# OOP Basics Retake Exam – WarCroft

## Overview

In this exam, you need to build a WarCraft-esque project, which has support for **characters**, **items** and **inventories** for storing each character’s items. Your senior colleagues have already implemented the game Engine, the IO and provided constants for all console messages as well as started work on some of the entity classes. Your task is to finish implementing the **entity classes** and also implement a **controller class**, which manages the **interaction** between the characters and items.

## Setup

* Upload **only the** WarCroftproject in every problem **except** **Unit Tests**
* **Do not modify the interfaces or their namespaces**
* Use **strong cohesion** and **loose coupling**
* **Use inheritance and the provided interfaces wherever possible**.
  + This includes **constructors**, **method parameters** and **return types**
* **Do not** violate your **interface** **implementations** by adding **more public methods** or **properties** in the concrete class than the interface has defined
* Make sure you have **no public fields** anywhere

## Task 1: Structure (50 points)

### Item

This is a **base class** for any **items** and it **should not be able to be instantiated**.

**This class is already implemented for you in the skeleton!** All that is left for you is to implement the **FirePotion** and **HealthPotion** child classes.

#### HealthPotion

The **health potion is a type of item.** It always has a **weight** of **5**.

##### Behavior

Each HealthPotion has the following **behavior**:

###### void AffectCharacter(Character character)

For an item to affect a character, the character **needs to** **be alive**.

The character’s **health** gets **increased** by **20 points**.

##### Constructor

An **item** should be able to be instantiated **without any parameters**.

#### FirePotion

The **Fire potion is a type of item**. It always has a **weight** of **5**.

##### Behavior

Each FirePotion has the following **behavior**:

###### void AffectCharacter(Character character)

For an item to affect a character, the character **needs to** **be alive**.

The character’s **health** gets **decreased** by **20 points**. If the character’s health **drops to zero**, the character **dies** (IsAlive **🡺** false).

##### Constructor

A FirePotion should be able to be instantiated **without any parameters**.

### Bag

This is a **base class** for any **bags** and it **should not be able to be instantiated**.

#### Data

Capacity – an **integer number**. Default value: 100

Load – Calculated property. The **sum of the weights** of the **items** in the bag.

Items – **Read-only collection** of type Item

#### Behavior

Each **bag** has the following **behavior**:

##### void AddItem(Item item)

If the current load + the weight of the item attempted to be added is **greater than** the bag’s **capacity**, throw an InvalidOperationExceptionwith the message **"**Bag is full!**"**

If the check passes, the **item** is added to the **bag**.

##### Item GetItem(string name)

If no items exist in the bag, throw an InvalidOperationException with the message **"**Bag is empty!**"**

If an item with that **name** **doesn’t exist** in the bag, throw an ArgumentException with the message **"**No item with name {name} in bag!**"**

If both checks pass, the **item** is removed from the **bag** and **returned** to the **caller**.

#### Constructor

A **Bag** should take the following values upon initialization:

int capacity

#### Backpack

This is a **type of bag** with 100 capacity.

#### Satchel

This is a **type of bag** with 20 capacity.

### Character

This is a **base class** for any **characters** and it **should not be able to be instantiated**. The class is just started in the skeleton so you will have to finish it.

#### Data

* Name – a **string (cannot be null or whitespace)**.
  + **Throw an** **ArgumentException with the message** **"Name cannot be null or whitespace!"**
* **BaseHealth** – a **floating-point number** – the starting and also the maximum health a character can have
* Health – a **floating-point number** (current health).
  + Health (current health) should never be more than the **BaseHealth** or less than 0.
* **BaseArmor** – a **floating-point number** – the starting armor a character has
* Armor – a **floating-point number** (current armor)
  + Armor – the current amount of armor left – can not be less than 0.
* **AbilityPoints** – a **floating-point number**
* Bag – a parameter of type **Bag**
* IsAlive – boolean value (default value: True)

#### Behavior

Each **character** has the following **behavior**:

##### void TakeDamage(double hitPoints)

For a character to take damage, they need to **be alive**.

The character takes damage equal to the **hit points**. Taking damage works like so:

The character’s **armor** is **reduced** by the **hit point amount**, then if there are **still hit points left**, they take that amount of **health damage**.

