SOEN6461: Take Home Exam Designing and Implementing (Some of) Dungeon and Dragons Character Classes

Yann-Gaël Guéhéneuc
Opening date: 2020/04/23
Submission date: 2020/04/30

Terms and Conditions

a) Following material is permitted: lecture notes, Moodle material, StackOverflow, Academic Code of Conduct and Concordia University's Academic Integrity, any Web site about Dungeon and Dragons.
b) Online searching of subject material/related exam questions is NOT permitted.
c) Communicating with classmates regarding any aspect of the exam or course is NOT permitted.
d) Posting or sharing the exam content, including exam questions, or your answers both during and after submission is NOT permitted

Questions During the Exam

Do not hesitate to contact me via e-mail or Moodle at any time. I will do my best to answer any question within 24h.

Academic integrity

Concordia University takes academic integrity very seriously and expects its students to do the same. I understand that any form of cheating, or plagiarism, as well as any other form of dishonest behaviour, intentional or not, related to the obtention of gain, academic or otherwise, or the interference in evaluative exercises committed by a student is an offence under the Academic Code of Conduct. I understand that the above actions constitute an offence by anyone who carries them out, attempts to carry them out or participates in them. By way of example only, academic offences include: Plagiarism; Contribution by a student to another student's work with the knowledge that such work may be submitted by the other student as their own; Unauthorized collaboration; Obtaining the questions or answers to an exam or other unauthorized resource; Use of another person's exam during an exam; Communication with anyone other than the instructor/invigilator during an exam and any unauthorized assistance during an exam; Impersonation; Falsification of a document, a fact, data or a reference. For more information about academic misconduct and academic integrity, refer to the Academic Code of Conduct and Concordia University's Academic Integrity webpage.

Problem Statement

Wikipedia reports that “Dungeons & Dragons (commonly abbreviated as D&D or DnD) is a fantasy tabletop role-playing game (RPG) originally designed by Gary Gygax and Dave Arneson. It was first published in 1974 by Tactical Studies Rules, Inc. (TSR). The game has been published by Wizards of the Coast (now a subsidiary of Hasbro) since 1997. It was derived from miniature wargames, with a variation of the 1971 game Chainmail serving as the initial rule system. D&D's publication is commonly recognized as the beginning of modern role-playing games and the role-playing game industry.

D&D departs from traditional wargaming by allowing each player to create their own character to play instead of a military formation. These characters embark upon imaginary adventures within a fantasy setting. A Dungeon Master (DM) serves as the game's referee and storyteller, while maintaining the setting in which the adventures occur, and playing the role of the inhabitants of the game world. The characters form a party and they interact with the setting's inhabitants and each other. Together they solve dilemmas, engage in battles, and gather treasure and knowledge. In the process, the characters earn experience points (XP) in order to rise in levels, and become increasingly powerful over a series of separate gaming sessions.” (Emphasis mine.)

We want to create character generator: a tool with which a player can create xyr own character, from a set of given character classes (e.g., the Ranger), a set of abilities, and pieces of equipment. Instead of using numerical values to characterise the character’ abilities, the system will offer explicit, first-class abilities (e.g., “intuition” or “leadership”). The system will also offer to equip characters with different internal and external pieces of equipment or capabilities (e.g., armours, weapons, bags… but also “stealth” or “shapeshifting”).

Rules

Explanations and Justifications Matter!

In every justification, report the (abstract) design problem to solve, discuss possible alternative solutions, and explain the trade-offs of each solutions before choosing one solution.

Favour short explanations. Be careful of grammar and vocabulary. In particular, use the proper terms as seen during the course.

If necessary, draw short but illustrative class and sequence diagrams. (You can draw these diagrams by hand if it is faster than by computer).

Design and Code Quality Matter!

Use the Maven project given to you as basis for your code.

Use Java 7. Do not worry about generics or the latest features of the Java language.

Make sure to choose the most appropriate design.

Make sure to write simple and clean code.

Make sure that your code compiles and runs.

Modify the Client class to illustrate via one or more examples your design/implementation.

Package types (interfaces) and classes appropriately.

Make types, classes, methods, and fields visible appropriately.

Name packages, classes, types, methods, fields, parameters, and local variable appropriately.

Deadlines and Submission

Submit your answers (this file with your answer in PDF as “Report”) and your code (five ZIP files “Question N.zip”) via Moodle before Thur. Apr. 30th, 11:59pm. No other format/names will be accepted.

