# 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.

1. State the Dependency Inversion Principle. Describe how you did, or could, use it in Project #2. Please be specific.

   Answer:

   High level components should not depend upon low level components. Instead, both should depend on abstractions.

   Implementing the Dependency Inversion Principle requires the use of an interface and object factory.

   All tests implement an ITest interface and provide a factory function get_Itests() that returns a vector of test objects, one for each test defined in a test Dll. The DllLoader uses that factory to create a collection of tests, each bound to the ITest interface, without ever binding to concrete test details.

   Also, the TestHarnessCore's Tester creates an instance of SingletonLogger and passes a pointer to that to each test throught the ITests::acceptHostedResource(Ilog*) method, which each test must implement. So each test couples to the ILog abstraction, not the details of the SingletonLogger. Here, the Tester acts as the logger factory.

2. Write code for a thread that receives messages in the Core Test Harness of Project #3. Please provide all the code to dequeue messages from the Process Pool Host's Comm instance, and deposit them in appropriate Host queues, using a lambda to define the thread's processing. For this question, you will find a block diagram for Project #3, including the Process Pool, attached at the end of this exam packet.

Answer:

```
Pseudo Code:  From Process Pool Diagram
while(true)
  get message from comm
  if ready msg post to readyQ
  if test request msg post to requestQ
```

```
/////////////////////////////////////////
// MT1Q2: Part A of answer

void TestHarness::recvMessages()
{
  while (true)
  {
    Message msg = comm_.getMessage();
    std::cout << "\n    TestHnRcv received msg: " + msg.command();
    if (msg.command() == "ready")
    {
      std::cout << "\n    TestHnRcv posting msg: " + msg.command();
      readyQ_.enQ(msg);
    }
    else if (msg.command() == "testReq")
    {
      std::cout << "\n    TestHnRcv posting msg: " + msg.command();
      requestQ_.enQ(msg);
    }
    else
    {
      std::cout << "\n    TestHarnessR - invalid message";
    }
  }
}

// MT1Q2: End of Part A of answer
/////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////
// MT1Q2: Part B of answer

std::cout << "\n    starting TestHarness receive thread listening on "
          << from_.toString();
recvr = std::thread([&]() { recvMessages(); });
recvr.detach();

// MT1Q2: End of Part B of answer
/////////////////////////////////////////////////////////////////////
```

Here, readyQ_, requestQ_, and comm_ are members of the TestHarness class that implements the recvMessages() method.

3. What relationships between classes does C++ support, and how did you use them for Project #2?

   Answer:

   C++ supports four relationships between classes – one example given of each[1]:

   a. Inheritance:
      i. Derived classes inherit all methods and data from their bases.
      ii. Used to define the hierarchy of test classes, rooted at the ITest interface. Also SingletonLogger implements ILog.

   b. Composition:
      i. A class holds an instance of another type as a data member.
      ii. Used by the Tester to hold an instance of DllLoader.

   c. Aggregation:
      i. An owning relationship that is defined by assigning a call to new to a pointer data member. Also defined when a method creates an instance of another class that goes out of scope when the method ends.
      ii. Used by the TestCollection to hold pointers to tests.

