Adapter Pattern:

```java
public class DataAdapter {
    private Connection connection;

    public DataAdapter(Connection connection) {
        this.connection = connection;
    }

    public Product loadProduct(int id) {
        try {
            String query = "SELECT * FROM Product WHERE ProductID = " + id;

            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(query);
            if (resultSet.next()) {
                Product product = new Product();
                product.setProductID(resultSet.getInt("ProductID"));
                product.setName(resultSet.getString("Name"));
                product.setPrice(resultSet.getDouble("Price"));
                product.setCount(resultSet.getDouble("Quantity"));
                product.setVendor(resultSet.getString("Vendor"));
                product.setExpirationDate(resultSet.getDate("ExpirationDate"));
                resultSet.close();
                statement.close();

                return product;
            }

        } catch (SQLException e) {
            System.out.println("Database access error!");
            e.printStackTrace();
        }
        return null;
    }

    public List<Product> loadAllProducts() {
        try {
            double totalRevenue = 0;
```

An adapter pattern is used to adapt Java objects into SQL columns and rows to be entered into the database.

Proxy Pattern

First we create a interface for User objects

```
 */
package SMDUI;

/**
 *
 * Creates the interface for the User
 */
public interface UserInterface {
    int getUserID();
    void setUserID(int ID);
    String getUserName();
    void setUserName(String username);
    boolean isManager();
    void setIsManager(boolean isManager);
    String getPassword();
    void setPassword(String password);
}
```

Proxy Pattern

Then we implement the interface to our real User class

```java
public class User implements UserInterface {
    private int UserID;
    private String Name;
    private String Password;
    private boolean isManager;

    public User(int UserID, String Name, String Password, boolean isManager){
        this.UserID = UserID;
        this.Name = Name;
        this.Password = Password;
        this.isManager = isManager;
    }

    public User() {}

    @Override
    public int getUserID(){
        return UserID;
    }

    @Override
    public void setUserID(int ID){
        UserID = ID;
    }

    @Override
    public String getUserName(){
        return Name;
    }

    @Override
    public void setUserName(String name){
        Name = name;
    }
```

Proxy Pattern

The ProxyUser class also implements the UserInterface and has a real User object as a private variable. This way the User object is only created when it needs to be

```
 */
package SMDUI;

/**
 *
 * @author alecshunnarah
 */
public class ProxyUser implements UserInterface {
    private User user;
    private int UserID;
    private String Name;
    private String Password;
    private boolean isManager;

    public ProxyUser(int UserID, String Name, String Password, boolean isManager){
        this.UserID = UserID;
        this.Name = Name;
        this.Password = Password;
        this.isManager = isManager;
    }

    public ProxyUser() {}

    @Override
    public int getUserID(){
        if(user == null){
            user = new User();
        }
        return user.getUserID();
    }

    @Override
    public void setUserID(int ID){
        if(user == null){
            user = new User();
        }
```

