Assignment 3
Virtuals & Abstract Classes

CS 202 - Summer 2022

Instructions

You need to submit just one file for this assignment: **player.cpp**. There are 2 functions you have to reimplement:

1. Player::useItem
2. Player::showPlayer (partially implemented)

You are not allowed to make any changes to the header files or the **item.cpp** file. The main.cpp file provided to you is the same as the previous assignment. It should print exactly what assignment 2 prints. The only significant difference in this assignment is the implementation of the classes.

Make sure to add a header in the following format at the top of **player.cpp**:

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

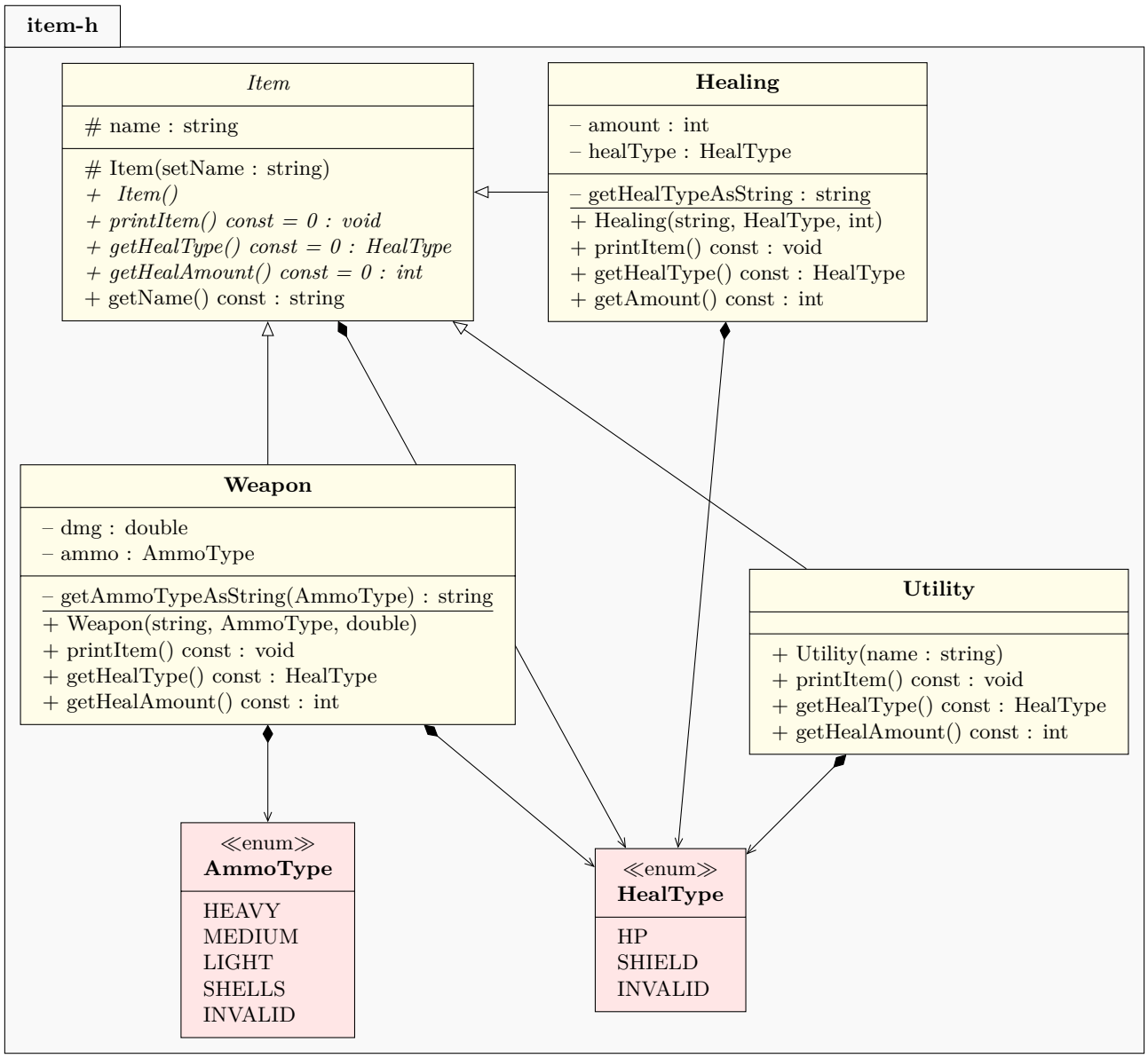
UML Diagrams

UML Diagrams for the `item.h` and `player.h` packages is given below. Note how using virtuals and abstract classes changes the structure of the program even though the final result remains the same. Also note that there's no ItemType enum anymore.

