Software Engineering Project

Part 3 - Domain Analysis (cont.): Defining Operations, Attributes, and Inheritance

The objective of this assignment is to proceed with the Domain Analysis of the Ice Skating Winter Olympic Scoring System by

1. modeling the ***Use Cases,***
2. identifying the ***Operations*** required for each Class,
3. assigning ***Attributes*** to each Class, and finally
4. by defining ***Inheritance***.

Use the description of the ***ISWOSS*** software system you received in Part 1 of the Project.

There are 3 separate steps indicated in this part of the project. Perform each step in the order listed here. While I understand that there are several elements that will be discovered in one step that will be useful in another step, it is very tempting to move the attention from one part of the analysis to another and to lose focus. In designing very large systems this may be cause of errors and missing parts. So it is imperative that you focus on each step and perform the analysis from the different points of view, the one of the use cases modeling, the one of the operations, and the one of the attributes and inheritance.

Before starting the first step let's prepare some material.

First Things First: Prepare a template code for each class (10% of the grade)

In the Project#2 you have identified a set of classes. You have also been asked to create the code for each of the classes identified for the system. Complete this step before you move to the next step. The classes at this point are almost empty since you have just identified their names and their constraints. While you will go through the remaining steps of this project, you will continue filling these classes with the operations and attributes that you will discover during the domain analysis you perform in this part of the project. While coding your classes, use the following style:

- **Coding Style** - Each class will consist of a *xx.h* file (see *Thing.h*) with the class definition, and a *xx.cpp* file (i.e. *Thing.cpp*) with the definition of the methods. An example of Coding Style has been given already in part 2.
- **Use Doxygen Commments** - Doxigen is a documentation system for C++, C, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP, and C#. Use Doxygen comments in your code. Instructions on how to use Doxigen have been already given in Part 2 of the project.

- **No main** - In OOP programs, the main (driver) is not that important unlike in structured programs. Use main mostly to test the code and then it can be thrown away since it will not be part of the final system.
- **Make sure it compiles** - Compile each class .cpp file to object code. Look at the example Makefile for the g++ command to use.
- **No Constructors** - Do not worry or implement constructors of any kind at the moment. If you need functionality, create a method. We are looking at high-level design (although expressing it directly in C++). The choice of whether to implement something in a constructor or a member function (or even a free function) is a lower-level design issue.
- **Empty Methods** - Do not implement the methods. Put enough in so that they compile, (e.g., return 0;).
- **Polish your design** Spend time on issues such class names, method names, parameters, etc.. Keep in mind the issues of object models including abstraction, encapsulation, modularity, hierarchy, typing, concurrency, and persistence. Consider how your design would affect how the program could be worked on by multiple programmers, and for testing individual parts. **This is VERY important!**

In addition to this in this part of the project you will be asked to produce some deliverables and to produce a printed version of them. **Be sure you add the date at the bottom of each page that gets printed** to keep track of the evolution process.

Deliverable #1 (10%) - Prepare the template of each class. Use Doxigen Comments for future easy documentation preparation.

Step 1 - Model Use Cases (40% of the grade)

This step is important to assign operations to the classes.

There are 2 ways to assign operations:

1. Define scenarios for each of the use cases in the system function statement and determine the operations that are needed to carry out those scenarios.
2. Examine each class and determine what is required.

The guidelines for designing Operations are:

- Have each operation perform one simple function
- Name each operation with a specific name that reflects the outcome fo the function, not its steps.
- Avoid having too many input and outputs – this may indicate many functions that should be split into separate operations
- Avoid input switches – they are often a sign of non primitive functions.

At this point model your use cases

1. Retrieve your system function statement and expand each Use Case into a detailed scenario to identify the operations needed to realize that use case. Modeling scenarios shows which objects collaborate in the Use Case and identifies operations needed in each object.
2. Decorate each Use Case with the use of a script as shown in your book Use Case example (see page 179 of your book).
3. While stepping through the scenario, several operations can be discovered. Use the operations discovered to update the Class Diagram. In this Class Diagram you will connect the classes with the relationship expressed by the discovered operation(s). In other words you will connect two classes by labeling the "uses" relationship with the name of the operation being used.
4. Update the code of each C++ class by adding the operations identified for each corresponding class.
5. **Polish your design** - Spend time on issues such method names, parameters, etc. Consider how your design would affect how the program could be worked on by multiple programmers, and for testing individual parts.

For example, assume you have to model a system in which a judge is judging a students' poster competition. You may have previously identified the class *JUDGE* and the class *COMPETITION*. While using a Use Case you have just identified the operation *is_assigned()* that assigns a judge to a poster competition. Then do the following:

- In your new Class Diagram label the connection between *JUDGE* and *COMPETITION* with the "uses" relationship from the *COMPETITION* to the *JUDGE* and label it with the label *is_assigned()*.
- Insert in the C++ class *JUDGE*, the method *is_assigned()*.

