Deliverable 4

For this assignment you will focus on associations, collections, and generics. You will expand off of your core objects you have in Deliverable 3. You will need to write new Weapon and Map classes. You will need to create proper associations amongst all of your classes as appropriate. You will need to create a UML class diagram. You will need to create a WPF form that tests your objects and classes.

Some things that might be useful to understand and use for this implementation will be interfaces, abstract classes, overriding of methods, and calling base objects. Object references in this project should be of the type specified.

You will need classes for the following:

# IRepeatable

The IRepeatable class will need to be modified to have the use a generic <T> type and have the CreateCopy method return a generic type (T).

# Monster

The IRepeatable implementation will need to be specified for type Monster. The CreateCopy method will need to be modified to return the type Monster.

# Potion

The IRepeatable implementation will need to be specified for type Potion. The CreateCopy method will need to be modified to return the type Potion.

# Weapon

A weapon class will need to be created. It will need to inherit from the Item class and it will need to implement the IRepeatable interface specified for type Weapon.

Weapons will not have a default constructor, but will have an overloaded constructor that will accept and set the attack speed modifier in addition to the name and value as did the base object.

Weapons have an attack speed modifier. This is how much the attack speed of a hero is decreased when the weapon is equipped by a hero. This needs a public getter, no setter.

Don’t forget to implement the interface as appropriately. This is similar to the Potion and Monster classes.

# Hero

The HasWeapon field should be removed and replaced with an Equipped Weapon object reference field. This reference will need to have a way to set and get the weapon object.

The HasWeapon setter will need to be removed and the getter will need to reference the equipped weapon field to see if the hero has a weapon equipped. (Hint: checking whether or not there is something in the Equipped Weapon field would be a great way to check this.)

Getting the AttackSpeed will no longer be modified universally by whether or not a weapon is equipped, but will now be modified to use the attack speed modifier of the weapon to decrement the attack speed relative to the weapon’s modifier. This is why a weapon has an attack speed modifier. Make sure to use the field on the Weapon object to modify this value.

For example, if the Hero has a base attack speed of 10 and the weapon modifier is 4, the AttackSpeed modified should return 6. Do not change the base attack speed, only modify the result. (Hint: if the hero does not have a weapon equipped, just return the hero’s attack speed.)

# MapCell

The HasItem field should be removed and replaced with an Item object reference field. This reference will need to have a way to set and get the associated object.

The HasItem setter will need to be removed and the getter will need to reference the item field to see if the mapcell contains an item.

The HasMonster field should be removed and replaced with an object reference field. This reference will need to have a way to set and get the associated object.

The HasMonster setter will need to be removed and the getter will need to reference the item field to see if the mapcell contains a monster.

# Map

The map will contain a collection of all of the map cells, available potions, available weapons, possible monsters, and the hero for our game. All weapons, potions, and monsters available in the game should be in this Map class. Do not set them on the form. The form should be independent of the game.

The map will need a Hero object reference. This will be your hero that will move around the map.

The map will need to contain collections for the following:

GameBoard – all of the MapCells that make up the game board in a two dimensional collection.

Potions – a list of all of the possible Potions that our game has available.

Weapons – a list of all of the possible Weapons that our game has available.

Monsters – a list of all of the possible Monsters in the game.

Each of these collections will need to be exposed through properties but will be set within the Map class. It would be a wise idea to have a method to do each of the following private methods:

FillMonsters – fills a collection with all of the monsters for the game.

FillPotions – fills a collection with all of the potions available in the game.

FillWeapons – fills a collection with all of the available weapons in the game.

FillMap – fills the gameboard with mapcells.

The monsters, potions, and weapons should have unique things which appear in the collection only once. e.g., the weapons list should have dagger, sword, club, etc. only appear once in the collection.

Map will not have a default constructor. Map will only have an overloaded constructor that accepts the number of columns and rows which are found in the game board. The game board is in a grid format. The game board will be will be assigned when map is instantiated. The collections of monsters, potions, and weapons are filled when the map is instantiated.

# Actor

Nothing will be done to Actor in this deliverable, it just needs to be included in the solution.

# Item

Nothing will be done to Item in this deliverable, it just needs to be included in the solution.

# UML Diagram

Include the UML diagram for your project. Make sure to update the diagram to properly show any and all associations.

# Grading

Points will be awarded as to how well the classes match the requirements given in this document. Points can be lost for unprofessional looking code or applications. There are no specific expectations on presentation of your form as long as it is able to show the object behaviors needed and is professional in appearance.

# Test Application

You will need to create a WPF application to test and prove your objects work the way expected. Use Grids, TextBoxes, TextBlocks, Buttons, StackPanels, and any other controls to create a test environment for your code. All existing object behavior is should still be available in your classes, but the application should specifically prove the following:

* A map object is created with a 5x5 map.
* Display all of the possible potions and monsters for your map.
* A button to fill each cell of the map with a random potion from the list of possible potions in your map object. A deep copy should be used.
* The name of each potion in each cell are displayed in a grid.
* Display all of the possible weapons.
* Create a hero object associated to the map object and display the hero’s stats. You will need to specifically show whether the hero has a weapon equipped and the hero’s attack speed.
* Give the hero in the map object a different weapon and update the display accordingly.

Here is an example of what your form might look like:













