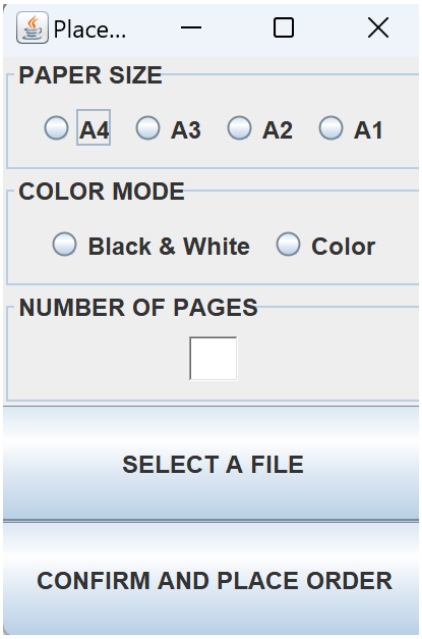
OOPS MINI PROJECT REPORT

PrintPal: Reimagining Print Services



NAME: SAMAH SYED

REGISTER NO.: 3122225001120

DEPARTMENT : CSE

SEMESTER : III

Contents

Problem Statement

PrintPal streamlines print services for printer shops, allowing customers to remotely upload documents, customize print options, and make payments, enabling print preparation. It generates collection tokens, allowing customers to visit the shop solely for hassle-free document collection, eliminating queues and enhancing customer experience.

Motivation for the Problem

Imagine this: It's midterms week, and the air at the college print shop crackles with tension. Ten students snake their way through a winding queue, clutching important documents and hoping to squeeze in those last-minute prints before their deadlines. Every click of the printer feels like an eternity, and the pressure builds with each frustrated sigh.

This familiar scene reflects the pain points of traditional print shops: excruciating queues, inconvenient ordering, and wasted time. PrintPal, our innovative solution, tackles these challenges head-on, promising a revolution in print services.
Following are the major benefits:

- Reduced waiting times: No more queueing anxiety! Get your prints on your own terms.
- Increased convenience: Upload documents and pay remotely, anytime, anywhere.
- Enhanced customization: Choose your preferred printing options for every job.
- Improved efficiency: Eliminate order discrepancies and streamline shop operations.
- Happy students, happy shopkeepers: A win-win solution for a smoother print experience.

Scope and Limitations

The solution has been implemented in Java Programming Language using Gradle, a build automation tool, and Java Swing for GUI. Additionally, MongoDB has been integrated to facilitate remote upload of documents and store user and order data.

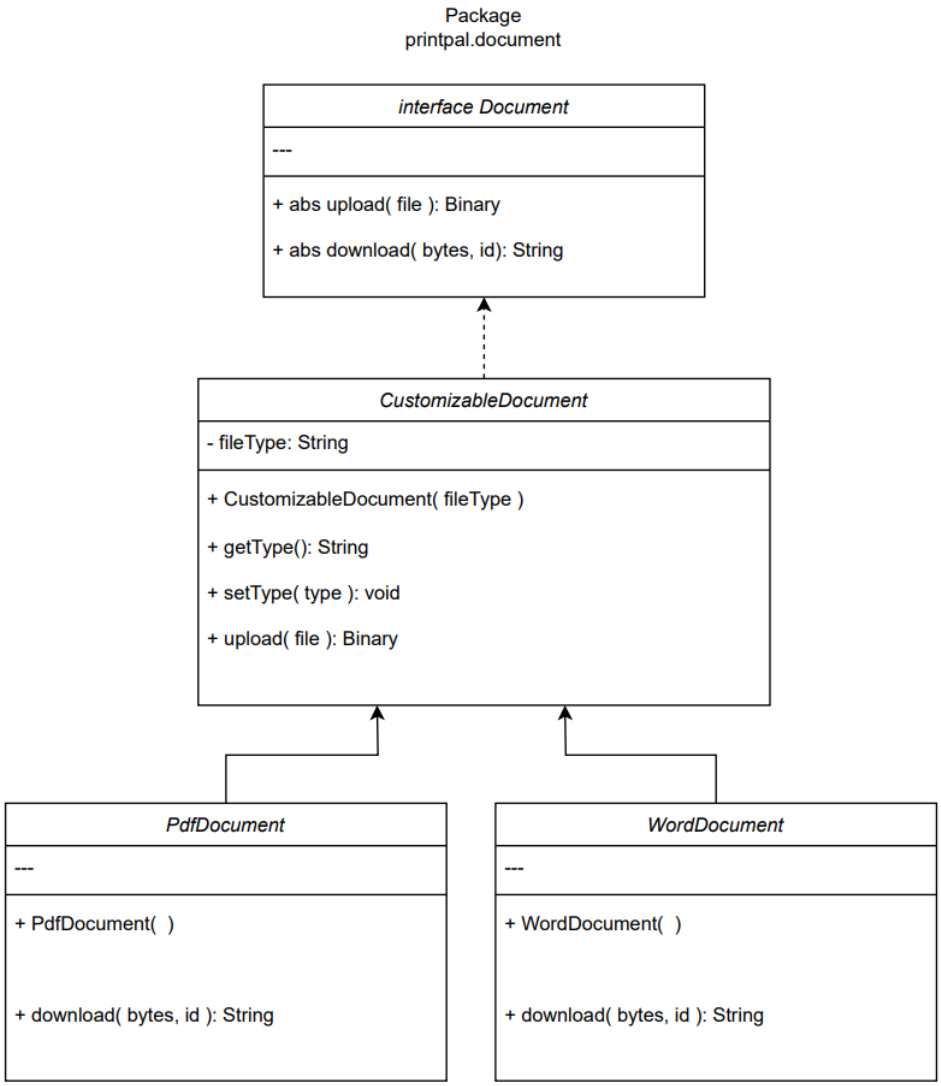
Scope:

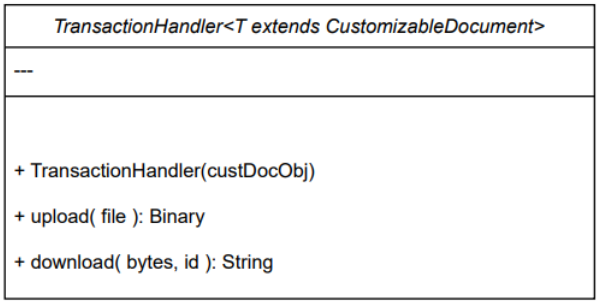
- Document types: Using generic types, the application supports popular formats like PDFs and Word documents. Integration with MongoDB facilitates remote document upload and access.
- Print options: Users can customize paper size, color mode, and quality. Additional options like duplex printing and binding could be added later.
- Token generation: Unique order IDs facilitate hassle-free document collection at the shop.
- Shopkeeper dashboard: Provides visibility into uploaded documents, order management, and invoice generation.
- Customer dashboard: Provides visibility into uploaded documents, order management, and invoice generation.

Limitations:

- No direct printing: PrintPal facilitates remote ordering document upload; and download, but actual printing happens at existing shop infrastructure.
- Shop integration: Requires cooperation from individual print shops to implement token-based document collection.
- Payment Integration: Payment and verification can be integrated with the application interface.
- Security: Robust data security measures must be implemented to protect user information and payment details.

Design of the Solution: Class Diagrams





Package
printpal.printResosources

PrintOptions
- paperSize: enum - colorMode: enum - numberOfPages: int
+ printOptions() + getPaperSize(): PaperSizePrices.PaperSize + getPaperSizeName(): String + setPaperSize(size): void + getColorModeName(): String + getColorMode(): ColorModePrices.PaperSize + setColorMode(color): void + getNumberOfPages(): int + setNumberOfPages(pages): void + getPrice( ): int

PaperSizePrices
+ paperSize: enum + pricePerPage: HashMap<PaperSize, Integer>
+ PaperSizePrices( )

ColorModePrices
+ colorMode: enum + pricePerPage: HashMap<colorMode, Integer>
+ colorModePrices( )

Package
printpal.printResosurces

Shop
- MONGO_DB_NAME: String - COLLECTION_NAME_ORDERS: String
+downloadPdf(orderId) byte[] + viewPendingOrders(): List<Document> + viewPendingCollection(): List<Document> + serveOrder(): Integer + serveOrder(orderID): String + undoServe(orderId): void + declineOrder(orderId): String

customer
- MONGO_DB_NAME: String - COLLECTION_NAME_ORDERS: String - COLLECTIOIN_NAME_CUSTOMERS: String + id: int - name: String - num: String
+ Customer(name, num) - addNewCustomer(id,name, num): void + getName: String + getId: int + getNewOrderId: long + placeOrder(pdf, status, color, size, pages): long + viewPendingOrders(): List<Document> + viewPendingCollection(): List<Document> + viewServedOrders(): List<Document> + viewDeclinedOrders(): List<Document> + viewOrderDetails(orderID): void + collectOrder(orderID): boolean + cancelOrder(orderId): boolean + placeOrder(document, options): int + callToPlaceOrder(customer, file, colorMode, paperSize, numerOfPages: void

Invoice
- invoiceDate: Date - orderId: long - customer: Customer - printObj: PrintOptions - totalAmount: double
+ Invoice (orderId, customer, printObj) + getInvoiceDate(): Date + setInvooiceDate(Date) : void + getCustomer(): Customer + serCustomer(customer) : void + getTotalAmount(): double + setTotalAmount(double) : void + saveAsFile(invoiceString): void + generateInvoice(): String + displayInvoice(): void

App
+ static main(args[]) : void

Modules Split-Up

1. Document Management:

- Interface: `Document` defines abstract methods for document upload and download.
- Classes:
  - `CustomizableDocument` implements `Document` and stores file type.
  - `PdfDocument` and `WordDocument` extend `CustomizableDocument` with specific upload/download implementations.
- Generic Handler: `TransactionHandler` handles document transactions (upload/download) generically for any `CustomizableDocument`.

