Assignment 2
Inheritance & Pointers

CS 202 - Summer 2022

Introduction

In this assignment we will be implementing an inventory system similar to the one used in Fortnite. We will be using inheritance, pointers, and dynamic memory allocation to achieve this. Note that this assignment has a potential for memory leaks, so I strongly advise you to use `valgrind` to check for memory errors.

Instructions

There are 2 files you have to complete for this assignment:

1. item.cpp
2. player.cpp

These are the only 2 files you are required to submit. Which means that you are not allowed to make any changes to the header files. The `main.cpp` file has been provided to you as a way to test your program. You can write your own main file to test your code as you write it.

Make sure to add a header in the following format at the top of **both files!**:

```
/*
* Name: YOUR_NAME, NSHE_ID_#, COURSE_SECTION, ASSIGNMENT_#
* Description: DESCRIPTION_OF_PROGRAM
* Input: EXPECTED_PROGRAM_INPUT
* Output: EXPECTED_PROGRAM_OUTPUT
*/
```

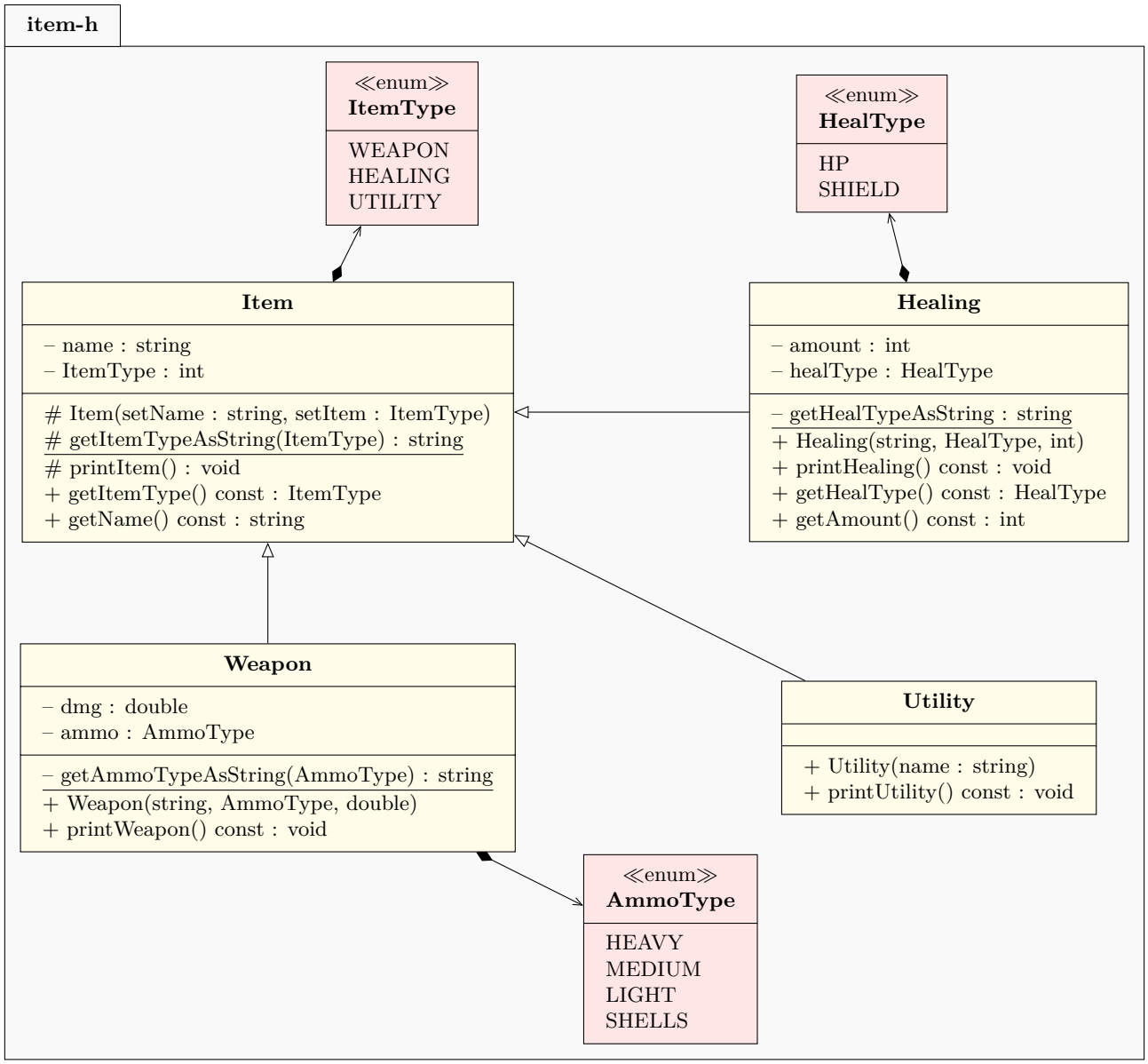
Helpful Tip

Make sure to test each function as you write it. Use terminal output liberally to ensure that your functions are doing exactly what they are supposed to do. Try to find edge cases which are likely to fail and test your code against them.

