Programming 3

UCN – Computer Science - C#

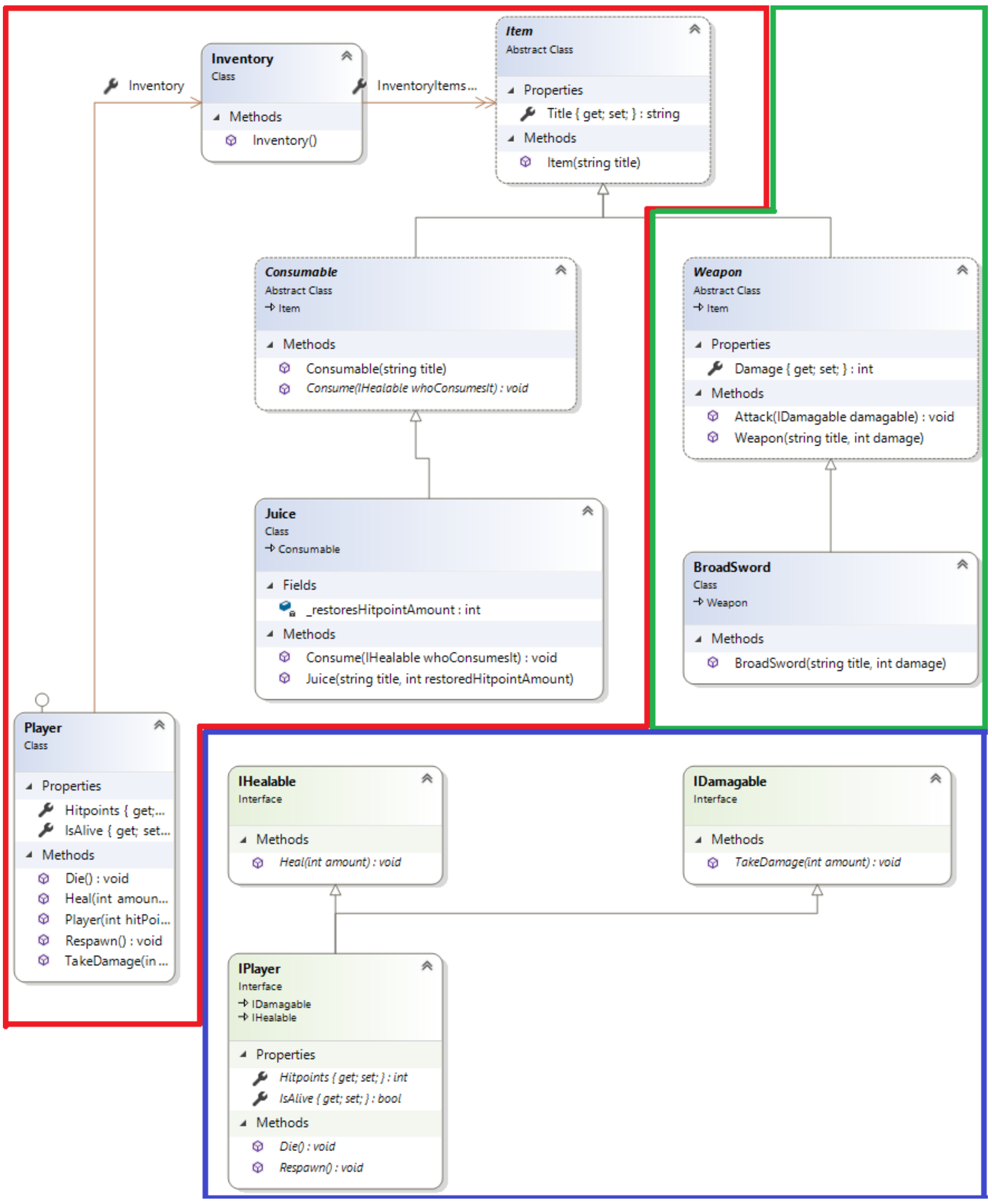
Inheritance Game

Content

1. Class Diagram

Exercise contains:
abstract classes and methods, interfaces, properties, methods, console application, class library, assembly references, for / foreach

Diagram for game



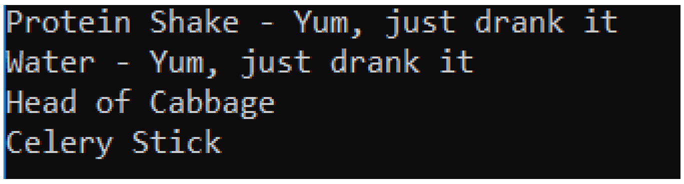
2. Part 1 – Game structure

1. Create an Empty solution
2. Create a .Net Framework Class Library project (GameLogic)
3. Create a Console application (TextGUI)
4. Create a reference from the Console application to the Class Library
5. Implement the classes in the diagram that are sorrounded by the RED line. Read the following text thoroughly.