2. Print Options:

- Class: `PrintOptions` encapsulates user-selected print preferences (paper size, color mode, number of pages).
- Price Mappings: `PaperSizePrices` and `ColorModePrices` map print options to prices for calculation.

3. Transaction Handling:

- TransactionHandler: Handles document upload and download, interacting with `CustomizableDocument` objects.

4. Invoice Generation:

- Class: `Invoice` generates and stores invoice information (date, order ID, customer, print options, total amount).
- Methods: Generate invoice strings, save as files, and display invoice details.

5. Shopkeeper Dashboard:

- Class: `Shop` provides order management functionality for shopkeepers.
- Methods: View pending/collected orders, serve/decline orders, download documents, and manage order status.

6. Customer Portal:

- Class: `Customer` represents customers and their interactions with the system.
- Methods: Place orders, view order status, collect orders, cancel orders, and provide customer information.

7. Main Application:

- Class: `App` contains the `main` method, orchestrating application flow.

Code Flow:

1. Customer Actions:
    - Customer calls `Customer.placeOrder` to initiate an order.
    - `Customer` creates `Invoice` and calls `TransactionHandler.upload` to handle document upload.
2. Shop Actions:
    - Shopkeeper views pending orders using `Shop` methods.
    - Shopkeeper serves orders using `Shop.serveOrder`.
    - Shopkeeper interacts with invoices using `Invoice` methods.
3. Document Handling:
    - `TransactionHandler` delegates upload/download to appropriate `CustomizableDocument` subclasses.
    - Documents are stored in the database (presumably).
4. Invoice Handling:
    - `Invoice` instances are created and managed, providing invoice details and file generation.

Data Storage:

- The code likely interacts with a database to store documents, orders, and customer information.

Implementation

PACKAGE PRINTPAL:

App.java

```
/*
 * This Java source file was generated by the Gradle 'init' task.
 */
package printpal;

import printpal.GUI.MainGUI;

public class App {


    public static void main(String[] args) {
        new MainGUI();


    }
}
```

Customer.java

```
package printpal;


// import com.lowagie.text.pdf.PdfDocument;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

import printpal.document.*;
import printpal.printResources.ColorModePrices;
```

```java
import printpal.printResources.PaperSizePrices;
import printpal.printResources.PrintOptions;

import org.bson.Document;
import org.bson.types.Binary;


import java.io.File;


import java.io.IOException;
import java.util.*;

public class Customer {

    private static final String MONGO_DB_NAME = "myDatabase";
    private static final String COLLECTION_NAME_ORDERS = "orders";
    private static final String COLLECTION_NAME_CUSTOMERS = "customers";
    public final int id;
    private final String name;
    private final String num;



    public Customer(int id, String name, String num) {
        this.id = id;
        this.name = name;
        this.num = num;
    }

    public String toString() {
        return "\nID: " + id + "\nNAME: " + name + "\nCONTACT NUMBER: " + num;
    }

    // new customer, so create a unique id
    public Customer(String name, String num) {
        int highestCusId = 111;
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        Document lastOrder =
database.getCollection(COLLECTION_NAME_ORDERS).find().sort(new Document("_id",
-1))
                .limit(1).first();
        if (lastOrder != null) {
            highestCusId = lastOrder.getInteger("customerId");
```

```
        }
        client.close();

        // Generate a new unique order ID (one higher than the highest
existing)
        int newCusId = highestCusId + 1;

        this.id = newCusId;
        this.name = name;
        this.num = num;

        addNewCustomer(newCusId, name, num);

    }

    private void addNewCustomer(int id, String name, String num) {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_CUSTOMERS);

        Document customerDoc = new Document()
                .append("customerId", id)
                .append("customerName", name)
                .append("customerNum", num);

        collection.insertOne(customerDoc);
        client.close();
    }

    public String getName() {
        return name;
    }

    public int getId() {
        return id;
    }

    public long getNewOrderId() throws IOException {
        // Get the highest existing order ID from the database
        long highestId = 11111;
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        Document lastOrder =
database.getCollection(COLLECTION_NAME_ORDERS).find().sort(new Document("_id",
```

```
-1))
                .limit(1).first();
        if (lastOrder != null) {
            highestId = lastOrder.getLong("orderId");
        }
        client.close();

        // Generate a new unique order ID (one higher than the highest
existing)
        long newOrderId = highestId + 1;

        return newOrderId;
    }

    public long placeOrder(File pdfFile, int status, String color, String size,
int pages) throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);

        // Binary uploadDoc = new PDFDocument().upload(pdfFile);
        PDFDocument obj = new PDFDocument();
        Binary uploadDoc = new
TransactionHandler<PDFDocument>(obj).upload(pdfFile);

        long orderId = getNewOrderId();

        // Build order document
        Document document = new Document()
                .append("orderId", orderId)
                .append("customerId", id)
                .append("customerName", name)
                .append("PhoneNumber", num)
                .append("status", status)
                .append("color", color)
                .append("size", size)
                .append("pages", pages)
                .append("pdf", uploadDoc);

        // Save order to database
        database.getCollection(COLLECTION_NAME_ORDERS).insertOne(document);
        client.close();

        return orderId;

    }
```

```java
    public static void inputsToPlaceOrder(Customer customer, File pdfFile)
throws IOException {

        Scanner scanner = new Scanner(System.in);

        // user input for color
        String colorInput = "";
        boolean validColor = false;
        while (!validColor) {
            System.out.println("Enter desired color mode (BlackWhite or Color):
");
            colorInput = scanner.nextLine().toUpperCase();
            validColor = ColorModePrices.ColorMode.valueOf(colorInput) != null;
            if (!validColor) {
                System.out.println("Invalid color mode. Please try again.");
            }
        }

        // Convert user input to ColorMode enum
        ColorModePrices.ColorMode colorMode =
ColorModePrices.ColorMode.valueOf(colorInput);

        // user input for paper size
        String sizeInput = "";
        boolean validSize = false;
        while (!validSize) {
            System.out.println("Enter desired paper size (A4, A3, A2, or A1):
");
            sizeInput = scanner.nextLine().toUpperCase();
            validSize = PaperSizePrices.PaperSize.valueOf(sizeInput) != null;
            if (!validSize) {
                System.out.println("Invalid paper size. Please try again.");
            }
        }

        // Convert user input to PaperSize enum
        PaperSizePrices.PaperSize paperSize =
PaperSizePrices.PaperSize.valueOf(sizeInput);

        // user input for number of pages
        int numberOfPages = 0;
        boolean validPages = false;
        while (!validPages) {
            System.out.println("Enter number of pages: ");
```

```
            try {
                numberOfPages = Integer.parseInt(scanner.nextLine());
                validPages = numberOfPages > 0;
                if (!validPages) {
                    System.out.println("Invalid number of pages. Please enter a
positive value.");
                }
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a number.");
            }
        }

        // Create PrintOptions object with user input
        PrintOptions printOptions = new PrintOptions(paperSize, colorMode,
numberOfPages);

        // Use the PrintOptions object to call the customer's placeOrder method
        customer.placeOrder(pdfFile, 1, printOptions.getColorModeName(),
printOptions.getPaperSizeName(),
                printOptions.getNumberOfPages());

        System.out.println("Order placed successfully! Total cost: $" +
printOptions.getPrice());
        scanner.close();
    }

    public List<Document> viewPendingOrders() throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);



        Document query = new Document();
        query.put("$and", Arrays.asList(
                new Document("status", 1),
                new Document("customerId", id)));

        FindIterable<Document> pendingOrders = collection.find(query);


        // Use an iterator to iterate through the results and add them to a
list
        List<Document> ordersList = new ArrayList<>();
```

```
        for (Document order : pendingOrders) {
            ordersList.add(order);
        }

        client.close();
        return ordersList;
    }

    public List<Document> viewPendingCollection() throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);

        // Filter documents with status = 2 (awaiting payment)
        Document query = new Document();
        query.put("$and", Arrays.asList(
                new Document("status", 2),
                new Document("customerId", id)));

        FindIterable<Document> pendingCollection = collection.find(query);


        // Use an iterator to iterate through the results and add them to a
list
        List<Document> ordersList = new ArrayList<>();
        for (Document order : pendingCollection) {
            ordersList.add(order);
        }

        client.close();
        return ordersList;

    }

    public List<Document> viewServedOrders() throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);

        // Filter documents with status = 3 (served)
        Document query = new Document();
        query.put("$and", Arrays.asList(
                new Document("status", 3),
```

```
                new Document("customerId", id)));

        FindIterable<Document> servedOrders = collection.find(query);


        // Use an iterator to iterate through the results and add them to a
list
        List<Document> ordersList = new ArrayList<>();
        for (Document order : servedOrders) {
            ordersList.add(order);
        }

        client.close();
        return ordersList;
    }

    public List<Document> viewDeclinedOrders() throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);

        // Filter documents with status = 4 (declined)
        Document query = new Document();
        query.put("$and", Arrays.asList(
                new Document("status", 4),
                new Document("customerId", id)));

        FindIterable<Document> declinedOrders = collection.find(query);


        // Use an iterator to iterate through the results and add them to a
list
        List<Document> ordersList = new ArrayList<>();
        for (Document order : declinedOrders) {
            ordersList.add(order);
        }

        client.close();
        return ordersList;
    }

    public Document viewOrderDetails(long orderId) throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
```

```
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);

        // Find document with matching order ID
        Document orderDetails = collection.find(new Document("orderId",
orderId)).first();

        client.close();
        return orderDetails;
    }

    public boolean cancelOrder(long orderId) throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);

        // Find the order document
        Document order = collection.find(new Document("orderId",
orderId)).first();

        if (order != null) {
            // Check if the order is pending (status = 1)
            int status = order.getInteger("status");
            if (status == 1) {
                // Delete the order document
                collection.deleteOne(new Document("orderId", orderId));
                client.close();
                return true;

            } else {
                client.close();
                return false;


            }
        } else {

            client.close();
            return false;
        }

    }

    public boolean collectOrder(long orderId) throws IOException {
```

```
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);

        // Find the order document
        Document order = collection.find(new Document("orderId",
orderId)).first();

        if (order != null) {
            // Check if the order is not already served (status != 3)
            int status = order.getInteger("status");
            if (status != 3) {
                // Update order status to declined
                if (status == 2) {
                    collection.updateOne(new Document("orderId", orderId),
                            new Document("$set", new Document("status", 3)));
                    client.close();
                    return true;
                } else {
                    client.close();
                    return false;
                }
            } else {
                client.close();
                return false;
            }
        } else {
            client.close();
            return false;
        }

    }

    public void callToPlaceOrder(Customer customer, File file,
ColorModePrices.ColorMode colorMode,
            PaperSizePrices.PaperSize paperSize, int numberOfPages) throws
IOException {

        PrintOptions printOptions = new PrintOptions(paperSize, colorMode,
numberOfPages);

        // Place an order with details
        long orderId = customer.placeOrder(
```

```
                file,
                1, // Status (1-3)
                colorMode.toString(),
                paperSize.toString(),
                numberOfPages);

        Invoice invoice = new Invoice(orderId, customer, printOptions);
        invoice.generateInvoice();

        System.out
                .println("Order placed successfully for customer: " +
customer.getName() + "; ID: " + customer.getId());
    }
}
```

