Project Description: Rental Management Application

Project Overview:
A Rental Management Application is a comprehensive software solution developed to facilitate the management of rental properties, ensuring that property owners, property managers, and tenants can effectively and efficiently handle all aspects of the rental process. The application serves as a centralized platform for various tasks, including property listing, tenant onboarding, lease management, maintenance requests, and financial transactions.

Key Features and Functionalities:

1. Property Listings:
    - Property owners can create detailed listings of their rental properties, including property descriptions, rental rates, and high-quality photos.
    - Property managers can easily organize and promote available properties for rent.
2. Tenant Management:
    - Property managers can manage tenant information, including tenant applications, references, and lease agreements.
    - Tenants can submit applications, review lease terms, and communicate with property managers.
3. Lease Tracking:
    - The application helps property managers and property owners keep track of lease agreements, including lease terms, renewal dates, and rent payments.
4. Maintenance Requests:
    - Tenants can submit maintenance requests and track their status.
    - Property managers can assign and manage maintenance tasks efficiently.
5. Financial Management:
    - The application enables property owners and managers to handle financial transactions, including rent collection.
6. Communication and Messaging:
    - The application includes communication tools for property managers, property owners, and tenants to exchange messages and notifications.
7. Document Management:
    - Users can store and manage essential documents such as lease agreements, rental applications, and property inspection reports.
8. Reporting and Analytics:
    - The application provides insights into property performance, financial data, occupancy rates, and maintenance history through reports and analytics.
9. Security and Data Protection:
    - Robust security measures ensure the safety of sensitive data, including personal and financial information.

Practice Assignment 6: Design Pattern

I. Factory Method Pattern (40%)

As an extension of Rental Management Application, look at "Maintenance Requests" requirement then complete the following tasks:

Low-Priority Requests:

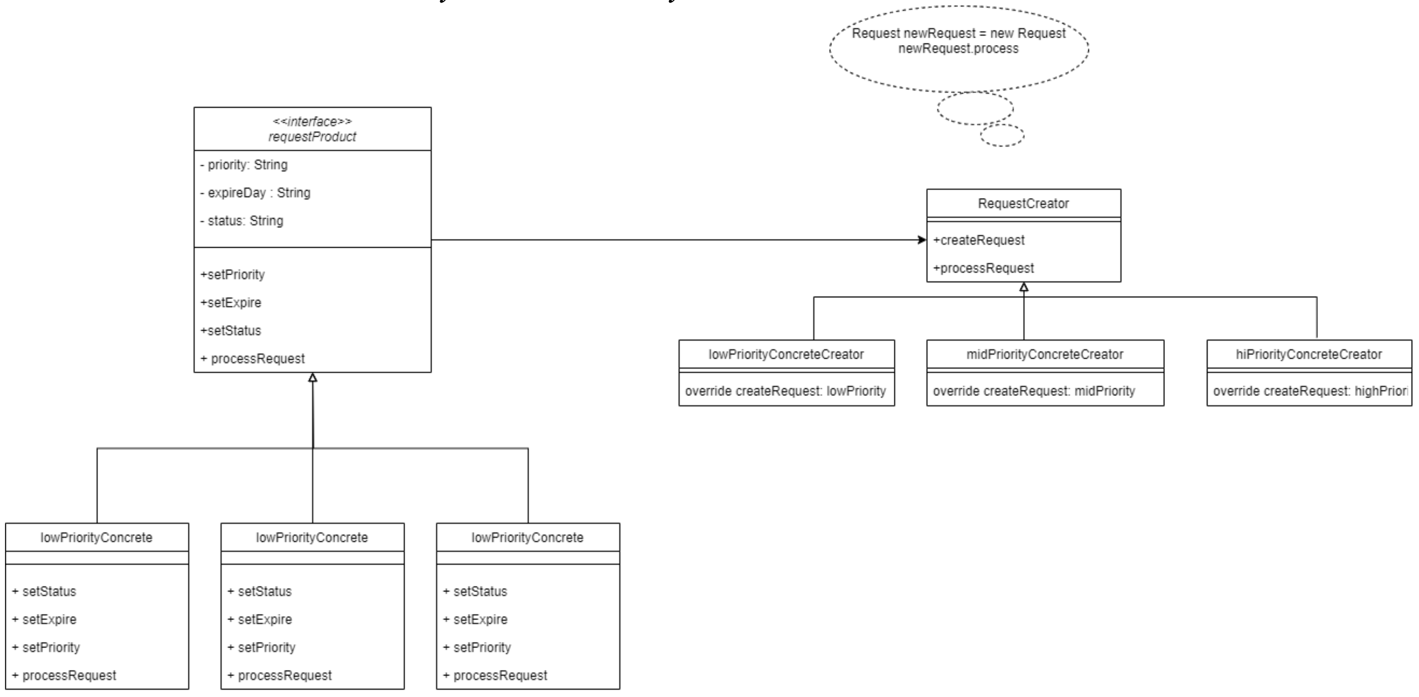
- *setPriority: Priority is set to "Ignore".*
- *setStatus: Status is set to "Done".*
- *setExpire: is set to current day.*
- *processRequest: Print out/Sent the following message "Request denied".*

