School Name

LOGO

Java Project on

GST Billing System

Submitted by -	Under the guidance of -
Student Name	Teacher Name
Class – X	
Roll No –	

CERTIFICATE

This is to certify that the project developed by **xxxxxxxxxxxxxx** of class **X –** entitled **GST Billing System**, is bonafide work of the student in the Computer Science Lab during the academic year 2023-24 is carried out with the consultation of Teacher Incharge.

______________ ______________

Examiner's Signature **Teacher Name**

(Teacher Incharge)

ACKNOLEDGEMENT

It is with great pleasure that I find myself penning down these lines to express my sincere thanks to various people who helped me a long way in completing this project.

The harmonious climate in our school provided proper guide for preparing the project. It was a privilege to have been guided by our computer teacher.

Thanks to all my classmates who helped me during development of this project with their constructive criticism and advice.

CONTENTS

OVERVIEW

The GST Billing System is a robust Java-based application designed for businesses to efficiently calculate, store, and manage invoices, taking into account the Goods and Services Tax (GST). It allows businesses to create and organize invoices, apply tax rates, and generate detailed bills for customers.

Features provided by this project are as follows:

1. **Product Management**:
   - Add, edit, or remove products from the system.
   - Specify product details such as name, description, price, and GST rate.
2. **Customer Management**:
   - Maintain a customer database for easy billing.
   - Store customer details like name, contact information, and GSTIN (if applicable).
3. **Invoice Generation**:
   - Create invoices by adding products, specifying quantities, and applying GST.
   - Automatically calculate subtotals, taxes, and total amounts.
   - Generate unique invoice numbers and date-stamp each invoice.

INTRODUCTION TO JAVA

Java is a versatile, object-oriented programming language celebrated for its platform independence, robustness, and extensive community support.

It enables developers to write code once and run it on various platforms using the Java Virtual Machine (JVM).

With strong typing, automatic memory management, and a rich standard library, Java simplifies development.

Its wide-ranging applications include web and mobile app development (Android), enterprise software, and embedded systems. Java's enduring popularity stems from its security features, large ecosystem of tools and libraries, and applicability in diverse domains, making it a preferred choice for both beginners and professionals in the software development field.

FEATURES OF JAVA PROGRAM

Platform Independence: Java programs can run on any platform or operating system with a Java Virtual Machine (JVM). This is achieved through the "Write Once, Run Anywhere" principle.

Object-Oriented: Java is primarily an object-oriented programming language, which promotes the use of objects and classes for modular and reusable code.

Strongly Typed: Java enforces strong type checking, which helps catch errors at compile-time rather than runtime, making it a safer language.

Automatic Memory Management: Java provides automatic memory management through garbage collection, freeing developers from managing memory explicitly.

Multi-threading: Java has built-in support for multithreading, making it easier to create concurrent and parallel applications.

Robustness: Java includes features like exception handling and type checking, which contribute to the robustness of programs and reduce the likelihood of crashes.

Security: Java has a strong security model, with features like a classloader, bytecode verification, and a security manager to protect against malicious code.

Rich Standard Library: Java comes with a comprehensive standard library, including packages for networking, I/O, data structures, and more, which simplifies development.

Architecture-Neutral: Java's architecture-neutral bytecode can be executed on various platforms, as long as there's a compatible JVM.

High Performance: While Java was initially criticized for being slower than natively compiled languages, modern JVMs and Just-In-Time (JIT) compilation have significantly improved its performance.

Dynamic: Java supports dynamic loading of classes and dynamic compilation, allowing for more flexibility in applications.

Community Support: Java has a large and active developer community with a wealth of resources, frameworks, and libraries available.

Portability: Java applications can be easily ported from one system to another as long as a compatible JVM is available.

Multithreading: Java's multithreading capabilities enable the concurrent execution of multiple threads within a single program, improving efficiency in tasks that benefit from parallel processing.

Rich Ecosystem: Java has a vast ecosystem of tools, frameworks, and third-party libraries, making it suitable for a wide range of applications, from web development to mobile apps to enterprise solutions.

Networking: Java provides libraries for network programming, making it a popular choice for developing networked and distributed applications.

Garbage Collection: Automatic garbage collection simplifies memory management, helping to prevent memory leaks and other memory-related issues.

Open-Source: Many Java implementations, like OpenJDK, are open-source, allowing developers to contribute to and modify the language.