Shop.java

```
package printpal;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

import org.bson.Document;
import org.bson.types.Binary;


import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.FindOneAndUpdateOptions;
import com.mongodb.client.model.ReturnDocument;

import printpal.document.*;

import java.util.*;

public class Shop {
```

```
    private static final String MONGO_DB_NAME = "myDatabase";
    private static final String COLLECTION_NAME_ORDERS = "orders";


    public static byte[] downloadPdf(long orderId) throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);

        Document document =
database.getCollection(COLLECTION_NAME_ORDERS).find(new Document("orderId",
orderId))
                .first();

        if (document == null) {
            throw new IOException("PDF not found for order ID: " + orderId);
        }

        byte[] pdfBytes = document.get("pdf", Binary.class).getData();
        client.close();
        return pdfBytes;
    }

    public List<Document> viewPendingOrders() throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);

        // Filter documents with status = 1 (pending)
        FindIterable<Document> pendingOrders = collection.find(new
Document("status", 1));

        // Use an iterator to iterate through the results and add them to a
list
        List<Document> ordersList = new ArrayList<>();
        for (Document order : pendingOrders) {
            ordersList.add(order);
        }

        client.close();
        return ordersList;
    }

    public List<Document> viewPendingCollection() throws IOException {
        MongoClient client = MongoClients.create();
```

```
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);

        // Filter documents with status = 2 (awaiting payment)
        FindIterable<Document> pendingCollection = collection.find(new
Document("status", 2));

        // Use an iterator to iterate through the results and add them to a
list
        List<Document> ordersList = new ArrayList<>();
        for (Document order : pendingCollection) {
            ordersList.add(order);
        }

        client.close();
        return ordersList;

    }

    public String serveOrder(long orderId) throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);

        // Find the order document
        Document order = collection.find(new Document("orderId",
orderId)).first();
        String message;
        if (order != null) {
            // Check if the order is pending (status = 1)
            int status = order.getInteger("status");
            if (status == 1) {
                // Update order status to served
                collection.updateOne(new Document("orderId", orderId), new
Document("$set", new Document("status", 2)));

                // Download and write the PDF
                try {
                    byte[] downloadedPdfBytes = downloadPdf(orderId);
                    String filename = "downloaded_pdf_" + orderId + ".pdf";
                    File outputFile = new File(filename);
                    try (FileOutputStream outputStream = new
FileOutputStream(outputFile)) {
```

```
                        outputStream.write(downloadedPdfBytes);
                    }
                } catch (IOException e) {


                    message = "Error serving order " + orderId + ": " +
e.getMessage();
                    client.close();
                    return message;


                }

                message = "Order " + orderId + " served successfully!";
            } else {
                message = "Order " + orderId + " cannot be served because it is
not pending.";
            }
        } else {
            message = "Order with ID " + orderId + " not found.";
        }
        client.close();
        return message;


    }

    public void undoServe(int orderId) {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);

        // Find and update the first pending order
        collection.findOneAndUpdate(
                new Document("orderId", orderId),
                new Document("$set", new Document("status", 1)),
                new
FindOneAndUpdateOptions().returnDocument(ReturnDocument.AFTER));
    }

    public Integer serveOrder() throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
```

```
database.getCollection(COLLECTION_NAME_ORDERS);

        // Find and update the first pending order
        Document order = collection.findOneAndUpdate(
                new Document("status", 1),
                new Document("$set", new Document("status", 2)),
                new
FindOneAndUpdateOptions().returnDocument(ReturnDocument.AFTER));



        Integer orderId = null;

        if (order != null) {
            orderId = Integer.parseInt(order.get("orderId").toString());

            // Download and write the PDF
            try {
                byte[] downloadedPdfBytes = downloadPdf(orderId);

                String filename = new
PDFDocument().download(downloadedPdfBytes, orderId);
                System.err.println("File downloaded: " + filename);

            } catch (IOException e) {

                System.err.println("Error serving a pending order: " +
e.getMessage());
            }

            System.out.println("A pending order served successfully!");
        } else {
            System.out.println("No pending orders found.");
        }

        client.close();
        return orderId;
    }

    public String declineOrder(long orderId) throws IOException {
        MongoClient client = MongoClients.create();
        MongoDatabase database = client.getDatabase(MONGO_DB_NAME);
        MongoCollection<Document> collection =
database.getCollection(COLLECTION_NAME_ORDERS);
```

```
        // Find the order document
        Document order = collection.find(new Document("orderId",
orderId)).first();
        String message = null;
        if (order != null) {
            // Check if the order is not already served (status != 3)
            int status = order.getInteger("status");
            if (status != 3 && status!=4) {
                // Update order status to declined
                collection.updateOne(new Document("orderId", orderId), new
Document("$set", new Document("status", 4)));
                message = "Order " + orderId + " declined successfully.";
            } else {
                message = "Order " + orderId + " cannot be declined because it
is already served/declined.";
            }
        } else {
            message = "Order with ID " + orderId + " not found.";
        }

        client.close();
        return message;
    }

}
```

Invoice.java

```
package printpal;

import java.util.*;

import printpal.printResources.PrintOptions;

import java.io.FileWriter;
import java.io.IOException;

public class Invoice {

    private Date invoiceDate;
    private long orderId;
```

```java
    private Customer customer;
    private PrintOptions printObj;
    private double totalAmount;

    public Invoice(long orderId, Customer customer, PrintOptions printObj) {

        this.invoiceDate = new Date();
        this.orderId = orderId;
        this.customer = customer;
        this.printObj = printObj;
        this.totalAmount = printObj.getPrice();

    }



    public Date getInvoiceDate() {
        return invoiceDate;
    }

    public void setInvoiceDate(Date invoiceDate) {
        this.invoiceDate = invoiceDate;
    }

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    public double getTotalAmount() {
        return totalAmount;
    }

    public void setTotalAmount(double totalAmount) {
        this.totalAmount = totalAmount;
    }

    public static void saveAsFile(String invoiceString) throws IOException {
        // Open the file for writing
        try (FileWriter writer = new FileWriter("Invoice.txt")) {
            // Write the invoice string to the file
            writer.write(invoiceString);
```

```
            writer.flush();

        } catch (IOException e) {
            // Handle and report any IOExceptions
            e.printStackTrace();
            System.err.println("Error saving invoice: " + e.getMessage());
        }
    }

    public String generateInvoice() {


        StringBuilder invoiceBuilder = new StringBuilder();

        invoiceBuilder.append("\nInvoice Date: " + invoiceDate);
        invoiceBuilder.append("\nOrder ID: " + orderId);
        invoiceBuilder.append("\n\nCUSTOMER DETAILS: " + customer); // override
to string method

        invoiceBuilder.append("\n\nPRINT CUSTOMIZATIONS:");
        invoiceBuilder.append("\n  • Paper size: " +
printObj.getPaperSizeName());
        invoiceBuilder.append("\n  • Color mode: " +
printObj.getColorModeName());
        invoiceBuilder.append("\n  • Number of pages: " +
printObj.getNumberOfPages());

        // Add the total amount of the invoice.
        invoiceBuilder.append("\n\nTOTAL AMOUNT: " + totalAmount);
        String invoiceString = invoiceBuilder.toString();
        try {
            saveAsFile(invoiceString);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }


        return invoiceString;
    }


}
```

