COS 214 Practical Assignment 2

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- Date Issued: **2 August 2022**
- Date Due: **16 August 2022** at **11:00am**
- Submission Procedure: **Upload to ClickUP**
- Submission Format: **archive (zip or tar.gz)**

1 Introduction

1.1 Objectives

In this practical you will:

- implement the Template Method design pattern;
- implement the Factory Method design pattern
- implement the Prototype design pattern;
- implement the Memento pattern; and
- integrate the patterns.

1.2 Outcomes

When you have completed this practical you should:

- understand the Template Method and be able to use C++ concepts like virtual functions and inheritance to implement it;
- understand how the Factory Method delegates object creation to its subclasses;
- notice the difference between the Prototype and the Factory Method and understand which to use where; and
- apply the Memento to store the state of objects and re-instate the state at a later stage.

2 Constraints

1. You must complete this assignment individually.
2. You may ask the Teaching Assistants for help but they will not be allowed to give you the solutions.

3 Submission Instructions

You are required to upload all your source files (that is `.h` and `.cpp`), your Makefile, UML diagrams as individual or a single PDF document and any data files you may have created, in a single archive to ClickUP before the deadline.

4 Mark Allocation

Task	**Marks**
Defining Enemies	27
Creating Enemies	48
Clone the Squad Members	15
Adventure of a Life Time	30
Bonus	10
TOTAL	130

5 Assignment Instructions

Hudson Soft has approached you to assist him in recreating the classic game title, "Adventure Island". He would like to incorporate ideas of modern games like Dota and World of Warcraft into this original game. The most significant changes he would like to implement, is to allow the user to control multiple players in a squad. He also wants to continuously extend and expand the game without much rework or recoding. Being a sensible computer scientist, you decide to model the game using Design Patterns and UML Class diagrams and implement the solution in C++ to produce a game that is portable between platforms.

Task 1: Defining Enemies ..... (27 marks)

Enemies can be one of four types:

- Snake
- Jaguar
- Gorilla
- Cannibal

1.1 Create an abstract class `Enemy`. Each Enemy has: (10)

- Health Points (HP)
- An attack move
- The damage it inflicts
- A defensive move

Enemies also have an `attack` method. Although different types of Enemies have different fighting styles, all attacks follow the same basic steps. The pseudocode for the attack method is given by:

```
function attack(SquadMember)
    while SquadMember still has Health Points and SquadMember still alive
        if hitSquadMember function returns true
            SquadMember died
            Enemy celebrates
        else
            if getHit function returns true
                Enemy dies.
            endif
        endif
    endwhile
endfunction
```

Each Enemy therefore has the following operations (functions or methods):

- relevant constructors and a destructor
- `attack` which takes an instance of a SquadMember as parameter and returns `void`
- `hitSquadMember` which takes an instance of a SquadMember as parameter and returns `bool`, defined by `bool hitSquadMember(SquadMember* z)`
- `celebrate` which takes no parameters and returns `void`
- `getHit` which takes an instance of a SquadMember as parameter and returns `bool`, defined by `bool getHit(SquadMember *z)`
- `die` which takes no parameters and returns `void`

1.2 Write the `Snake`, `Jaguar`, `Gorilla` and `Cannibal` classes which inherit from the `Enemy`. These subclasses will not override Enemy's `attack` method, because their attacks always follows the same pattern. Instead, the subclasses will implement the 4 methods (primitive operations) as follows: (16)

1. `bool hitSquadMember(SquadMember* z)` – The enemy hits the squad member by calling the SquadMember's `takeDamage` method. `takeDamage` takes as a parameter the damage done by the enemy and it returns the squad members remaining HP. `hitSquadMember` should return true if the squad member is killed (i.e. when its HP is ≤ 0).
2. `void celebrate()` – If the squad member dies, the enemy celebrates triumphantly.
3. `bool getHit(SquadMember *z)` — If the squad member is still alive, it attacks the enemy. The damage done by the squad member can be obtained by calling the SquadMember's `getDamage` method. The damage done by the squad member should be subtracted from the enemy's HP. The `getHit` function should return true if the enemy is killed.
4. `void die()` – If the squad member's hit kills the enemy, the enemy dies.

Write your own comprehensive main program to test your code.

The following table shows what each of the operations should output for each of the Enemy subclasses:

Method Name	**Snake**	**Jaguar**	**Gorilla**	**Cannibal**
hitSquadMember (SquadMember* z)	Snake wraps around <memberName> and uses <primaryWeapon>.	Jaguar leaps toward the <memberName> and deliver's a forceful <primaryWeapon>.	Gorilla slams his fists on the ground, growls and hits <memberName> with <primaryWeapon>.	Cannibal rushes towards <memberName> with a <primaryWeapon>.
celebrate()	Player tried strongly till the end.	Should have fought harder soldier.	Player tried in vain trying to save himself.	Screams with his last breath, "I am your father'.
getHit (SquadMember* z)	Slithers rapidly searching for safety and employs <defensiveMove>.	Growls in pain as he is maimed. Jaguar turns around and delivers <defensiveMove> against <memberName>.	Roars and hits his chest in anger.	The other villagers come running deploying <defensiveMove>.
die()	Hisses and curls up as he is defeated.	Gives one last growl before falling over.	The earth shakes as the gorilla falls to the ground.	Shakes his <primaryWeapon> at the player's remains.

