Deliverable 3

For this assignment you will focus on inheritance, abstraction, and interfaces. You will expand off of your core objects you made in Deliverable 2. You will need to write an IRepeatable interface and new classes for Hero, Monster, and Potion which will inherit from exiting classes you wrote already. You will also need to create a UML class diagram, as well as a WPF form to test 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.

You will need the following classes:

# Actor

The Actor class from Deliverable 2 will need to be converted to an abstract class.

# Item

The Item class from Deliverable 2 will need to be converted to an abstract class.

# IRepeatable

The IRepeatable interface will be used on items that can have several things of the same type on a map. You will need to create this interface. This interface will require a method to create a deep copy of an object. The interface needs to require a method that should be named something like CreateCopy and should return type object.

# Hero

A Hero class will need to be created.

Hero is the player character that will be moving around the dungeon. The Hero class will need to inherit from Actor.

Heroes can have a weapon equipped, so you will need to track whether a hero has a weapon equipped or not. A Hero does not start with a weapon. We will worry about which weapon is equipped later; for now we just need to know whether or not a weapon is equipped.

The hero constructor will need to call the base constructor and require all of the same parameters as the base constructor and it will pass all of these values to the base constructor.

Hero will need to override the AttackSpeed property/method on the base class. If the hero has a weapon equipped, then return the attack speed as half of the base attackspeed.

The move method on the base class will also need to be overridden. Make sure to call the move method from the actor class in the hero class move method. We will make more use of this in later deliverables.

# Monster

Monster is the class for fightable creatures on the map. Monster will need to inherit from Actor and implement the IRepeatable interface because we can have multiple monsters of the same type on the map at the same time.

Monster will need an attack value, this is how many points of damage that they can inflict on a hero.

Monster will also need an overloaded constructor that will need all of the information to pass to the base object, plus the attack value.

Monster will need to implement the CreateCopy method required by IRepeatable to create a Deep Copy.

# Potion

Potions is the class for items that can heal a hero. The potion class will need to inherit the item class and implement the IRepeatable interface because multiple items of the same type can appear on a map.

Potions have a color. This color could be a system media color, but a string name is also fine.

Potion will need an overloaded constructor that will need all of the information to pass to the base object, plus the color.

Potion will need to implement the CreateCopy method required by IRepeatable to Create a Deep Copy.

# MapCell

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

# UML Diagram

Include the UML diagram for your project.

# 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 your objects. The test application needs to create an object for Hero, Monster, and Potion. Use TextBoxes, TextBlocks, Buttons, and any other controls to create a test environment for your code. The applications should prove the following:

* All existing object behavior is still available.
* A hero object is created and displayed.
* A hero can have a weapon equipped and by equipping the weapon the attackspeed is cut in half.
* A monster object is created and displayed.
* The potion object is created and displayed.
  + Multiple potions can be created using the CreateCopy method.
    - Each potion created must be displayed on the form.
* The first potion object must be changed in the list proving that the CreateCopy method performs a deep copy.
* I have added graphics, these are not required.

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



 