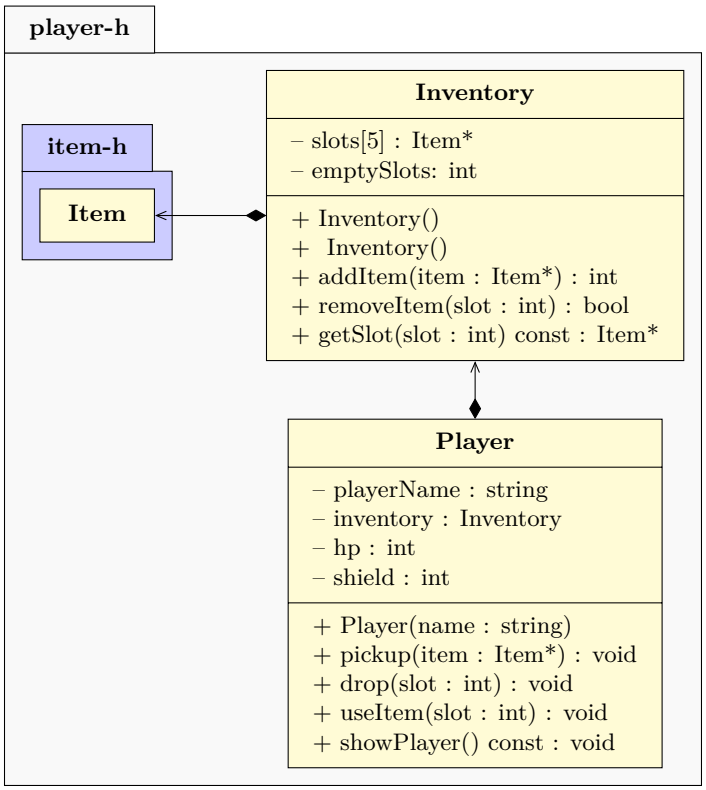
UML Diagrams

UML Diagrams for the `item.h` and `player.h` packages is given below.

item.h



player.h



Functions and Class Details

All functions, classes and important variables used in the assignment are described below. Variables are in green. Functions that are to be implemented as part of this assignment are given in red., Functions that have been implemented for you are in blue. Getters and setters that have already been implemented for you will not be shown here.

Enum ItemType

This is an enumerator used to define the type of Item we are addressing. There are 3 types of items possible:

- **WEAPON**
- **HEALING**
- **UTILITY**

This acts a way for us to check which subclass should we be typecasting to. An example of this has been given in the **showPlayer()** function.

Class Item

This class defines the properties of a generic Item. The properties of the item specific to its type are defined in the subclasses.

- **string name** - Name of the item.
- **ItemType item** - Item type. This can be either **WEAPON**, **HEALING** or **UTILITY**.
- **Item(string setName, ItemType setItem)** - This is the default constructor for the class. Note that the constructor is protected meaning that an Item object cannot be created by anyone except subclasses.

- **static string getItemTypeAsString(ItemType)** - This is a helper class used for the printItem function. It will return a string for the corresponding ItemType. Note that this is a static function so every object does not have an independent copy of it.
- **void printItem() const** - Prints an item. See how the getItemTypeAsString function is being used here.

Enum AmmoType

This is an enumerator used to define the type of ammo used by a weapon. There are 4 ammo types available:

- **HEAVY**
- **MEDIUM**
- **LIGHT**
- **SHELLS**

Enumerators like this are used to limit the possible values that an object can have.

Class Weapon : Item

This class is publicly inherited from the Item class. Therefore, it contains the properties of a generic item, and some additional properties that specifically define a weapon.

- **double dmg** - The damage done by this weapon. We have set the data type to double here because it sounds cool. Also because damage numbers can also be affected by factors such as elemental protection, armor, hit location, which may result in fractional damage numbers.
- **AmmoType ammo** - Ammo type that is used by this weapon. This can take any value from the AmmoType enumerator.
- **static string getAmmoTypeAsString(ItemType)** - This is a helper class used for the printWeapon function. It will return a string for the corresponding AmmoType. It works just like the getItemTypeAsString function. Use the Sample Output as a reference for the string values that should be returned by this function.
- **Weapon(std::string, AmmoType, double)** - This is a parameterized constructor for this class. The parameters taken by this constructor are name of the item, ammo type used by it, and the damage it does respectively. Note that this constructor cannot access the item name variable. So, we call the Item class constructor here to construct an item with the given name, and then set the other parameters as required for the Weapon class.
- **void printWeapon() const** - Prints a weapon. See how the getAmmoTypeAsString function is being used here. Also note how this function uses the printItem function from the parent class because it does not have access to the item name and item type variables.