If the character’s **health** drops to **zero**, the character **dies** (IsAlive become **false**)

Example: Health: **100**, Armor: **30**, Hit Points: **40** 🡺 Health: **90**, Armor: **0**

##### void UseItem(Item item)

For a character to use an item, they need to be **alive**.

The item affects the character with the item effect.

#### Constructor

A **character** should take the following values upon initialization:

string name, double health, double armor, double abilityPoints, Bag bag

#### Warrior

The **Warrior** is a special class, who can **attack** other characters (implements IAttacker).

##### Data

The Warrior class always has **100 Base Health**, **50 Base Armor**, **40 Ability Points**, and a **Satchel** as a bag.

##### Constructor

The **Warrior** only needs a **name** for initialization:

string name

##### Behavior

The warrior only has **one special behavior** (every other behavior is **inherited**):

###### void Attack(Character character)

For a character to attack another character, **both of them need to be alive**.

If the character they are trying to attack is the same character, throw an InvalidOperationException with the message **"**Cannot attack self!**"**

If all of those checks pass, the **receiving character takes damage** equal to the **attacking character’s** **ability points**. The damage is subtracted from the **armor points first** and once there is no more armor points, from the **health points** of the receiver.

#### Priest

The **Priest** is a special class, who can **heal** other characters (implements IHealer).

##### Data

The Priest class always has **50 Base Health**, **25 Base Armor**, **40 Ability Points**, and a **Backpack** as a bag.

##### Constructor

The **Priest** only needs a **name** for initialization:

string name

##### Behavior

The Priest only has **one special behavior** (every other behavior is **inherited**):

###### void Heal(Character character)

For a character to heal another character, **both of them need to be alive**.

If this is true, the **receiving character’s health increases by the healer’s ability points**.

## Task 2: Business Logic (150 points)

### The Controller Class

The business logic of the program should be concentrated around several **commands**. Implement a class called WarController, which will hold the **main functionality**.

The War Controller keeps track of the **character party** and the **item pool** (the items in the game, which can be picked up).

***Note: The*** *WarController* ***class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!***

The main functionality is represented by these **public** **methods**:

|  |
| --- |
| WarController.cs |
| public string JoinParty(string[] args)  {  throw new **NotImplementedException**();  }  public string AddItemToPool(string[] args)  {  throw new **NotImplementedException**();  }  public string PickUpItem(string[] args)  {  throw new **NotImplementedException**();  }  public string UseItem(string[] args)  {  throw new **NotImplementedException**();  }  public string GetStats()  {  throw new **NotImplementedException**();  }  public string Attack(string[] args)  {  throw new **NotImplementedException**();  }  public string Heal(string[] args)  {  throw new **NotImplementedException**();  } |

**NOTE: WarController class** methods are called from the Engine so these methods **must** **NOT** receive the command parameter (the **first argument** from the input line) as part of the arguments!

**ALSO NOTE: The** WarController **class should not handle any exceptions. That should be the responsibility of the Engine.**

### Commands

There are several commands that control the business logic of the application and you are supposed to build.   
They are stated below.

#### JoinParty Command

##### Parameters

* characterType – **string**
* name – **string**

##### Functionality

**Creates a character** and **adds them** to the **party**.

If the **character type** is invalid, throw an ArgumentException with the message **"**Invalid character type **"**{characterType}**"**!**"**

Returns the **string** **"**{name} joined the party!**"**

#### AddItemToPool Command

##### Parameters

* itemName – **string**

##### Functionality

**Creates an item** and **adds it** to the **item pool**.

If the **item type** is **invalid**, throw an ArgumentException with the message **"**Invalid item **"**{name}**"**!**"**

Returns the **string** **"**{itemName} added to pool.**"**

#### PickUpItem Command

##### Parameters

* characterName – **string**

##### Functionality

Makes the character with the specified name **pick up the last item in the item pool and add it to his/her bag**.

If the character doesn’t exist in the **party**, throw an ArgumentException with the message **"**Character {name} not found!**"**

If there’s **no items left** in the pool, throw an InvalidOperationException with the message **"**No items left in pool!**"**

Returns the **string** **"**{characterName} picked up {itemName}!**"**

#### UseItem Command

##### Parameters

* characterName – a **string**
* itemName – **string**

##### Functionality

**Makes the character** with that name use an **item with that name** **from their bag**.

If the character doesn’t exist in the **party**, throw an ArgumentException with the message **"**Character {name} not found!**"**

The rest of the exceptions should be processed by the called functionality (empty bag, etc.)