Community and Support: Java has a large and active developer community, which means you can find plenty of resources, documentation, and support online.

Backward Compatibility: Java has a strong commitment to backward compatibility, which means older Java code can often run on newer JVMs without modification.

OOP CONCEPTS IN JAVA

Java is an object-oriented programming (OOP) language, and it incorporates several key OOP concepts. Understanding these concepts is fundamental to writing effective Java programs. Here are the main OOP concepts in Java:

1. **Class**: A class is a blueprint or template for creating objects. It defines the attributes (data members) and behaviors (methods) that objects of that class will have.
2. **Object**: An object is an instance of a class. It represents a real-world entity with attributes and behaviors defined by the class.
3. **Encapsulation**: Encapsulation is the concept of bundling the data (attributes) and methods (functions) that operate on the data into a single unit called a class. Access to the internal state of an object is controlled through access modifiers like public, private, and protected.
4. **Inheritance**: Inheritance is a mechanism that allows one class (subclass or derived class) to inherit the properties and behaviors of another class (superclass or base class). This promotes code reuse and the creation of a hierarchy of classes.
5. **Polymorphism**: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables the same method to have different implementations in different classes, typically through method overriding and interfaces.

6. **Abstraction**: Abstraction is the process of simplifying complex reality by modeling classes based on the essential properties and behaviors while hiding unnecessary details. Abstract classes and interfaces are used to achieve abstraction in Java.

7. **Method Overriding**: Method overriding is a feature that allows a subclass to provide a specific implementation of a method that is already defined in its superclass. It enables the invocation of the subclass's method when an object of the subclass is used.

8. **Polymorphic References**: Java allows you to declare a reference variable of a superclass or interface type and use it to refer to objects of subclasses. This is an important aspect of polymorphism.

9. **Interfaces**: Interfaces in Java define a contract specifying a set of methods that a class implementing the interface must provide. Multiple inheritance-like behavior can be achieved through interfaces.

10. **Abstract Classes**: Abstract classes are classes that cannot be instantiated themselves. They are meant to be subclassed, and they often contain abstract methods (methods without implementation) that must be implemented by concrete subclasses.

11. **Constructors**: Constructors are special methods used for initializing objects when they are created. They have the same name as the class and do not have a return type.

12. **Access Modifiers**: Access modifiers (public, private, protected, and default) control the visibility and accessibility of classes, fields, and methods in different parts of a program.

13. **This Keyword**: The **this** keyword refers to the current instance of a class. It is often used to disambiguate between instance variables and method parameters with the same name.

14. **Super Keyword**: The **super** keyword is used to access the superclass's members (fields and methods) and can be used in constructors to call a superclass constructor.

15. **Instance and Static Members**: Java allows you to define both instance members (associated with objects) and static members (associated with the class itself).

CLASSES USED

1. Product Class

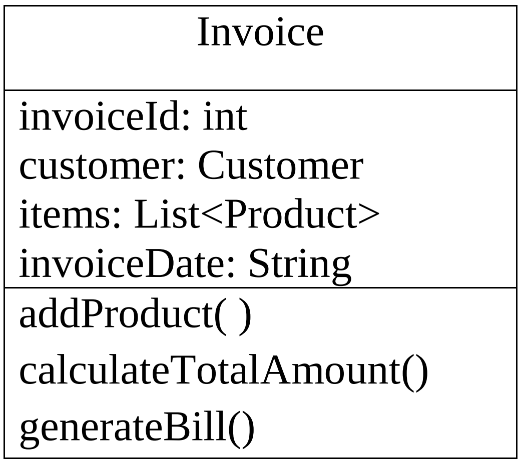
Product
productId: int name : String description : String price : double gstRate : double quantity : int

getAllProducts( )
updateProduct( )
deleteProduct( )
calculateTotalPrice( )

2. Customer Class

Product
customerId :int name:String contactInfo:String gstNumber : String
getAllCustomers ( ) updateCustomers ( ) deleteCustomers ( )

3. Invoice Class



SOFTWARE AND HARDWARE REQUIREMENTS

SOFTWARE REQUIREMENTS:

Platform: windows 10/11.

