COMP3506 Assignment 1 Rundown

COMP3506 Tutoring Team

August 7, 2018

Semester 2, 2018

The Brief

Air traffic management simulation - based on the OneSKY Australia project

The Brief

Air traffic management simulation - based on the OneSKY Australia project

Meant to be able to represent the entire Australian airspace

- For simplicity, we will say the Australian airspace is a 5321x3428x35km cube
- Each cell in the cube will represent one cubic kilometer
- Each cell may contain more than one aircraft
- The airspace will contain a maximum of 20,000 aircraft at one time
- Simulation should make efficient use of memory and be able to be run on a computer with 8GB of memory

The Brief

Air traffic management simulation - based on the OneSKY Australia project

Meant to be able to represent the entire Australian airspace

- For simplicity, we will say the Australian airspace is a 5321x3428x35km cube
- Each cell in the cube will represent one cubic kilometer
- Each cell may contain more than one aircraft
- The airspace will contain a maximum of 20,000 aircraft at one time
- Simulation should make efficient use of memory and be able to be run on a computer with 8GB of memory

Keep these points in mind when designing your implementation

The Task

- Two data structures
    - An IterableQueue (with an iterator)
    - A Cube
- JUnit tests for each of the data structures
- Analysis of the CDTs and methods you write

IterableQueue ADT & TraversableQueue CDT

A queue data structure (first in, first out) that can be iterated over

IterableQueue ADT & TraversableQueue CDT

A queue data structure (first in, first out) that can be iterated over

IterableQueue extends the java.lang.Iterable interface

IterableQueue ADT & TraversableQueue CDT

A queue data structure (first in, first out) that can be iterated over

IterableQueue extends the java.lang.Iterable interface

TraversableQueue must implement the iterator() method that returns an iterator for the queue

TraversableQueue Iterator

Iterator must implement the java.util.Iterator interface (including the next() and hasNext() methods)

TraversableQueue Iterator

Iterator must implement the java.util.Iterator interface (including the next() and hasNext() methods)

Iterator should be able to access each element of the queue in order

TraversableQueue Iterator

Iterator must implement the java.util.Iterator interface (including the next() and hasNext() methods)

Iterator should be able to access each element of the queue in order

Possible implementation of the iterator: make a private subclass inside the TraversableQueue class

IterableQueue ADT Methods

- `enqueue(T element);`

  add a new element to the end of the queue

- `T dequeue();`

  remove and return the element at the head of the queue

- `int size();`

  returns the number of elements in the queue

Cube ADT and BoundedCube CDT

Holds items in a 3D grid

Cube ADT and BoundedCube CDT

Holds items in a 3D grid

Each item accessed by an $< x, y, z >$ co-ordinate tuple

Cube ADT and BoundedCube CDT

Holds items in a 3D grid

Each item accessed by an $< x, y, z >$ co-ordinate tuple

Each cell could contain more than one object

Cube ADT Methods

- `void add(int x, int y, int z, T element);`

  adds an element at $< x, y, z >$
- `T get(int x, int y, int z);`

  returns the oldest element at $< x, y, z >$
- `IterableQueue<T> getAll(int x, int y, int z);`

  returns all elements at $< x, y, z >$
- `boolean isMultipleElementsAt(int x, int y, int z);`

  whether there is more than one element at $< x, y, z >$

Cube ADT Methods

- `boolean remove(int x, int y, int z, T element);`

  removes the specified element at $< x, y, z >$

- `void removeAll(int x, int y, int z);`

  removes all elements at $< x, y, z >$

- `void clear();`

  removes all elements in the cube

Functionality Marking Criteria

35% of your overall mark

Functionality	▪ Passes at least 90% of test cases, only failing sophisticated or tricky tests.	▪ Passes at least 80% of test cases, only failing one or two simple tests.	▪ Passes at least 70% of test cases.	▪ Passes less than 70% of test cases.
	7	6	5 4	3 2 1 0

Code Quality and Design Marking Criteria

10% of your overall mark

Code & Design Quality	▪ Code is well structured with excellent readability and clearly conforms to a coding standard. ▪ Comments are clear, concise and comprehensive, enhancing comprehension of method usage and implementation details. ▪ All design choices are appropriate for the context and the ADT.	▪ Code is well structured with very good readability and conforms to a coding standard. ▪ Comments are clear, mostly concise and comprehensive, and in most cases enhancing comprehension of method usage and implementation details. ▪ All, but one, design choice is appropriate for the context and the ADT.	▪ Code is well structured with good readability and mostly conforms to a coding standard. ▪ Comments are clear and fairly comprehensive, providing useful information about method usage and implementation details. ▪ Most design choices are appropriate for the context and the ADT.	▪ Parts of the code are not well structured, are difficult to read, or do not conform to a coding standard ▪ Comments at times are not clear, or provide little useful information about method usage or implementation details. ▪ Most design choices are inappropriate for the context or the ADT.
	2	1.5	1	0.5 0

JUnit Tests

Some basic tests provided in BoundedCubeTest.java and TraversableQueueTest.java

JUnit Tests

Some basic tests provided in BoundedCubeTest.java and TraversableQueueTest.java

You will need to write more tests in MyBoundedCubeTest.java and MyTraversableQueueTest.java

JUnit Tests

Some basic tests provided in BoundedCubeTest.java and TraversableQueueTest.java

You will need to write more tests in MyBoundedCubeTest.java and MyTraversableQueueTest.java

Tests will be marked on code coverage (i.e. the percentage of your code the tests) and how well they cover boundary conditions

