Bloomify: Flower Shop Management System

Data Structures and Object-Oriented Programming

Lyna Selami

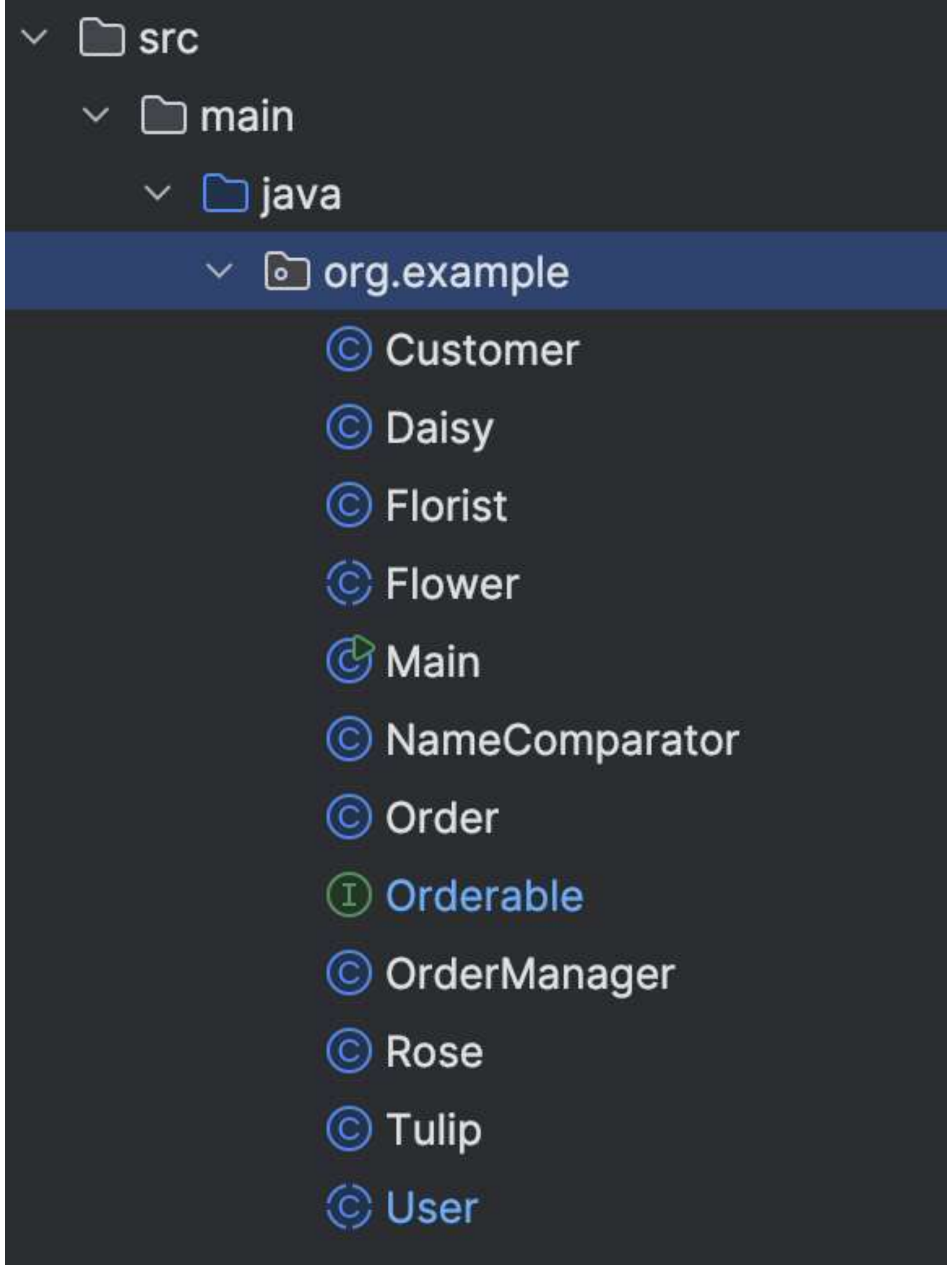
Table of Contents

Project Description

- Bloomify is a flower shop management system written in Java that demonstrates object-oriented programming concepts such as class hierarchies, interfaces, polymorphism, exception handling, and stream processing. Customers can browse, search, and filter flowers, while florists can manage inventory. Orders can be placed and saved to text files, and all functionality is verified using unit tests.

