CS 100 – Software Construction
Exam 1

July 10, 2019

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

Name: _____

Student ID: _____

1. (3 points) Which of the following is **not** a step in the waterfall method?
    A. Requirements
    B. Design
    C. Development
    D. Testing
    E. Iteration

2. (3 points) Which of the following is **not** an agile principal?
    A. Individuals and interactions over processes and tools
    B. Working software over comprehensive documentation
    C. Small teams over large organizations
    D. Customer collaboration over contract negotiation
    E. Responding to change over following a plan

3. (5 points) What property of Kanban is meant to increase throughput?

    Solution: Limiting work in progress

4. (3 points) Which of the following are scrum rolls (circle all that apply)?
    A. Scrum Master
    B. Scrum Operator
    C. Quality Assurance (QA)
    D. Scrum Team
    E. Product Owner

5. (3 points) What is the name of daily meeting typical in scrum practics?
    A. Plank meeting
    B. Daily sync meeting
    C. Support meeting
    D. Standup meeting
    E. Scrum is not associated with a daily meeting

6. (3 points) Which of the following types of tests are seen by users?
    A. Unit Testing
    B. Integration Testing
    C. System Testing
    D. Smoke Testing
    E. Acceptance Testing

7. (5 points) What is the process by which previously developed functional and non-functional tests are re-run to make sure that previously developed and tested software still functions correctly (performs) after a change?

Solution: Regression testing

8. (5 points) What is the process by which changes merged into the main branch are validated by running automated tests against them?

Solution: Continuous Integration (CI). I will also accept Continuous Delivery or Deployment, although that is not technically correct.

9. (3 points) Which of the following are general stages of unit testing (circle all that apply)
   - **A. Arrange Phase**
   - B. Apply Phase
   - **C. Act Phase**
   - D. Accept Phase
   - **E. Assert Phase**

10. (2 points) Drivers are typically used in a bottom-up approach, true or false?
    - **A. True**
    - B. False

You have access to a GIS system similar to the one described in the composite pattern exercise. It is made up of an abstract base class `GISNode` which has two pure virtual public functions:

- `Point find_closest_poi(string name, Point location)` which returns the `Point` of the closest point of interest to the parameter `location` with a name matching the parameter `name`
- `vector<Point> find_all_poi(string name)` which returns the `Point`s of all the points of interest with a name matching the parameter `name`

This `GISNode` class is the parent class for a composite class `Area` and a leaf class `Poi`.

The `Area` class represents some area of the map and represents everything from the entire world to one city block. It contains a `vector<GISNode*> children` private member which holds all the data contained within the area. It overrides the inherited function `find_closest_poi` such that it will return the closest point to the `location` parameter of all of it's `children` `find_closest_poi` returned `Point`s, and if none of those values are valid it will return the invalid point `Point(-1,-1)`. It overrides the inherited function `find_all_poi` such that it will return the sum of all of it's `children` `find_all_poi` returned `vector<Point>`s.

The `Poi` class represents a single point of interest on the map. It contains `string name` and `Point location` private members which represent the name of the point of interest and it's location on the map (respectively). It overrides the inherited function `find_closest_poi` such that it will return the the `Poi`'s `location` assuming the `name` parameter to the function matches the `name` private member, otherwise it returns the invalid point `Point(-1,-1)`. It overrides the inherited function `find_all_poi` such that it will return a `vector<Point>` containing only the private member `location` as long as the `name` parameter to the function matches the `name` private member, otherwise it returns an empty `vector<Point>`.

All the code for the above classes except for their constructors and destructors has been provided at the end of this exam for you to reference if necessary.

You can assume there is a pre-defined struct `Point` which represents a map location and has correctly overloaded `==` and `!=` operators. For all `Point` classes you can assume that the value `(-1,-1)` is reserved to represent an invalid `Point`. You can also assume that the function `double distance(Point a, Point b)` exists which returns the absolute distance between two `Point` objects.

11. (65 points) Using the system described above, you will use the **decorator pattern** to create a class `Cache` which will be used to reduce the number of function calls necessary in the system to retrieve certain types of data. Caches are a system where copies of commonly requested data are stored in a way that makes them faster to retrieve than the traditional method. Your cache will be used to reduce the number of function calls needed to perform a `find_all_poi` call with one specific `name` parameter to part of the system. The cache will only be able to reduce the number of function calls necessary to `find_all_poi` calls with the same `name` parameter, and the `name` value they cache for will not change after the `Cache` object is constructed (although additional `Cache` objects for other `name` parameters can be created). An object of `Cache` should be capable of being placed anywhere in the composite structure described above and should not modify the execution of any functionalities except `find_all_poi`. It should be able to immediately return the results that would normally be returned from a `find_all_poi` call to the portion of the system it is caching for **without any additional calls to the portion of the system it is caching for**. In order to initialize the cache with the proper values and update it when necessary `Cache` should have a function `refresh_cache` which **is allowed** to make calls to the portion of the system it is caching. This function can have any return value and parameters necessary. This function will only be called by the system which constructs and places `Cache` objects and should be accessible only to that system and **not** to the client of the GIS system. This function will be called infrequently while calls to `find_all_poi` will be called frequently, leading to a amortized reduction (meaning reducing as the number of calls grows) in function calls to the system. You do not need to implement constructors or destructors for your class but must have any members necessary for your class to function. If you assume some members are filled during construction please write your assumption as a comment above, below, or next to the member.

Write the code necessary to implement the previously described cache class below

Solution:

```
class Cache : public GISNode {
    private:
        GISNode* child;
        vector<Point> locations;
        string name;
    public:
        virtual Point find_closest_poi(string name, Point location) {
            return this->child->find_closest_poi(name, location);
        }
        virtual vector<Point> find_all_poi(string name) {
            if(this->name == name) {
                return this->locations;
            }
        }
        void refresh_cache() {
            this->locations = this->child->find_all_poi(this->name);
        }
};
```

```
class GISNode {
    public:
        virtual Point find_closest_poi(string name, Point location) = 0;
        virtual vector<Point> find_all_poi(string name) = 0;
};

class Area : public GISNode {
    private:
        vector<GISNode*> children;
    public:
        virtual Point find_closest_poi(string name, Point location) {
            Point closest = Point(-1,-1);
            for(int i = 0; i < this->children.size(); i++) {
                Point child_location = this->children.at(i)->find_closest_poi(name, location);
                if(child_location != Point(-1,-1)) {
                    if(closest == Point(-1,-1) ||
                       distance(closest, location) > distance(child_location, location)) {
                        closest = child_location;
                    }
                }
            }
            return closest;
        }
        virtual vector<Point> find_all_poi(string name) {
            vector<Point> locations;
            for(int i = 0; i < this->children.size(); i++) {
                vector<Point> child_locations = this->children.at(i)->find_all_poi(name);
                for(int j = 0; j < child_locations.size(); j++) {
                    locations.push_back(child_locations.at(j));
                }
            }
            return locations;
        }
};

class Poi : public GISNode {
    private:
        Point location;
        string name;
    public:
        virtual Point find_closest_poi(string name, Point location) {
            if(this->name == name) { return this->location; }
            return Point(-1,-1);
        }
        virtual vector<Point> find_all_poi(string name) {
            vector<Point> ret;
            if(this->name == name) { ret.push_back(this->location); }
            return ret;
        }
};
```