Testing Marking Criteria

20% of your overall mark

Testing	▪ All public methods in the CDTs are comprehensively tested. (e.g. 90% of branches and almost all boundary conditions are tested.) ▪ Tests are clearly designed with an exemplary understanding of the algorithm and data structure design.	▪ Almost all public methods in the CDTs are well tested. (e.g. 80% of branches and most boundary conditions are tested.) ▪ Tests seem to be designed with a very good understanding of the algorithm and data structure design.	▪ Most public methods in the CDTs are moderately well tested. (e.g. 70% of branches and some boundary conditions are tested.) ▪ Tests seem to be designed with a fairly good understanding of the algorithm and data structure design.	▪ Some public methods in the CDTs are adequately tested. (e.g. less than 70% of branches and a few boundary conditions are tested.) ▪ Tests seem to be designed with little understanding of the algorithm and data structure design.
	4	3	2	1 0

Analysis

Each public method in your CDT classes should state the runtime efficiency in the JavaDoc for that method (using big-O notation)

Analysis

Each public method in your CDT classes should state the runtime efficiency in the JavaDoc for that method (using big-O notation)

Each CDT class should state the memory efficiency in the JavaDoc for the class (also using big-O notation)

Analysis

In a comment at the end of each CDT class, provide a justification of the design choices you made in implementing the CDT (should consider the memory and run-time efficiency of the data structure in the context of the air traffic management simulation)

Analysis

In a comment at the end of each CDT class, provide a justification of the design choices you made in implementing the CDT (should consider the memory and run-time efficiency of the data structure in the context of the air traffic management simulation)

For the BoundedCube class, ensure your justification considers memory usage in the context of the simulation (in the simulation there will be a very small number of occupied cells)

- Describe any limitations of your CDT and alternative implementations that may be more appropriate (you dont need to write code for these implementations, only describe them)

Analysis Marking Criteria

35% of your overall mark - equally as important as functionality!

Analysis	▪ Correctly determined run-time and space efficiency of every method and CDT. ▪ Justification of CDT design choices are valid, clearly expressed and relevant to the application. ▪ BoundedCube discussion demonstrates a good understanding of the issues related to its memory usage in this context and the trade-offs of different potential implementations.	▪ Correctly determined run-time efficiency of almost all methods and space efficiency of all CDTs. ▪ Justification of CDT design choices are valid and relevant to the application. ▪ BoundedCube discussion demonstrates a good understanding of the issues related to its memory usage in this context and some trade-offs of different potential implementations.	▪ Correctly determined run-time and space efficiency of most methods and CDTs. ▪ Justification of CDT design choices are mostly valid and somewhat relevant to the application. ▪ BoundedCube discussion demonstrates a good understanding of the issues related to its memory usage in this context.	▪ Correctly determined run-time and space efficiency of some methods and CDTs. ▪ Justification of CDT design choices are at best partially valid or not relevant to the application. ▪ BoundedCube discussion demonstrates little understanding of the issues related to its memory usage in this context.
	7	6	5 4	3 2 1 0

Constraints

You may **not** change the ADT interfaces

Constraints

You may **not** change the ADT interfaces

You may **not** change the package structure of the program - your code **must** adhere to the directory structure in the supplied ZIP file

Constraints

You may **not** change the ADT interfaces

You may **not** change the package structure of the program - your code **must** adhere to the directory structure in the supplied ZIP file

You may **only** use basic Java constructs - you may **not** import any data structures from libraries (including the Java Collections Framework)

- Exceptions are:
    - The Iterable and Iterator interfaces
    - The Comparable and Comparator interfaces (optional usage)

Constraints

You may **not** change the ADT interfaces

You may **not** change the package structure of the program - your code **must** adhere to the directory structure in the supplied ZIP file

You may **only** use basic Java constructs - you may **not** import any data structures from libraries (including the Java Collections Framework)

- Exceptions are:
  - The Iterable and Iterator interfaces
  - The Comparable and Comparator interfaces (optional usage)

You may **not** change the provided JUnit test classes

Constraints

You **must** provide a constructor for each CDT class that corresponds to the CDT object creation in the sample tests

- You may provide as many other public or non-public constructors to your CDTs as you see fit

Constraints

You **must** provide a constructor for each CDT class that corresponds to the CDT object creation in the sample tests

- You may provide as many other public or non-public constructors to your CDTs as you see fit

You may **not** add public methods or fields to your CDT classes that dont implement ADT methods

- You may add extra non-public methods and fields to your classes
  - Keep method logic simple and avoid repeating code
- You may add non-public subclasses

Important Note

Your code must compile using the standard Oracle JDK version 8 or Open JDK version 8

Important Note

Your code must compile using the standard Oracle JDK version 8 or Open JDK version 8

It is not acceptable that your code only works in your IDE

- Last year some students submitted assignments that would only work in the IntelliJ IDE - if you are using IntelliJ make sure your code works outside of the IDE
    - Code that only works in IntelliJ may be treated as non-compiling code (and hence you will be capped at a maximum of 5 marks)
- IntelliJ also gives innacurate code coverage reports - do not rely on these reports if you are using this IDE

Important Note

Your code must compile using the standard Oracle JDK version 8 or Open JDK version 8

It is not acceptable that your code only works in your IDE

- Last year some students submitted assignments that would only work in the IntelliJ IDE - if you are using IntelliJ make sure your code works outside of the IDE
    - Code that only works in IntelliJ may be treated as non-compiling code (and hence you will be capped at a maximum of 5 marks)
- IntelliJ also gives innacurate code coverage reports - do not rely on these reports if you are using this IDE

However, don't let this disencourage you from using one of the best Java IDEs out there ;)