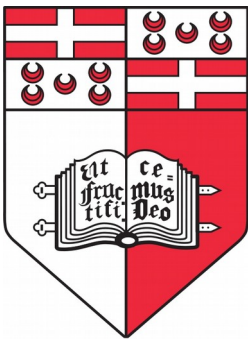
Object Oriented Programming Assignment

CPS2004
Object Oriented Programming

Lecturer
Dr Jean-Paul Ebejer

Stefan Mallia

Bachelor of Science (Honours)
Computing Science
and
Statistics and Operations Research



University of Malta
Department of Computer Science

Table of Contents

Task 1: QuadTree....4
- Composition relations....4
- The RC class....4
- The QuadTree class....5
  - Constructors....5
  - Saving and Opening file....5
  - Insert method....6

Task 2: Quiz Test....7
- QuizObject class....7
- QuestionObject class....8

Task 3: Web Server....9
- SimpleServer class....9
- ServerRunnable class....9
- ServerParser class....10
- FunctionalityChecker class....10
- HomePage class....10

Table of Figures

Figure 1: Task 1 Class Diagram....4
Figure 2: 3x3 region....5
Figure 3: Task 2 Class diagram....7
Figure 4: Task 3 Class diagram....9

Task 1: QuadTree

A QuadTree template was implemented using C++ using the UML class design shown in Figure 1.