Question 1 (20 pts): The given code provides examples of character classes (e.g., Ranger). Innate abilities are numerous, but we will consider the following ones:

- Strength: burly, fit, scrawny, plump
- Dexterity: slim, sneaky, awkward, clumsy
- Constitution: strong, healthy, frail, sick
- Intelligence: inquisitive, studious, simple, forgetful
- Wisdom: good judgement, empathy, foolish, oblivious
- Charisma: leadership, confidence, timid, awkward

Redesign the given code to allow adding abilities to the characters so that new "values" for innate abilities could be added later without having to modify (much) the implementation of the character classes (e.g., a new Constitution: invincible). Before implementing your design, justify your choice in the space below (no more). Then, implement your design based on the given code. ZIP it as "Question1.zip".

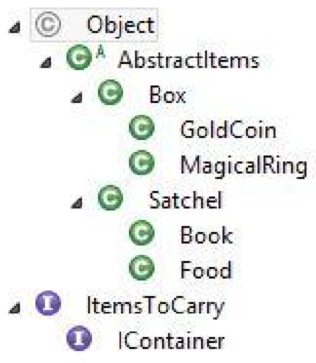
As we need to have a character generator, I think the best fit would be Factory DP and for this question as we need to have these innate abilities values to be added later freely without having much modification in character classes, the best design pattern will be **Visitor** design pattern so each ability (concrete element in visitor DP) can accept a new value by a visitor which can add a new value to the list of values of any wanted ability.
I was first thinking about the extension design pattern and to design adding values as a new extension to our character objects but I realized that it is not good enough because it will not give me a good new operation for adding a new value and it is mostly more useful when you want each object choose the extensions they want to offer.
As it is mentioned in the lecture slides of visitor design pattern it decouples the data structure from the from the algorithm on the data structures. Visitor lets you define a new operation without changing the classes of the elements on which it operates. So here I can implement the visitor design pattern so without altering the characters classes and attributes we can have a new value for innate abilities like invincible in constitution.
Visit method would add the value and the abilities can accept them as a visitor.
So if client wants to add this new value xe can do so by just calling the accept method of that ability in the main method.
So here my concrete elements will be 6 innate abilities each with one accept method and my concrete visitor will be a class with 6 addValue (actually visit) methods.

Question 2 (30 pts): Characters can wear various types of clothing: boots, hats, helmets, cloaks, armour. Redesign the given code to allow adding types of clothing to the characters in such a way that new types can be added and worn by characters without having to modify (much) the implementation of the character classes. Before implementing your design, justify your choice in the space below (no more) and explain how you enforce that certain types of clothing must be worn before or after other certain types, e.g., how do you enforce that armours must always be on top of clothes? Then, implement your design based on the given code. ZIP it as “Question2.zip”.