Language: JDK 1.8 above

IDE : BlueJ

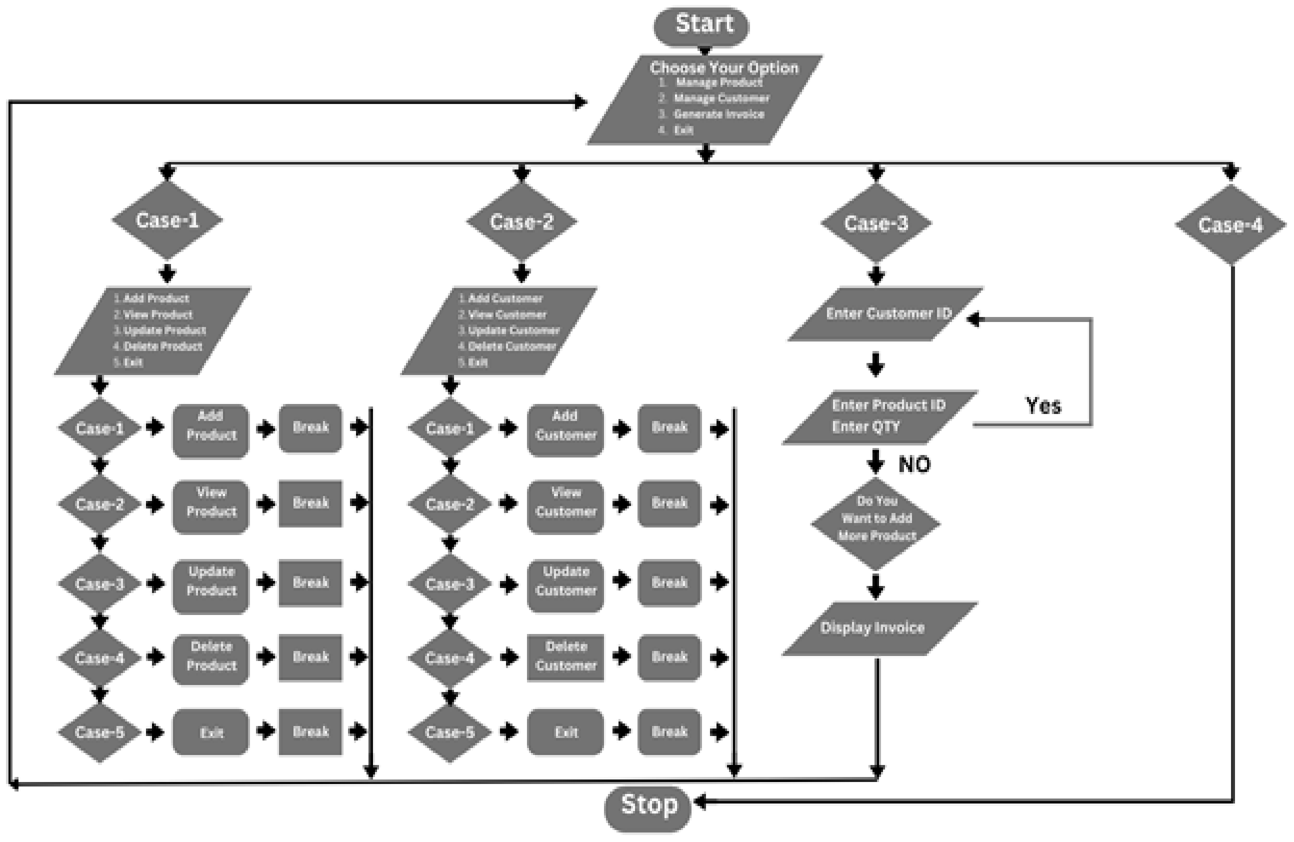
HARDWARE REQUIREMENTS:

Processor (CPU): Pentium Core 5 or Core 7

Memory (RAM) : 4 GB Minimum

Storage (Hard Drive or SSD): 250 GB minimum

FLOWCHART


Start
Choose Your Option
1. Manage Product
2. Manage Customer
3. Generate Invoice
4. Exit
Case-1
Case-2
Case-3
Case-4
1. Add Product
2. View Product
3. Update Product
4. Delete Product
5. Exit
1. Add Customer
2. View Customer
3. Update Customer
4. Delete Customer
5. Exit
Case-1
Add Product
Break
Case-2
View Product
Break
Case-3
Update Product
Break
Case-4
Delete Product
Break
Case-5
Exit
Break
Case-1
Add Customer
Break
Case-2
View Customer
Break
Case-3
Update Customer
Break
Case-4
Delete Customer
Break
Case-5
Exit
Break
Enter Customer ID
Enter Product ID
Enter QTY
Yes
NO
Do You Want to Add More Product
Display Invoice
Stop