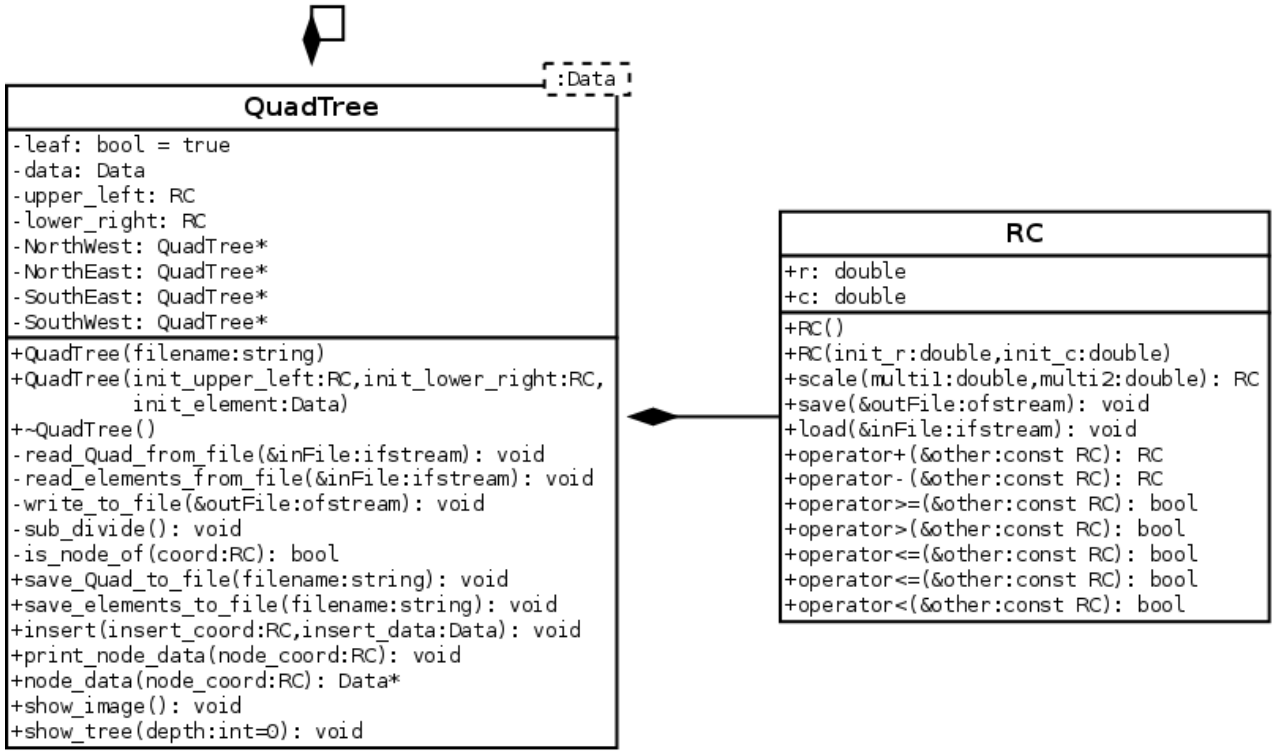


Figure 1: Task 1 Class Diagram

Composition relations

The QuadTree is a tree structure where each node could either have be a leaf or be the parent of 4 sub-nodes. Because of this, the QuadTree class could be a parent of 4 other QuadTrees where the lifetime of those sub QuadTrees cannot extend beyond the life of the parent QuadTree, hence the composition relation with itself.

Another composition relation exists with another class, RC (RowColumn), which is used to define the boundary of the QuadTree region. For example, an 8x8 QuadTree (which is also a root node) would have an upper_left of (0,0) and a lower_right of (8,8).

The RC class

The RC class is used as to define point coordinates inside the QuadTree. It is also used to define the QuadTree's boundaries, the upper left and lower right.

The values in the RC class are declared as double. This is to deal with pixel regions that do not divide evenly. For example, a 6x6 QuadTree divides into 4 3x3 QuadTrees which cannot be further subdivided neatly using ints.

However, with doubles, a 3x3 QuadTree sub divides into 4 1.5x1.5 QuadTrees where the NorthWest region takes over 4 pixels, the NorthEast and SouthWest regions takes over 2 pixels each, and the SouthEast region takes over 1 pixel. This is shown in Figure 2.

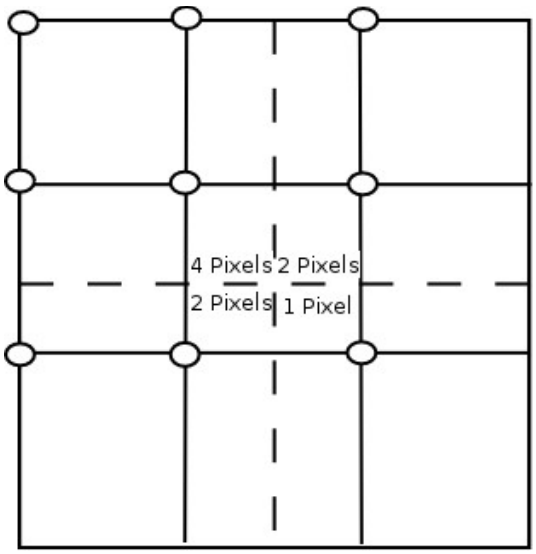


Figure 2: 3x3 region

Note that only the SouthEast region need not be divided any further due to containing only one pixel. Also note that if SouthWest and NorthEast are further sub divided then 4 new QuadTrees are created of which 2 are ignored. This is due to there being only 2 pixels in each of these regions.

Pixel indexes are considered to start from (0,0). So for an 8x8 QuadTree, the last picture is located at (7,7). Note that pixels are only located on integer coordinates inside the QuadTree and only touch the QuadTree's left and upper boundary due to rows being directed downwards and columns directed rightwards.

The RC class also defines several operators to simplify operations between coordinates. The equality/inequality operators require that both r and c satisfy the condition. For example, for (r1,c1) > (r2,c2) to be true, it is required that both r1>r2 and c1>c2 to be true.

The save and load functions in the RC class where created to allow for binary saving of RC instances alongside their respective QuadTree node. Since RC is a class that contains methods, it was necessary to create functions that save only the RC class's attributes. These two functions take in an ifstream/ofstream to perform the write/read operations on already opened files. The ifstream is opened from the QuadTree class's save_Quad_to_file() method.

The QuadTree class

Constructors

There are two constructors for the QuadTree class. The first constructor takes a file name as argument and loads the the QuadTree from there. The second initializes a r-by-c QuadTree where all values are set to the value specified by argument, hence a QuadTree with one node.

