CSE687 Midterm #1

Name: ___Instructor's Solution______________ SUID: _____________

This is a closed book examination. Please place all your books on the floor beside you. You may keep one page of notes on your desktop in addition to this exam package. All Exams will be collected promptly at the end of the class period. Please be prepared to quickly hand in your examination at that time.

If you have any questions, please do not leave your seat. Raise your hand and I will come to your desk to discuss your question. I will answer all questions about the meaning of the wording of any question. I may choose not to answer other questions.

You will find it helpful to review all questions before beginning. All questions are given equal weight for grading, but not all questions have the same difficulty. Therefore, it is very much to your advantage to answer first those questions you believe to be the easiest.

//

A good answer to a question about ideas and principles discusses the ideas and principles, how they will be used, and gives brief examples.

A good answer to a code question is brief, clear code that does what is asked and very little else. You can discuss elaborations of the code in comments.

I note the irony that I sometimes elaborate functionality of my code due to an over-abundance of enthusiasm, although that is usually done to instruct.

Grades are based on both contents of answer and clarity of presentation.

//

1. State the Liskov Substitution Principle and describe ways to write code that can cause substitution errors.

   Functions that use pointers or references statically typed to some base class must be able to use objects of classes derived from the base through those pointers or references without any knowledge specialized to the derived classes.

   Code constructs that may cause substitution errors:
   a. Overloading base class methods in a class derived from the base – causes hiding
   b. Overloading base class virtual functions – causes hiding when those functions are overridden in the derived class.
   c. Redefining in a derived class a non-virtual method in the base – non-virtual method calls are based on the type of the pointer, not the type of the object bound to the pointer.
   d. Failure to provide a virtual destructor in a class that will serve as a base for inheritance – causes incomplete destruction of derived objects on the heap when deleted from a base class pointer bound to the derived instance.
   e. Use of default parameters in both base and derived with different values if accessed through a base pointer or reference – will use values determined by the static type of the pointer or reference, not the type of the bound instance.

2. What is object oriented design (OOD)? How have you used OOD in your first project? Please be specific.

   OOD is the design of software using classes and class relationships.

   The Project #1 NoSqlDb is composed of many classes with single responsibilities. The list, below, describes each class and shows their relationships with other classes:

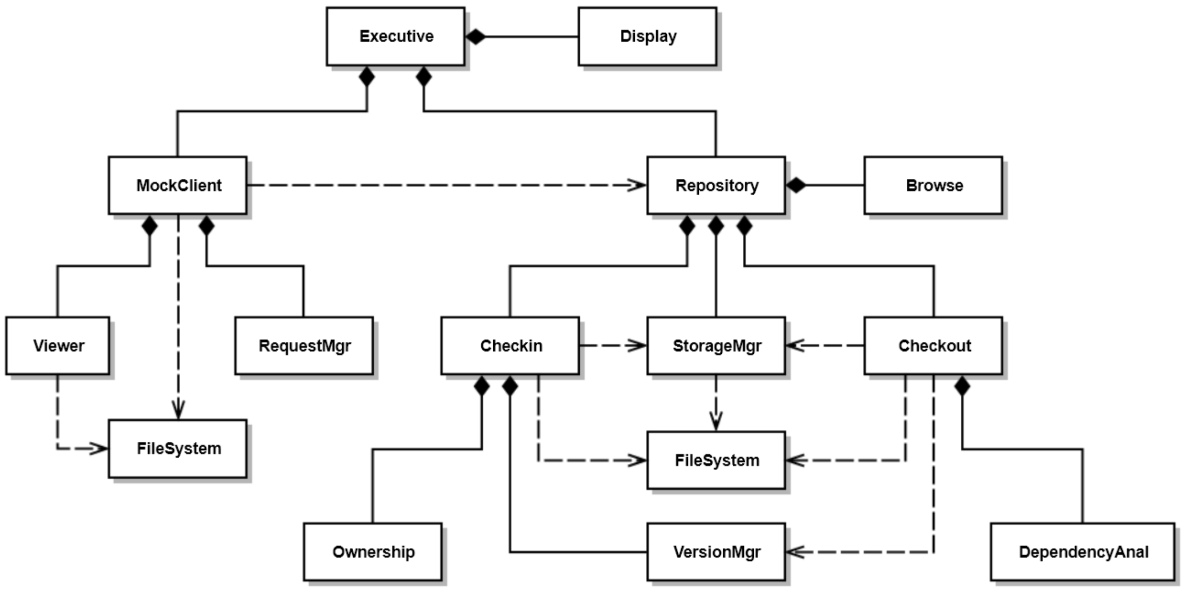
   a. DbCore – record management
   b. PayLoad – application specific data management
   c. Edit – modify existing db records
      uses instance of DbCore
   d. Query – extract information from db
      uses instance of DbCore
   e. Persist – Save and retrieve db records
      owns XmlDocument, uses instance of DbCore
   f. XmlDocument – support creating XML representation of db and recreation of db records
   g. DateTime – manage time information
   h. Utiltites – support for type conversions and display
   i. Executive – compose all Project #1 functionality and deploy
      owns instances of DbCore, PayLoad, Edit, Query, and Persist
      uses Utilities
   j. TestClass – demonstrate satisfaction of requirements
      owns instances of DbCore, PayLoad, Edit, Query, and Persist
      uses Utilities