PACKAGE PRINTPAL.DOCUMENT:

Document.java

```
package printpal.document;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;



import org.bson.types.Binary;

public interface Document {

    public String download(byte[] bytes, long id) throws IOException;
    public Binary upload(File fileName) throws FileNotFoundException,
IOException;


}
```

CustomizableDocument.java

```
package printpal.document;

import org.bson.types.Binary;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

import java.io.IOException;

public abstract class CustomizableDocument implements Document {
    private String fileType;
```

```
    public CustomizableDocument(String fileType) {
        this.fileType = fileType;

    }

    public Binary upload(File pdfFile) throws FileNotFoundException,
IOException{


            // Read PDF file as bytes
            byte[] pdfBytes;
            try (FileInputStream inputStream = new FileInputStream(pdfFile)) {
                pdfBytes = inputStream.readAllBytes();
            }

            return new Binary(pdfBytes);

    }



    public String getType() {
        return fileType;
    }

    public void setType(String type) {
        this.fileType = type;
    }


}
```

PDFDocument.java

```
package printpal.document;

import java.io.File;

import java.io.FileOutputStream;
import java.io.IOException;
```

```java
public class PDFDocument extends CustomizableDocument {


    public PDFDocument() {
        super("PDF");
    }


    public String download(byte[] pdfBytes, long orderId) throws IOException{
        String filename = "downloaded_pdf_" + orderId + ".pdf";
        File outputFile = new File(filename);
                try (FileOutputStream outputStream = new
FileOutputStream(outputFile)) {
                    outputStream.write(pdfBytes);
                }
             catch (IOException e) {

                System.err.println("Error downloading the document: " +
e.getMessage());
            }

        return filename;
    }

}
```

TransactionHandler.java

```java
package printpal.document;

import org.bson.types.Binary;
import java.io.File;

import java.io.IOException;



public class TransactionHandler<T extends CustomizableDocument> {

    private T document;
```

```
    public TransactionHandler(T document) {
        this.document = document;
    }

    public String download(byte[] documentBytes, long orderId) throws
IOException {
        return document.download(documentBytes, orderId);
    }

    public Binary upload(File file) throws IOException {
        return document.upload(file);
    }
}
```

PACKAGE PRINTPAL.PRINTRESOURCES

ColorModePrices.java

```
package printpal.printResources;

import java.util.*;


public class ColorModePrices{

    public static enum ColorMode {
        BLACK_AND_WHITE,
        COLOR
    }

    public HashMap<ColorMode, Integer> pricePerPage = new HashMap<>();
    public ColorModePrices(){
        pricePerPage.put(ColorMode.BLACK_AND_WHITE, 1);
        pricePerPage.put(ColorMode.COLOR, 5);
    }
}
```

PaperSizePrices.java

```
package printpal.printResources;

import java.util.*;


public class PaperSizePrices{

    public static enum PaperSize {
        A4,
        A3,
        A2,
        A1
    }

    public HashMap<PaperSize, Integer> pricePerPage = new HashMap<>();

    public PaperSizePrices(){
        pricePerPage.put(PaperSize.A1, 4);
        pricePerPage.put(PaperSize.A2, 3);
        pricePerPage.put(PaperSize.A3, 2);
        pricePerPage.put(PaperSize.A4, 1);

    }

}
```

PrintOptions.java

```
package printpal.printResources;

public class PrintOptions {

    private PaperSizePrices.PaperSize paperSize;
    private ColorModePrices.ColorMode colorMode;
    private int numberOfPages;

    public PrintOptions(PaperSizePrices.PaperSize paperSize,
ColorModePrices.ColorMode colorMode, int numberOfPages) {
```

```
        this.paperSize = paperSize;
        this.colorMode = colorMode;
        this.numberOfPages = numberOfPages;
    }

    public PaperSizePrices.PaperSize getPaperSize() {
        return paperSize;
    }

    public String getPaperSizeName() {
        switch (paperSize) {
          case A4: return "A4";
          case A3: return "A3";
          case A2: return "A2";
          case A1: return "A1";
          default: throw new IllegalStateException("Unexpected paper size: " +
paperSize);
        }
      }

    public void setPaperSize(PaperSizePrices.PaperSize paperSize) {
        this.paperSize = paperSize;
    }

    public ColorModePrices.ColorMode getColorMode() {
        return colorMode;
    }

    public String getColorModeName() {
        switch (colorMode) {
          case BLACK_AND_WHITE: return "BLACK_AND_WHITE";
          case COLOR: return "COLOR";
          default: throw new IllegalStateException("Unexpected color mode: " +
colorMode);
        }
      }

    public void setColorMode(ColorModePrices.ColorMode colorMode) {
        this.colorMode = colorMode;
    }

    public int getNumberOfPages() {
        return numberOfPages;
    }
```

```
    public void setNumberOfPages(int numberOfPages) {
        this.numberOfPages = numberOfPages;
    }

    public int getPrice(){
        PaperSizePrices paperSizePrices = new PaperSizePrices();
        ColorModePrices colorModePrices = new ColorModePrices();
        int sizePrice = paperSizePrices.pricePerPage.get(this.paperSize);
        int colorPrice = colorModePrices.pricePerPage.get(this.colorMode);
        int pricePerPage = sizePrice * colorPrice;
        int totalAmount = pricePerPage * this.numberOfPages;

        return totalAmount;
    }
}
```