The constructor that loads from file will load from binary files unless the QuadTree is initialized to contain bool values as a specialization. In the case of bool values, the QuadTree would load from a text file where all pixel values are either 'T' or 'F'.

Saving and Opening file

In the generic case, this constructor accepts two save file methods, both of which are binary files.

The first file save method is a recursive function where the QuadTree was saved using a recursive function. This implementation can be found in the save_Quad_to_file() method where all QuadTree attributes are saved for each node recursively and then reconstructed recursively upon loading. This

is a very efficient method when there are many neighboring pixels with the same value.

The second file save method simply saves the QuadTree iteratively by checking each pixel within the bounded region.

The two file save methods are distinguishable to the program because a flag is saved at the beginning of the file to indicate which save method was used.

The boolean specialization of this template class changes the implementation to save the QuadTree such the QuadTree is saved iteratively, i.e. element-by-element, to a text file. The specialization also disables the recursive save method.

Insert method

The insert function navigates through the tree recursively until the pixel is found. Subdivision is done if the node is larger than one pixel and the inserted value is not the same as the node's. At the end of the function all children are checked to determine whether the node should be collapsed.

Task 2: Quiz Test

A multiple choice quiz was implemented using Java. Figure 3 shows the UML class diagram.

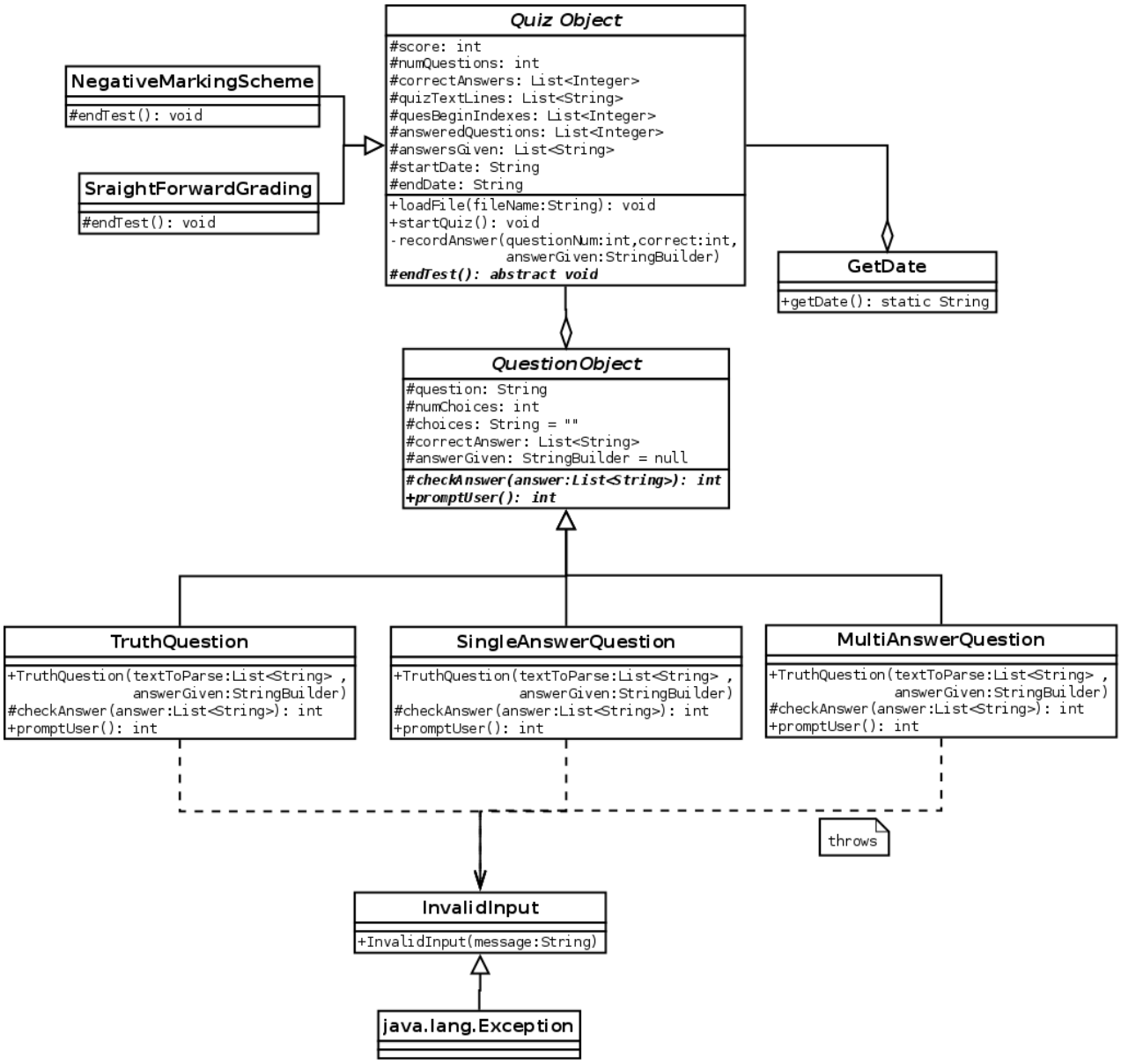


Figure 3: Task 2 Class diagram

QuizObject class

