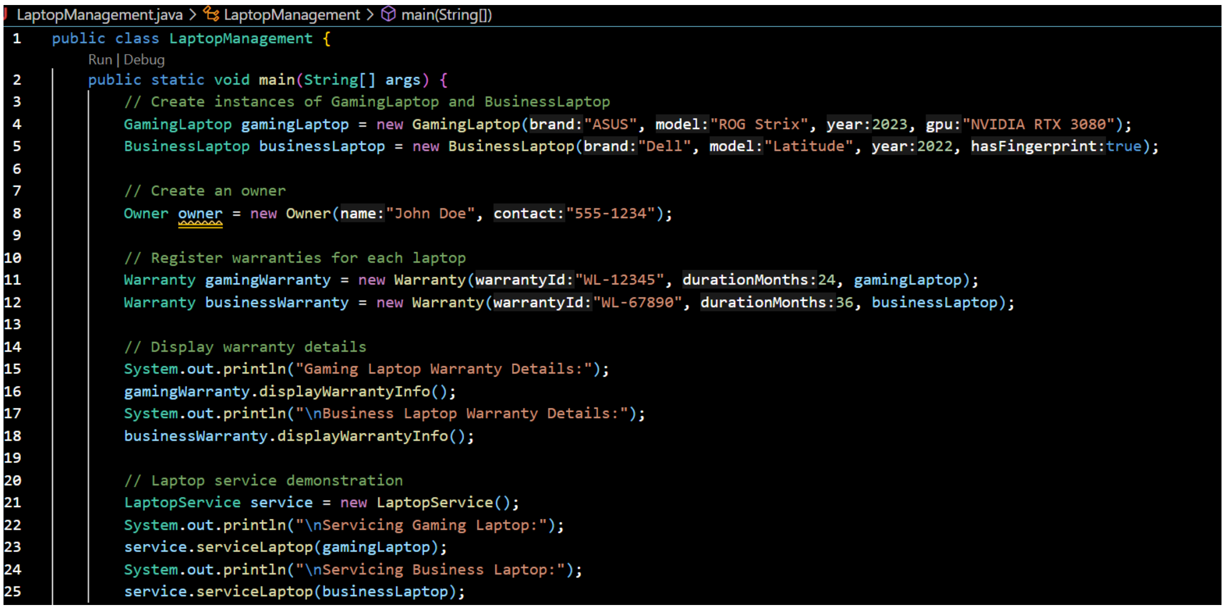


Warranty.java > ...

```java
public class Warranty {
    private String warrantyId;
    private int durationMonths;
    private Laptop laptop;

    // Constructor
    public Warranty(String warrantyId, int durationMonths, Laptop laptop) {
        this.warrantyId = warrantyId;
        this.durationMonths = durationMonths;
        this.laptop = laptop;
    }

    // Display warranty info
    public void displayWarrantyInfo() {
        System.out.println("Warranty ID: " + warrantyId);
        System.out.println("Duration (months): " + durationMonths);
        laptop.displayInfo();
    }
}
```

<u>Main Class: LaptopManagement</u>

The `LaptopManagement` class demonstrates the creation and manipulation of `Laptop` objects and their derived instances:
- Instances of GamingLaptop and BusinessLaptop are created and assigned to variables.
- Owner and Warranty objects are instantiated, showcasing relationships between classes.
- The LaptopService class is used to service laptops, demonstrating method calls with different objects.
- The `displayWarrantyInfo()` method illustrates how class attributes can be displayed cohesively.

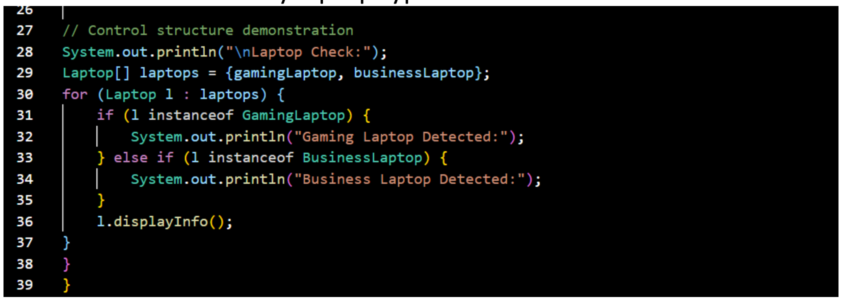


LaptopManagement.java > LaptopManagement > main(String[])

```java
public class LaptopManagement {
    Run | Debug
    public static void main(String[] args) {
        // Create instances of GamingLaptop and BusinessLaptop
        GamingLaptop gamingLaptop = new GamingLaptop(brand:"ASUS", model:"ROG Strix", year:2023, gpu:"NVIDIA RTX 3080");
        BusinessLaptop businessLaptop = new BusinessLaptop(brand:"Dell", model:"Latitude", year:2022, hasFingerprint:true);

        // Create an owner
        Owner owner = new Owner(name:"John Doe", contact:"555-1234");

        // Register warranties for each laptop
        Warranty gamingWarranty = new Warranty(warrantyId:"WL-12345", durationMonths:24, gamingLaptop);
        Warranty businessWarranty = new Warranty(warrantyId:"WL-67890", durationMonths:36, businessLaptop);

        // Display warranty details
        System.out.println("Gaming Laptop Warranty Details:");
        gamingWarranty.displayWarrantyInfo();
        System.out.println("\nBusiness Laptop Warranty Details:");
        businessWarranty.displayWarrantyInfo();

        // Laptop service demonstration
        LaptopService service = new LaptopService();
        System.out.println("\nServicing Gaming Laptop:");
        service.serviceLaptop(gamingLaptop);
        System.out.println("\nServicing Business Laptop:");
        service.serviceLaptop(businessLaptop);
```

