# Week 9 – Session 1: Inheritance and composition

In the laboratory sessions this week you are going to explore some advanced class features in OOP programming with C++ namely inheritance and composition. These are two techniques which you can use to extend your class architectures but are very different in nature.

- Inheritance (aka generalisation) is an 'is a' relationship. For example, a student 'is a' person. And a lecturer 'is a' person also. So, it makes sense for them to be derived from a common base class person.

- Composition is different in that it is a 'has a' relationship. For example, an airplane 'has a' rudder and 'has a' number of engines.

This week we are going to build a multi class hierarchy for simulated role-playing game (RPG).

The class hierarchy is shown on the next page.

**Note: The class getters and setters are not shown but should be implemented**.

## Challenge 1: Implement the Item base class

The class hierarchy may look complex, but the beauty of object-oriented design and implementation is that we can treat each class as an independent unit and start implementing the code at the base of the architecture.

### Task 1 Declare the Item class - duration 10 minutes

Create a new VS solution and win32 project. Add an Item class and define the class interface in the `Item.h` file

### Task 2 Item class constructors - duration 5 minutes

Implement the Item class constructors in the `Item.cpp`.

### Task 3 Implement the Getters and setters - duration 15 minutes

Implement the Item class getters and setters in the `Item.cpp`.

### Task 4 Implement ToString function- duration 5 minutes

Implement the `ToString` function of the Item class. The function should simply return a string in the format:

```
<ItemName, worth <ItemValue> Gold coins
```

After you have a fully implemented item class, create some items and test it.