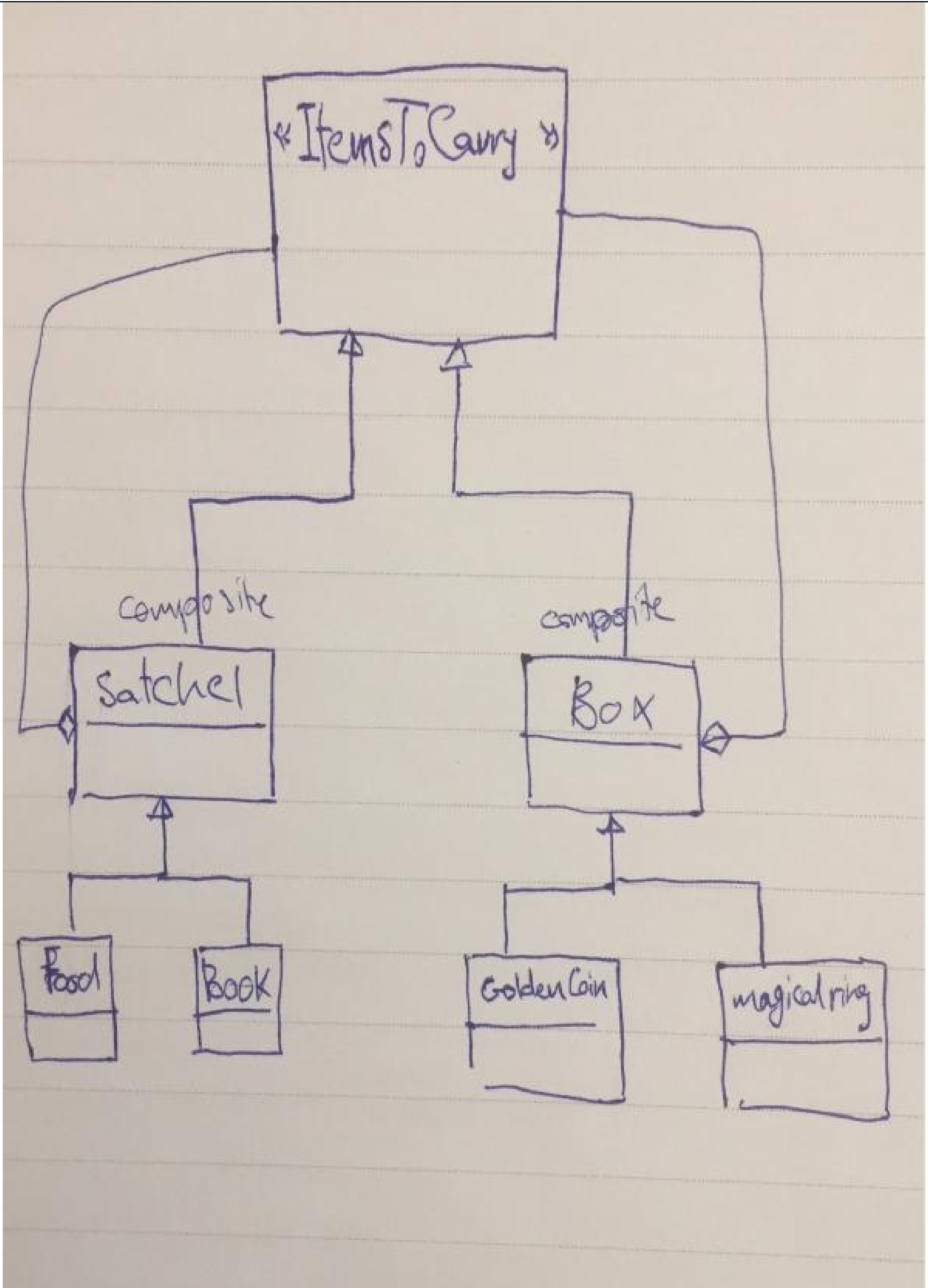
In this Question the main problem is to add new pieces of clothing to the characters without making much changes in the characters classes. It means that we should wrap the object (a character) in to new layers of clothes. So if we think of the character as the center of an onion the clothes will be the layers on that. So the answer will be **Decorator** design pattern.
As we want to enforce an order for wearing each clothing It means that we have to prioritise the clothing .I think the best way to do that is to have a **Template method** pattern and same as the lecture slides that we had some pre-process sort operations and some post-process sort operation and we made a framework for that as a template method in the decorator class , this is the exact same situation here, we are having decorators that should be applied on the object in a certain order so in addition to decorator we need **Template method** design pattern.
Based on the UML diagram of decorator pattern each type of character (barbarian, wizard, etc) will be concrete components and each piece of clothes will be concrete decorator which modifies and adds the behaviour of the character object at the runtime. There should be also a decorator class which will have template method pattern with the certain order (I define the order myself) and the hook methods which will have the default implementation in decorator class and the main implementation in the concrete decorators.
By getting the dress of character we will see the string which says what clothes has been put on in what order.

Question 3 (20 pts): Characters can carry various items in satchels or boxes: food items, books, gold coins, rings. Satchels are useful for food items, books, etc. while boxes protect gold coins, magical rings, etc. Separate for the given code, design and implement a hierarchy that offer satchels and boxes in which various items can be stored. Before implementing your design, justify your choice in the space below (no more) and explain how you enforce that certain items can only be put in satchels, e.g., food items, and not boxes (because boxes are too small). Then, implement your design independently of the given code. ZIP it as "Question3.zip".