Returns the **string** **"**{character.Name} used {itemName}.**"**

#### GetStats Command

##### Parameters

No parameters.

##### Functionality

Returns info about **all characters**, sorted by **whether** **they are alive** (**descending**), **then by** their **health** (**descending**)

The format of a single character is:

"{name} - HP: {health}/{baseHealth}, AP: {armor}/{baseArmor}, Status: {Alive/Dead}"

Returns the formatted character info for each character, **separated by new lines**.

#### Attack Command

##### Parameters

* attackerName – **string**
* receiverName – **string**

##### Functionality

Makes the **attacker** **attack** the **receiver**.

If any character doesn’t exist in the **party**, throw an ArgumentException with the message **"**Character {name} not found!**"** Check the Attacker first and then the receiver.

If the **attacker cannot attack**, throw an ArgumentException with the message **"**{attacker.Name} cannot attack!**"**

The command output is in the following format:

"{attackerName} attacks {receiverName} for {attacker.AbilityPoints} hit points! {receiverName} has {receiverHealth}/{receiverBaseHealth} HP and {receiverArmor}/{receiverBaseArmor} AP left!"

If the attacker ends up **killing** the receiver, add a **new line**, plus **"**{receiver.Name} is dead!**"** to the output.

Returns the **formatted string**

#### Heal Command

##### Parameters

* healerName – a **string**
* healingReceiverName – **string**

##### Functionality

Makes the **healer** **heal** the **healing** **receiver**.

If any character doesn’t exist in the **party**, throw an ArgumentException with the message **"**Character {name} not found!**".** Check the Healer first and then the receiver.

If the **healer cannot heal**, throw an ArgumentException with the message **"**{healerName} cannot heal!**"**

The command **output** is in the following format:

**"{healer.Name} heals {receiver.Name} for {healer.AbilityPoints}! {receiver.Name} has {receiver.Health} health now!"**

Returns the **formatted string**

### Input / Output

#### Input

* You will receive commands **until the "END" command is received**.

Below, you can see the **format** in which **each command** will be given in the input:

* JoinParty {Alliance/Horde} {class} {name}
* AddItemToPool {itemName}
* PickUpItem {characterName}
* UseItem {characterName} {itemName}
* GetStats
* Attack {attackerName} {attackTargetName}
* Heal {healerName} {healingTargetName}

#### Output

Print the output from each command when issued. When the game is over, print **"**Final stats:**"** and the output from the GetStats command.

If an exception is thrown during any of the commands’ execution, print:

* **"**Parameter Error: **"** plus the message of the exception if it’s an ArgumentException
* **"**Invalid Operation: **"** plus the message of the exception if it’s an InvalidOperationException

#### Constraints

* The commands will always be in the provided format.

#### Examples

|  |
| --- |
| **Input** |
| JoinParty Warrior Gosho  JoinParty Warrior Pesho  AddItemToPool HealthPotion  AddItemToPool FirePotion  PickUpItem Gosho  JoinParty Priest Ivan  Attack Gosho Pesho  Attack Gosho Pesho  Attack Gosho Pesho  Heal Ivan Pesho  Attack Gosho Ivan  Attack Gosho Ivan  Attack Gosho Pesho  Attack Gosho Pesho  Attack Gosho Pesho  END |
| **Output** |
| Gosho joined the party!  Pesho joined the party!  HealthPotion added to pool.  FirePotion added to pool.  Gosho picked up FirePotion!  Ivan joined the party!  Gosho attacks Pesho for 40 hit points! Pesho has 100/100 HP and 10/50 AP left!  Gosho attacks Pesho for 40 hit points! Pesho has 70/100 HP and 0/50 AP left!  Gosho attacks Pesho for 40 hit points! Pesho has 30/100 HP and 0/50 AP left!  Ivan heals Pesho for 40! Pesho has 70 health now!  Gosho attacks Ivan for 40 hit points! Ivan has 35/50 HP and 0/25 AP left!  Gosho attacks Ivan for 40 hit points! Ivan has 0/50 HP and 0/25 AP left!  Ivan is dead!  Gosho attacks Pesho for 40 hit points! Pesho has 30/100 HP and 0/50 AP left!  Gosho attacks Pesho for 40 hit points! Pesho has 0/100 HP and 0/50 AP left!  Pesho is dead!  Invalid Operation: Must be alive to perform this action!  Final stats:  Gosho - HP: 100/100, AP: 50/50, Status: Alive  Pesho - HP: 0/100, AP: 0/50, Status: Dead  Ivan - HP: 0/50, AP: 0/25, Status: Dead |