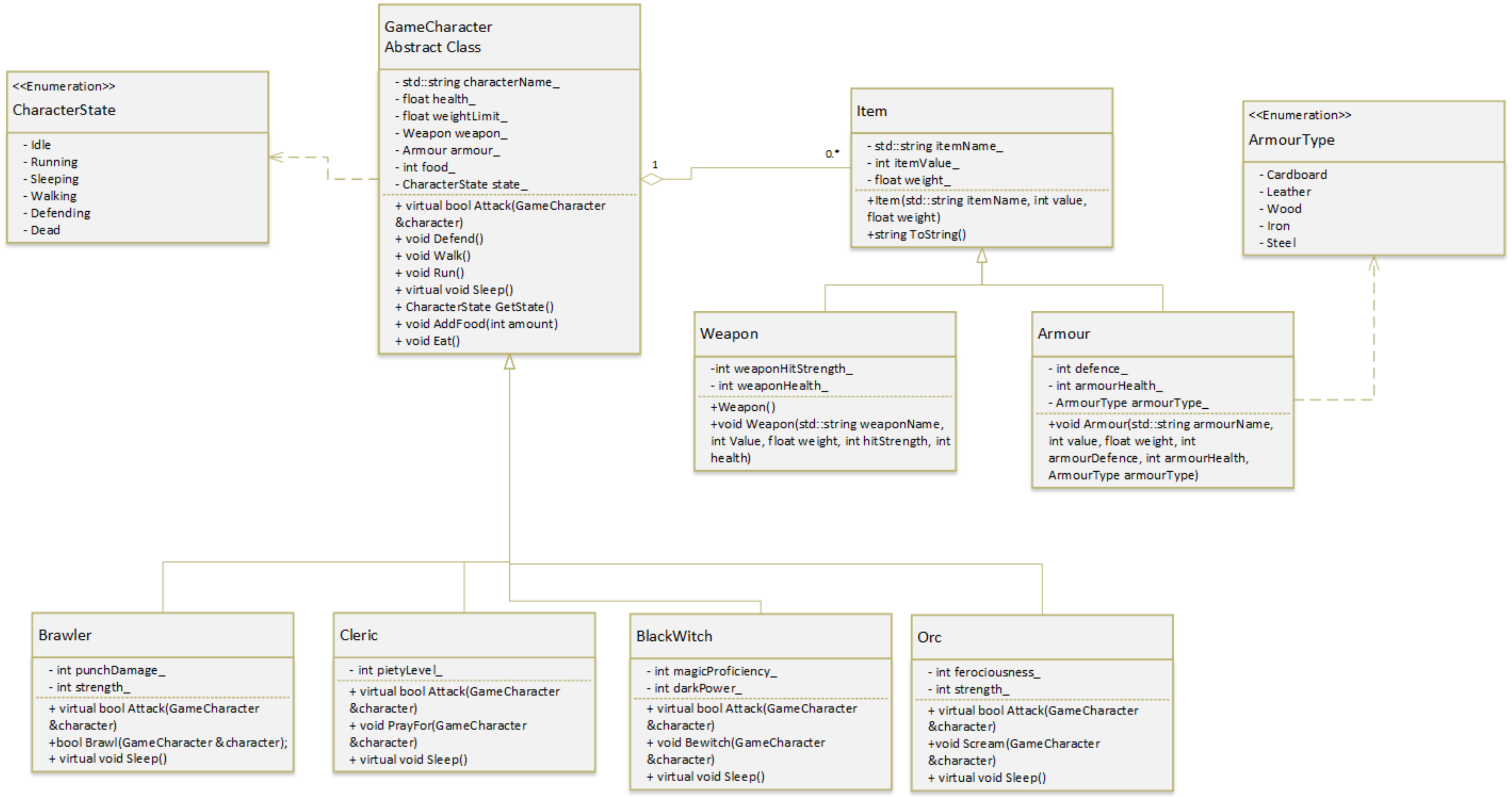

Figure 1: UML class diagram to complete

## Challenge 2: Implementing the weapon class

Now we have implemented our item base class we can proceed by extending this base class and derive a weapon and armour class from this. We will start with the weapon class.

### Task 1 Declare the Weapon class - duration 10 minutes

Create a `Weapon` header and implementation file. Declare the `Weapon` class and inherit from the `Item` base class.

### Task 2 Weapon class constructors - duration 5 minutes

Implement the `Weapon` class constructors in the `Weapon.cpp`.

### Task 3 Implement the Getters and setters - duration 15 minutes

Implement the Weapon class getters and setters in the `Weapon.cpp`.

### Task 4 Implement ToString function- duration 10 minutes

Implement the `ToString` function of the `Weapon` class. The function should simply return a string in the format:

```
Weapon: <ItemName>, Worth: <ItemValue> gold coins, Hit strength:
<HitStrength>, Weapon health: <WeaponHealth>
```

Create some weapon objects.

### Task 4 Explore a little - duration 10 minutes

Create a weapon object and print its details to screen. Use the debugger to explore the internals of the weapon object. You should see that the weapon consists of an item object and additional attributes that are unique to the weapon class.

## Challenge 3: Implementing the Armour class

It is all well and good running around our imaginary RPG with weapons, but we also need some armour to protect our game characters.

### Task 1 Declare the Armour class - duration 10 minutes

Create an `Armour` header and implementation file. Declare the `Armour` class and inherit from the `Item` base class. `Armour` in our game can be made from five different types of material. We can use an enumeration for this. Add the following enumeration to your `Armour.h` file

```
enum ArmourType { Cardboard, Leather, Wood, Iron, Steel };
```

## Task 2 Armour class constructors - duration 5 minutes

Implement the `Armour` class constructors in the `Armour.cpp`.

## Task 3 Implement the Getters and setters - duration 15 minutes

Implement the `Armour` class getters and setters in the `Armour.cpp`.

## Task 4 Implement ToString function- duration 10 minutes

Implement the `ToString` function of the Armour class. The function should simply return a string in the format:

```
Armour: <ItemName>, Worth: <ItemValue> gold coins, Defence:
<Defence>, Armour health: <ArmourHealth> Armour material:
<ArmourType>
```

Create some armour objects.