SOURCE CODE

Product.java

```
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;

public class Product {
	private int productId;
	private  String name;
	private  String description;
	private  double price;
	private  double gstRate;
	private  int quantity;

    public int getQuantity() {
		return quantity;
	}

	public void setQuantity(int quantity) {
		this.quantity = quantity;
	}

	public int getProductId() {
		return productId;
	}

	public void setProductId(int productId) {
		this.productId = productId;
	}

	public String getDescription() {
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public double getPrice() {
		return price;
	}

	public void setPrice(double price) {
		this.price = price;
	}

	public double getGstRate() {
		return gstRate;
	}

	public void setGstRate(double gstRate) {
		this.gstRate = gstRate;
	}
```

```
        public void setName(String name) {
               this.name = name;
        }

        public String getName() {
          return name;
     }

        //Constructor
        public Product() {};
     public Product(int productId, String name, String description, double price,
double gstRate) {
          this.productId = productId;
          this.name = name;
          this.description = description;
          this.price = price;
          this.gstRate = gstRate;
     }

     public double calculateTotalPrice(int quantity) {
          return price * quantity * (1 + gstRate / 100);
     }

     public void getAllProducts(List<Product> products){
        System.out.println("ProductID | Product Name | Description | Price|Rate");
        System.out.println ("-------------------------------------------------");
        for(Product prod : products) {

        System.out.println(prod.productId+"|"+prod.name+"|"+prod.description+"|"+p
rod.price+"|" +prod.gstRate);

        }
        System.out.println("-------------------------------------------------");
     }

     public List<Product> updateProduct(int productID,List<Product> products){
        for(Product prod : products) {
               if(prod.getProductId() == productID) {
                      Scanner sc = new Scanner(System.in);
                      System.out.println("Update Product Name :");
                      prod.setName(sc.nextLine());
                      System.out.println("Update Product Description :");
                      prod.setDescription(sc.nextLine());
                      System.out.println("Update Product Price :");
                      prod.setPrice(sc.nextFloat());
                      System.out.println("Update Product GSTRate :");
                      prod.setGstRate(sc.nextFloat());
                      System.out.println("Product Updated Successfully !!!");
               }
        }

        return products;
     }
     public List<Product> deleteProduct(int productID,List<Product> products){

          for(Iterator<Product> itr = products.iterator();itr.hasNext();){
                      Product prod =itr.next();
                             if(prod.getProductId() == productID){
                             itr.remove(); // right call
```

```
                        System.out.println("Product Deleted Successfully !!!");
                    }
                }

            return products;
        }

    }
```

Customer.java

```
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;

public class Customer {
	private int customerId;
	private String name;
	private String contactInfo;
	private String gstNumber;
	 public int getCustomerId() {
		return customerId;
	}

	public void setCustomerId(int customerId) {
		this.customerId = customerId;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getContactInfo() {
		return contactInfo;
	}

	public void setContactInfo(String contactInfo) {
		this.contactInfo = contactInfo;
	}

	public String getGstNumber() {
		return gstNumber;
	}

	public void setGstNumber(String gstNumber) {
		this.gstNumber = gstNumber;
	}

    public Customer() {}
```

```java
        public Customer(int customerId, String name, String contactInfo, String
gstNumber) {
         this.customerId = customerId;
         this.name = name;
         this.contactInfo = contactInfo;
         this.gstNumber = gstNumber;
     }

     public void getAllCustomers(List<Customer> customers){

        System.out.println("customerId | Name | ContactInfo | GST Number ");
        System.out.println("---------------------------------------------");
        for(Customer customer : customers) {

               System.out.println(customer.customerId+" | "+customer.name+" |
"+customer.contactInfo+" | "+customer.gstNumber);

        }
        System.out.println("----------------------------------------------");
     }

     public List<Customer> updateCustomer(int customerId,List<Customer>
customers){

        for(Customer customer : customers) {
               if(customer.getCustomerId() == customerId) {
                      Scanner sc = new Scanner(System.in);
                      System.out.println("Update Customer Name :");
                      customer.setName(sc.nextLine());
                      System.out.println("Update Customer ContactInfo :");
                      customer.setContactInfo(sc.nextLine());
                      System.out.println("Update GST Number :");
                      customer.setGstNumber(sc.nextLine());
                      System.out.println("Customer Record Updated Successfully
!!!");
               }
        }

        return customers;
     }

     public List<Customer> deleteCustomer(int customerID,List<Customer>
customers){

        for(Iterator<Customer> itr = customers.iterator();itr.hasNext();){
               Customer customer =itr.next();
                      if(customer.getCustomerId() == customerId){
                      itr.remove(); // right call
                      System.out.println("Customer Deleteded Successfully !!!");
                 }
               }

               return customers;
     }
}
```