|  |
| --- |
| **Input** |
| JoinParty Warrior Gosho  JoinParty Warrior Pesho  AddItemToPool HealthPotion  AddItemToPool FirePotion  PickUpItem Pesho  PickUpItem Gosho  PickUpItem Pesho  UseItem Pesho HealthPotion  UseItem Pesho FirePotion  AddItemToPool FirePotion  PickUpItem Gosho  JoinParty Warrior Ivan  Attack Ivan Gosho  Attack Ivan Gosho  Attack Ivan Gosho  Attack Gosho Ivan  Attack Ivan Gosho  Attack Ivan Pesho  Attack Ivan Pesho  Attack Ivan Pesho  END |
| **Output** |
| Gosho joined the party!  Pesho joined the party!  HealthPotion added to pool.  FirePotion added to pool.  Pesho picked up FirePotion!  Gosho picked up HealthPotion!  Invalid Operation: No items left in pool!  Parameter Error: No item with name HealthPotion in bag!  Pesho used FirePotion.  FirePotion added to pool.  Gosho picked up FirePotion!  Ivan joined the party!  Ivan attacks Gosho for 40 hit points! Gosho has 100/100 HP and 10/50 AP left!  Ivan attacks Gosho for 40 hit points! Gosho has 70/100 HP and 0/50 AP left!  Ivan attacks Gosho for 40 hit points! Gosho has 30/100 HP and 0/50 AP left!  Gosho attacks Ivan for 40 hit points! Ivan has 100/100 HP and 10/50 AP left!  Ivan attacks Gosho for 40 hit points! Gosho has 0/100 HP and 0/50 AP left!  Gosho is dead!  Ivan attacks Pesho for 40 hit points! Pesho has 80/100 HP and 10/50 AP left!  Ivan attacks Pesho for 40 hit points! Pesho has 50/100 HP and 0/50 AP left!  Ivan attacks Pesho for 40 hit points! Pesho has 10/100 HP and 0/50 AP left!  Final stats:  Ivan - HP: 100/100, AP: 10/50, Status: Alive  Pesho - HP: 10/100, AP: 0/50, Status: Alive  Gosho - HP: 0/100, AP: 0/50, Status: Dead |

|  |
| --- |
| **Input** |
| JoinParty Warrior Gosho  JoinParty Warrior Ivan  Attack Gosho Gosho  PickUpItem Gosho  AddItemToPool InvalidItem  AddItemToPool HealthPotion  UseItem Gosho InvalidItem  UseItem Gosho HealthPotion  PickUpItem InvalidCharacter  Attack Ivan Ivan  Attack Pesho Ivan  Attack Ivan Pesho  Attack A B  Attack Ivan Gosho  Attack Ivan Gosho  Attack Ivan Gosho  Attack Ivan Gosho  END |
| **Output** |
| Gosho joined the party!  Ivan joined the party!  Invalid Operation: Cannot attack self!  Invalid Operation: No items left in pool!  Parameter Error: Invalid item "InvalidItem"!  HealthPotion added to pool.  Invalid Operation: Bag is empty!  Invalid Operation: Bag is empty!  Parameter Error: Character InvalidCharacter not found!  Invalid Operation: Cannot attack self!  Parameter Error: Character Pesho not found!  Parameter Error: Character Pesho not found!  Parameter Error: Character A not found!  Ivan attacks Gosho for 40 hit points! Gosho has 100/100 HP and 10/50 AP left!  Ivan attacks Gosho for 40 hit points! Gosho has 70/100 HP and 0/50 AP left!  Ivan attacks Gosho for 40 hit points! Gosho has 30/100 HP and 0/50 AP left!  Ivan attacks Gosho for 40 hit points! Gosho has 0/100 HP and 0/50 AP left!  Gosho is dead!  Final stats:  Ivan - HP: 100/100, AP: 50/50, Status: Alive  Gosho - HP: 0/100, AP: 0/50, Status: Dead |