QuizObject is the class that takes care of the quiz as a whole. It keeps record of information such as the number of answers given, what amount of answers are correct, etc. The class has three methods. One loads the file containing the quiz and parses it to determine on what lines questions are found on. The second, recordAnswer(), is a method to keep record of what answers are given by the quiz taker. The third, startQuiz(), is the method that prompts the user with a menu interface and creates new instances of QuestionObject, taking a subsection of the quiz text as input, to handle the parsing of an individual quiz question.

Quiz object is an abstract class where only one function is abstract, endTest(). The implementation

of this method is left up to the subclasses NegativeMarkingScheme and StraightForwardGrading to apply different styles of grading.

QuestionObject class

Like QuizObject, QuestionObject is also an abstract class. However, unlike QuizObject, all its method implementation was left to its subclasses which work around the different question type requirements.

This object is constructed with a sub-List of strings, where each element is a line, of the relevant quiz section. This list is parsed to determine the correct answers, and distinguish between the question line and choice lines. The constructor also takes as argument a StringBuilder object, which is passed by reference. The StringBuilder is used to record the answer given to print later in the question selection menu when control goes back to QuizObject.

The promptUser() method prompts the user with the question and choices represented by that particular class instance. It returns a -1 for incorrect answers and a 1 for correct answers. It also contains a form of exception handling where an InvalidInput class is thrown if either no answer was given, too many answers were given, or some other invalid form of input was entered.

Task 3: Web Server

A simple server to service GET requests was implemented using Java. C++ would also have been a viable option for consructing a server. However, the advantages that C++ has over Java, such as better performance and memory handling, are not enough to overcome Java's robustness and superior library support.

The server listens to a port until an HTTP request is received, parses the request, and sends the requested file to the client.

