Concordia University
Department of Computer Science
and Software Engineering

Advanced program design with C++
COMP 345 --- Fall 2016

Individual assignment #3

Deadline:	Friday, November 27rd, 2016
Evaluation:	5% of final mark
Late submission:	not accepted
Teams:	this is an individual assignment

Problem statement

This is an individual assignment. It is divided into four distinct parts. Each individual student is expected to select one of these parts as his/her assignment. Each part is about the development of a part of the topic presented as the team project. Even though it is about the development of a part of your team project, each assignment has to be developed independently of the others and is not to be presented as an integrated part of the team project, or include the implementation of one another's aspects. Each member of your team is free to choose to do any part, and is expected to follow a different design approach than the other team members that have selected the same assignment topic. Note that the following descriptions describe the baseline of the assignment, and are related to the project description (see the course web page for a full description of the team project).

Part 1: Character builder

Implement a Builder pattern for the Character class to create characters (player character or enemy character) of various levels (fighter class only), and enabling various types of fighter style to be chosen.

Ability scores generation method: Any character has the same 6 ability scores (Strength, Intelligence, Dexterity, Constitution, Wisdom, Charisma). Upon creation of the character, the values associated with each ability score is randomly determined. For the generation of ability scores, for each ability score, roll 4d6 and selects the 3 highest dice values. After all 6 scores have been generated, they are assigned to an ability depending on the type of fighter that this character is: (1) a *"bully"* uses brute strength to destroy his enemies, (2) a *"nimble"* favors dexterity and better armor class to evade blows, (3) a *"tank"* favors survival by more hit points through a high constitution score. Create one Concrete Builder for each of these three types of fighter.

Type of fighter	Ability scores in decreasing order of importance
Bully	Strength, Constitution, Dexterity, Intelligence, Charisma, Wisdom
Nimble	Dexterity, Constitution, Strength, Intelligence, Charisma, Wisdom
Tank	Constitution, Dexterity, Strength, Intelligence, Charisma, Wisdom

Level-dependent characteristics: As a character goes up levels, the following are increasing: (1) his hit points go up by (1d10+constitution modifier), (2) his attack bonus goes up by one, and his number of attacks per round increase by one every five levels, according to the following table:

level	1 att/round
1	+1
2	+2
3	+3
4	+4
5	+5

level	2 att/round
6	+6/+1
7	+7/+2
8	+8/+3
9	+9/+4
10	+10/+5

level	3 att/round
11	+11/+6/+1
12	+12/+7/+2
13	+13/+8/+3
14	+14/+9/+4
15	+15/+10/+5

level	4 att/round
16	+16/+11/+6/+1
17	+17/+12/+7/+2
18	+18/+13/+8/+3
19	+19/+14/+9/+4
20	+20/+15/+10/+5

Part 2: Character Strategy

Implement the character actions (either player or non-player) as a Strategy pattern. Each turn, a character is allowed to proceed with a move, an attack, and free actions. Implement 3 different ConcreteStrategies: 1) a HumanPlayerStrategy that lets the user decide where to move, who to attack, and what free actions to take; 2) an AggressorStrategy that make the character automatically move towards and attack the player character; 3) a FriendlyStrategy that makes the character automatically move towards the character, but not attack unless attacked, in which case it adopts the AggressorStrategy.

Part 3: Items as Decorators

Implement worn items using a Decorator pattern, where the items are a subclass of the Character class and provide an overridden version of the accessor methods for the characteristics that they augment. The decoration mechanism should allow at most one of each item type to be worn, as well as to provide a remove() method to remove any item currently worn.

Part 4: Game logger as Observer

Implement a game log using the observer pattern. The log should record game events from different observables in real-time as they are happening: 1) the game controller records the events happening in every game phase (game setup, map loading, player turns switching, end phase) 2) the map records every movement made on the map, 3) the character records every attack attempted and its result 4) the dice records every dice roll. All the log entries should be recorded in a unified log. It should be possible to switch on/off any of the different logging sources.

Assignment submission requirements and procedure

You are expected to submit a group of C++ files implementing a solution to one of the problems stated above (Part 1, 2, 3 or 4). Your code must include a *driver* that allows the marker to compile and execute your code on a standard lab computer. The driver should simply create a character, map, item container, or dice object and somehow demonstrate that the character, map, item container or dice was created following the above-mentioned specifications, as well as following the applicable d20 game rules.

For each part, you must design a CppUnit test suite composed of at least two relevant test cases. The test cases you provide must be documented. The header file(s) of your implementation must include in Doxygen documentation a statement of: 1) the game rules involved in their respective implementation, 3) a brief description of your design, 3) the libraries used in the code, including a rationale for the selection of these libraries. The submitted code must include a driver (i.e. a main function) that demonstrates possible uses of the implemented code. The focus of this course being the coding aspect of software development, you are discouraged to submit external documentation.

You have to submit your assignment before midnight on the due date using the ENCS Electronic Assignment Submission system under the category "programming assignment 2". Late assignments are not accepted. The file submitted must be a .zip file containing all your code. You are allowed to use any C++ programming environment as long as they are usable in the labs. No matter what programming environment you are using, you are responsible to give proper compilation and usage instructions to the marker in a README file to be included in the zip file.

Evaluation Criteria

Analysis:	
Completeness/correctness of statement of game rules involved (Doxygen) :	5 pts (indicator 4.1)
Design:	
Compliance of solution with stated problem and game rules:	15 pts (indicator 4.4)
Simplicity and appropriateness of the solution:	7 pts (indicator 4.3)
Clarity/completeness of Doxygen documentation:	3 pts (indicator 7.3)
Use of tools:	
Rationale for selection of tools/libraries used:	2 pts (indicator 5.2)
Proper use of language/tools/libraries:	3 pts (indicator 5.1)
Implementation:	
Code readability: naming conventions, clarity, use of comments:	2 pts (indicator 7.3)
Coding style: `.h` and .cpp files:	2 pts (indicator 5.1)
Relevance of test cases provided:	4 pts (indicator 4.4)
Relevance of driver and completeness of presented results:	7 pts (indicator 4.4)
Total	**50 pts (indicator 6.4)**