As we are having two different containers satchels and boxes and we are having different Items inside each of these containers this reminds me of the **composite** design pattern. Same as the lectures that we had set of sort algorithms (internal and external) and we had also each sort algorithm as a leaf.
Here is the same situation there are two different composites : boxes and satchels which can contain certain items each. Like a hierarchy which will represent **part-whole hierarchies** of objects.
Following the structure mentioned in the lecture slides when we want to implement the composite creator inside the factory class we will add all of its items into the composite and this way we will enforce each specific item to be put in the specific container and we will prevent putting items in the container that it shouldn't be stored in. so we build the composite objects and the client will use these composite objects.
I want to have an if statement for each container so if the object(item) that we want to add to each container is not the **instanceof** that container it doesn't allow the user to add that item in that container. Example: `if(item instanceof Box )`I will add that item to the box (gold coins and magical rings classes are extending the Box class so they are instances of Box and they can be added to the box) and otherwise system can throw an exception or show a message that you cannot put this item in the box.
Here you can the Hierarchy that I created in my code.



```
Object
  AbstractItems
    Box
      GoldCoin
      MagicalRing
    Satchel
      Book
      Food
ItemsToCarry
  IContainer
```

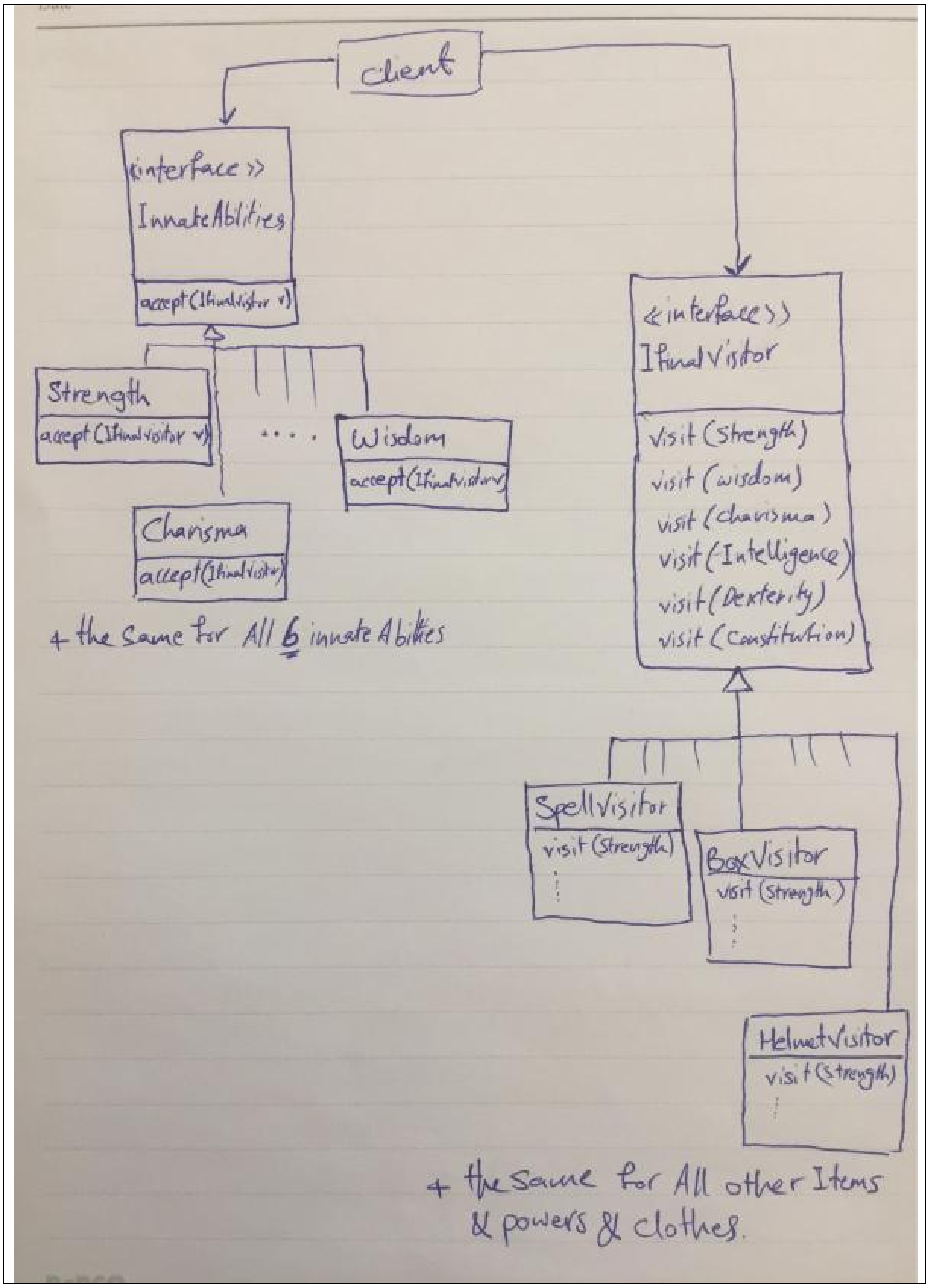

«ItemsToCarry»
composite
Satchel
composite
Box
Food
Book
GoldenCoin
magicalring

Question 4 (15 pts): Redesign the given code to allow characters to carry satchels and boxes (themselves possibly containing various items) and to allow characters to possess powers, e.g., spells, infravision, summons, etc. Before implementing your design, justify your choice in the space below (no more) and explain how you distinguish items (e.g., satchels) from powers (e.g., infravision). Then, implement your design based on the given code. ZIP it as "Question4.zip".

The main problem in this question is to allow characters to have two different abilities of carrying Items and possessing powers.
First I wanted to implement the Extension object design pattern to add these by extensions, but then I realized that we do not want to add these abilities of carrying Items or possessing powers at runtime and just to one instance of our object of characters so Extension object did not help.
I Also thought about Visitor pattern as it lets you define a new operation without changing the classes of the elements on which it operates so I could add Carry operation and Possess operation but when I looked into the question more carefully I understood that we just want our characters to be able to carry and possess things so I need to actually implement them as an attribute of my character classes and same as the previous question I will implement the **composite** Design pattern I can have lists of the powers to be possess and the Items to be carried as an attribute for all characters. And In the **Factory** class I can create the objects of each Item and each Power and add it to the lists of my characters.