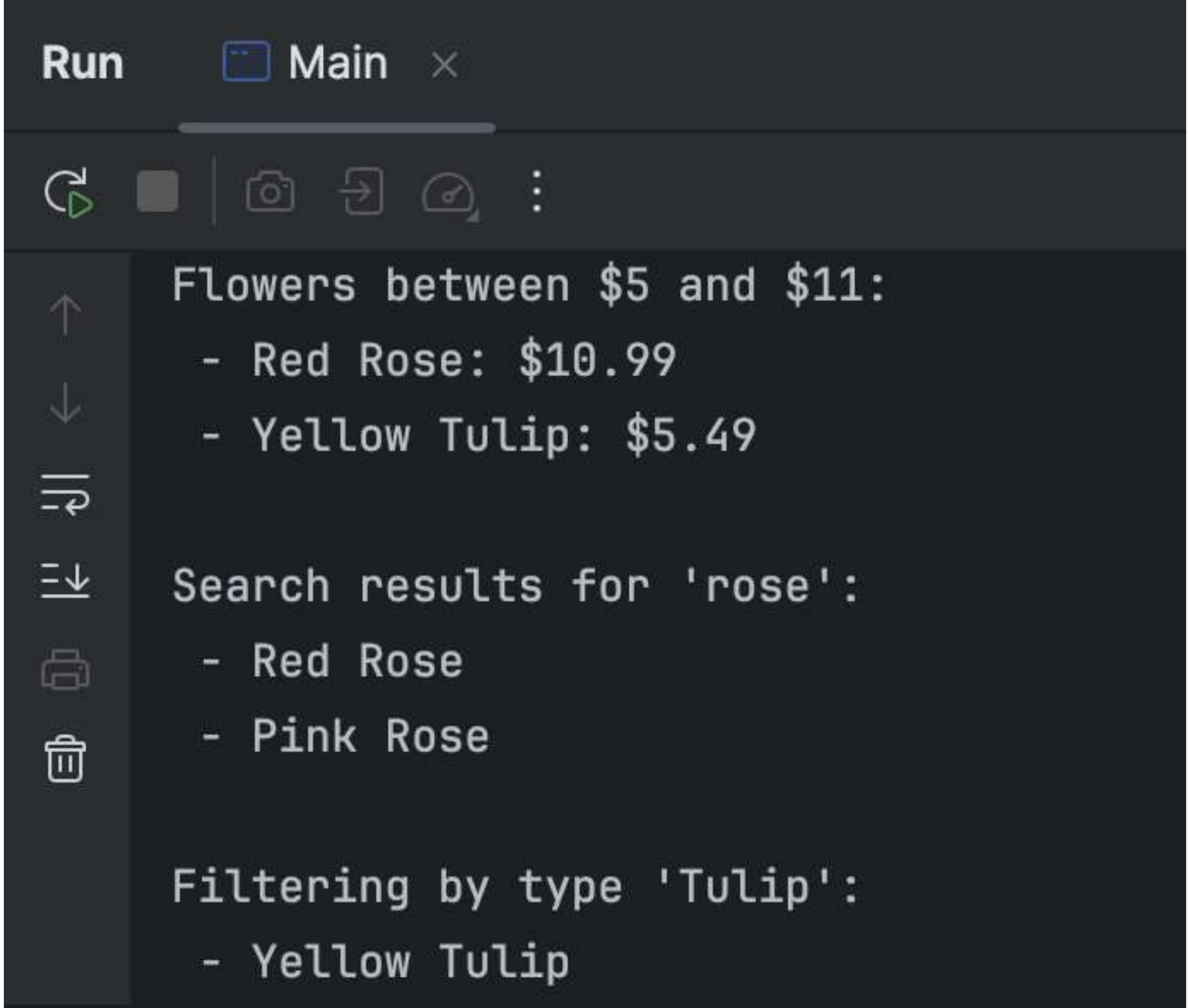
Program Features – Object-Oriented Design

- Bloomify uses two class hierarchies:
  - User → Customer / Florist
  - Flower → Rose, Tulip, Daisy
- The program includes:
  - One user-defined interface (Orderable)
  - One Comparable implementation (Flower)
  - One custom Comparator (NameComparator)
  - Method overriding via viewOptions()


src
main
java
org.example
Customer
Daisy
Florist
Flower
Main
NameComparator
Order
Orderable
OrderManager
Rose
Tulip
User

Program Features - Stream + Lambda Filters

- Can view flower options
- Add flowers to cart
- Place orders
- Cart totals are calculated and displayed
- Example:“Customer adds a Red Rose and a Daisy to their cart. The system displays the total: $14.24.”



```
Flowers between $5 and $11:
 - Red Rose: $10.99
 - Yellow Tulip: $5.49

Search results for 'rose':
 - Red Rose
 - Pink Rose

Filtering by type 'Tulip':
 - Yellow Tulip
```

Program Features – File Input/Output

- Can view/add/remove flowers
- Inventory can be saved to and loaded from inventory.txt
- Example:“Florist removes 'Yellow Tulip' and adds a 'Purple Orchid'. The updated inventory is saved to file.”

Program Features – Unit Testing

- I created a complete test suite using JUnit 5.
- Tests were written for the following classes:

  - FlowerTest: Tests getName(), getPrice(), and compareTo()

  - OrderTest: Tests addItem() and calculateTotal()

  - CustomerTest: Tests addToOrder() and getCart()

  - OrderManagerTest: Confirms addOrder() and loadInventory() run without errors

Execution Output Example

- The Main.java file demonstrates the core features:
  - A user views options (polymorphism)
  - Flowers are created and compared
  - Filters are applied
  - A customer places an order
  - The order is saved to file

Challenges

- Learning how to properly structure a multi-class Java project with Maven and Junit
- Managing file I/O while ensuring proper data formatting in text files
- Ensuring compatibility between test packages and the main application code
- Organizing realistic test cases and separating functionality into logical commits
- Keeping track of all project requirements and ensuring full feature coverage

Learning Outcomes

- I gained confidence in building a complete Java application from scratch
- I better understand inheritance, interfaces, abstract classes, and polymorphism
- I learned how to filter and search using streams and lambda expressions
- I now know how to write and run unit tests with Junit
- I feel more comfortable using Git, IntelliJ, and Maven for real projects