Builder Pattern

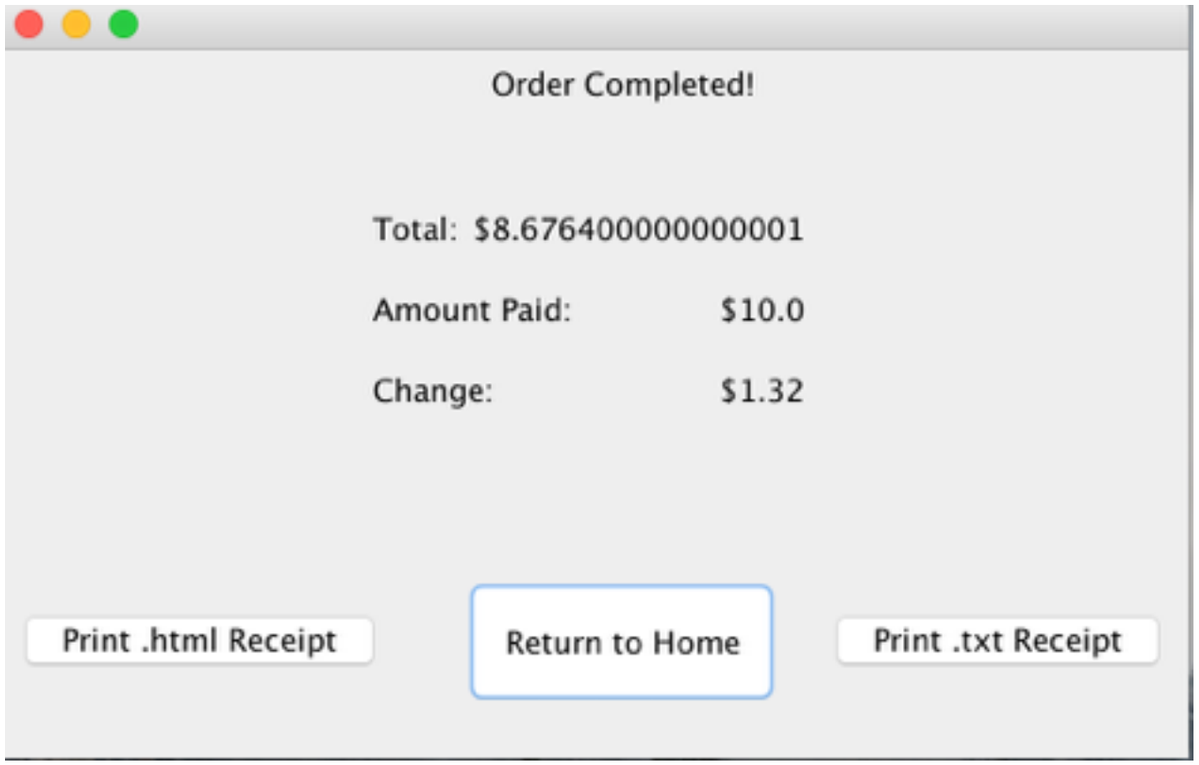
The builder pattern is used to create a method that prints a .txt receipt and an .html receipt

```
public class ReceiptBuilder {
    private double total;
    private double amtPaid;
    private double change;
    private String message = "Thank you for shopping with us today!";

    public ReceiptBuilder(double total, double amtPaid, double change, String message){
        this.total = total;
        this.amtPaid = amtPaid;
        this.change = change;
        this.message = message;
    }
```

```
public void CreateTxtReceipt() throws FileNotFoundException {
    PrintStream out = new PrintStream(new FileOutputStream("Store_Receipt.txt"));
    System.setOut(out);

    System.out.println("\tTotal: " + total);
    System.out.println("\tAmount Paid: " + amtPaid);
    System.out.println("\tChange: " + change);
    System.out.println("\n\t" + message);
}

public void CreateHTMLReceipt() throws FileNotFoundException {
    PrintStream out = new PrintStream(new FileOutputStream("Store_Receipt.html"));
    System.setOut(out);

    System.out.println("<p>Total: " + total + "</p>");
    System.out.println("<p>Amount Paid: " + amtPaid + "</p>");
    System.out.println("<p>Change: " + change + "</p>");
    System.out.println("<p>" + message + "</p>");
}
```

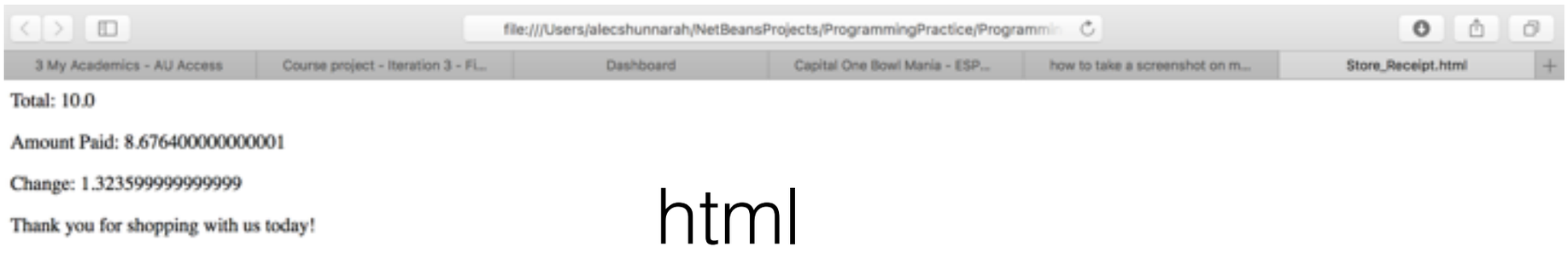
Builder Pattern

Clicking “Print .txt Receipt” will generate a .txt file with the same information on the checkout page, and “Print .html Receipt” will generate an .html file with the same content.



Builder Pattern

The two files output by the two buttons:



html

```
Total: 10.0
Amount Paid: 8.676400000000001
Change: 1.323599999999999

Thank you for shopping with us today!
```

.txt