<u>Use of Control Structures</u>

Control structures, including `for` loops and `if` statements, are utilized in LaptopManagement to iterate over and identify laptop types:



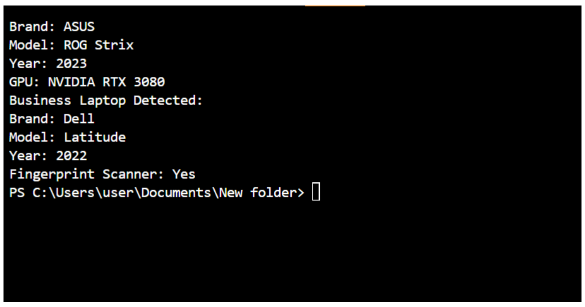
```
// Control structure demonstration
System.out.println("\nLaptop Check:");
Laptop[] laptops = {gamingLaptop, businessLaptop};
for (Laptop l : laptops) {
    if (l instanceof GamingLaptop) {
        System.out.println("Gaming Laptop Detected:");
    } else if (l instanceof BusinessLaptop) {
        System.out.println("Business Laptop Detected:");
    }
    l.displayInfo();
}
}
}
```

This logic demonstrates object identification and polymorphic behavior during runtime.

Encapsulation and Data Privacy

Encapsulation is a core feature throughout all classes, ensuring that attributes are kept private and accessible only through public methods (getters and setters). This design preserves data privacy and protects against unauthorized attribute modifications.

Output :



```
Brand: ASUS
Model: ROG Strix
Year: 2023
GPU: NVIDIA RTX 3080
Business Laptop Detected:
Brand: Dell
Model: Latitude
Year: 2022
Fingerprint Scanner: Yes
PS C:\Users\user\Documents\New folder>
```

1. Laptop (Base Class)

```
public class Laptop {
    private String brand;
    private String model;
    private int year;

    // Constructor
    public Laptop(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    // Getters
    public String getBrand() {
        return brand;
    }

    public String getModel() {
        return model;
    }
    public int getYear() {
        return year;
    }

    // Setters
    public void setBrand(String brand) {
        this.brand = brand;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public void setYear(int year) {
        this.year = year;
    }

    // Display method
    public void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
    }
}
```

2. GamingLaptop (Derived Class)

```
public class GamingLaptop extends Laptop {
    private String gpu;

    // Constructor
    public GamingLaptop(String brand, String model, int year, String gpu) {
        super(brand, model, year);
        this.gpu = gpu;
    }

    // Getter
    public String getGpu() {
        return gpu;
    }

    // Setter
    public void setGpu(String gpu) {
        this.gpu = gpu;
    }

    // Overriding displayInfo() to include GPU
    public void displayInfo() {
        super.displayInfo();
        System.out.println("GPU: " + gpu);
```

```java
    }
}
```

3. BusinessLaptop (Derived Class)

```java
public class BusinessLaptop extends Laptop {
    private boolean hasFingerprint;

    // Constructor
    public BusinessLaptop(String brand, String model, int year, boolean hasFingerprint) {
        super(brand, model, year);
        this.hasFingerprint = hasFingerprint;
    }

    // Getter
    public boolean hasFingerprint() {
        return hasFingerprint;
    }

    // Setter
    public void setFingerprint(boolean hasFingerprint) {
        this.hasFingerprint = hasFingerprint;
    }

    // Overriding displayInfo() to include fingerprint scanner info
    public void displayInfo() {
        super.displayInfo();
        System.out.println("Fingerprint Scanner: " + (hasFingerprint ? "Yes" : "No"));
    }
}
```

4. Owner (Additional Class)

```java
public class Owner {
    private String name;
    private String contact;

    // Constructor
    public Owner(String name, String contact) {
        this.name = name;
        this.contact = contact;
    }

    // Getters
    public String getName() {
        return name;
    }

    public String getContact() {
        return contact;
    }
    // Display owner info
    public void displayOwnerInfo() {
        System.out.println("Owner Name: " + name);
        System.out.println("Contact: " + contact);
    }
}
```

5. LaptopService (Utility Class)

```
public class LaptopService {
    public void serviceLaptop(Laptop laptop) {
        System.out.println("Servicing the laptop...");
        laptop.displayInfo();
    }
}
```