As the type of my Items list are from the type ItemsToCarry and the type of the powers in list is the type PowersToPossess (interfaces), this way I can distinguish Items from powers.
At last in my **AbstractFactory** of the game will use the factories of Items and powers and characters to create a character with proper Items to be carried and powers to be possessed.

Question 5 (15 pts): Characters can wear different types of clothing, carry items, and possess powers, which all combine with their innate abilities. These combinations result in the final values for a character's abilities. For example, a character's final strength could be the sum of the character's innate strength (Question 1) plus the strength added from wearing (or not) armour, helmets, etc. (Question 2) plus the strength added from carrying special items and having powers (Questions 3 and 4). Combine and redesign the given code and your previous code to allow computing a character's final abilities, e.g., strength, without having to modify your design every time the rules of the game change. Before implementing your design, justify your choice in the space below (no more). Then, implement your design based on the given code. ZIP it as "Question5.zip".

The main problem in this question is to add values from other abilities to the innate abilities and to calculate the final value for a character's final innate abilities.
I was thinking about **Extension object** design pattern to add an extension of e.g., strength calculator to each object of for calculating the total strength and I could have 6 extensions (6 innate abilities) but the code of each extension class would get so complex as I had to pass all values of each item and power and piece of cloth to that extension I decided to implement the **visitor pattern** to visit every Item and cloth and power by one object of each innate ability and (6 visit methods for each visitor) an this way when we create an object of our character and we want to calculate the strength we just call the accept methods of the strength and it visits all visitors and collect the values added to strength and by the getStrength() method of the character we will have the total final value for strength .
As it is mentioned in the lecture slides of visitor design pattern it decouples the data structure from the from the algorithm on the data structures. Visitor lets you define a new operation without changing the classes of the elements on which it operates.


client
«interface»
InnateAbilities
accept(IfinalVisitor v)
Strength
accept(IfinalVisitor v)
Wisdom
accept(IfinalVisitor v)
Charisma
accept(IfinalVisitor)
+ the same for All 6 innate Abilites
«interface»
IfinalVisitor
visit (Strength)
visit (wisdom)
visit (charisma)
visit (Intelligence)
visit (Dexterity)
visit (Constitution)
SpellVisitor
visit (Strength)
BoxVisitor
visit (Strength)
HelmetVisitor
visit (Strength)
+ the same for All other Items & powers & clothes.

I have neither given nor received unauthorized aid on this exam and I agree to adhere to the specific Terms and Conditions that govern this exam.

Name	Negar Ashouri
Student ID	40071530
Unique e-mail ID	emailforthiscourse@gmail.com
Signature	N.A