PRINTPAL.GUI

CustomerDashboardGUI.java

```
package printpal.GUI;

import org.bson.Document;

import printpal.Customer;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import java.util.List;


public class CustomerDashboardGUI extends JFrame implements ActionListener {

    private final Customer customer;

    private final JButton placeOrderButton, pendingOrdersButton,
pendingCollectionButton, servedOrdersButton,
            declinedOrdersButton, orderDetailsButton, cancelOrderButton,
```

```
collectOrderButton;

    private final JTextField orderIdField;

    public CustomerDashboardGUI(Customer customer) {
        super("Customer Dashboard - " + customer.getName());
        this.customer = customer;

        setLayout(new FlowLayout());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Buttons for functionalities
        placeOrderButton = new JButton("Place Order");
        placeOrderButton.addActionListener(this);
        add(placeOrderButton);

        pendingOrdersButton = new JButton("View Pending Orders");
        pendingOrdersButton.addActionListener(this);
        add(pendingOrdersButton);

        pendingCollectionButton = new JButton("View Pending Collection");
        pendingCollectionButton.addActionListener(this);
        add(pendingCollectionButton);

        servedOrdersButton = new JButton("View Served Orders");
        servedOrdersButton.addActionListener(this);
        add(servedOrdersButton);

        declinedOrdersButton = new JButton("View Declined Orders");
        declinedOrdersButton.addActionListener(this);
        add(declinedOrdersButton);

        orderIdField = new JTextField(10);
        orderIdField.setHorizontalAlignment(SwingConstants.CENTER);
        JLabel orderIdLabel = new JLabel("Order ID:");
        add(orderIdLabel);
        add(orderIdField);

        orderDetailsButton = new JButton("View Order Details");
        orderDetailsButton.addActionListener(this);
        add(orderDetailsButton);

        cancelOrderButton = new JButton("Cancel Order");
        cancelOrderButton.addActionListener(this);
        add(cancelOrderButton);
```

```
        collectOrderButton = new JButton("Collect Order");
        collectOrderButton.addActionListener(this);
        add(collectOrderButton);

        pack();
        setVisible(true);
    }



    public void printOrdersTable(List<Document> Orders) {
        // Build the table header
        StringBuilder header = new StringBuilder();

header.append("+--------").append("+---------").append("+------").append("+----
---").append("+---------").append("+----------").append("+------").append("\n")
;
        header.append("| Order ID | Customer | Color | Size | Pages | Status |
PDF |").append("\n");

header.append("+--------").append("+---------").append("+------").append("+----
---").append("+---------").append("+----------").append("+------").append("\n")
;

        // Build the table rows

        for (Document document : Orders) {
            header.append(formatTableRow(document)).append("\n");
        }

header.append("+--------").append("+---------").append("+------").append("+----
---").append("+---------").append("+----------").append("+------").append("\n")
;


        JOptionPane.showMessageDialog(this, header.toString(), "Orders",
JOptionPane.INFORMATION_MESSAGE);
    }

    private static String formatTableRow(Document document) {
        StringBuilder row = new StringBuilder();
        row.append("| ").append(document.get("orderId")).append(" | ");
        row.append(document.get("customerName")).append(" | ");
        row.append(document.get("color")).append(" | ");
```

```
        row.append(document.get("size")).append(" | ");
        row.append(document.get("pages")).append(" | ");
        row.append(document.get("status")).append(" | ");
        // Add "Yes" or "No" based on PDF presence
        String pdfStatus = document.get("pdf") != null ? "Yes" : "No";
        row.append(pdfStatus).append(" |");
        return row.toString();
    }

    private void displayOrderDetails(Document orderDetails) {
        if (orderDetails == null) {
            return;
        }



        // Build the vertical table format
        StringBuilder textBuilder = new StringBuilder();
        textBuilder.append("*** Order Details for
").append(orderDetails.get("customerName")).append(" (Order ID:
").append(orderDetails.get("orderId")).append(") ***\n");

        textBuilder.append("- Color:
").append(orderDetails.get("color")).append("\n");
        textBuilder.append("- Size:
").append(orderDetails.get("size")).append("\n");
        textBuilder.append("- Pages:
").append(orderDetails.get("pages")).append("\n");
        textBuilder.append("- Status:
").append(orderDetails.get("status")).append("\n");
        if (orderDetails.get("pdf")!=null) {
            textBuilder.append("- PDF Uploaded: YES\n");
        } else {
            textBuilder.append("- PDF Uploaded: NO\n");
        }


        JOptionPane.showMessageDialog(this, textBuilder.toString(), "Order
Details", JOptionPane.INFORMATION_MESSAGE);


    }
```

```
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == placeOrderButton) {
            new PlaceOrderGUI(customer);
        } else if (e.getSource() == pendingOrdersButton) {
            // handle viewPendingOrders() call and display results
            List<Document> pendingOrders = null;
            try {
                pendingOrders = customer.viewPendingOrders();
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }

            if (pendingOrders != null) {

                printOrdersTable(pendingOrders);;
            } else {
                JOptionPane.showMessageDialog(this, "No pending orders
found.");
            }
        } else if (e.getSource() == pendingCollectionButton) {
            // handle viewPendingCollection() call and display results
            List<Document> pendingCollection = null;
            try {
                pendingCollection = customer.viewPendingCollection();
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }

            if (pendingCollection != null) {

                printOrdersTable(pendingCollection);
            } else {
                JOptionPane.showMessageDialog(this, "No pending collection
orders found.");
            }
        } else if (e.getSource() == servedOrdersButton) {
            // handle viewServedOrders() call and display results
            List<Document> servedOrders = null;
            try {
                servedOrders = customer.viewServedOrders();
            } catch (IOException e1) {
                // TODO Auto-generated catch block
```

```
                e1.printStackTrace();
            }

            if (servedOrders != null) {

                printOrdersTable(servedOrders);
            } else {
                JOptionPane.showMessageDialog(this, "No served orders found.");
            }
        } else if (e.getSource() == declinedOrdersButton) {
            // handle viewDeclinedOrders() call and display results
            List<Document> declinedOrders = null;
            ;
            try {
                declinedOrders = customer.viewDeclinedOrders();
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }

            if (declinedOrders != null) {

                printOrdersTable(declinedOrders);
            } else {
                JOptionPane.showMessageDialog(this, "No declined orders
found.");
            }
        } else if (e.getSource() == orderDetailsButton) {
            // Get order ID from user input
            long orderId = getOrderIdInput();
            if (orderId == -1) {
                // Handle invalid input scenario
                JOptionPane.showMessageDialog(this, "Invalid order ID
entered.");
            } else {
                Document orderDetails = null;
                ;
                try {
                    orderDetails = customer.viewOrderDetails(orderId);
                    this.displayOrderDetails(orderDetails);
                } catch (IOException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
                if (orderDetails != null) {
```

```
                } else {
                    JOptionPane.showMessageDialog(this, "Order not found.");
                }
            }
        } else if (e.getSource() == cancelOrderButton) {
            // Get order ID from user input
            long orderId = getOrderIdInput();
            if (orderId == -1) {
                // Handle invalid input scenario
                JOptionPane.showMessageDialog(this, "Invalid order ID
entered.");
            } else {
                boolean successful = false;
                try {
                    successful = customer.cancelOrder(orderId);
                } catch (IOException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
                if (successful) {
                    JOptionPane.showMessageDialog(this, "Order successfully
cancelled.");
                    // Update any relevant lists on the dashboard based on
cancelled order
                } else {
                    JOptionPane.showMessageDialog(this, "Failed to cancel
order. Please try again later.");
                }
            }
        } else if (e.getSource() == collectOrderButton) {
            // Get order ID from user input
            long orderId = getOrderIdInput();
            if (orderId == -1) {
                // Handle invalid input scenario
                JOptionPane.showMessageDialog(this, "Invalid order ID
entered.");
            } else {
                boolean successful = false;
                try {
                    successful = customer.collectOrder(orderId);
                } catch (IOException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
```

```
                if (successful) {
                    JOptionPane.showMessageDialog(this, "Order successfully
collected.");
                    // Update any relevant lists on the dashboard based on
collected order
                } else {
                    JOptionPane.showMessageDialog(this, "Failed to collect
order. Please try again later.");
                }
            }
        }
    }

    private long getOrderIdInput() {
        try {
            return Long.parseLong(orderIdField.getText().trim());
        } catch (NumberFormatException e) {
            return -1;
        }
    }


}
```

CustomerLoginGUI.java

```
package printpal.GUI;
import javax.swing.*;

import printpal.Customer;

import java.awt.*;
import java.awt.event.*;

public class CustomerLoginGUI extends JFrame implements ActionListener {

    private final JTextField cusIdField, nameField, numberField;
    private final JButton loginButton;
    private Customer customer;

    public CustomerLoginGUI() {
```

```
        super("Customer Login");
        setLayout(new FlowLayout());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel cusIdLabel = new JLabel("Customer ID:");
        add(cusIdLabel);
        cusIdField = new JTextField(10);
        add(cusIdField);

        JLabel nameLabel = new JLabel("Name:");
        add(nameLabel);
        nameField = new JTextField(20);
        add(nameField);

        JLabel numberLabel = new JLabel("Phone Number:");
        add(numberLabel);
        numberField = new JTextField(15);
        add(numberField);

        loginButton = new JButton("Login");
        loginButton.addActionListener(this);
        add(loginButton);

        pack();
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == loginButton) {
            String cusIdString = cusIdField.getText().trim();
            String name = nameField.getText().trim();
            String number = numberField.getText().trim();


            if (!cusIdString.isEmpty() && !name.isEmpty() && !number.isEmpty())
{
                try {
                    int cusId = Integer.parseInt(cusIdString);
                    customer = new Customer(cusId, name, number);

                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(this, "Invalid Customer ID
format.");
                    return;
```

```
                }
            } else if (!name.isEmpty() && !number.isEmpty()) {
                customer = new Customer(name, number);
                JOptionPane.showMessageDialog(this, "Your Customer ID is:
"+customer.id);
            } else {
                JOptionPane.showMessageDialog(this, "Please enter at least Name
and Phone Number.");

                return;
            }


            new CustomerDashboardGUI(customer);
        }
    }




}
```

PlaceOrderGUI.java

```
package printpal.GUI;

import javax.swing.*;

import printpal.Customer;
import printpal.printResources.ColorModePrices.ColorMode;
import printpal.printResources.PaperSizePrices.PaperSize;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;

public class PlaceOrderGUI extends JFrame implements ActionListener {

    private Customer customer;
    private PaperSize paperSize = PaperSize.A4;
```

```java
    private ColorMode colorMode = ColorMode.BLACK_AND_WHITE;
    private int numberOfPages = 1;
    private File file;


    public PlaceOrderGUI(Customer customer) {
        this.customer = customer;

        initUI();
    }

    private JRadioButton blackAndWhiteButton, colorButton;
    private JRadioButton a4Button, a3Button, a2Button, a1Button;
    private JButton fileButton, orderButton;
    private TextField pageField;

    private void initUI() {
        // this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(new GridLayout(5,1));
        this.setTitle("Place an order");




        // size ----------------------------------------
        a4Button = new JRadioButton("A4");
        a3Button = new JRadioButton("A3");
        a2Button = new JRadioButton("A2");
        a1Button = new JRadioButton("A1");


        JPanel sizePanel = new JPanel(new FlowLayout());

        ButtonGroup sizeGroup = new ButtonGroup();
        sizeGroup.add(a4Button);
        sizeGroup.add(a3Button);
        sizeGroup.add(a2Button);
        sizeGroup.add(a1Button);

        sizePanel.setBorder(BorderFactory.createTitledBorder("PAPER SIZE"));


        sizePanel.add(a4Button);
        sizePanel.add(a3Button);
        sizePanel.add(a2Button);
```

```
        sizePanel.add(a1Button);

        a4Button.addActionListener(this);
        a3Button.addActionListener(this);
        a2Button.addActionListener(this);
        a1Button.addActionListener(this);




        this.add(sizePanel);


        // color ----------------------------------------
        JPanel colorPanel = new JPanel(new FlowLayout());


        blackAndWhiteButton = new JRadioButton("Black & White");
        colorButton = new JRadioButton("Color");
        ButtonGroup colorGroup = new ButtonGroup();
        colorGroup.add(blackAndWhiteButton);
        colorGroup.add(colorButton);



        colorPanel.setBorder(BorderFactory.createTitledBorder("COLOR MODE"));
        colorPanel.add(blackAndWhiteButton);
        colorPanel.add(colorButton);

        blackAndWhiteButton.addActionListener(this);
        colorButton.addActionListener(this);




        this.add(colorPanel);


        // number of pages ----------------------------------------
        JPanel pagePanel = new JPanel(new FlowLayout());
        pagePanel.setBorder(BorderFactory.createTitledBorder("NUMBER OF
PAGES"));
        pageField = new TextField();
        pagePanel.add(pageField);
        this.add(pagePanel);
```

```
        // File selector ----------------------------------------
        fileButton = new JButton("SELECT A FILE");
        fileButton.addActionListener(this);
        this.add(fileButton);


        // Place Order --------------------------------------------
        orderButton = new JButton("CONFIRM AND PLACE ORDER");
        orderButton.addActionListener(this);
        this.add(orderButton);



        this.pack();
        this.setVisible(true);

    }



    public void actionPerformed(ActionEvent e) {

        if (e.getSource() == a4Button) {
            paperSize = PaperSize.A4;
        } else if (e.getSource() == a3Button) {
            paperSize = PaperSize.A3;
        } else if (e.getSource() == a2Button) {
            paperSize = PaperSize.A2;
        } else if (e.getSource() == a1Button) {
            paperSize = PaperSize.A1;
        }

        if (e.getSource() == blackAndWhiteButton) {
            colorMode = ColorMode.BLACK_AND_WHITE;
        } else if (e.getSource() == colorButton) {
            colorMode = ColorMode.COLOR;
        }

        if (e.getSource() == fileButton){
            JFileChooser fileChooser = new JFileChooser();
            int response = fileChooser.showOpenDialog(null);
            if(response == JFileChooser.APPROVE_OPTION){
                file = new
```

```
File(fileChooser.getSelectedFile().getAbsolutePath());

            }
        }

        if (e.getSource() == orderButton){
            numberOfPages = Integer.parseInt(pageField.getText());
            try {
                customer.callToPlaceOrder(customer, file, colorMode, paperSize,
numberOfPages);

                JOptionPane.showMessageDialog(this, "ORDER HAS BEEN
PLACED.\nINVOICE FILE CREATED AS: Invoice.txt");
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }

        }

    }


}
```