6. Warranty (Additional Class)

```
public class Warranty {
    private String warrantyId;
    private int durationMonths;
    private Laptop laptop;

    // Constructor
    public Warranty(String warrantyId, int durationMonths, Laptop laptop) {
        this.warrantyId = warrantyId;
        this.durationMonths = durationMonths;
        this.laptop = laptop;
    }

    // Display warranty info
    public void displayWarrantyInfo() {
        System.out.println("Warranty ID: " + warrantyId);
        System.out.println("Duration (months): " + durationMonths);
        laptop.displayInfo();
    }
}
```

7. Main Class (LaptopManagement)

```
public class LaptopManagement {
    public static void main(String[] args) {
        // Create instances of GamingLaptop and BusinessLaptop
        GamingLaptop gamingLaptop = new GamingLaptop("ASUS", "ROG Strix", 2023, "NVIDIA RTX
3080");
        BusinessLaptop businessLaptop = new BusinessLaptop("Dell", "Latitude", 2022, true);

        // Create an owner
        Owner owner = new Owner("John Doe", "555-1234");
        // Register warranties for each laptop
        Warranty gamingWarranty = new Warranty("WL-12345", 24, gamingLaptop);
        Warranty businessWarranty = new Warranty("WL-67890", 36, businessLaptop);

        // Display warranty details
        System.out.println("Gaming Laptop Warranty Details:");
        gamingWarranty.displayWarrantyInfo();
        System.out.println("\nBusiness Laptop Warranty Details:");
        businessWarranty.displayWarrantyInfo();

        // Laptop service demonstration
        LaptopService service = new LaptopService();
        System.out.println("\nServicing Gaming Laptop:");
        service.serviceLaptop(gamingLaptop);
        System.out.println("\nServicing Business Laptop:");
        service.serviceLaptop(businessLaptop);

        // Control structure demonstration
        System.out.println("\nLaptop Check:");
        Laptop[] laptops = {gamingLaptop, businessLaptop};
        for (Laptop l : laptops) {
            if (l instanceof GamingLaptop) {
                System.out.println("Gaming Laptop Detected:");
            } else if (l instanceof BusinessLaptop) {
                System.out.println("Business Laptop Detected:");
            }
            l.displayInfo();
        }
    }
}
```

RUBRIC FOR ASSIGNMNET 2

Criteria	Excellent (4)	Good (3)	Satisfactory (2)	Needs Improvement (1)	Marks
OOP Understanding and Base Class (Object Creation and Attributes/Methods) **(5%)**	Excellent understanding of OOP principles. The base class is impeccably designed with well-thought-out attributes/methods, showcasing a deep understanding of OOP.	Good understanding of OOP principles. The base class is well-defined with appropriate attributes/methods, demonstrating a solid grasp of OOP concepts.	Basic understanding of OOP principles but needs improvement. The base class has some attributes/methods, but they are incomplete or poorly designed, reflecting a need for further development.	Limited or no understanding of OOP principles. The base class definition is incorrect or missing, showing a lack of comprehension of fundamental OOP concepts.	
Inheritance and Polymorphism **(20%)**	Inheritance is clear, and polymorphism is skilfully demonstrated with a common interface and distinct implementations in derived classes, showcasing mastery of these OOP principles.	Inheritance and polymorphism are present but require refinement. There's an understanding of these concepts, but they need further improvement for clear and effective implementation.	Attempted inheritance and polymorphism, but not effectively implemented. Basic attempts at inheritance and polymorphism exist but lack clarity and effectiveness.	No inheritance or polymorphism demonstrated, indicating a complete absence of these key OOP concepts.	
Constructor and Setter/Getter Methods(**5%**)	Constructor and setter/getter methods are well-designed and fully functional, demonstrating a strong understanding of class construction and data management.	Constructor and setter/getter methods are well-implemented but may lack some details, showcasing a reasonable understanding of class construction.	Constructor and setter/getter methods exist but are incomplete or incorrect, indicating a need for further development.	Missing constructor and setter/getter methods, demonstrating a lack of basic class structure.	

Main Class and Attribute/Method Manipulation **(20%)**	The main class expertly showcases the manipulation of attributes and methods within the application, indicating a high level of proficiency in class interaction and utilization.	The main class effectively instantiates and manipulates the required classes, demonstrating a reasonable understanding of class interaction.	The main class exists but lacks proper instantiation and manipulation of classes, indicating the need for improvement in demonstrating class interaction.	The main class is missing or not functional, showing a lack of understanding in how to integrate and manipulate classes.	
Creativity, Code Clarity, and Documentation **(20%)**	High creativity, excellent code clarity, and comprehensive documentation explaining the thought process, purpose of attributes/methods, and control structure usage, reflecting strong creativity and effective code explanations.	Some creativity and code clarity. Documentation is present but incomplete, showing moderate creativity and an attempt at code explanation.	Limited creativity and code clarity. Minimal or unclear documentation is present, demonstrating minimal effort in adding creativity and explaining the code.	Lack of creativity and code clarity. No documentation is provided, indicating poor creativity and an absence of code explanations.	
Team Member Evaluation (30%) 1) Task Completion 2) Quality of Work 3) Able to explain clearly 4) Problem-Solving &Technical skills 5) Explain things using OOP terms 6) Q&A					
TOTAL MARKS					

REMARKS

END OF QUESTION