

National University of Singapore  
School of Computing  
CS1010X: Programming Methodology  
Semester II, 2021/2022

**Mission 14**  
**Hungry Games Training, Part II**

Release date: 18 April 2022

**Due: 19 May 2022, 23:59**

## Required Files

- mission14-template.py
- hungry\_games.py

## Background

Grandwizard Ben is extremely pleased that you managed to successfully model the objects that would be present in the Hungry Games. Following your success in the previous mission, he has now tasked you with the responsibility of preparing our volunteers to take part in the Hungry Games.

You will have to teach our sheltered mages the skills necessary to survive in a hostile environment. They will have to learn how to hunt, satisfy their hunger, and manage their inventory efficiently.

You have to mould them to become worthy tributes of District M.

## Introduction

In this mission, we will create a new `Tribute` class, extended from the `Person` class in the `hungry_game.py` file. Our `Tribute` class would be the base object that would participate in the Hungry Games, and as such, we would need to implement additional methods and properties so as to satisfy the requirement of the games.

The template file `mission14-template.py` has been provided for your use. Do not copy the test code onto Coursemology.

This mission consists of **five** tasks.

## Task 1: Hunger (7 marks)

We have not modelled hunger in the Person class, as we have previously assumed that food is always abundant and that one would always have access to food. That assumption is not true in the Hungry Games - it is quite possible that one would die of starvation. We will need to model this in our Tribute object.

- Augment the Tribute class by adding a new property, hunger, which will describe the level of hunger for the Tribute. The initial value for hunger should be 0, as all the Tributes will start the game with full stomach.
- Create a method, `get_hunger()`, which return the current hunger level of the tribute.

```
>>> cc = Tribute("Chee Chin", 100)
>>> cc.get_hunger()
0
```

- Create a method, `add_hunger(hunger)`, which will add a hunger value to the Tribute's hunger. When the hunger of a Tribute is equal or more than 100, he/she will `go_to_heaven()`.

```
>>> Base = Place("base")
>>> cc = Tribute("Chee Chin", 100)
>>> Base.add_object(cc)
>>> cc.get_place().get_name()
base
>>> cc.get_hunger()
0
>>> cc.add_hunger(50)
>>> cc.get_hunger()
50
>>> cc.add_hunger(50)
Chee Chin went to heaven!
>>> cc.get_hunger()
100
>>> cc.get_place().get_name()
Heaven
```

- Create a method, `reduce_hunger(hunger)`, which will minus a hunger value to the Tribute's hunger. Your function should ensure that the hunger of the Tribute is always greater or equal to 0.

```
>>> cc = Tribute("Chee Chin", 100)
>>> cc.add_hunger(10)
>>> cc.get_hunger()
10
>>> cc.reduce_hunger(20)
>>> cc.get_hunger()
0
```

## Task 2: Eat (3 marks)

As a natural extension from the previous Task, we will need a way for our Tribute object to reduce their hunger.

Create a method `eat(food)`, which takes in a Food object. It will reduce the hunger of the player by the `food.get_food_value()`. In addition, if the object passed to the `eat` method is of the type Medicine, it will increment the player's health by `medicine.get_medicine_value()`.

There are several rules you need to take note. Firstly, your function should ensure that the health of the player never exceeds 100 and hunger of the player never less than 0. Secondly, the player cannot eat food that is out of his/her inventory. If he/she tries to eat something that is not in the inventory, there's simply no effect. Thirdly, after eating the food, your player should remove the Food or Medicine object from the inventory, even if the Food/Medicine didn't help increase health or reduce hunger.

```
>>> cc = Tribute("Chee Chin", 100)
>>> chicken = Food("chicken", 5)
>>> aloe_vera = Medicine("aloe vera", 2, 5)

>>> Base = Place("base")
>>> Base.add_object(cc)
>>> Base.add_object(chicken)
>>> Base.add_object(aloe_vera)

>>> cc.reduce_health(10)
>>> cc.add_hunger(4)
>>> named_col(cc.get_inventory())
[]

# Note that hunger did not decrease because
# chicken is not in Chee Chin's inventory
>>> cc.eat(chicken)
>>> cc.get_hunger()
4

>>> cc.take(chicken)
Chee Chin took chicken
>>> cc.take(aloe_vera)
Chee Chin took aloe vera

>>> named_col(cc.get_inventory())
['chicken', 'aloe vera']

>>> cc.eat(aloe_vera)
>>> cc.get_health()
95
>>> cc.get_hunger()
2
```

```
>>> named_col(cc.get_inventory())
['chicken']
>>> cc.eat(chicken)
>>> cc.get_hunger()
0

>>> named_col(Base.get_objects())
['Chee Chin']
```

### Task 3: Managing Inventory (6 marks)

In the Person object, there is already a support for an inventory, and when the Person object takes a Weapon object or Food object, the object would go to the inventory. For our Tribute object, we want a way for us to retrieve the Weapon and Food objects from the inventory.

- Create a new method in the Tribute class, `get_weapons()`, which would return a tuple of Weapon objects that the Tribute currently has in his inventory.
- Create a new method in the Tribute class, `get_food()`, which would return a tuple of Food objects that the Tribute currently has in his inventory.
- Create a new method in the Tribute class, `get_medicine()`, which would return a tuple of Medicine objects that the Tribute currently has in his inventory.

```
>>> cc = Tribute("Chee Chin", 100)
>>> bow = RangedWeapon("bow", 4, 10)
>>> sword = Weapon("sword", 2, 5)
>>> chicken = Food("chicken", 5)
>>> aloe_vera = Medicine("aloe vera", 2, 5)

>>> Base = Place("base")
>>> Base.add_object(cc)
>>> Base.add_object(bow)
>>> Base.add_object(sword)
>>> Base.add_object(chicken)
>>> Base.add_object(aloe_vera)

>>> cc.take(bow)
Chee Chin took bow
>>> cc.take(sword)
Chee Chin took sword
>>> cc.take(chicken)
Chee Chin took chicken
>>> cc.take(aloe_vera)
Chee Chin took aloe vera

>>> named_col(cc.get_inventory())
['bow', 'sword', 'chicken', 'aloe vera']

>>> named_col(cc.get_weapons())
```

```

('bow', 'sword')

>>> named_col(cc.get_food())
('chicken', 'aloe vera')

>>> named_col(cc.get_medicine())
('aloe vera',)

```

## Task 4: Attack!! (4 marks)

Our Tribute must definitely be able to attack other objects in the Hungry Games - it is the essence of the Hungry Games. In order to attack something, we will need to know the object that we are attacking, as well as the Weapon that the Tribute will use to attack. We will assume that we can only attack LivingThing objects.

Create a new method in the Tribute class, `attack(living_thing, weapon)`, which would take in 2 parameters, a LivingThing object, and a Weapon object. It should determine the damage that should be dealt to the LivingThing, and reduce the health of the LivingThing by that amount.

**NOTE:** The attack method needs check if the Tribute currently has the weapon object in his/her inventory and whether the target is in the same location. Otherwise it has no effect!

```

>>> Base = Place("base")
>>> cc = Tribute("Chee Chin", 100)
>>> bear = Animal("bear", 20, 10)
>>> sword = Weapon("sword", 10, 10)

>>> Base.add_object(cc)
>>> Base.add_object(sword)
>>> Base.add_object(bear)

>>> bear.get_health()
20

# Attack has no effect, because Chee Chin don't have the sword
# in his inventory
>>> cc.attack(bear, sword)
>>> bear.get_health()
20

>>> cc.take(sword)
Chee Chin took sword
>>> cc.attack(bear, sword)
>>> bear.get_health()
10
>>> cc.attack(bear, sword)
bear went to heaven!
>>> named_col(Base.get_objects())
['Chee Chin', 'bear meat']

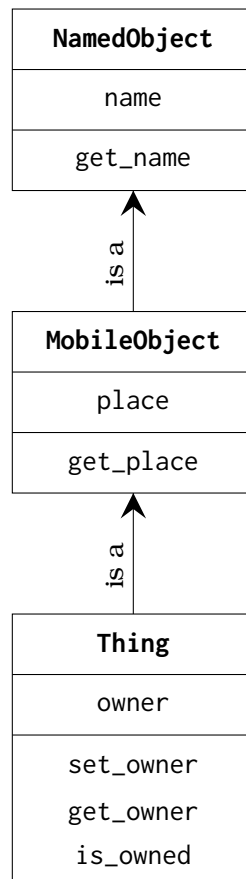
```

## Task 5: Object Hierarchy (5 marks)

Draw a simple inheritance diagram showing all the objects (classes) defined in the complete Hungry Games simulation and the inheritance relations between them, as well as the properties and methods defined for each class.

You may include your inheritance diagram as an attachment when you submit the mission.

**Hint:** There should be a total of 13 classes. Did you get all of them?



A partial class diagram of some of the classes defined in the Hungry Games