- **Deliverable #2** (15%) - Create a new Class Diagram with the fully annotated operations required to carry out the use cases in the system function statement. Print a copy of the new Class Diagram and insert it into the 3 ring folder.

- **Deliverable #3** (25%) - Prepare a report containing the Use Case Diagrams in UML required for your project and all the scripts associated with those Diagrams (follow the style of the example of the book). Print your report and insert it into the 3 ring folder.

PROGRESS SO FAR: After completing this part you have:

- Discovered the major operations required to carry out the use cases in the system function statement
- Documented the needed inputs and outputs of those operations
- Assigned the operations to the appropriate class.

DELIVERABLES – **Release 3.1)**

- A class diagram showing the operations and code preparation (10% - which corresponds to Deliverable #1)

- Class diagram showing the “uses” relationships (15% - which corresponds to Deliverable #2)
- Use Case Diagrams showing the key scenarios (25% - which corresponds to Deliverable #3)

Step 2 - Defining attributes and inheritance (50% of the grade)

Part of the domain analysis is the identification of the ***attributes.*** The attributes are the properties that describe the class. An attribute is equivalent to an ***aggregation*** association where the label is the attribute name and the cardinality is exactly one.

The identification of the inheritance is also another very important part of the domain analysis. If, while discovering classes, you have found yourself saying "This class is almost the same but different" or "This attribute applies to most of the instances of this class but not to all of them", these are situations in which you have found that 2 classes characterize an instance: one class that is specific to the type and one that is more general. In other words, you have discovered a **superclass** (a more general class) and a **subclass** (a more specific one).

1. Identify attributes of each class and the type of each attribute (ex. *string*, a *int*, a *date* class, a *date_range* class, etc.). Add them to the C++ classes.
2. Identify inheritances, add the inheritance description in the C++ classes and add the generalization relationship to the same Class Diagram you have created in Step 1.
3. **Polish your design** Spend time on issues such class names, attribute names, etc.. Keep in mind the issues of object models including abstraction, encapsulation, modularity, hierarchy, typing, concurrency, and persistence.

PROGRESS SO FAR: After completing this part you have:

- Located most data attributes of the key abstraction
- Discovered and assigned the types to these attributes
- Discovered some additional key abstractions and relationships
- Discovered some relationships that will become classes because they have data attributes
- Found any subclasses and superclasses

DELIVERABLES – **Release 3.2)**

Deliverable #4 (10%) - Create an augmented class diagram that shows the inheritance structure, newly discovered classes and relationships, and cloud compartments showing important properties of the classes.

Deliverable #5 (10%) – Updated class diagram with the specifications for the new classes and relationship discovered.

Deliverable #6 (10%) - Update the code for your classes by adding the attributes you have discovered and by creating the newly discovered classes. Perform any additional modification that you have discovered so far.

Deliverable #7 (10%) - **Required Readme file** - Prepare a small "Readme" file in which you discuss your experience in the group so far. Include any alternative decisions that you had to make, or issues that you faced, or domain analysis decisions that changed over time. Also indicate how the work has been distributed in the group and how the teamwork has been organized. If changes have occurred over time indicate the data of the change. The ReadMe file should also guide the instructor through the final folder report that contains all the deliverables by providing and index (for example, the list of the folder and what each folder contain) and indicate any information that many be necessary to communicate to the instructor while reading the final report of the deliverables. Add any information generated in the template report that has been given to you in the part 1 of the project.

Deliverable #8 (10%) – Print the documentation developed in this part of the project by running Doxigen in the folder with all the C++ classes generated. Use the html file produced by the application, print it and attach it to your report.

Step 3 – Validate your Model and Iterate

To validate the model, you check that the abstraction, operations, and relationship are sufficient to allow you to implement a system that satisfies your charter statement, which you produced from the requirement analysis.

One method of validation is to pick one or more key use cases of the system and walk through each path, noting all operations that need to be performed.

As you probably have noticed, object oriented analysis is not as rigidly ordered as the steps required in this project. Once you have experienced you can combine several steps at once, and you will iterate though several these steps several times before the analysis is complete. Then, when do you stop your iteration? You will stop when you have accomplished these goals:

1. You have identified all domain entities that will play a role and defined their classes.
2. You have specified the relationships between each of these classes.
3. You have associated with each class all the operations performed by or on it.
4. You have analyzed each operation to the point where you understand what it needs to do and what other classes are involved.

Instructions for electronic submission - Collect all the deliverables in a folder called *GroupNumberPart3,* zip the folder and attach it to the project assignment before the deadline.

The **grading rubric** of the project has been given in detail in each of the above steps.

Deadline: 11:59 pm, April 6, 2018