ShopDashboardGUI.java

```
package printpal.GUI;

import java.awt.*;
import java.awt.event.*;
import java.io.IOException;

import javax.swing.*;
import java.util.List;

import org.bson.Document;

import printpal.Shop;

public class ShopDashboardGUI extends JFrame implements ActionListener {
```

```java
    private final Shop shopService;

    private final JTextField orderIdField = new JTextField(10);
    private final JButton viewPendingOrdersButton = new JButton("View Pending
Orders");
    private final JButton viewPendingCollectionButton = new JButton("View
Pending Collection");
    private final JButton serveOrderButton = new JButton("Serve Order");
    private final JButton serveOrderIdButton = new JButton("Serve Order (by
ID)");
    private final JButton declineOrderButton = new JButton("Decline Order");

    public ShopDashboardGUI(Shop shopService) {
        this.shopService = shopService;
        initializeUI();
        addActionListeners();
    }

    private void initializeUI() {
        setLayout(new FlowLayout());
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);



        add(viewPendingOrdersButton);
        add(viewPendingCollectionButton);

        add(serveOrderButton);
        add(new JLabel("Order ID:"));
        add(orderIdField);
        add(serveOrderIdButton);
        add(declineOrderButton);

        setTitle("Shop Dashboard");
        pack();
        setVisible(true);
    }

    private void addActionListeners() {
        viewPendingOrdersButton.addActionListener(this);
        viewPendingCollectionButton.addActionListener(this);
        serveOrderButton.addActionListener(this);
        serveOrderIdButton.addActionListener(this);
        declineOrderButton.addActionListener(this);
```

```
    }

    public void printOrdersTable(List<Document> Orders) {
        // Build the table header
        StringBuilder header = new StringBuilder();

header.append("+--------").append("+---------").append("+------").append("+----
---").append("+---------")
                .append("+----------").append("+------").append("\n");
        header.append("| Order ID | Customer | Color | Size | Pages | Status |
PDF |").append("\n");

header.append("+--------").append("+---------").append("+------").append("+----
---").append("+---------")
                .append("+----------").append("+------").append("\n");

        // Build the table rows

        for (Document document : Orders) {
            header.append(formatTableRow(document)).append("\n");
        }

header.append("+--------").append("+---------").append("+------").append("+----
---").append("+---------")
                .append("+----------").append("+------").append("\n");

        JOptionPane.showMessageDialog(this, header.toString(), "Orders",
JOptionPane.INFORMATION_MESSAGE);
    }

    private static String formatTableRow(Document document) {
        StringBuilder row = new StringBuilder();
        row.append("| ").append(document.get("orderId")).append(" | ");
        row.append(document.get("customerName")).append(" | ");
        row.append(document.get("color")).append(" | ");
        row.append(document.get("size")).append(" | ");
        row.append(document.get("pages")).append(" | ");
        row.append(document.get("status")).append(" | ");
        // Add "Yes" or "No" based on PDF presence
        String pdfStatus = document.get("pdf") != null ? "Yes" : "No";
        row.append(pdfStatus).append(" |");
        return row.toString();
    }

    @Override
```

```
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == viewPendingOrdersButton) {
            // Call and handle results of viewPendingOrders() function
            try {
                List<Document> pendingOrders = shopService.viewPendingOrders();
                printOrdersTable(pendingOrders);
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }

        } else if (e.getSource() == viewPendingCollectionButton) {
            // Call and handle results of viewPendingCollection() function
            try {
                List<Document> pendingCollection =
shopService.viewPendingCollection();
                printOrdersTable(pendingCollection);
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }

        } else if (e.getSource() == serveOrderButton) {
            Integer orderId = null;

            try {
                orderId = shopService.serveOrder();
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            int result = JOptionPane.showConfirmDialog(this, "Serve order " +
orderId + "?");
            if (result == JOptionPane.YES_OPTION) {
                JOptionPane.showMessageDialog(this,
                        "Order " + orderId + " served successfully.\nFILE HAS
BEEN DOWNLOADED");
            } else {
                JOptionPane.showMessageDialog(this, "Order serving
cancelled.");
                shopService.undoServe(orderId);
            }
        } else if (e.getSource() == serveOrderIdButton) {
            // Serve the order specified in the textField
            Long orderId = null;
```

```
            try{
            orderId = Long.parseLong(orderIdField.getText());
            String message = null;
            int result = JOptionPane.showConfirmDialog(this, "Serve order " +
orderId + "?");
            if (result == JOptionPane.YES_OPTION) {
                try {
                    message = shopService.serveOrder(orderId);
                } catch (IOException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
                JOptionPane.showMessageDialog(this, message);
            } else {
                JOptionPane.showMessageDialog(this, "Order serving
cancelled.");
            }
            } catch (NumberFormatException ne){
                JOptionPane.showMessageDialog(this, "ORDER ID IS NILL");
            }
        } else if (e.getSource() == declineOrderButton) {
            // Decline the current order based on orderIdField
            Long orderId = null;
            try{
            orderId = Long.parseLong(orderIdField.getText());
            String message = null;
            int result = JOptionPane.showConfirmDialog(this, "Decline order " +
orderId + "?");
            if (result == JOptionPane.YES_OPTION) {
                try {
                    message = shopService.declineOrder(orderId);
                } catch (IOException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
                JOptionPane.showMessageDialog(this, message);
            } else {
                JOptionPane.showMessageDialog(this, "Order decline
cancelled.");
            }
            } catch (NumberFormatException ne){
                JOptionPane.showMessageDialog(this, "ORDER ID IS NILL");
            }
        }
    }
```

```
}
```

MainGUI.java

```
package printpal.GUI;

import java.awt.*;
import java.awt.event.*;


import javax.swing.*;

import printpal.Shop;


public class MainGUI extends JFrame implements ActionListener {
    private final JButton shopButton = new JButton("Shop Login");
    private final JButton customerButton = new JButton("Customer Login");

    public MainGUI() {

        initializeUI();

    }

    private void initializeUI() {
        setLayout(new FlowLayout());
        setTitle("PrintPal");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        add(shopButton);
        add(customerButton);
        shopButton.addActionListener(this);
        customerButton.addActionListener(this);

        pack();
        setVisible(true);
    }
```

```
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == shopButton) {

            new ShopDashboardGUI(new Shop());

        } else if (e.getSource() == customerButton) {

            new CustomerLoginGUI();

        }

    }
}
```

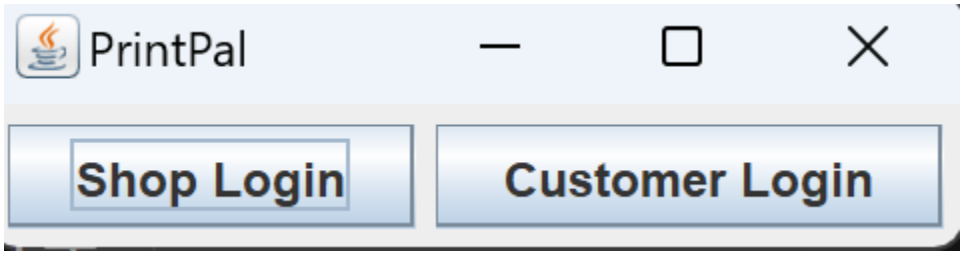
BUILD FILE FOR GRADLE PROJECT

```
/*
 * This file was generated by the Gradle 'init' task.
 *
 * This generated file contains a sample Java application project to get you
started.
 * For more details take a look at the 'Building Java & JVM projects' chapter
in the Gradle
 * User Manual available at
https://docs.gradle.org/8.0.2/userguide/building_java_projects.html
 */

plugins {
    // Apply the application plugin to add support for building a CLI
application in Java.
    application

}


repositories {
    // Use Maven Central for resolving dependencies.
    mavenCentral()


}
```

