CS 100 – Software Construction
Exam 1

May 8, 2017

Do not start working until you have been told to. Write down any assumptions you make. You must show all your work in order to get credit for questions, and if you use any of the blank sheets for your answer please indicate it on the page that has the relevant question

Name and Section: Jerry Lee Section 22

Student ID: 861208515

1. What is the "traditional" method of development that is still used for most government projects?

   A. thunderstorm development
   B. iterative development
   C. stair step development
   D. waterfall development
   E. lockstep development

2. What are the three main rolls in Scrum?

   Scrum Master
   Scrum Team
   Product Owner

3. In scrum, a span of time in which tasks assigned to workers cannot be changed is known as what?

   Sprint

4. What are the three C's of user stories?

   Confirmation
   Conversation
   Card

5. A large user story that cannot be worked on until it has been broken down is known as what?

   Epic

6. If you called the following commands in the following order

```
git init
touch README.md
git commit -m "initial push"
```

   and your local hard drive was destroyed, would you be able to recover your code from GitHub (assuming your git client was configured correctly and there were no errors with the above commands)?

   A. Yes
   B. No

7. You are tasked with designing an order tracking system for a large shipping company using the **composite pattern.** This company ships items in three ways: individually, in boxes, or in trucks; which make up the types of orders your system will need to handle. Trucks can hold any combination of boxes and individual items (contained within an internal `vector`), and boxes can hold any number of individual items (contained within an internal `vector`). Each individual item has an `int` upc that identifies what product it represents.

   Any callers in the system will use an `Order` reference (defined below), and shouldn't need to know the difference between individual item, box, and truck orders. All orders should have a `vector<int> scan()`, which when called should return a `vector<int>` containing all the upc codes for all the items in the order.

   Using the `Order` class below as a base, write all the other necessary classes and functions to create the above described system.

```
class Order {
    public:
        virtual vector<int> scan() = 0;
}
```

```
class Trunk : public Node {
  private:
      vector<int> branches;
      Node* trunk;


  public:
      Trunk() : Node() {};
      Trunk(Node* trunk) : Node(), trunk(trunk) {};
      void add_branch(Node* branch) {
          this->branches.push_back(branch);
      }
       void remove_branch(Node* branch){
           for (unsigned int i = 0; i < branches.size(); i++){
               if (branches[i] == branch)
                   branches.erase(branches.begin() + i);
                   return;
               }
           }
       }

       vector<int> [illegible] {
           vector<int> [illegible];
           for (unsigned int i = 0; i < branches.size(); i++){
               for (unsigned int j = 0; j < [illegible]; j++){
                   [illegible].push_back([illegible]);
               }
           }

           return [illegible];
       }
```

8. Next, you will use the **strategy pattern** to create a system for fulfilling the orders you created in the previous question in multiple ways. Fulfillment is the process of taking items out of the warehouse and packing them for shipment.

   The company has two different types of fulfillment warehouses (automated and manual) which require two different methods for performing the fulfillment. The manual warehouse requires the `int` upc of each item to be printed (using `std::cout`) as an `int`. The automated warehouse requires the `int` upc of each item to be printed as a `hex` value (you can use `std::cout << std::hex <<` to print an `int` value in `hex`). All types of order can be fulfilled at either type of warehouse, and each order type should be able to be fulfilled with a call to a new `void fulfill()` function.

   Below, create the classes and functions you need to perform the two methods of fulfillment, as well as any changes to previously defined classes that you would need to make this new fulfillment system work properly. For previously defined classes, simply write which class is being changed and the new code, it will be assumed all previous code from that class remains unchanged (you do not need to re-copy it here).