ECE 3574 • Applied Software Engineering
Midterm Examination (19 questions, 50 points)
Fall 2008

Name: Student ID#:

Questions 1-15: 2 points each.

Question 1. For overloading arithmetic symbols (+ - * /) on Fraction objects (see Example 2.14; page 65, Ezust), which is better, member functions or non-member global operators?

[solution] Non-member global operators are preferred because then the first or second operand can be converted just easily to a Fraction. As a member function, only the right hand side is convertible, which means the operator won't work in a symmetric way.

Question 2. Which member functions cannot be inherited from the base class? Explain why.

[solution] The constructors, destructors, copy and assignment operators are all not inherited. They get generated for classes depending on certain rules, and the generated versions call base class versions.

Question 3. What is a design pattern? What do most design patterns have in common?

[solution] Most design patterns describe how to separate code by responsibility. They have a name, a description, and describe a problem (anti-pattern) that it attempts to solve.

Question 4. There are three kinds of design patterns: structural, creational, and behavioral. Which kind is the Visitor pattern? Why?

[solution] Behavioral - it describes how code is organized, and assigns the responsibility of visiting objects to a specific class.

Question 5. Why does the FileVisitor need to be recursive?

[solution] Because hierarchical directory structures are recursive data structures.

Question 6. What does it mean when QObject A is the parent of QObject B?

[solution] B is managed by A, and B will be destroyed when A is.

Question 7. How can QObject be both composite and component?

[solution] When it has both parents and children.

Question 8. Qt's container classes are used to collect value types. What kinds of things are not appropriate to store by value in a value collection?

[solution] QObjects, polymorphic collections in general, and any class that cannot be copied because it has no copy constructor.

Question 9. Which containers provide a mapping from key to value? List and describe at least two, and tell how they are different from each other.

[solution] QMap and QHash. QMap maintains its values sorted by keys, where QHash does not. Also, QHash provides much faster value lookups than QMap.

Question 10. What is a QLayout? What is its purpose? What is an example of a concrete QLayout class?

[solution] A QLayout is a class whose sole purpose it is to arrange other QWidgets or sub-layouts. It has no visual representation itself. There are Boxes, Grids and Stacks.

Question 11. QWidget is derived from QObject and QPaintDevices. What does this multiple inheritance allow QWidget -derived classes to do?

[solution] They are QObjects, so they can have parents, children, properties, signals, and slots. They are QPaintDevices, so they have a height, width, depth, and can be painted on the screen.

Ouestion 12. What is a QAction? How are actions triggered?

[solution] A QAction is an object representing something that the user can do to the program. It can be triggered through menu items, button clicking, keyboard presses, etc.

Ouestion 13. It is possible to create QMenus without using QActions. What are the advantages of using a QAction?

[solution] By creating a unique QAction for each action, you can add it to different menus and toolbars, and from one place, disable/enable it, or change other properties (such as its label or tooltip) and it will reflect in all views (menu-items, toolbar buttons, etc) that refer to it.

Ouestion 14. What does it mean for a container to "manage" its heap objects? How can a container of pointers to heap objects become a „ managed container"?

[solution] A container is said to "manage" its heap objects if it is responsible for the lifetimes of those heap objects (i.e., it destroys the objects when it is destroyed). A container of pointers to heap objects can become a managed container by calling qDeleteAll(container), which will call delete on each contained pointer, thereby invoking the destructors of the pointed-to objects.

Ouestion 15. Give 3 examples of Qt classes that implement the Flyweight pattern.

[solution] QList, QString, QSet.

Ouestion 16. What is the Model View pattern? How is it useful? [4 points]

[solution] The Model View pattern is a generic solution to the problem of visually rendering a collection of data items. Instead of combing the code for storing, indexing, and accessing the data with that for visually rendering them, the pattern uses different classes for doing this. The Model class provides an interface for accessing the data and the View class visually renders that data. The pattern thus allows decoupling of the model and the view - i.e., it allows one to use different View classes (for obtaining different views) for a given model, and different Model classes (for storing and accessing data in different ways) for a given view.

Question 17. See an example use of the QStringListModel given in Example 11.24 of Ezust (page 272-273). If you didn't use the Model View pattern and Qt classes that support this pattern (i.e., QStringListModel, QListView) to write this application, how would you then write this application? Explain the general steps that you would take, the Qt classes that you will use, the order in which you will use the APIs, and any possible signal/slot connections that will be needed. [5 points]

There are several "low-level" ways to do this. An example approach would be:

Use a container class (e.g., QHash, QMap) to store the QStrings ("Mandaag .. '). For rendering this data, use a QApplication, define a central QWidget for it, and set up a parent QLayout (e.g., QGridLayout) for the widget. To the parent QLayout, add a QVBoxLayout (for rendering the QStrings "Mandaag...") and the "insert" QPushButton widget (for rendering the output QStrings "item #....").