Invoice.java

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

public class Invoice {
    int invoiceId;
    Customer customer;
    List<Product> items;
    String invoiceDate;

   public Invoice(int invoiceId, Customer customer, String invoiceDate) {
       this.invoiceId = invoiceId;
       this.customer = customer;
       this.invoiceDate = invoiceDate;
       this.items = new ArrayList<>();
   }

   public void addProduct(Product product) {
       // Add a product and quantity to the invoice
       items.add(product);
   }

   public double calculateTotalAmount() {
       double totalAmount = 0;
       for (Product product : items) {
           totalAmount += product.calculateTotalPrice(product.getQuantity());
       }
       return totalAmount;
   }
   public void generateBill(List<Product> products) {
       System.out.println("Invoice:");

System.out.println("ProductId\\t\\tProductName\t\tPrice\tQuantity\tGST\tTotal");
       System.out.println("--------------------------------------------------");
       for (Product product : products) {

System.out.println(product.getProductId()+"\t\t"+product.getName()+"\t\t"+product
.getPrice()+"\t\t"+product.getQuantity()+"\t\t"+product.getGstRate()+"\t\t"+produ
ct.calculateTotalPrice(product.getQuantity()));
       }
       System.out.println("---------------------------------------------------");
       System.out.println("Total Amount: $" + calculateTotalAmount());
   }
}
```

GSTBillingSystem.java

```
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;
import java.util.Date;
public class GSTBillingSystem {
         static Scanner scanner = new Scanner(System.in);
      static List<Product> products = new ArrayList<>();
      static List<Customer> customers = new ArrayList<>();
      static List<Invoice> invoices = new ArrayList<>();
      static int productCounter = 1;
      static int customerCounter = 1;
      static int invoiceCounter = 1;

    public static void main(String[] args) {

        while (true) {
            // Menu for user actions (e.g., add products, customers, create
invoices)
             displayMenu();
            int option = scanner.nextInt();
            scanner.nextLine(); // Consume the newline character
            if (option == 10) {
                break;
            }
            switch (option) {
                case 1:// Add Product
                    addProduct();
                    break;
                case 2: //View Product
                    System.out.println("View Products");
                    new Product().getAllProducts(products);
                    break;
                case 3://Update Product
                    System.out.println("Enter ProductID to Update :");

                    int pid =scanner.nextInt();
                    products=new Product().updateProduct(pid, products);
                   break;
                case 4://Delete Product

                    System.out.println("Enter ProductID to Delete :");
                    int pid1 = scanner.nextInt();
                    products=new Product().deleteProduct(pid1,products);
                    break;
                case 5://Add Customer
                    addCustomer();
                    break;

                case 6://View Customer
                    new Customer().getAllCustomers(customers);
                    break;
                case 7://Update Customer
                    System.out.println("Enter CustomerID to Update :");
                    int cid =scanner.nextInt();
```

```
                        customers=new Customer().updateCustomer(cid, customers);
                        break;
                    case 8 ://Delete Customer
                        System.out.println("Enter CustomerID to Delete :");
                        int cid1 = scanner.nextInt();
                        customers=new Customer().deleteCustomer(cid1,customers);
                        break;

                    case 9:
                        if (products.isEmpty() || customers.isEmpty())
                        {
                            System.out.println("Please add products and customers 
before creating an invoice.");
                            break;
                        }

                        System.out.print("Select a customer by entering the customer 
ID: ");
                        int selectedCustomerId = scanner.nextInt();
                        scanner.nextLine(); // Consume the newline character