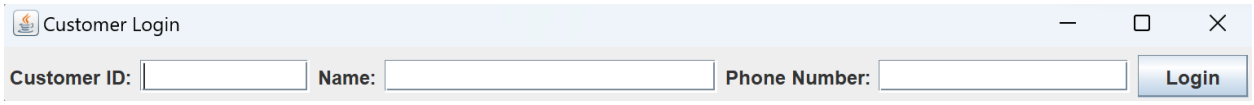
```
dependencies {
    // Use JUnit Jupiter for testing.
    testImplementation("org.junit.jupiter:junit-jupiter:5.9.1")

    // This dependency is used by the application.
    implementation("com.google.guava:guava:31.1-jre")




    implementation("com.google.auth:google-auth-library-oauth2-http:1.11.0")
    implementation("com.google.code.gson:gson:2.9.1")


    implementation("com.lowagie:itext:2.1.7")


implementation("com.fasterxml.jackson.dataformat:jackson-dataformat-xml:2.13.0"
)


    implementation("org.mongodb:mongodb-driver-sync:4.11.1")
    implementation("org.apache.poi:poi-ooxml:5.2.3")



}

application {
    // Define the main class for the application.
    mainClass.set("printpal.App")
}

tasks.named<JavaExec>("run") {
    standardInput = System.`in`
}

tasks.named<Test>("test") {
    // Use JUnit Platform for unit tests.
    useJUnitPlatform()
}
```

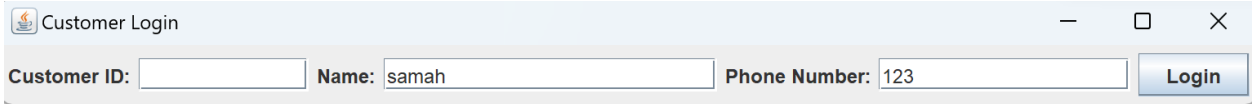
Output Screenshots

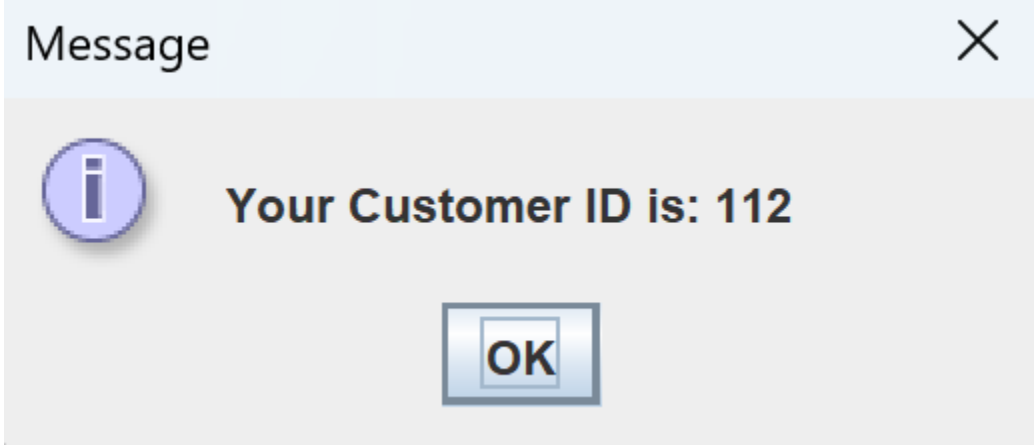
Main Frame



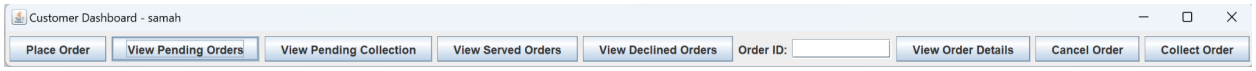
Customer login and registration



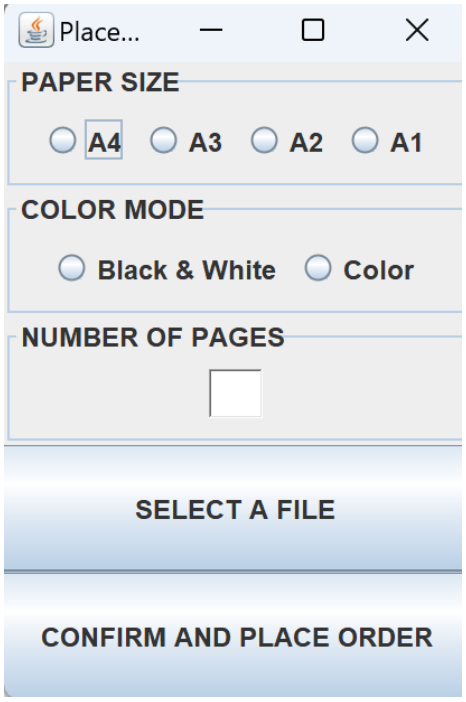




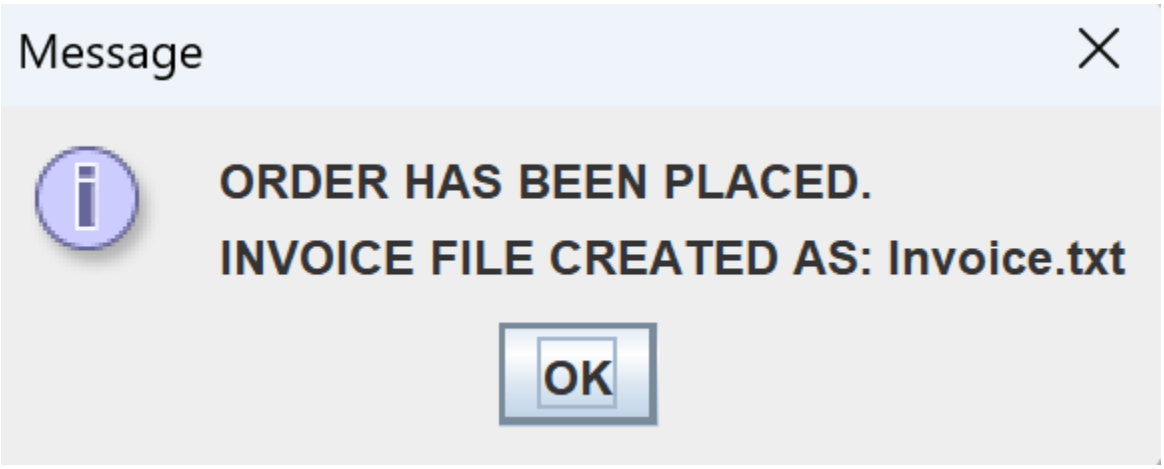
Customer Details



Order customization



Invoice creation



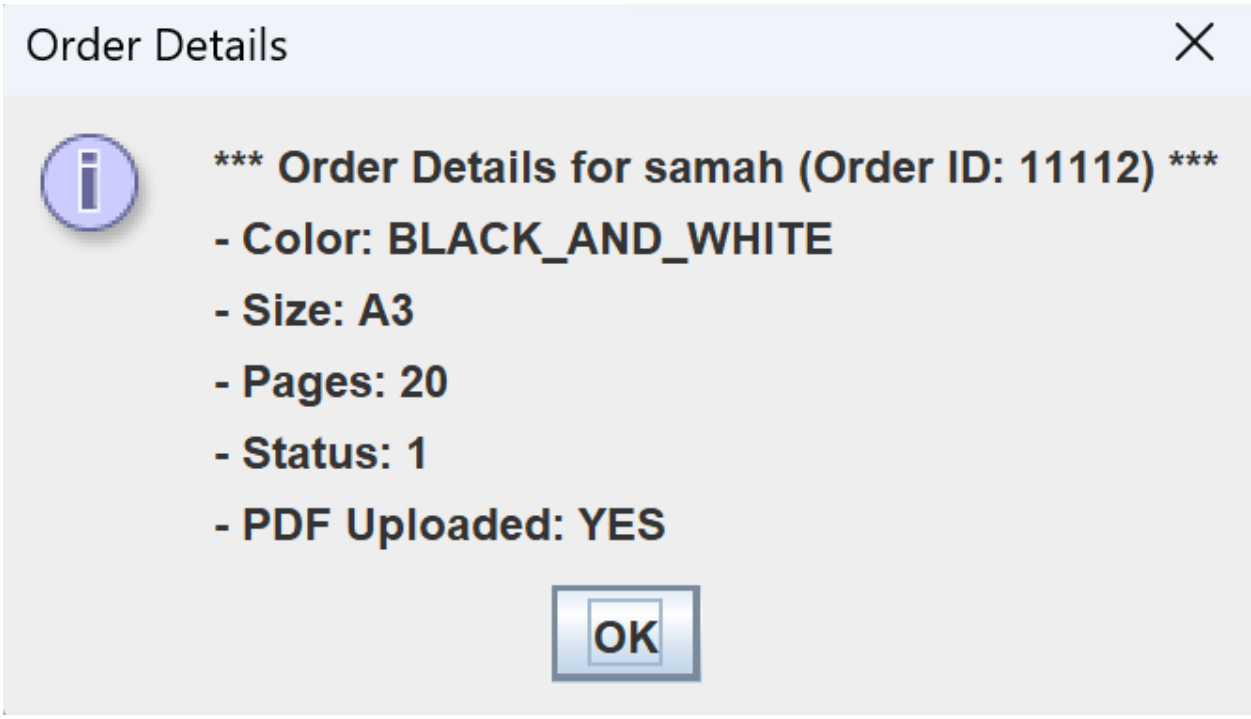
Invoice.txt

```
Invoice Date: Fri Dec 22 23:07:01 IST 2023
Order ID: 11112

CUSTOMER DETAILS:
ID: 112
NAME: samah
CONTACT NUMBER: 123

PRINT CUSTOMIZATIONS:
  • Paper size: A3
  • Color mode: BLACK_AND_WHITE
  • Number of pages: 20

TOTAL AMOUNT: 40.0
```