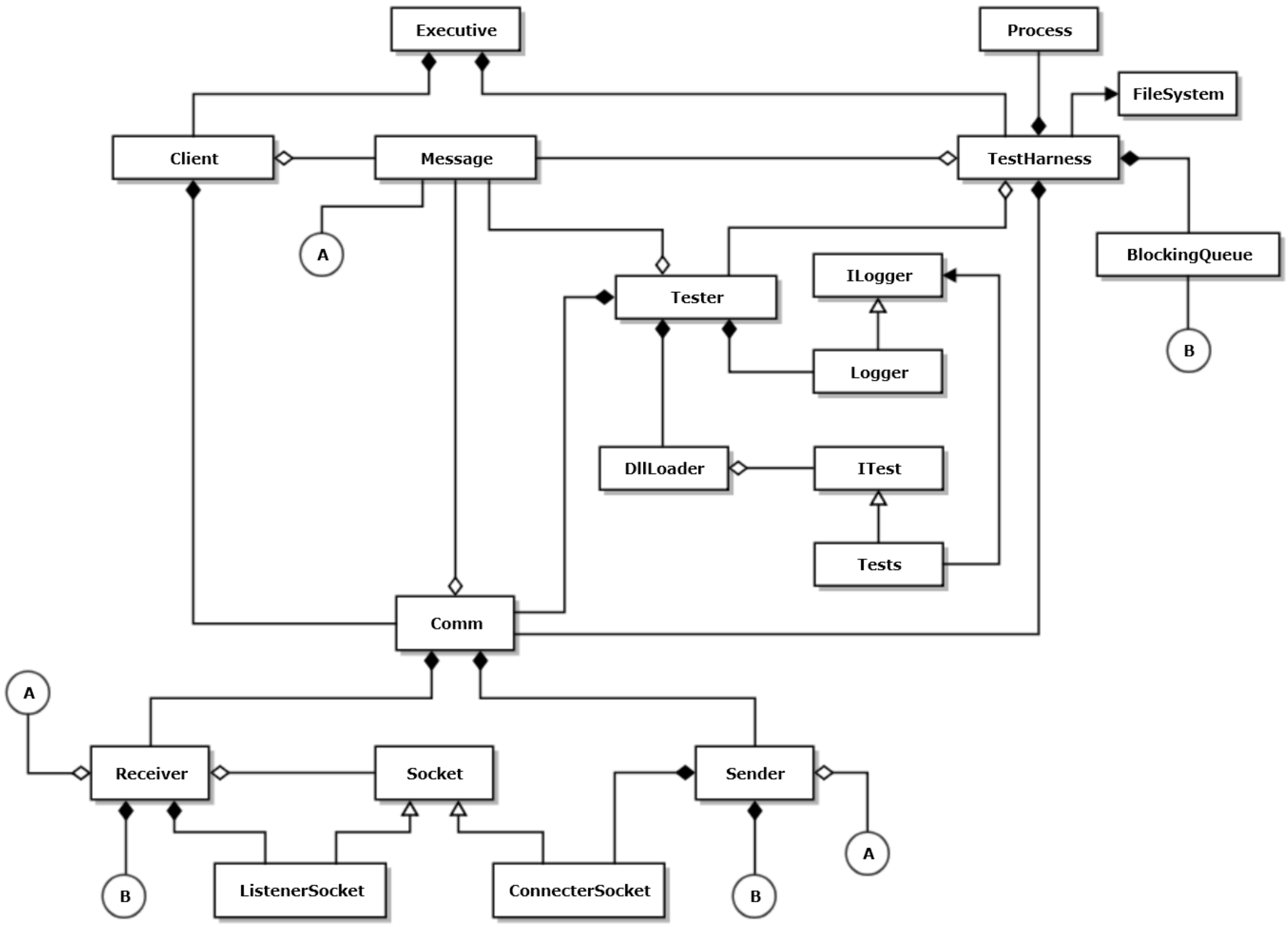
   d. Using:
      i. defined when a member function accepts a pointer or reference to an instance created by another class.
      ii. Used by SingleTestUtilities to hold a pointer to the Tester's logger.

[1] See answer to MT1Q4 for more complete picture of class relationships.

4. Draw a class diagram for a design for your implementation of Project #3. Please guide your design by the Single Responsibility Principle.

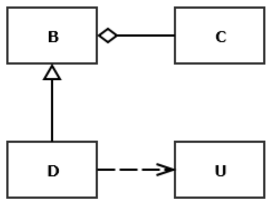
   Answer:

   No points taken off if you didn't include the Socket classes and the Sender and Receiver.



Note that the diamonds for composition and aggregation relationships are attached to the owner, not the owned.