   Several of you have said that OOD is the use of Object Oriented Languages. That is true for Java and C#, but is incomplete. One can use C++ without using any classes even though it is an excellent Object Oriented Language.

   Some have cited using templates, but that is not OOD. It is generic programming. That, of course, does not mean that a template class is not an artifact of OOD, but rather, the template has nothing to do with its OOD-ness.

   Several of you also discussed packages instead of classes in Project #1. Packages are not OOD specific. C language programs use packages, for example.

3. Draw a class diagram for your design of Project #2. Discuss, briefly, the responsibilities of each. You do not have to include the NoSqlDb classes.



Class	Responsibility
Executive	Create MockClient and Repository and start execution
Display	Writes demonstration messages to console
MockClient	Initiate Checkin and Checkout operations
Viewer	Write request traffic to console (will become view in GUI – Pr#3), uses process to display file text. Diagram assumes process part of Viewer.
Repository	Create Checkin, Checkout, and StorageMgr Respond to MockClient requests
Checkin	Respond to checkin request, using Ownership rules and VersionMgr
Checkout	Respond to checkout request using DependencyAnal, VersionMgr
StorageMgr	Manages Repository Folders and placement of files
Browse	Provides name, description, and children of specified file
VersionMgr	Creates new versions and finds latest versions
Ownership	Validate checkin, may track check-outs for multiple owner policies
DependencyAnal	Evaluate transitive dependency relationships
FileSystem with classes: File, Path, and Directory	Provide directory and file management utilities. This is a stand-in for the classes File, Path, and Directory, to lessen complexity of diagram

This is a fairly elaborate design for Project #2. A simpler design may be acceptable.

Note that only two classes do not have owners: Executive and FileSystem which has only classes with static methods. Also note that the responsibilities describe actions, and in some cases, action flows.

4. Write all the code for a File Manager that searches a directory tree, rooted at a specified path. passing file names and directory names to a using application[1], without including any application details.

```
template<class App>
class FileMgr
{
public:
  using Patterns = std::vector<std::string>;
  using Files = std::vector<std::string>;
  using Dirs = std::vector<std::string>;

  bool search(const std::string& path);
  void addPattern(const std::string& pattern)  // not required for answer
  {
    patterns_.push_back(pattern);
  }
private:
  std::vector<std::string> patterns_;
};

template<typename App>
bool FileMgr<App>::search(const std::string& path)
{
  if (!FileSystem::Directory::exists(path))
  return false;
  App app;
  std::string fullPath = FileSystem::Path::getFullFileSpec(path);
  app.doDir(fullPath);

  for (auto patt : patterns_)
  {
    Files files = FileSystem::Directory::getFiles(fullPath, patt);
    for (auto file : files)
      app.doFile(file);
  }
  Dirs dirs = FileSystem::Directory::getDirectories(path);
  for (auto dir : dirs)
  {
    if (dir == "." || dir == "..")
      continue;
    std::string fullDir = fullPath + "\\" + dir;
    search(fullDir);
  }
  return true;
}
```

```
// Pseudo Code:
//   In class FileMgr<App>
//   Search(path)
//      App.doDir(path)
//      Find files
//      For each file
//          App.doFile(file)
//       Find Dirs
//       For each dir
//           Search(path + /dir)
//       return
```