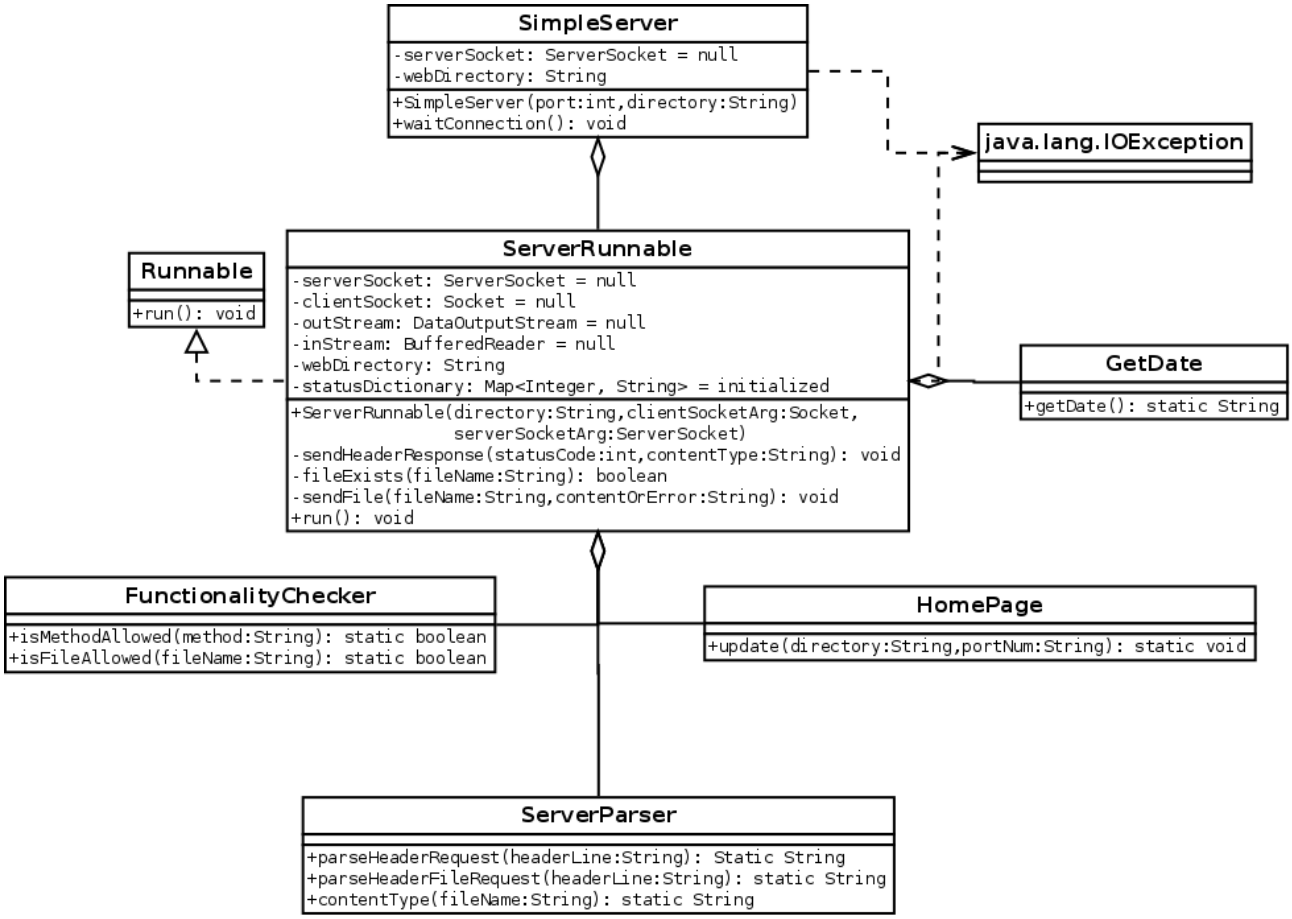


Figure 4: Task 3 Class diagram

SimpleServer class

This class is mainly used execute the method, waitConnection(). This method waits to accept a new connection. Once a new connection is accepted, a new thread is launched to handle the HTTP request and the waitConnection() method goes back to waiting for connections.

ServerRunnable class

The majority of the work is handled by ServerRunnable. This class sets up the data stream and reads in the HTTP request. The ServerParser and FunctionalityChecker classes are used by this class to parse the request. If the request is processed successfully, the requested file is sent using the sendFile function. If no file is requested the homepage html file is sent. This file contains links to the available files in the web directory. Otherwise, if the request is not processed successfully, an error message is sent to the client.

ServerParser class

ParseHeaderRequest() and parseHeaderFileRequest() return the method and the filename respectively. ContentType() returns whether the content requested is html or 'plain' (txt) to be used in forming the header response.

FunctionalityChecker class

This class contains functions to check whether the method or filetypes are permitted by the server. This particular server only implements GET requests and sends html and text files so if anything else is inputted into this class it returns a false.

HomePage class

The HomePage class updates the homepage.html file located in the web directory. This file contains links to the available files in the web directory.