Medium-Priority Requests:

- *setPriority: Priority is set to "Medium".*
- *setStatus: Status is set to "Accepted".*
- *setExpire: is set to one month from current day.*
- *processRequest: Print out/Sent the following message "Request accept, estimated completion date is [expireDay]".*

High-Priority Requests:

- *setPriority: Priority is set to "Emergency".*
- *setStatus: Status is set to "Accept".*
- *setExpire: is set to current day.*
- *processRequest: Print out/Sent the following message "Emergency request, our Administer will contact you immediately !".*



Permanent
Contract

1. Add one "**User story**" to your Trello which labeled "**Maintenance Requests**" with 2 technical tasks: "**Factory Method Pattern**"; "**Update GitHub**".
2. Set yourself as the sole "**Member**".
3. Set "**Contract"** due day to 28/06/2025. It can have any Prioritize.
4. Implement Factory Method Pattern according to Class Diagram.

II. Code Smells

5. Add one "**User story**" to your Trello which labeled "**Code Smells**" with 3 technical tasks: "**Communication and Messaging**"; "**Reporting and Analytics**"; "**Update GitHub**".
6. Set due day to 28/06/2025. It can have any Prioritize.

1. Code smell 1 (20%)

Look at the implementation of "Communication and Messaging" provided in your Laboratory and see if you can identify any code smells or areas that might need improvement. Then provide your own implementation.

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
class Message {
    private final String content;
    private final String sender;
    private final String recipient;
    public Message(String content, String sender, String recipient) {
        this.content = content;
        this.sender = sender;
        this.recipient = recipient;
    }
    // Getters for message properties
    public String getContent() {
        return content;
    }
    public String getSender() {
        return sender;
    }
    public String getRecipient() {
        return recipient;
    }
    public void printSummary() {
        System.out.println("Content: " + content);
        System.out.println("Sender: " + sender);
        System.out.println("Recipient: " + recipient);
    }
    public void printDetails() {
        System.out.println("Content: " + content);
        System.out.println("Sender: " + sender);
        System.out.println("Recipient: " + recipient);
        System.out.println("Content Length: " + content.length());
        System.out.println("Sender Uppercase: " + sender.toUpperCase());
        System.out.println("Recipient Lowercase: " +
recipient.toLowerCase());
    }
}
class MessagingService {
    private final Map<String, List<Message>> inbox;
    public MessagingService() {
        this.inbox = new HashMap<>();
    }
    public void sendMessage(String content, String sender, String
recipient) {
        Message message = new Message(content, sender, recipient);
        inbox.computeIfAbsent(message.getRecipient(), k -> new
ArrayList<>()).add(message);
    }
    public List<Message> getMessagesForRecipient(String recipient) {
        return inbox.getOrDefault(recipient, new ArrayList<>());
    }
    public void printAllMessages() {
        for (String recipient : inbox.keySet()) {
            List<Message> messages = inbox.get(recipient);
            for (Message message : messages) {
                System.out.println("Recipient: " +
message.getRecipient());
                System.out.println("Sender: " + message.getSender());
                System.out.println("Content: " + message.getContent());
            }
        }
    }
}