[1] You may assume you have a FileSystem package with methods:
std::vector<std::string> FileSystem::getFiles(const std::string& path, const std::string& pattern)
std::vector<std::string> FileSystem::getDirectories(const std::string& path);

5. Assume you have designed, for Project #1, a Query class that provides a template method accepting callable objects. Write a query, using that method, that finds all files with a specified category as recorded in the payload instance[2] for that file.

```
template<typename P>
class Query
{
public:
  Query(DbCore<P>& db) : db_(db) { keys_ = db_.keys(); }
  static void identify(std::ostream& out = std::cout);
  Query& select(Conditions<P>& conds);

  template<typename CallObj>
  Query& select(CallObj callObj);  // applies callObj to each DbElement in
                                   // query’s key set
  Query& query_or(Query<P>& q);
  Query& from(const Keys& keys) { keys_ = keys; return *this; }
  void show(std::ostream& out = std::cout);
  Keys& keys() { return keys_; }
private:
  DbCore<P>& db_;
  Keys keys_;
};

template<typename P>
template<typename CallObj>
Query<P>& Query<P>::select(CallObj callObj)
{
  Keys newKeys;
  for (auto key : keys_)  // searches current keyset
  {
    if (callObj((*pDb_)[key]))
      newKeys.push_back(key);
  }
  keys_ = newKeys;       // return holding new keyset
  return *this;
}
```

//
// answer starts here:

```
Query<PayLoad> q1(db_);

std::string category = "thirdCategory";
std::cout << "\n  select on payload categories for \"" << category << "\"\n";

auto hasCategory = [&category](DbElement<PayLoad>& elem) {
  return (elem.payLoad()).hasCategory(category);
};

q1.select(hasCategory).show();  // using template method
```

[2] You may assume that the PayLoad class has a method:

bool PayLoad::hasCategory(const std::string& category)

6. Write a list of tasks that your check-in process must execute for Project #2. Please consider complete and incomplete inputs provided by the client.

Check-in tasks:

a. Validate check-in using ownership rules
    i. If invalid, fail checkin, notify client, and exit
b. Copy file from path specified by Client
c. Determine if this is a new check-in or continuation of an open check-in
    i. Check closed status in payload of last checkin if there is one
    ii. If closed or closed pending, find latest version, generate new version as latest + 1
    iii. If not closed or closed pending, previous checkin was incomplete, so keep same version
    iv. If previous checkin is closed or there is no previous checkin, create new db element, add to db with new key (see d)
    v. Otherwise, edit existing DbElement for open checkin (there can only be one)
d. Define unique key, i.e., namespace::filename.ext.ver
e. Create (new version) or update (open checkin) metadata
    i. Copy author, existing child keys, description from previous version if new version, or create from user supplied checkin information for new item, e.g., ver 1.
    ii. Update any changes in metadata specified in checkin information
f. Append version number to file
g. Submit file to StorageMgr for saving
h. Close checkin if requested
    i. May need to set as close pending
i. Notify client of successful checkin (open, closed pending, or closed)

Note: Do Not Close checkins for child keys !!!
Note: If you make a task list like this, implementing the process is very straight-forward.

7. Given a function with declaration:

   Result meaningOfLife(DateTime dt);[3]

   where result contains a size_t value and a std::string, write code to run the method asynchronously.

```
struct Result
{
  size_t mol;
  std::string blahBlah;
};

///////////////////////////////////////////////////////////
// function definition is not required for answer
Result meaningOfLife(DateTime dt)
{
  std::this_thread::sleep_for(std::chrono::milliseconds(2000));
  Result result;
  result.mol = 42;
  result.blahBlah = "The meaning of life on " + std::string(dt) +
    " is ,as any fool can see, ";
  return result;
}
///////////////////////////////////////////////////////////

int main()
{
  std::cout << "\n  MT1Q7 - Meaning of life";
  std::cout << "\n =========================\n";

  std::future<Result> fut = std::async(meaningOfLife, DateTime().now());
  Result result = fut.get();
  std::cout << "\n  " << result.blahBlah << result.mol << "!\n";
  std::cout << "\n  Thanks, Douglas Adams, for all the fish.\n\n";
  return 0;
}
```

//

Definition of struct Result and the 3rd and 4th lines of main are a complete answer.

//

[3] You don't need to specify the body of the function, but will need to show the definition of Result, to fully answer this question.