**Exercise**

**Develop the following:**

**1.** Create a class called *Player* and add in different member variables, like *name*, *age*, *velocity*, *health*, *stamina*, etc.

**2.** Create a member function called *Initialize()* that will prompt the user to enter the player’s stats. Do as much error trapping here as possible.

**3.** Create a member function called *DisplayStatus()* to display all of the above player data on screen. Use your best ASCII art skills here.

**4.** Instantiate a player object on the heap and get the user to initialize him with values. Don’t forget to delete the player again. If you are comfortable with smart pointers, use those instead.

**5.** Use a set of constant values where applicable and assign those to the player as well. Perhaps his health should begin with a const value of 100?

**6.** Add member functions to the class to give the player behaviour, such as eating, running, attacking, etc.

**7.** Use the class’ data members interchangeably within these functions, considering the following:

**a.** Eating should improve health and stamina

**b.** Running tires the player out

**c.** Attacking uses power to attack with but could affect health

**d.** If player’s stamina is low or they are too old, they won’t have the energy to do much

**e.** Make sure player’s health and stamina remain between **0** and **100%**

**8.** Create a function in the *main.cpp* source file called *PlayGame()* that takes in the player object. How should we pass the player object through?

**9.** Within *PlayGame()*, present the user with a menu of options for the player to perform. Each option will invoke one of the above created member functions.

For instance, if the user chooses to eat, call the player class’ *Eat()* function. If the wants to run, call the *Run()* function, etc.

**10.** The *PlayGame()* function should be called from within a main game loop within *main()*. End the loop as soon as the player decides to quit or their health expires. Error trap wherever necessary.

**11.** Turn the *Player* class into an *Abstract Base Class* and turn the *Eat()*, *Run()* and *Fight()* member functions into *pure virtual functions*.

**12.** Create two or three child classes called *Wizard*, *Thief*, *Mercenary*, etc. and make sure they derive from the *Player* class. Make sure that they have their own member variables and that they successfully override the base class’ member functions.

**13.** In the child classes, add in the specific behaviour to each member function. For example, the Wizard might eat very little and cast a spell when fighting. The Mercenary will fight like mad and run far without becoming exhausted, etc.

**14.** In *main.cpp*, add options to the menu to allow the user to choose what type of player they wish to create and play with. Make sure to dynamically create the correct player on the heap.

**15.** Create an *Enemy* class that has its own properties and behaviour. Write the appropriate constructors, destructors, getters and setters and create a robust class.

**16.** Create a *Game* class that will control the entire game, a menu system and the overall gameplay. The main game loop should exist within this class.

**17.** In the game, instantiate a player and an array of enemies and try to get the player to attack the enemies and vice versa. Think about how you want the *Player* and *Enemy* classes to interact with each other

**18.** The *Player* object could be passed into the *Enemy* object and the enemy will access the player’s health when attacking him (or vice-versa). You could also create a *Fight()* function in the *Game* class that takes in a *Player* and *Enemy* object. Both classes will use their getters and setters to subtract health when they attack each other

void Game::Fight(const Player& player, const Enemy& enemy)

{

//if player attacks enemy decrease enemy health

//if enemy attacks player, decrease player health

}

**19.** Create an *Inventory* class and use *aggregation* to contain it inside the *Player* class

**20.** The *Inventory* class should contain all data and functions that relate to inventory such as *item name*, *value of item*, *DisplayItem()*, etc.

**21.** Instantiate an array of *Inventory* objects in the *Player* class’s constructor. Don’t forget to delete the array. Do you remember how to do this?

**22.** Add a *AddInventory()* function to *Player* so that items can be added. This routine should call into the *Inventory* class to then assign item details.

**23.** Create a *DisplayInventory()* class in *Player* that loops through the inventory array and display all items the player is carrying

**24.** Update your *Enemy* class so that it too contains some components. What should an enemy carry? Think of the has-a model for this.

**25.** Update your *Game* class so that it contains the *Player* and *Enemy*. This makes good sense since the game has a player and the game has an enemy.

**26.** Update your *Enemy* class such that classes such as *Troll*, *Goblin* or *Dragon* derive from it.