public class Main {
    public static void main(String[] args) {
        MessagingService messagingService = new MessagingService();

        // sending messages
        messagingService.sendMessage("Hello, tenant!", "Property
Manager", "Tenant A");
        messagingService.sendMessage("Important notice: Rent due next
week.", "Property Owner", "Tenant A");
```

```
        messagingService.sendMessage("Maintenance request received.", 
"Tenant A", "Property Manager");

        // retrieving messages for a recipient
        List<Message> tenantAMessages = 
messagingService.getMessagesForRecipient("Tenant A");
        for (Message message : tenantAMessages) {
            System.out.println("From: " + message.getSender() + ", 
Content: " + message.getContent());
        }

        // Calling the new method
        Message exampleMessage = new Message("Test", "Sender", 
"Recipient");
        exampleMessage.printDetails();

        messagingService.printAllMessages();
    }
}
```

2. Code smell 2 (20%)

Look at the implementation of "Reporting and Analytics" provided in your Laboratory and see if you can identify any code smells or areas that might need improvement. Then provide your own implementation.

```
import java.util.List;
import java.util.ArrayList;

class Property {
    private String name;
    private double rentAmount;
    private String ownerName;
    private String location;

    public Property(String name, double rentAmount, String ownerName, String location) { // 
Constructor with a data clump
        this.name = name;
        this.rentAmount = rentAmount;
        this.ownerName = ownerName;
        this.location = location;
    }

    public String getName() {
        return name;
    }
    public double getRentAmount() {
        return rentAmount;
    }

    public String getOwnerName() {
        return ownerName;
    }

    public String getLocation() {
        return location;
    }

    public void printPropertyDetails() {
        System.out.println("Property: " + name);
        System.out.println("Rent Amount: $" + rentAmount);
        System.out.println("Owner: " + ownerName);
        System.out.println("Location: " + location);
    }
}

class FinancialReport {
    private String reportTitle;
    private List<Property> properties;
    private double totalRent;

    public FinancialReport(String reportTitle, List<Property> properties) {
        this.reportTitle = reportTitle;
        this.properties = properties;
    }

    public void generateReport() {
        totalRent = 0;
        System.out.println("Financial Report: " + reportTitle);
        System.out.println("---------------------------");
        for (Property property : properties) {
            property.printPropertyDetails();
            totalRent += property.getRentAmount();


            if (property.getRentAmount() > 2000) {
                System.out.println("This is a premium property.");
            } else {
                System.out.println("This is a standard property.");
            }
            double yearlyRent = property.getRentAmount() * 12;
            System.out.println("Yearly Rent: $" + yearlyRent);
            System.out.println("--------------------");
        }

        System.out.println("Total Rent Amount: $" + totalRent);
    }
}

public class ReportGenerator {
    public static void main(String[] args) {
        Property property1 = new Property("Apartment A", 1500.0, "John Doe", "City Center");
        Property property2 = new Property("House B", 2000.0, "Jane Smith", "Suburb");
        Property property3 = new Property("Condo C", 1800.0, "Bob Johnson", "Downtown");

        FinancialReport financialReport = new FinancialReport("Monthly Rent Summary",
List.of(property1, property2, property3));
        financialReport.generateReport();

    }
}
```

III. Make sure that your board mirrors your current progress. (20%)

~The end~