1.3 Which design pattern have you just implemented? (1)

Task 2: Creating Enemies ..... (48 marks)

In this task, you will use a factory to create the enemies and initialise all their member variables. The class definitions for the concrete creator participant is given by:

```
class EnemyFactory
{
public:
    EnemyFactory() {}
    virtual ~EnemyFactory() {}

    // This pure virtual function should be overridden by subclasses.
    // Notice that it returns a pointer to a Enemy.
    // Remember this in your implementation.
    virtual Enemy* createEnemy(string) = 0;

protected:
    string getName();
```

```
};
```

2.1 Create the subclasses of the EnemyFactory. These are defined as SnakeFactory, JaguarFactory, GorillaFactory and CannibalFactory (the ConcreteCreator participants). These subclasses need to be separated into different `.h` and `.cpp` files named according to their classnames. Write your own comprehensive main program to test your code. (12)

2.2 Implement the `getName()` function in EnemyFactory to return a randomised name each time the function is called. Write your own comprehensive main program to test your code. (5)

2.3 The subclasses must implement the `createEnemy` method that requires the enemy's attack and defence moves and returns a pointer to the newly created Enemy. Write your own comprehensive main program to test your code. (12)

To set the name of the enemy, use the `getName()` in the base class of EnemyFactory to set the enemy's name upon construction.

```
Enemy* createEnemy(string attack, string defense);
```

Below is a table containing the values to which the member variables of the different enemies should be initialised (by their respective `createEnemy` functions). $\mathcal{N} \sim (\mu, \sigma)$ represents a value drawn from a normal distribution with a mean of μ and a standard deviation of σ. $\mathcal{U} \sim (a, b)$ represents a value drawn from a uniform distribution with a minimum value of a and maximum value of b, both (inclusive).

Attribute	**Snake**	**Jaguar**	**Gorilla**	**Cannibal**
HP	$\mathcal{N} \sim (6, 1)$	$\mathcal{N} \sim (10, 3)$	$\mathcal{U} \sim (4, 12)$	$\mathcal{U} \sim (30, 8)$
Damage	2	4	1	6

Hint #1: In order for the Concrete Creators to be able to assign values to an enemy's member variables, you will have to add setters to the Enemy class. For easy construction, call a parameterised constructor of the Enemy class from the derived classes.

Hint #2: To implement a function to return a value drawn from a $\mathcal{N} \sim (\mu, \sigma)$ distribution, refer to the Box–Muller transform `https://en.wikipedia.org/wiki/BoxMuller_transform`. Also methods in C++11 Numerics library may be used.

2.4 Use Visual Paradigm to draw a UML Class diagram of the classes and their relationships. (10)

2.5 Use Visual Paradigm to draw a UML Object diagram of the objects and their relationships when the first enemy is defeated. (5)

2.6 In your PDF describe how the Gang of Four Factory Method pattern is realised by your implementation, i.e. which participants of the GoF structure map to your classes, and which methods in GoF map to your methods. (3)

2.7 Which design pattern was implemented in this part? (1)

Task 3: Clone the Squad Members ..... (15 marks)

3.1 Create and implement SquadMember.cpp file which keeps track of the member's HP, damage that it deals and name. You should create appropriate setters and getters for HP and damage. Name should only have a getter. (10)

3.2 Add a `clone` function to the SquadMember class. The clone function should return a pointer to a new SquadMember. The member variables of the new SquadMember should be initialised to the same values, except the name, as those of the SquadMember it was cloned from. You are free to implement the approach on how the name should be set, however your implementation will be considered when marked. (5)

```
SquadMember* clone();
```

Task 4: Adventure of a Life Time ..... (30 marks)

4.1 Implement a new incarnation of "Adventure Island" in a terminal that uses all of the patterns that you have used thus far in this assignment. You **MUST** use your existing code to assist in building your game. Create a main.cpp file and makefile to allow for building and executing your game. Bonus marks will be awarded for effort and creativity. (10)

4.2 Expand your game by implementing the Memento pattern in **TWO** distinct aspects of the game, e.g. (a) to undo a move and (b) to save and restore your progress. Your are required to explain in your PDF how you expanded your game, what the functions of the Memento patterns are, and what of your code realises the GoF description of the pattern. (10)

4.3 Draw the final UML class diagram showing all the classes and relationships between the classes for your game. Save the diagram as *SystemUMLClassDiagram.pdf* and make sure you upload it along with all your source code and other files. (10)

Task 5: Bonus ..... (10 marks)

Implement a Save and Restore function using the **Memento** pattern to allow a user to save the current state of the game to a file on disk. When the game is launched, the user should be requested whether a new game is to be started or an existing saved game state be loaded.