Enum HealType

See Healing class for usage.

- **HP** - Item will regenerate player's HP.
- **SHIELD** - Item will regenerate player's shield.

Class Healing : Item

This class publicly inherits from the Item class and defines the properties of healing items.

- **int amount** - Amount of HP or shields regenerated by this item. Note that the type of regeneration provided is defined by the **healType** variable.
- **HealType healType** - Defines whether this item will regenerate the player's HP or shields.
- **static string getHealTypeAsString(HealType)** - Implement this function in the same way that you implemented the getAmmoTypeAsString function. Use the Sample Output as a reference for the string values that should be returned by this function.

- **Healing(std::string, HealType, int)** - Parameterized constructor that takes 3 parameters: name of item, healing type provided, and healing amount provided. Use the example provided in the Weapon class to implement this constructor.
- **void printHealing() const** - Prints a healing item. Use the Sample Output and printWeapon function as reference for how to implement this function. You will need to implement the getHealTypeAsString function before implementing this function.

  > Hint
  >
  > Copy and paste the code from the printWeapon function and change only the necessary parts of it. You can leave the setw values as they are.

Class Utility : Item

This class publicly inherits from the Item class. All items that do are not weapons or healing items are classified as Utility items.

- **Utility(string name)** - The parameterized constructor for this class has been provided to you.
- **void printUtility() const** - This is the only function this class has. Since this class doesn't have any properties, the print function will simply call the print function from the Item class.

  > Hint
  >
  > Make sure to print an end line!

Class Inventory

This class defines a player inventory. Note that this class uses Item class pointers to instantiate item objects.

- **Item* slots[5]** - Inventory objects only have 5 slots. Each slot holds a pointer to an item object.
- **int emptySlots** - Number of empty inventory slots currently present. This number will be 5 for an empty inventory and 0 for a full inventory.
- **Inventory()** - Default constructor. This sets each slot to `NULL` or `nullptr` and the number of empty slots to 5.
- **~Inventory()** - Destructor. Since we are dynamically allocating memory for each item inserted into the inventory slots, the destructor must deallocate this memory when the object is deleted. Otherwise, it will result in a memory leak. Check each slot for a valid item pointer and delete it.

  > Hint
  >
  > If the number of empty slots is 5, the inventory is empty. If any slot contains a nullptr, nothing needs to be deleted.

- **int addItem(Item *item)** - Insert an item into the inventory. If the number of empty slots is 0, then no item should be inserted and a -1 must be returned. Otherwise, the item must be inserted in the first free slot found in the slots array. For example, if slots 0, 2, and 4 are filled, then the new item should be inserted into slot 1. Remember to decrement the emptySlots counter. Return the slot number where the new item was inserted. For example, if the item was inserted in slot index 1, then 1 will be returned.

  > Hint
  >
  > You can assume that the item pointer passed in as a parameter points to a valid item.

- **bool removeItem(int slot)** - Removes the item from the slot number given in the function parameter. If no item is found at that slot, return false. Otherwise, remove the item and return true. Make sure to increment the emptySlots counter.

Hint

Set the value in slots[slot] to a nullptr AFTER deleting the item present in it.

- **Item* getSlot(int slot) const** - Returns a pointer to the item present in the slot. Note that since the getter is constant, it cannot modify any slot item. Also note that this getter returns a pointer to an Item and not an Item object itself!

Class Player

The player class has an inventory object. Note that player functions interact with the player's inventory. Functions in this class depend on the Inventory class. Make sure to implement those first.

- **string playerName** - Player name given to a player on creation. Note that Player object cannot be created without a playerName because a default constructor has not been provided for this class.
- **Inventory inventory** - Player's inventory. This will call the default constructor of the inventory class. Therefore, the player will always start with an empty inventory with 5 slots.
- **int hp** - Current hp of the player. Starts at 100.
- **int shield** - Current shield of the player. Starts at 0.
- **Player(std::string name)** - Parameterized constructor. Note that this class does not have a default constructor. So, a player name MUST be provided to create an object of this class.
- **void pickup(Item*)** - Adds an item to the player's inventory. This function calls the addItem function from the Inventory class. If the addItem function returns -1, then it is assumed that the player inventory is full and the item is deleted.
- **void drop(int slot)** - Drops the item present in the slot index given as a parameter. This function checks if there's an item in the given slot. If there is, then it's dropped using the removeItem function from the Inventory class.
- **void useItem(int slot)** - Uses the item in the given slot. Only Healing items can be used. Other items should make the player say "Cannot use <item name>". Note that when an item is used, it must be removed from the inventory. If there's no item in the slot, the player must say "No item in this slot." When a healing item is used, the player must say "Used <item name>". Use static_cast to cast parent class to the required subclass. See the showPlayer function for reference.