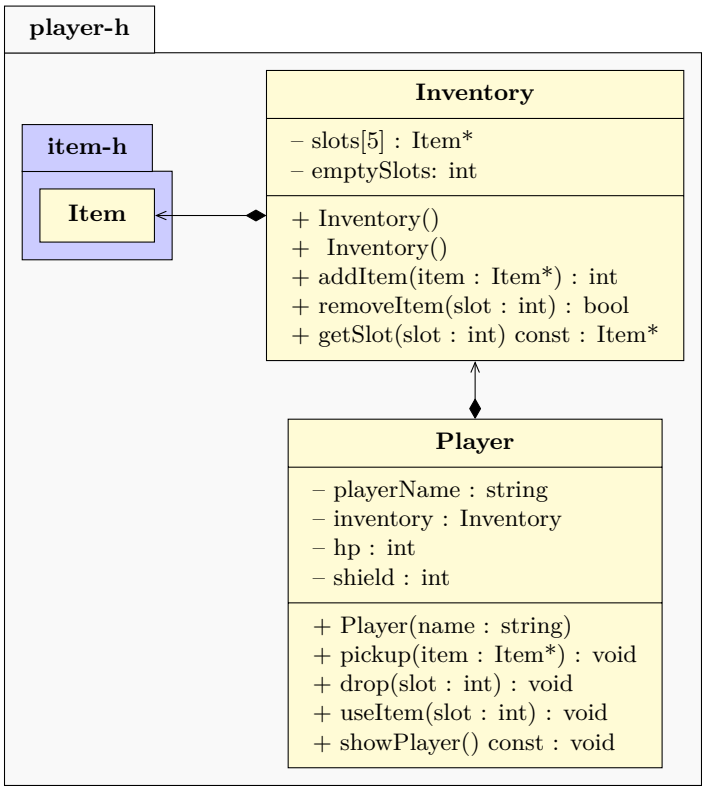
Helpful Tip

Abstract classes and virtual functions are shown in *italics*.

item.h



player.h



Functions and Class Details

Since most of the functions are exactly the same as in Assignment 2, **only the ones which have changed will be detailed here.** 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.

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.
- **Item(string setName)** - 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.
- **virtual void printItem() const = 0** - Purely virtual function, meaning that it needs to be implemented in derived classes.
- **virtual HealType getHealType() const = 0** - Purely virtual. We have to implement it in each and every derived class that we don't want treated as abstract.
- **virtual int getHealAmount() const = 0** - Purely virtual. Same as above. We need to implement it in every derived class whether it's relevant or not.

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**
- **INVALID** - Indicates that this item is not of Weapon type.

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.

- **void printItem() const** - This is a reimplementation of the printItem function from the Item class. If the printItem function is called from an Item pointer pointing to a Weapon object, this function will execute.
- **HealType getHealType() const** - Since we want to ensure that the Weapon class is not abstract, we have to implement this function. In this case, the function will return **HealType::INVALID**. When we learn exception handling, we can have this function throw an exception instead.
- **int getHealAmount() const** - We also have to implement this function in this class to prevent it from being abstract. This function will simply return **-1 since we don't have the amount variable in this class. Similar to the above function, we can have this function throw an exception when we learn about exception handling.**

Enum HealType

See Healing class for usage.

- **HP** - Item will regenerate player's HP.
- **SHIELD** - Item will regenerate player's shield.
- **INVALID** - This item is not a Healing type item.

Class Healing : Item

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

- **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 printItem() const** - Prints a healing item. This is again a reimplementation of the printItem function from the Item class.
- **HealType getHealType() const** - This function will return the value stored in the **healType** variable. This value will either be **HealType::HP** or **HealType::SHIELD**. It will never be **HealType::INVALID**.
- **int getHealAmount() const** - In this class, this function will return the value of the **amount** variable.

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 printItem() const** - This is a reimplementation of the printItem function from the Item class. It essentially just does what the printItem function did in assignment 2.

- **HealType getHealType() const** - In this class, the function will return **HealType::INVALID**.
- **int getHealAmount() const** - This function will simply return **-1 since we don't have the amount variable in this class.**

Class Inventory

There are no changes in this class. All functions have been implemented for you.

Class Player

You have to implement 2 functions here.

- **void useItem(int slot)** - Use the inventory.getSlot() function to get an item from the respective slot. If it's a nullptr, make the player say "No item in slot <slot index>" and then return. If there is an item in the slot, call the getHealType() function directly. If it's HealType::INVALID, then the item is not a Healing item and can't be used. If it's a HealType::HP item then add the amount to the player's HP, otherwise if it's a HealType::SHIELD item then add the amount to the player's shield. Remove the item from the inventory using the removeItem function. Make sure to print the appropriate messages for all above cases.

  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.
  – You won't need to use static_cast in this function.

- **void showPlayer() const** - Prints the player stats and inventory. Use a **for loop** similar to the one used in Assignment 2. You won't have to use a switch case in this situation. Simply call the printItem function using the item pointer in the inventory slot. Since the base function is virtual, the correct derived class function will be called automatically!

Sample Output

Output for this program is exactly the same as the output for Assignment 2. You can download the Assignment 2 solution and run the given **main.cpp** with the files from Assignment 2. Your output from this assignment should match this output exactly. If you want to store the output of the program in a text file, you can do this:

```
make
./a.out > output.txt
```

This will redirect all terminal output into the output.txt file so that you can easily compare them. If you redirect both the assignment 2 and assignment 3 outputs to separate text files, you can use Meld to do a side by side comparison.