Laboratory 6

Objective
Become familiar with the OOP concepts

Problem 1
Let's have some fun and implement a little game skeleton, using what we already know about Python programming. This game will use a class hierarchy, modeling an army, as follows.

The game will have one or more instances of a `Player` class. This class represents the player of the game. The game starts with a player having 100 coins in currency. These coins will be used to buy soldiers of other resources. Also the `Player` class will have an `Army` instance in which the soldiers will be recruited. When a soldier is recruited, the currency amount will decrease with the corresponding cost value.

Let's say we start with 100 coins, then a Swordsman is recruited, which costs 20 coins, the Player's amount will become 80 coins.

The `Army` class has a name and holds a list with `Soldier` instances. It also has a method `fight(self, army)` modeling a fight between armies.

There are some classes that should be implemented. A base class called `Soldier`, which is the superclass for all the other militaries involved in this game. It has a `type` attribute, which is filled in in subclasses, an `attack` and a `defense` attributes which signify how attack and defense capabilities, a boolean attribute called `armored` (if it wears armor or not) and a `cost`, saying how much coins will spend a `Player` for it. The `Soldier` class will have a `weapon` attribute, filled in in subclasses, saying what kind of weapon this soldier will handle.

The `Soldier` can yell hurray (`say_hurray()` method), but we'll implement this in the subclasses, every specific soldier yell in a specific way. The Soldier has an `acknowledge(self)` method which will say `"I was informed by the Scouter"` when the `Scouter` calls `inform_army(self, army)` method. The Soldier has a `die(self)` method which will remove it from its army if he was killed.

A `Soldier` has a `cost`, it can be bought only if the Player can afford.

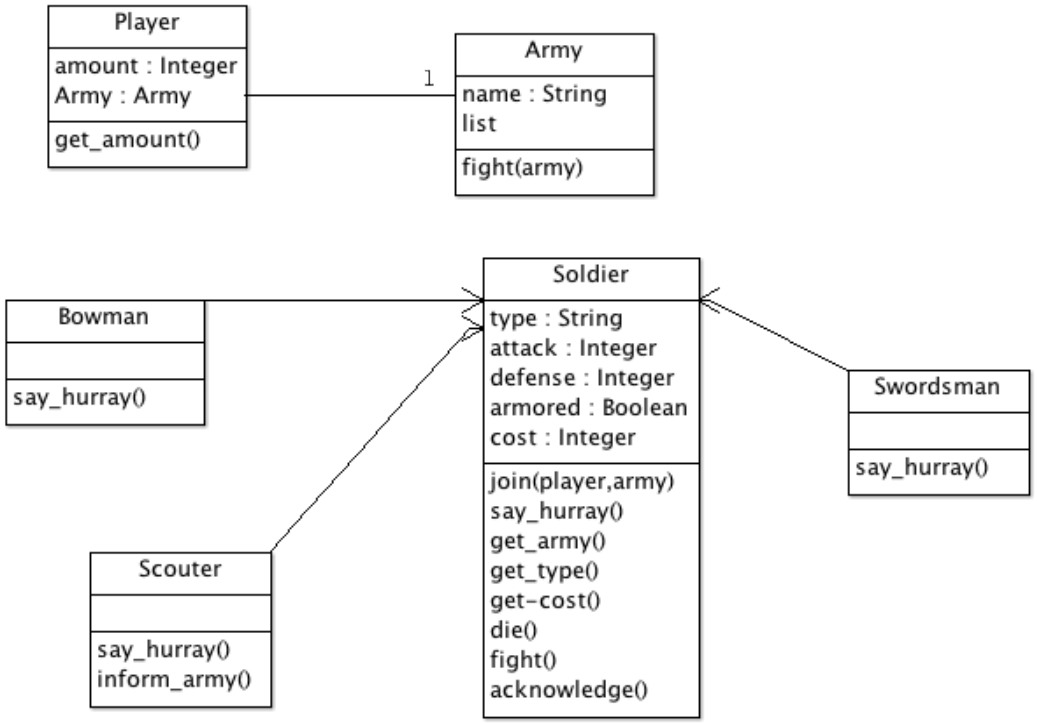
Also it can `join()` to a specific `Army`. We can have as many armies we want in this game, but they are strictly related to the number of `Player` instances.

Also, we'll have a `Scouter` which will discover the map, informing the armies about the terrain and the enemies (`inform_army(self, army)` method). We'll have a `Swordsman` and a `Bowman` which are part of the army.

The main program should be able to instantiate soldiers, allow them to join to different armies, fight each other and eventually die.

	Scouter	**Bowman**	**Swordsman**
cost	15	30	20
attack	50	110	100
defense	20	30	50
armored	No	No	Yes

A more detailed UML class diagram is below:



The main program should look like below (just a suggestion):

```
from swordman import Swordman
from bowman import Bowman
from scouter import Scouter
from army import Army
from player import Player


red_army = Army("Red Army", [])


player_one = Player(100, red_army)


black_army = Army("Black Army", [])


player_two = Player(100, black_army)

swordman_one = Swordman("Short sword")
swordman_one.join(player_one, red_army)
swordman_one.say_hurray()

swordman_two = Swordman("Short sword")
swordman_two.join(player_one, red_army)
swordman_two.say_hurray()

bowman_one = Bowman("Long bow")
bowman_one.join(player_two, black_army)
bowman_one.say_hurray()

scouter_one = Scouter("Spear")
scouter_one.join(player_two, black_army)
scouter_one.say_hurray()

scouter_one.inform_army(black_army)

print(red_army.list)
print(black_army.list)

print(red_army.list)

red_army.fight(black_army)
print(black_army.list)
print(red_army.list)
```