Since the QStrings ("Mandaag...") must be rendered and editable, use

an input widget such as a QLineEdit. Retrieve each of the QString from the container (using an iterator pattern with keys(), values(), etc.), insert it into a QLineEdit (using insert() or setText()), and add the QLineEdit as a widget to the QVBoxLayout (note that QLineEdit is a QWidget). This will display each of the QStrings in the displayed window.

Connect the textChanged() signal of QLineEdit to a slot, which will insert the signal's returned QString parameter into the container class. (textChanged() signal is emitted whenever the text changes and the user-entered text is returned as the signal's parameter.)

Connect the "insert" QPushButton's clicked() signal to the slot addItem() as before. But this slot will now create the QString "item #. . insert it into a QLineEdit, and add the QLineEdit as a widget to the QVBoxLayout. It will also add the QString to the container class.

Question 18. What is the difference between aggregation and composition relationships? [3 points]

[solution] In an aggregate relationship, the lifetimes of the objects on either end are not related to each other. In contrast, in a composition relationship, the "whole" object is responsible for the lifetime of the „ part" object.

Question 19. Draw a UML diagram with three or more classes, different from all the examples given in Ezust, and make sure that all of these different kinds of relationships are represented:

- aggregation
- composition
- one-to-one and one-to-many
- unidirectional

The classes should represent real-world concepts, and the relationships should attempt to represent reality. Justify why there is a relationship of each kind in your diagram. [8 points]

//

ECE 3574 Applied Software Engineering

Midterm Examination Solutions (16 questions, 50 points)

Fall 2010

Name: Student ID#:

Name: Student ID#: Questions 1-15: 3 points each. Use your own words in your answers.

Question 1. Consider the function call graduate( ) in Example 6.5 of Ezust (page 141). Why is that (a) this call prints [Student] and not [GradStudent] and (b) this call does not print GradStudent's support (Support) information?

[solution]

(a) getC1assName( ) is not defined as a virtual function in the class Student . Thus, the call graduate ( ) will invoke Student's getClassName(), which returns the string "Student". If getC1assName( ) was defined as virtual in class Student, then graduate(&gs) will invoke GradStudent's getC1assName( ) , which returns the string "GradStudent", as result of dynamic binding.

(b) tostring( ) is not defined asa virtual function in the class Student. Thus, the call graduate( ) will invoke Student's tostring( ) function, which doesn't print any Support information. In contrast, GradStudent's tostring( ) function prints the Support information.

Question 2. What is an abstract class? What is it used for? What can you do with a concrete class that you cannot do with an abstract class?

[solution] An abstract class contains at least one pure virtual function, and cannot be instantiated like a concrete class. An abstract class is used to group features that are common to many classes in a design, but the class by itself doesn't have a real world physical counterpart.

Question 3. For each of these items, determine whether it would normally be found in a header (.h) file or an implementation (.cpp) file and explain why.

(a) function definitions

[solution] implementation - they should be only compiled once, and not included by other modules that use it.

(b) function declarations

[solution] usually the header file, to refer to functions defined in other separate implementation files.

(c) static object declarations

[solution] header files, like most declarations.

(d) static object definitions

[solution] implementation file, just like functions. We do not wish to have redundant storage allocation for statics.

(e) class definitions

[solution] these go in the header file.

(t) class declarations

[solution] header file - usually they are forward declarations.

(g) inline function definitions

[solution] header files - unlike regular functions which can be linked together, inline functions need to be fully defined before they can be used.

(h) inline function declarations

[solution] Header file - Rarely used for member functions except when the declaration is separated from the definition - but kept in the same header file.

(i) default argument specifiers

[solution] header file - default arguments to a function changes the way the function is called. This is the realm of the header file.

Question 4. What is a design pattern? What do most design patterns have in common?

[solution] Most design patterns describe how to separate code by responsibility. They have a name, a description, and describe a problem (anti-pattern) that it attempts to solve.

Question 5. What happens to a QObject when it is reparented?

[solution] It is removed from its former parent's child list, and added to the new parent's child list.