- Juice --> Inherits from abstract class Consumable. Consumable inherits from abstract class Item
- Create a Player class, that contains the properties IsAlive and Hitpoints. Also, Create an Inventory class, and give the Player class a reference to a Inventory. The Inventory class should contain a List<Item>'s as a property; remember to instantiate it in the Inventory constructor
- Create a constructor on the Player class that accepts hitpoints, and set the Hitpoints property value inside of it, along with a new Inventory and IsAlive = true;
- Let the abstract class Item contain the string property "Title" --> Let the constructor accept a string title, and set the property
- Let the abstract class Consumable contain an abstract method Consume that accepts an input of type Player; in the Consumable class, remember to call the base class constructor, and pass along the title to the superclass
- Let the concrete class Juice inherit from the abstract class Consumable. Create a field on consumable that contains the amount of hitpoints it restores.
- Remember to set the field value in the constructor, along with the input parameters that you need to send along to the superclass. Implement the abstract method Consume(you could do a Console.WriteLine("I ate something");

6. Create another class that inherits from Consumable called Bread; set it up as you did with the Juice class
7. In your Program.cs, instantiate a new Player. Instantiate some "Juice" and "Bread" instances to the Players Inventory List<>
8. Loop through the Players Inventory List, Print the Items title. In your loop, use the "is" operator, to check if the current item is Juice. If it is, drink it.

Example output:


```
Protein Shake - Yum, just drank it
Water - Yum, just drank it
Head of Cabbage
Celery Stick
```

3. Part 2 – Attack

1. Expand your program by implementing new types of items. You currently have Consumables; how about implementing Weapons. You can do this by creating an abstract Weapon class and inherit from Item

2. When you have created Weapon as you did with the Consumable class, you can create two concrete weapons: eg. a BroadSword and a Katana. You do this by creating concrete classes and inheriting from Weapon.

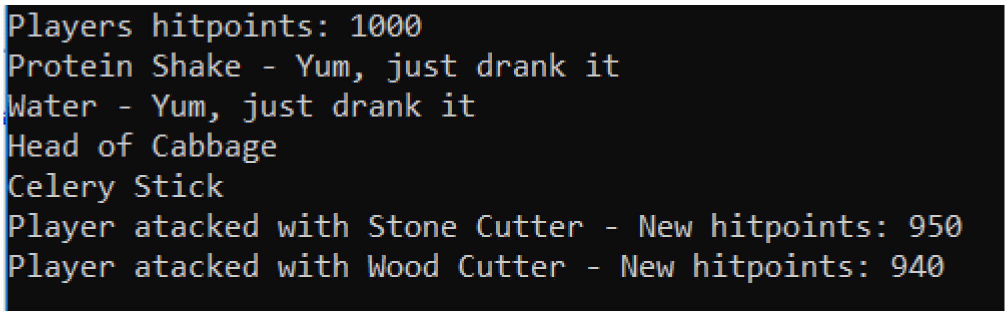
3. Implement the abstract method Attack(Player playerToAttack), and decrement the Health of the input player.

4. Go back to your Program.cs, and add your new Weapons to the inventory List

5. Loop through it, and see that you inventory can contain everything that is an Item.

6. If you want, you can check if the current loop iteration Item is a Weapon, if it is, type cast it to Weapon and run the Attack method on it.

Exampe output:


```
Players hitpoints: 1000
Protein Shake - Yum, just drank it
Water - Yum, just drank it
Head of Cabbage
Celery Stick
Player atacked with Stone Cutter - New hitpoints: 950
Player atacked with Wood Cutter - New hitpoints: 940
```

Note: Attack decrements the hit points. But not implemented that hit points increment when player consumes juice or bread.

4. Part 3 – Interfaces

Expand your program with generalized behavior using interfaces.

1. Implement a IPlayer interface with two methods, Die() and Respawn(). Furthermore, try to implement your properties IsAlive and Hitpoints here. This means that all IPlayers needs to have Hitpoints and an Alive status

2. Let your player class implement this interface, and implement the methods

3. Implement a IDamagable interface, and let your Player Implement it. You could also let the IPlayer interface inherit from the IDamagable interface. It´s up to you.

4. On the IDamagable interface, create the method TakeDamage(int amount)

5. Implement the TakeDamage method on your Player, and decrement the Health he has.

6. Notice that ANY type that implements the IDamagable interface, now can take damage, and work in any context we set up.

7. Now you can finally examine your implementation and find the places where you use a concrete player as input to methods, and change this to IDamagable. This loosens coupling.

5. Part 4 – Freestyle

1. Continue developing your Inventory system by creating the IHealable interface.

2. You could also implement a IPickup interface, and let your Item (or something else) implement it, enabling that player can pick up "things".