Tutor Marked Exercises 2, Questions, Design Document, Testing Plan

Unit 4,5 Learning Objective Questions

Meaning of a `package` Statement (TIJ Page 212)

The `package` statement in Java is used to organize classes, interfaces, and sub-packages into namespaces. It helps to:

1. Avoid naming conflicts by grouping related classes.
2. Manage large codebases by categorizing similar functionality.

Example:

```
package project2;
public class MyClass {
    // Class definition
}
```

Inner Classes (TIJ Pages 345-346)

An **inner class** is a class defined within another class. It is associated with an instance of the enclosing class and has access to its members.

Example:

```
class OuterClass {
    class InnerClasss {
        void display() {
            System.out.println("Innner class meathod");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        OuterClass.InnerClasss inner = new Outer().new Inner
();
        inner.display();
    }
}
```

TME 2: Test Plan and Design Document

Course: Computer Science 308 – Java for Programmers

Assignment: TME 2

Author: Boris Bojanov

Student ID: 3608550

Date: Jan 28 2025

Part 1: GenericOrder, Subclasses, and OrderProcessor (60%)

1. Overview

This part requires designing a generic container `GenericOrder`, with subclasses `ComputerOrder` and `PartyTrayOrder`, and an `OrderProcessor` class for managing and processing orders. A data generator and a client class are used for testing purposes.

---

2. Testing Plan

2.1 Testing Objectives

- Verify `GenericOrder` and its subclasses handle arbitrary numbers of objects.
- Validate `OrderProcessor` correctly processes, sorts, and dispatches orders.
- Confirm data generation works as expected.
- Test under normal, abnormal, and limiting conditions.

2.2 Compile & Run Instructions

1. **Compilation:**
   - `javac Products.java GenericOrder.java ComputerOrder.java PartyTrayOrder.java OrderProcessor.java ClientPart1.java`
2. **Execution:**
   - Run the client: `java ClientPart1`

2.3 Test Cases

Test Case 1: Adding Items to GenericOrder

- **Purpose:** Verify items are correctly added to `GenericOrder`.
- **Input:** Add three objects (e.g., `ComputerPart`, `Peripheral`, `Service`).
- **Expected Result:** All objects appear in the order, with unique identifiers.
- **Actual Result:** [Record Actual Result]

Test Case 2: Processing Orders in OrderProcessor

- **Purpose:** Test sorting and processing logic in `OrderProcessor`.
- **Input:** Create a `ComputerOrder` with two `ComputerPart` objects and one `Peripheral` object.

- **Expected Result:**
    - Objects are sorted by type.
    - Correct identifiers are associated.
- **Actual Result:** [Record Actual Result]

Test Case 3: Dispatching Sorted Orders

- **Purpose:** Verify `dispatchXXX` methods output correctly.
- **Input:** Dispatch collections of `ComputerPart` and `Peripheral`.
- **Expected Result:** Console output in specified format (e.g., `Motherboard - name=Asus, price=$37.5, order number=123456`).
- **Actual Result:** [Record Actual Result]

Test Case 4: Abnormal Data Handling

- **Purpose:** Test robustness with invalid inputs.
- **Input:**
    - Null object in `GenericOrder`.
    - Empty order passed to `OrderProcessor`.
- **Expected Result:**
    - Program handles gracefully with appropriate error messages.
- **Actual Result:** [Record Actual Result]

Test Case 5: Limiting Conditions

- **Purpose:** Ensure program handles edge cases (e.g., maximum order size).
- **Input:** Add 1000 objects to a `ComputerOrder`.
- **Expected Result:** Program processes efficiently without errors.
- **Actual Result:** [Record Actual Result]

---

3. Design Document

3.1 Classes and Methods

- **GenericOrder:**
  - Fields: `id`, `collection` (List).
  - Methods: `add(T item)`, `remove(T item)`, `getId()`.
- **ComputerOrder:**
  - Subclass of `GenericOrder` for `ComputerPart`, `Peripheral`, `Service`.
- **PartyTrayOrder:**
  - Subclass of `GenericOrder` for `Cheese`, `Fruit`, `Service`.
- **OrderProcessor:**
  - Methods: `accept(GenericOrder)`, `process()`, `dispatchXXX()`.
- **DataGenerator:**
  - Generates sample data for testing.

3.2 Algorithms

- **Order Sorting in `process`:**
  - Iterate through orders, categorize objects by type.
  - Use data structures (e.g., Map<String, List>).
- **Dispatch Logic:**
  - Iterate sorted collections, format output.

3.3 Enhancements

- Implement additional validation (e.g., duplicate IDs).
- Optimize sorting using streams.

Part 2: ComputerPartyOrder (35%)

1. Overview

This part extends `GenericOrder` to include `ComputerPartyOrder`, handling a mix of `ComputerPart`, `Peripheral`, `Cheese`, `Fruit`, and `Service`. It modifies `OrderProcessor` as necessary.

---

2. Testing Plan

2.1 Testing Objectives

- Verify `ComputerPartyOrder` handles mixed object types.
- Confirm `OrderProcessor` accommodates `ComputerPartyOrder`.

2.2 Compile & Run Instructions

1. **Compilation:**
    - `javac Products.java ComputerPartyOrder.java OrderProcessor.java ClientPart2.java`
2. **Execution:**
    - Run the client: `java ClientPart2`

2.3 Test Cases

Test Case 1: Adding Mixed Items to ComputerPartyOrder

- **Purpose:** Confirm mixed objects can coexist in `ComputerPartyOrder`.
- **Input:** Add `ComputerPart`, `Cheese`, `Fruit`, and `Service` objects.
- **Expected Result:** All objects are added with unique IDs.
- **Actual Result:** [Record Actual Result]

Test Case 2: Processing ComputerPartyOrder

- **Purpose:** Test `OrderProcessor` with mixed objects.
- **Input:** Create a `ComputerPartyOrder` with multiple object types.
- **Expected Result:** Objects sorted into respective collections.
- **Actual Result:** [Record Actual Result]

Test Case 3: Dispatching Mixed Collections

- **Purpose:** Verify dispatch methods handle all object types.
- **Input:** Dispatch collections from `ComputerPartyOrder`.
- **Expected Result:** Output lists objects grouped by type, correctly formatted.
- **Actual Result:** [Record Actual Result]

Test Case 4: Limiting Conditions

- **Purpose:** Ensure program handles edge cases.
- **Input:** Create an order with one object per type and verify processing.
- **Expected Result:** Correct processing and dispatch.
- **Actual Result:** [Record Actual Result]

---

3. Design Document

3.1 Classes and Methods

- **ComputerPartyOrder:**
  - Fields: Same as `GenericOrder`.
  - Methods: Extend as needed.
- **OrderProcessor Modifications:**
  - Adjust `accept` and `process` to handle mixed orders.

3.2 Enhancements

- Add validation for object type mismatches.
- Optimize dispatch methods for large collections.

NOTE:

The following link is not working properly!!!

“Please see testplan.html for a sample test plan.” (https://scis.lms.athabascau.ca/file.php/422/tme_files/guidelines.htm)

Results in a black page. DegreeWorks does not present me with a way to find the template of a test plan. I have sent Emails requesting an example test plan.

```
Click here to access DegreeWorks using myAU login
```