Question 6. Why is the copy constructor of QObject not public?

[solution] To prevent it from being copied. QObjects are supposed to have identity, and so there should be no "exact" copy. In addition, copying a QObject would mean copying all its children, a potentially expensive operation.

Question 7. How can you access the children of a QObject?

[solution] The children of a QObject can be obtained through the findChildren() member function.

Question 8. What is the difference between value types and object types? Give examples.

[solution] Value types are simple types that can be copied or compared quickly—e.g., int, char, QString, QDate, pointer types, etc. In contrast, object types are complex, maintain an identity, and can rarely be copied—e.g., QObject.

Question 9. When connecting a signal to a slot, are there any restrictions on the parameters of the signal and the slot?

[solution] A signal of one object can be connected to the slots of one or more other objects, provided the parameter lists are assignment compatible from the signal to the slot. That is, the slot must have at least as many parameters as the signal (the slot can ignore extra parameters), and those parameters must be value types.

Question 10. How can a container of pointers to heap objects become a "managed container"?

[solution] There are several ways to do this. One way is for the container to call the Qt algorithm qDe1eteA11(container) . This calls delete on each (pointer) element in the container, which invokes the destructors of the heap-allocated pointed-to objects and destroys them.

Question 11. What is the purpose of the QLayout class? What is an example of a concrete QLayout class?

[solution] A QLayout is a class whose sole purpose is to arrange other QWidgets or sub-layouts. It has no visual representation of itself. Concrete QLayouts are Boxes, Grids and Stacks.

Question 12. Which containers provide a mapping from key to value? List and describe at least two, and tell how they are different from each other.

[solution] QMap and QHash. QMap maintains its values sorted by keys, where QHash does not. Also, QHash provides much faster value look-ups than QMap.

Question 13. What is the Flyweight pattern? Give 3 examples of Qt classes that implement the Flyweight pattern.

[solution] The Flyweight pattern is a generic solution to the problem of avoiding creating multiple copies of an object, when a single shared instance of the object will do. The pattern uses a handler class that manages a single shared instance of the object and keeps track of references to the shared instance through a reference counter. The counter is incremented when new references to the shared instance

are added, and decremented when references are deleted. When the counter reaches zero, the handler destroys the shared object.

Qt example implementations of this pattern include QList, QString, and QSet.

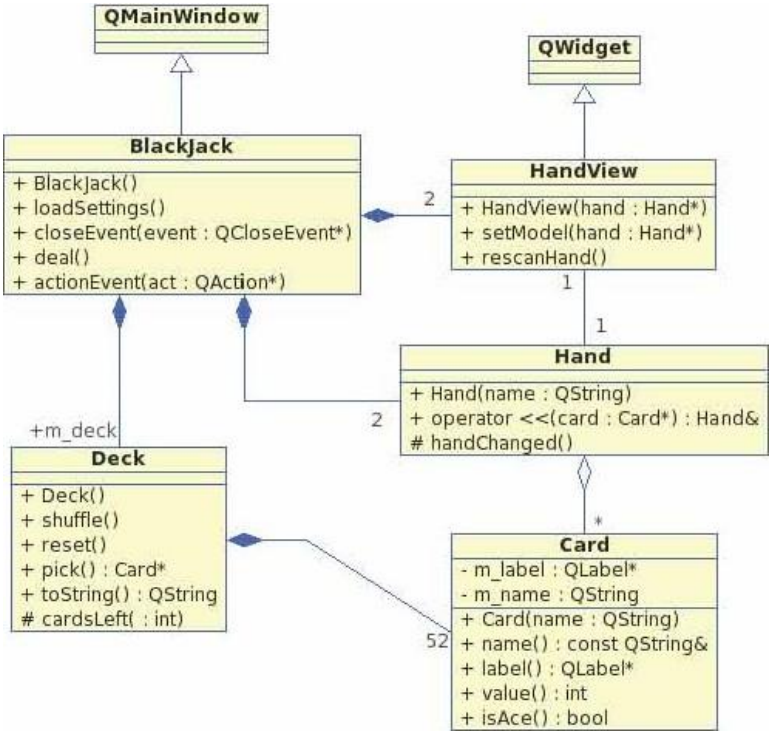
Question 14. What is the difference between a spacer and a stretch?

[solution] A spacer is a fixed amount of space that does not stretch, while a stretch starts with a certain amount of space and expands to gobble up available space. In the case of multiple stretches in the same layout, the ratio can be used to determine which stretch expands faster.