                        Customer selectedCustomer = null;
                        for (Customer customer : customers) {
                            if (customer.getCustomerId() == selectedCustomerId) {
                                selectedCustomer = customer;
                                break;
                            }
                        }
                        if (selectedCustomer == null) {
                            System.out.println("Customer not found.");
                            break;
                        }

                        Date date = new Date();
                        SimpleDateFormat formatter = new 
SimpleDateFormat("dd/MM/yy");
                        String str = formatter.format(date);
                        Invoice newInvoice = new Invoice(invoiceCounter, 
selectedCustomer,str);
                        invoiceCounter++;
                        String choice="";
                        do{
                            System.out.print("Select a product to add to the invoice 
by entering the product ID (or enter 0 to finish): ");
                            int selectedProductId = scanner.nextInt();
                            scanner.nextLine(); // Consume the newline character
                            Product selectedProduct = null;
                            for (Product product : products) {
                                if (product.getProductId() == selectedProductId) {
                                    selectedProduct = product;
                                    break;
                                }
                            }

                            if (selectedProduct != null) {
                                System.out.print("Enter quantity: ");
                                int quantity = scanner.nextInt();
                                scanner.nextLine(); // Consume the newline character
                                selectedProduct.setQuantity(quantity);
```

```
                              newInvoice.addProduct(selectedProduct);
                              System.out.println("Product added to the invoice.");
                          }else {
                            System.out.println("Product not Found !!");
                          }
                          System.out.println("do you want to add more
Product(Y/N):");
                          choice =scanner.nextLine() ;
                      }while(choice.equals("Y"));
                      invoices.add(newInvoice);
                      printInvoice();
                      break;
              }

          }


      }
      static void printInvoice() {
          // Display all invoices
          System.out.println("Invoices:");
          for (Invoice invoice : invoices) {
              System.out.println("Invoice ID: " + invoice.invoiceId);
              System.out.println("Customer: " + invoice.customer.getName());
              //System.out.println("Total Amount: $" +
invoice.calculateTotalAmount());
              System.out.println("-------------------------");
              invoice.generateBill(invoice.items);
          }
      }
      static void displayMenu() {
         System.out.println("------------------------------------------------------
---------");
         System.out.println("\t\tGST BILLING APPLICATION");
         System.out.println("------------------------------------------------------
---------\n");
         System.out.println("********************** MANAGE PRODUCT
**************************");
         System.out.println("1. Add Product");
          System.out.println("2. View Product");
          System.out.println("3. Update Product");
          System.out.println("4. Delete Product");
          System.out.println("********************** MANAGE CUSTOMER
*************************");
          System.out.println("5. Add Customer");
          System.out.println("6. View Customer");
          System.out.println("7. Update Customer");
          System.out.println("8. Delete Customer");
          System.out.println("**********************GENERATE
INVOICE**************************");
          System.out.println("9.Create Invoice");
          System.out.println("10.Exit");
          System.out.println("------------------------------------------------------
---------");
          System.out.print("Select an option: ");
      }
      static void addProduct() {
         System.out.print("Enter product name: ");
          String productName = scanner.nextLine();
```

```
        System.out.print("Enter product description: ");
        String productDescription = scanner.nextLine();
        System.out.print("Enter product price: ");
        double productPrice = scanner.nextDouble();
        System.out.print("Enter GST rate: ");
        double productGstRate = scanner.nextDouble();
        scanner.nextLine();
        Product newProduct = new Product(productCounter, productName, 
productDescription, productPrice, productGstRate);
        products.add(newProduct);
        productCounter++;
        System.out.println("Product added.");
    }
    static void addCustomer() {
       System.out.print("Enter customer name: ");
        String customerName = scanner.nextLine();
        System.out.print("Enter customer contact information: ");
        String customerContactInfo = scanner.nextLine();
        System.out.print("Enter customer GST number (if applicable): ");
        String customerGstNumber = scanner.nextLine();

        Customer newCustomer = new Customer(customerCounter, customerName, 
customerContactInfo, customerGstNumber);
        customers.add(newCustomer);
        customerCounter++;
        System.out.println("Customer added.");
    }

}
```

OUTPUTS

BLIOGRAPHY

- ✓ Javatpoint.com
- ✓ www.wikipedia.com

Teachers Report

Name of the student -	
Class -	X
Board Roll No. -	1
School -	
Topic -	GSTBilling System
Teachers Remark	