5. Write all the code for a copy constructor for class D shown in the class diagram given here. You may assume that all of the bases and members of each class are shown in the diagram. When will this method be called?



Answer:

```
D(const D& d) : B(d)
{
  std::cout << "\n  D copy constructed";
}
```

Note this uses the B copy constructor to take care of C. U is simply used by D, so should not be copied.

D’s copy constructor will be called when an instance of D is passed or returned by value to a function or when we explicitly invoke it as here:

```
D d1;
D d2 = d1;  // copy construction
```

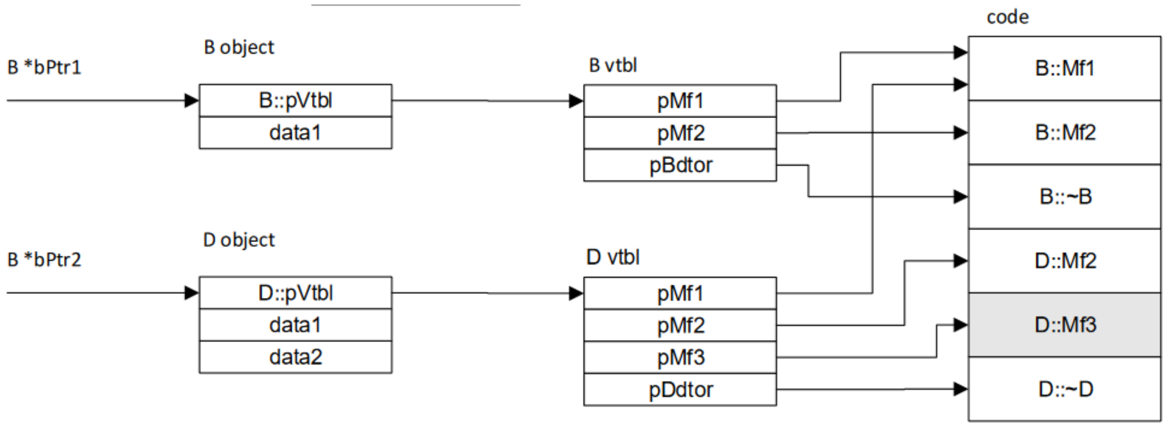
6. What is a virtual function pointer table? When are its contents defined and when is it used?

Answer:

A Virtual Function Pointer Table (vtbl) is a table of pointers to virtual methods created for every class that contains at least one virtual method. It is used for polymorphic dispatching, e.g., invoking a derived class method via a base class pointer, to which it is bound.

Vtbls are created at compile time, one for each class with virtual methods.

Vtables are used at run time to dispatch virtual function calls whenever we invoke a method of a derived class by means of a pointer to its base class.



Note that you were not obligated to draw this diagram.

7. Write a class declaration for a container for test results. Assume that the ITest interface declares name and author functions, in addition to its test method, which each test driver implements. The container will store results using the test driver name as a key and the value associated with the key will be the author, time-date[2], test result, and an instance of an unspecified type which is test specific. The container will also provide a means to persist the test results collection to an XML string[3].

Answer:

```
template<typename P>  // this class declaration is not required
class TestResult
{
public:
  Property<std::string> name;
  Property<std::string> author;
  Property<std::string> date;
  Property<bool> result;
  Property<P> appSpecific;  // will use here for log
};


using Key = std::string;
template<typename P>
using iterator
   = typename std::unordered_map<Key, TestResult<P>>::iterator;

template<typename P>
class TestResults
{
public:
  void add(const TestResult<P>& tr);  // key is tr.name()
  bool contains(const Key& key);
  typename iterator<P> begin();
  typename iterator<P> end();
  size_t size() const;
  TestResult<P>& operator[](const Key& key);
  TestResult<P> operator[](const Key& key) const;
  std::string toXml() const;
private:
  std::unordered_map<Key, TestResult<P>> results_;
};
```

[2] The date-time is passed as a string, probably generated by the DateTime class.
[3] You may assume that the unspecified type provides a toXml method.