Practical 4

COS214

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Deadline: 09/09/2024 at 23:59

Marks: 120

1 General Instructions

- This assignment should be completed in pairs.
- Be ready to upload your practical well before the deadline, as no extensions will be granted.
- You can use any version of C++.
- You can import any library but it can't do the pattern for you:
- You will upload your code with your main to FitchFork as proof that you have a working system.
- **If you meet the minimum FitchFork requirements (working code with at least 60% testing coverage), you will be marked by tutors during your practical session. Please book in advance (Instructions will follow on ClickUp).**
- **You will not be allowed to demo if you do not meet the minimum FitchFork requirements.**

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's

work without consent, copying a friend's work (even with permission) and copying material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm. **If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.**

3 Mark Distribution

Activity	**Mark**
Task 1: UML Diagrams	30
Task 2: Implementation of Patterns	60
Task 3: FitchFork Testing Coverage	10
Task 4: Demo Preparation	20
Task 5: Bonus Marks	10
Total	**120**

Table 1: Mark Distribution

Farm Management System Practical Assignment

You are tasked with designing and implementing a simulation system for managing multiple farms. Each farm consists of several crop fields, each of which has a soil state that influences the crop yield. Your system must manage soil states, apply fertilizers, track crop growth, and manage logistics (e.g., trucks) based on certain triggers.

Scenario Overview:

Your software company is developing a smart farm management system. The system must:

- Track and update soil states for each crop field.
- Apply fertilizers to improve crop yield
- Manage logistics using trucks to deliver fertilizers and collect harvested crops based on crop thresholds.
- Build extra barns to increase storage capacity.
- Implement traversal strategies for iterating through farms and their crop fields using breadth-first or depth-first strategies.

System Components and Tasks:

Each section describes the minimum requirements to complete that task. Often you will need to implement additional functions in order for the pattern to perform its function.

Hint: Before implementing anything identify the patterns used and draw a UML class diagram of the complete system.

Component 1: Farmlands and Crop Fields

The farm system is composed of various farmland units, each containing multiple crop fields. Your task is to:

1. Create a hierarchical structure representing farmlands composed of multiple `FarmUnit` objects. A `FarmUnit` could be a `CropField`, `Barn`, or another farm-related structure.
2. Each `CropField` should store:
   - The type of crop (e.g., wheat, corn).
   - The total storage capacity and the current amount of stored crops.
   - The current soil state, which affects crop yield.
3. Implement the following methods:
   - `getTotalCapacity()`: Returns the total capacity of the crop field.
   - `getCropType()`: Returns the type of crop in the field.
   - `getSoilStateName()`: Returns the name of the current soil state.

Composite

Leaf: Barn, CropField
Composite: Farmland
Component: FarmUnit
Client: Farmland?

Hint: *Consider how to treat a group of fields and individual fields uniformly within your structure.*

Component 2: Soil States and Transitions

Each crop field's productivity depends on its soil state, which can change based on actions taken. Your task is to:

1. Define the different soil states for a crop field (you can add more): `DrySoil`, `FruitfulSoil`, and `FloodedSoil`.
2. Soil states should influence the crop yield:
   - `DrySoil`: Yields minimal (x1) crops.
   - `FruitfulSoil`: Yields x3 crops.
   - `FloodedSoil`: Prevents further crop growth.
3. Implement at least the following methods within the soil states:
   - `harvestCrops()`: Harvest crops based on the current soil state.
   - `rain()`: Simulate rainfall that may change the soil state.
   - `getName()`: Return the name of the soil state (e.g., "Dry", "Fruitful").

State

State: State
ConcreteState: Dry, etc
Context: CropField

Hint: *Think about how to encapsulate state-specific behavior in a modular way.*

Component 3: Fertilizer Application and Storage Enhancement

In cases where the soil state is `DrySoil`, fertilizers can be applied to make fields more productive. Additionally, extra barns can be added to crop fields or farms to expand their storage capacity. Your task is to:

1. Implement a mechanism that allows fertilizers to be applied to `CropFields` to improve their productivity by transitioning them from `DrySoil` to `FruitfulSoil`.
2. Implement a mechanism that allows an `ExtraBarn` to be added to a crop field to increase its storage capacity.
3. Both the fertilizer and extra barn should dynamically enhance functionality without changing the existing class structure.

Decorator
Component: CropField
Decorator: CropFieldDecorator
ConcreteDecorator: FertilizedField, ExtraBarn
ConcreteComponent: CropField

4. Implement the following methods:
   - `increaseProduction()`: Enhance crop yield after applying fertilizer.
   - `harvest()`: Harvest crops based on the enhanced soil state.
   - `getLeftoverCapacity()`: Return the remaining storage capacity after adding the extra barn.

Hint: *Consider how to dynamically add functionality or attributes to crop fields without modifying their class.*

Component 4: Truck Logistics and Notifications

Trucks play a crucial role in delivering fertilizers and collecting crops. Your task is to:

1. Implement the truck logistics system, which includes:
   - `FertilizerTruck`: Delivers fertilizer to a crop field when needed.
   - `DeliveryTruck`: Collects harvested crops when the storage capacity reaches a predefined threshold.
2. Trucks should be notified automatically when certain events occur:
   - When fertilization is needed. You can decide when this is necessary, eg. Soil is dry.
   - When a crop field's storage capacity is nearing its limit.