Question 15. What is the Model View pattern? How is it useful?

[solution] The Model View pattern is a generic solution to the problem of visually rendering a collection of data items. Instead of combing the code for storing, indexing, and accessing the data with that for visually rendering them, the pattern uses different classes for doing this. The Model class provides an interface for accessing the data and the View class visually renders that data. The pattern thus allows decoupling of the model and the view - i.e., it allows one to use different View classes (for obtaining different views) for a given model, and different Model classes (for storing and accessing data in different ways) for a given view.

Question 16. Consider the following UML diaglam:



Explain this diagram by identi&ing the different types of class relationships (i.e., which class is related to which class and how) and the multiplicities in the relationships.

This design follows the model/view paradigm Identi& the model class(es) and the view class(es). [5 points]

[solution]

Class relationships:

- Class BlackJack is derived from class QMainWindow
- Class HandView is derived from class QWidget
- Class Blackjack is composed of Wo Hand View objects
- Class Blackjack is composed of two Hand Objects
- Class Blacl$ack is composed of one Deck object
- Class Hand View and class Hand has a 1:1 bidirectional association relationship between them
- Class Hand is an aggregate of zero or more instances of Card objects
- Class Deck is composed of fifty two Card objects

Model/view classes:

- Class HandView is the view class
- Class Hand is the model class

//

Chapter 1 Review Questions:

A pointer is an address of memory in which you have stored some data. It is usually a pointer to a data type

and what you pass is a-

What is the address of operator? - is the & and gives you the address of a variable

What is the de-reference operator? - it is the asterisk *. It gives you value to which you are pointing to.

What is a null pointer? a Null pointer is a pointer to any data type, but the value is 0. You would want to define one

to help find bugs and is a good way to program to test.

What might cause segmentation fault? - Usually if you have a pointer pointing to something in memory that isn't mapped then it could cause it.

What are the possible uses of cost when dealing with pointers? -skipped

What is a function signature? - skipped

Why is it an error to have two functions with the same signature but different return types in one scope? - Because it is not allowed in C++ (C++11).

Chapter 2 Review Questions:

2.16 Review Questions.

1) Struct and class is equivalent but in Struct, all members are public. But in class, you have public, private, and protected members which is not possible in a Struct.

3) Friend functions are functions that are able to access private and public member of the class.

4) A static data member differs from non-static because -

6) to declare a member function to be cost means that function does not change any data member of our class.

Chapter 4 Review Questions:

4.5 Review Questions

4) the iterator pattern is the first pattern we discussed. With it, we are able to, in a standard way, walk through a data structure.

Is very powerful as with an iterator, you don't need to know which data structure your class implements.

6) the difference between composition and aggregation. Composite relation will manage the data over lifecycle of container. In Aggregate, container does not manage the lifecycle of the data.

Chapter 5 Review Questions:

5.13 review questions

1) what is difference between function declaration and function definition? We are declaring a type when we are writing it. We are defining it instead

when we are putting the prototype and body.

2)for every member function, it can have default functions. Meaning you can provide standard values to one of the or multiple arguments. So if

user doesn't set one of these arguments, then it will have default value. Default values part of the definition.

6) By default in C, they are pass by value. Every copy is initialized on stack before calling the function. But in C++, we can also pass by reference.

Chapter 6 Review Questions:

6.11

1) what is diff between a function and a method? due to if a function in class was a virtual, it would thus be method.

2) what does it mean for base class to be hidden

3) constructor, copy constructor, copy assignment I

Chapter 7 Review Questions:

Abstract cl

What is a design patter? What'd most design patterns that in common.

Chapter 8 Review Questions:

8.9 Review Question

m

1) what does it mean when QObject A is aren't of QObject B. Means that specifies at run time who is pointing to

\\\\\\\\'

event loop allows take representing'

Chapter 10 Review Questions

QMainWindow and Dialog both inherit from QWidget. QMainWindow is divided in multiple lllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllll

Chapter 11 Review Questions

2) when we define template. We define template parameters. Can be multiple and can be type but also constant. instead the PARAM of function can only be types.

3) this is because to use same template, then you get an error.

Chapter 12 review

Very important is MainobjectFramework. Divided into two parts. One part allows you to know everything about QObjectType and the other about QValue Type. Like this because in QT, our types are divided into two categories. One has value types so you can clone and the other being dynamic from QObject. Everything inherited from QObject is not closable. Only one instance.

QObjectType and Value Type. Different types. Value Types can be copied.

QMetaObject