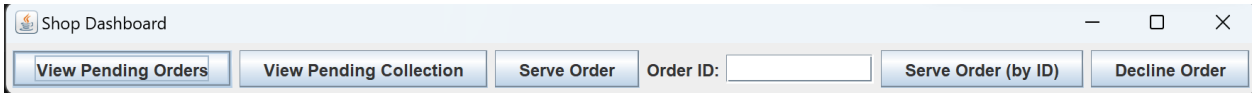
Orders

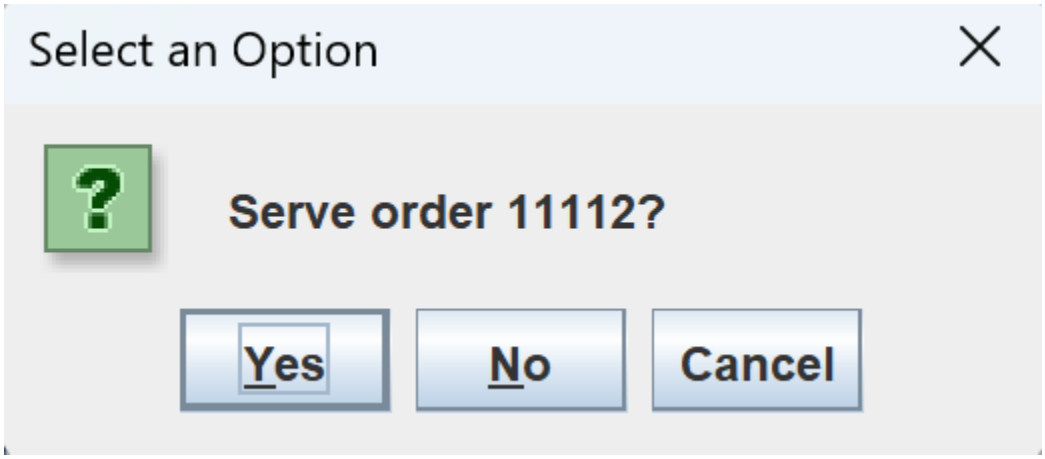
```
+--------+---------+------+-------+---------+----------+------
| Order ID | Customer | Color | Size | Pages | Status | PDF |
+--------+---------+------+-------+---------+----------+------
| 11112 | samah | BLACK_AND_WHITE | A3 | 20 | 1 | Yes |
+--------+---------+------+-------+---------+----------+------
```

OK

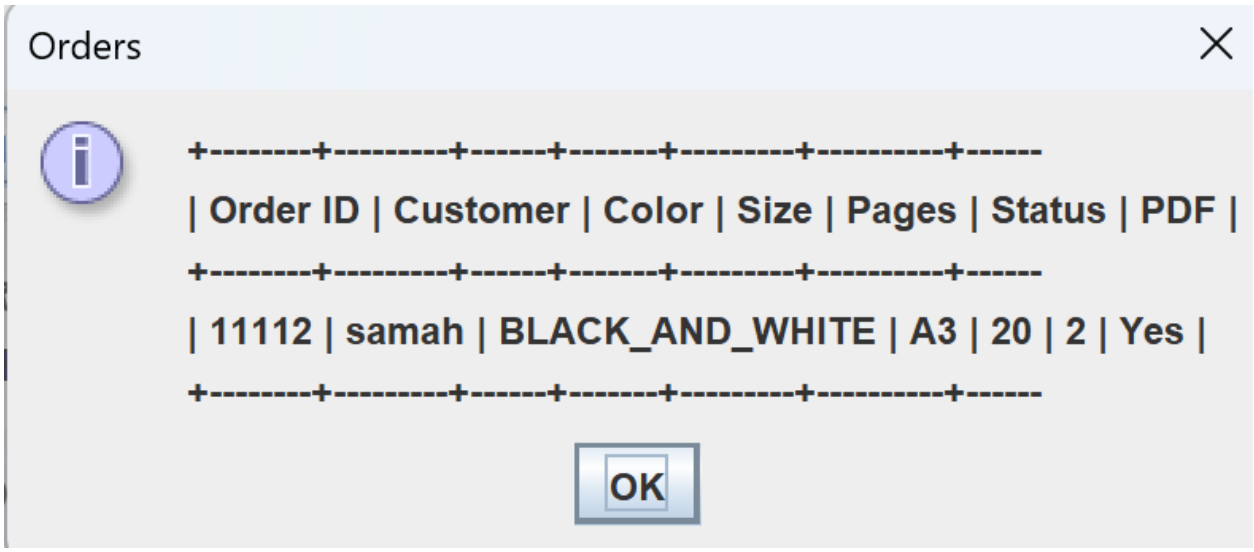


Shop Dashboard





Served orders pending for collection



Object-Oriented Features Used

1. Polymorphism (Overriding and Overloading):

- Overriding:
  - `PdfDocument.download` and `WordDocument.download` override the abstract `download` method in `Document` to provide specific implementations for each document type.
- Overloading:
  - `PrintOptions` has two constructors: one with default values and one with custom values for print options.
  - `Customer` constructors are overloaded, just providing the name and mobile number as input acts are registration wherein the application provides the customer with a `customerId`, whereas, providing the `customerId` as input along with the name and phone number fetches the existing customer object and acts as customer login.

2. Collection Frameworks:

- HashMap:
  - `PaperSizePrices` and `ColorModePrices` use `HashMap` to map print options to prices for calculation.
  - Likely used elsewhere for storing orders, invoices, and customer information.

3. Generic Programming:

- TransactionHandler:
  - Declared as `TransactionHandler<T extends CustomizableDocument>` to handle any document type that extends `CustomizableDocument` generically.

4. Interfaces:

- Document:

    - Defines a contract for document functionality, allowing for different concrete implementations (`PdfDocument`, `WordDocument`) while ensuring common behavior.

5. Parent-Child Relationship (Inheritance):

- CustomizableDocument`extends`Document`:
    - Inherits general document properties and methods, adding file type handling.
- PdfDocument`and`WordDocument`extend`CustomizableDocument`:
    - Inherit common document functionality while specializing in specific document formats.

6. Delegation:

- CustomizableDocument`has a`PrintOptions` object:
    - Delegates print option handling to the `PrintOptions` class, promoting loose coupling and code reusability.

7. File Handling:

- Invoice.saveAsFile`:
    - Saves invoice information to a text file, demonstrating file I/O operations.

8. Exception Handling:

    - The code uses error handling for robustness.

9. Database Interaction:

    - The code likely interacts with a database (MongoDB) for data persistence, though the specific implementation isn't provided.

Inference and Future Extension

PrintPal demonstrates the effectiveness of object-oriented principles in streamlining and digitizing traditional print services. By leveraging features like inheritance, polymorphism, and delegation, the platform offers a customizable and efficient experience for both students and shopkeepers. The potential for future extensions, such as payment integration and multi-shop support, signifies PrintPal's adaptability and scalability, paving the way for a revolutionary transformation in the print industry.

Potential for future extension:

1. Payment Integration and Verification:

- Integrate with online payment gateways (PayPal, Stripe) to enable secure payment processing within the app.
- Implement robust verification mechanisms to prevent fraudulent transactions and ensure payment accuracy.

2. Multi-Shop Support:

- Facilitate the onboarding of multiple print shops with distinct profiles and dashboards.
- Allow customers to select their preferred shop for order placement.
- Implement a system for managing shop-specific settings, orders, and inventory.

3. Expanded File Type Support:

- Add support for additional document types beyond PDFs and Word documents (e.g., PowerPoint, Excel, images).
- Implement implicit conversion to PDF for non-PDF file types to ensure compatibility with printing systems.

4. Direct Printing Capabilities:

- Explore integration with printer drivers or cloud-based printing services to enable direct printing from the app.
- This would streamline the process and eliminate the need for manual collection at the shop.

5. Notifications:

- Send notifications to customers when orders are ready for collection or when payment is successful.

6. Order History and Analytics:

- Provide users with a history of their past orders, including details on documents, print options, and costs.
- Offer shopkeepers insights into their sales data, popular print options, and customer behavior.