Observer

Subject: FarmUnit
ConcreteS: CropField
Observer: Truck
CObserve: FT, DT

3. Implement the following methods:
   - `buyTruck()`: Simulate a farm unit purchasing a truck.
   - `sellTruck()`: Farm unit sells an unused truck.
   - `callTruck()`: Notify and dispatch a truck for a specific operation from a FarmUnit.
   - `startEngine()`: Start the truck's operation.

Hint: *Consider how to create a notification system that automatically triggers truck operations.*

Component 5: Farm Traversal Strategies

To monitor and manage the farm, the system needs to iterate over farms and crop fields. Your task is to:

1. Implement a traversal mechanism for navigating through farms and crop fields using both:
   - **Breadth-First Traversal**: Prioritize farms based on proximity or surface area.
   - **Depth-First Traversal**: Handle deeper farm hierarchies.
2. These traversals will allow you to check crop conditions and trigger logistics operations when needed.
3. Implement the following methods:
   - `firstFarm()`: Return the first farm for traversal.
   - `next()`: Move to the next farm in the traversal.
   - `isDone()`: Check if traversal is complete.
   - `currentFarm()`: Resturns current farm

Iterator

Iterator: FarmIterator
CI: BFS, DFS
Aggregate: Farmland

CA: CropFields, Barn

Hint: *Think about how to create flexible iterators that work regardless of the underlying structure.*

4 Task 1: UML Diagrams (30 marks)

In this task, you need to construct UML Class, Object and State Diagram for the given scenario.

4.1 Class Diagram (10 marks)

- Create a single comprehensive UML class diagram that shows how the classes in the scenario interact and participate in various patterns.
- Ensure it includes all relevant classes, their attributes, and methods.
- Identify and indicate the design patterns each class is involved in, as well as their roles within these patterns. (Remember a class can participate in more than one pattern)
- Display all relationships between classes.
- Add any additional classes that are necessary for a complete representation.

4.2 State Diagram (10 marks)

- Draw a UML state diagram showing how the harvest can be effected by various events.

4.3 Sequence Diagram (10 marks)

- Draw a UML sequence diagram showing how the trucks are notified.

5 Task 2: Implementation (60 marks)

In this task, you need to implement the system components as described in the scenario.

- Use your UML class diagram as a guide to integrate the design patterns. Note that a single class can be part of multiple patterns.
- The provided scenario outlines the minimum requirements for this practical. **You may need to add extra classes/functions to complete the pattern.** Feel free to add any additional classes or functionalities to better demonstrate your understanding of the patterns and tasks.
- Thoroughly test your code (more details will be provided in the next task). Tutors will only mark code that runs.
- You may use arrays, vectors, or other relevant containers. Avoid using any libraries not mentioned in the general instructions, as your code must be compatible with FitchFork. If you really need a specific library please email u21689432@tuks.co.za at least a week before the due date.
- Tutors may require you to explain specific functions to ensure you understand the pattern being implemented.

6 Task 3: FitchFork Testing Coverage (10 marks)

In this task, you are required to upload your completed practical to FitchFork.

- Ensure that your code has at least **60% coverage** (6/10) in your testing main. This is necessary for demonstrating your work to the tutors.
- You will need to show your FitchFork submission with sufficient coverage to the tutor before being allowed to demo.
- You will be required to download your code from FitchFork for demonstration purposes.
- It might be useful to have two main files: a testing main for FitchFork and a demo main with a more user-friendly interface (e.g., a terminal menu) for demoing. This is just a suggestion, not a requirement. **You are required to have at least a testing main.** If you choose to have a demo main, upload it to FitchFork as well, but ensure it does not compile and run with `make run`.
- Name your testing main file `TestingMain.cpp`.
- If you create a demo main, name it `DemoMain.cpp` so it can be excluded from the coverage percentage. Note that you do not have to test your demo main.
- Additionally, upload a makefile that will compile and run your code with the command `make run`.

7 Task 4: Demo Preparation (20 marks)

You will have 5-7 minutes to demo your system and answer any questions from the tutors. Proper preparation is crucial.

- Ensure you have all relevant files open, such as UML diagrams and source code.
- Set up your environment so that you can easily run the code during the demo after downloading it.
- Be ready to demonstrate specific function implementations upon request.
- Practice your demo to make sure it fits within the allotted time and leaves time for questions.
- Prepare a brief overview (30 seconds) of your system to quickly explain its functionality and design.
- Make sure your testing and demo mains (if applicable) are ready to show their respective functionalities.
- Have a clear understanding of the design patterns used and be prepared to discuss how they are implemented in your code. Also, think about other potential use cases for the patterns.
- Be prepared to answer questions. If you do not know the answer, inform the marker and offer to return to the question later to avoid running out of time.

8 Task 5: Bonus Marks (10 marks)

Utilize both Github and Doxygen comprehensively to be awarded the full 10 bonus marks.

9 Submission Instructions

You will submit on both ClickUp and FitchFork.

- FitchFork submission:
  - Zip all files.
  - Ensure you are zipping the files and not the folder containing the files.
  - Upload to the appropriate slot on FitchFork well before the deadline, as no extensions will be granted.
  - **Ensure that you have at least 60% coverage.**
- ClickUp submission. You should submit the following in an archived folder:
  - Your UML diagrams (both as images and Visual Paradigm projects).
  - Your source code.