Lab 05: ADTs

CMPT 145

Laboratory 05 Overview

Part 1 : Pre-Lab Reading (Slide 4)
Part 2 : Laboratory Activities (Slide 16)
Hand In : Brief answers to reflection questions, and a transcript of your work (Slide 24)

Part I

Pre-Lab Reading

ADTs for Adaptability

ADTs enhance Adaptability

- Software always changes!
- ADTs help software designers manage change and new demands
- We'll see how to use ADTs to make future changes easier!

Example: The MM1 Queueing algorithm

- Did you ever wonder what would happen if the MM1 simulation used LIFO order instead of FIFO?
- Humans value fairness, and queues (FIFO) seem to be fair.
- Just how unfair would it be if customers were served in LIFO order?
- To answer, we could simply edit the MM1 program to use a Stack instead of a Queue.

Why naive editing is bad

- To answer the FIFO vs. LIFO question, we could simply edit the MM1 program.
- Changing a working program to have different behaviour is fine, as long as the old behaviour is no longer needed.
- In this experiment we want both behaviours:
    - MM1 with a FIFO queue (standard).
    - MM1 with a LIFO stack (experimental).
- Editing back and forth is a waste of programmers' time!

Why copying is bad

- To answer the FIFO vs. LIFO question, we could simply copy the MM1 program, and change the copy.
- Suppose there are errors you didn't notice before you copied.
  - Copying the program copies the bugs!
  - Twice as much code to fix!
- Suppose we want to add code to the MM1 program, say, to collect more data about wait times.
  - Copying the program forces us to modify both copies the same way.
  - Takes twice as long to make the changes.
- Having two copies of the same program means that you have twice as much code.

Changing the ADT operations

- We could edit the Queue ADT and change the code:
  - Make enqueue behave like Stack's push
  - Make dequeue behave like Stack's pop
- The would affect every program that already uses your Queue ADT!
- You'd have to change it back some time!

List of bad ideas for adaptable software

- Copying code is a bad idea.
  - More code means more errors, and more time debugging.
- Editing code repeatedly is a bad idea.
  - Wastes programmer time!
- Modifying an established ADT is a bad idea.
  - Changes the behaviour of every working application that uses it.

Abstraction to the rescue!

- Stacks and Queues are similar, but not exactly the same.
- The similarity can be expressed by a Container ADT with the following operations.
    - Create a container
    - Add a value to the container
    - Remove a value from the container
    - Check the container's size, if it's empty
    - Peek at the upcoming value without removing it
- The Container ADT generalizes Stack and Queue.
- We'll create 2 different implementations of this ADT.

Creating an adapter ADT

- Here's one of the implementations: ContainerQ

```
# CMPT 145: ContainerQ
# Simple adapter for Queues
# documentation removed to conserve space

import Queue as Queue

def create():
    return Queue.create()
def add(container,value):
    Queue.enqueue(container,value)
def remove(container):
    return Queue.dequeue(container)
def is_empty(container):
    return Queue.is_empty(container)
def size(container):
    return Queue.size(container)
def peek(container):
    return Queue.peek(container)
```

ContainerQ is an Adapter

- The ContainerQ operations call the Queue operations.
    - It does nothing else.
    - Makes the container behave like a FIFO Queue.
- We can edit the MM1 application so that it imports the Container ADT instead of the Queue ADT.
    - The code has changed, but the behaviour has not.
- Key idea: We can also create a ContainerS ADT, an adaptor for the Stack ADT.
    - Makes the container behave like a LIFO Stack.
- From the outside, ContainerQ and ContainerS look the same

Abstraction is the way

- When you need LIFO, import the Stack ADT.
- When you need FIFO, import the Queue ADT.
- When you need to swap between FIFO or LIFO, import a Container ADT.
- Swapping Queues and Stacks is now easy!
    - Just change the `import` line!
- Creating the Container ADTs is an investment of time and effort, but it pays off in programmer time saved later.

Part II

Laboratory Activities

Activity Step 1: Preparation

- Download the following files from the Laboratory.
  - `MM1.py`
  - `Queue.py`
  - `Statistics.py`
  - `ContainerQ.py`
- Create a new project (named Lab05), and add these files to it.
- Open a text editor with an empty file. This will contain your transcript for the lab.

Activity Step 2: Preparation

- Run `MM1.py` to be sure it's working. Use the following inputs:

`arrival_rate`:	1.9
`service_rate`:	2.0
`sim_length`:	100000

- We'll use those same settings for all of our experiments in today's lab.
- Run it a few times, and make note of the average values reported.
- Copy/paste the output to the transcript file.
    - Give the pasted output a heading like "Before"

ACTIVITY Step 3: Adapting ADTs

1. Open the file `ContainerQ.py`.
   - Its functions simply call the Queue ADT
   - `ContainerQ.py` adapts the Queue ADT.
2. Modify `MM1.py` so that it uses `ContainerQ.py`.
   - Hint: Be very careful to change every line in `MM1.py` that mentions the Queue ADT specifically!
   - Hint: `import ContainerQ as Container`
3. Run `MM1.py` to be sure it's (still) working.
   - Use the same inputs as before, and be sure that it gives more or less the same output as before!
4. Copy/paste the output from these runs to the transcript file.
   - Give the pasted output a heading like "After editing for ContainerQ"

Activity Step 4: A new container

1. Make a copy of `ContainerQ.py`, call it `ContainerS.py`
2. Modify all the functions in `ContainerS.py` so that it is an adapter for `Stack.py`
3. You'll need to change the import, and all 6 operations!

ACTIVITY Step 5: Changing the simulation

- Modify `MM1.py` so that it uses `ContainerS.py`.
  - Hint: You should only need to change the import line!
  - Hint: `import ContainerS as Container`
- Run `MM1.py` a few times, and make note of the average values reported.
- Did the average change a lot from Step 2?
- Copy/paste the output from these runs to the transcript file.
  - Give the pasted output a heading like “After editing for ContainerS”

ACTIVITY Step 6: Experimenting

- Run `MM1.py` a few times using `ContainerS.py`. Make note of the Statistics report.
  - Hint: `import ContainerS as Container`
- Run `MM1.py` a few times using `ContainerQ.py`. Make note of the Statistics report.
  - Hint: `import ContainerQ as Container`
- Copy/paste the output from these runs to the transcript file.
  - Give the pasted output a heading like “Some experimental output”

Activity Step 7: Reflection

- Answer the following questions with a sentence or two, and put your responses in your `lab05-transcript.txt` file, near the top, just after your name/student number, etc.
  1. Does the average waiting time increase when you use LIFO instead of FIFO in the MM1 simulation?
  2. Does the maximum waiting time increase when you use LIFO instead of FIFO in the MM1 simulation?
  3. Now that you have both ContainerS and ContainerQ, how hard is it to switch between them in MM1?
- Give your answers a heading like “Reflections”

Part III

Hand In

What To Hand In

Hand in your `lab05-transcript.txt` file showing:

- The answers to Reflection questions (Step 7). This should be near the beginning of the document so that it's easier to grade.
- The results of running the `MM1.py` program in Steps 1-6.