  Hints

  - You can check the item type using:

    ```
    item->getItemType()
    ```

  - Make sure to check the HealType of the item and regenerate the correct parameter.
  - HP and Shield do not have a maximum. They can regenerate to infinity.
  - Use the removeItem function once the item has been used.

- **void showPlayer() const** - Prints the player stats and inventory. Note the use of static_cast in this function. Use this as a reference for the useItem function.

Make Files

A make file has been provided for this project. In order to use make files you will need to install the `make` package. For Debian/Ubuntu and WSL users, you can install make using the following command in your Linux shell:

```
sudo apt install make
```

For Mac users with homebrew installed, you can use this command:

```
brew install make
```

For VS Code users, I highly recommend installing the Makefile Tools extension.

Object Code

A static library containing the solution object code has been provided in the ObjectCode folder in this handout. You can make changes to the sample main file provided here, and compile the project using `make`. This will allow you to obtain whatever output you desire from the finished code to help you implement the assignment.

This is how libraries are typically shipped where only the required public functions are exposed to the users while the libraries themselves are closed source. This is how engines like Unity or Unreal Engine provide an API to game developers while still being closed source.

Sample Output

This is a sample run of the program with the given `main.cpp` file in the ObjectCode folder. Feel free to make changes to this file to help you understand how the finished code should work.

```
piyush@Piyush-Desktop:/.../ast2/solution$ ./a.out
Player 1: Picked up HammerAssault!
Player 1: Picked up SidearmPistol!
Player 1: Picked up PumpShotgun!
Player 1: Picked up Medkit!
Player 1: Picked up FishingRod!
Player 1: My inventory is full!

Player 1 : 100(0)
Inventory:
0    Weapon :        HammerAssault : AMMO   Medium : DMG  31
1    Weapon :        SidearmPistol : AMMO    Light : DMG  25
2    Weapon :          PumpShotgun : AMMO   Shells : DMG 103.2
3   Healing :               Medkit : TYPE       HP : AMT 100
4   Utility :           FishingRod
Player 1: Dropped FishingRod!
Player 1: Picked up SmallShield!

Player 1 : 100(0)
Inventory:
0    Weapon :        HammerAssault : AMMO   Medium : DMG  31
1    Weapon :        SidearmPistol : AMMO    Light : DMG  25
2    Weapon :          PumpShotgun : AMMO   Shells : DMG 103.2
3   Healing :               Medkit : TYPE       HP : AMT 100
4   Healing :          SmallShield : TYPE   Shield : AMT  25
Player 1: Used Medkit
Player 1: Used SmallShield

Player 1 : 200(25)
Inventory:
0    Weapon :        HammerAssault : AMMO   Medium : DMG  31
1    Weapon :        SidearmPistol : AMMO    Light : DMG  25
2    Weapon :          PumpShotgun : AMMO   Shells : DMG 103.2
Player 1: No item in slot 4.
Player 1: Cannot use SidearmPistol
Player 1: Picked up HeavySniper!

Player 1 : 200(25)
Inventory:
0    Weapon :        HammerAssault : AMMO   Medium : DMG  31
1    Weapon :        SidearmPistol : AMMO    Light : DMG  25
2    Weapon :          PumpShotgun : AMMO   Shells : DMG 103.2
3    Weapon :          HeavySniper : AMMO    Heavy : DMG 120
Player 1: Dropped SidearmPistol!

Player 1 : 200(25)
Inventory:
0    Weapon :        HammerAssault : AMMO   Medium : DMG  31
2    Weapon :          PumpShotgun : AMMO   Shells : DMG 103.2
3    Weapon :          HeavySniper : AMMO    Heavy : DMG 120
Player 1: Picked up ShieldPotion!
```

```
Player 1 : 200(25)
Inventory:
0    Weapon :         HammerAssault : AMMO   Medium : DMG  31
1   Healing :          ShieldPotion : TYPE   Shield : AMT  50
2    Weapon :           PumpShotgun : AMMO   Shells : DMG 103.2
3    Weapon :           HeavySniper : AMMO    Heavy : DMG 120
piyush@Piyush-Desktop:/.../ast2/solution$
```