Joint program

Algebra X Epitech

Object Oriented Programming

Exam

1.1

All exercices are to be done either in Java or in C++.
Examples are in Java but should work the same in C++.

Unless specified otherwise, all messages must be followed by a newline.

Unless specified otherwise, the getter and setter name will always add “get” or “set” in front of the name of the attribute, in CamelCase.

EXERCISE 01

Files to hand in: ./Character.(java|cpp)

Create an abstract `Character` class that is composed of the following protected attributes: “name”, “life”, “agility”, “strength”, “wit”, and a constant “RPGClass” string attribute, with the corresponding getters.
These attributes must have the following values by default:

- name: first argument passed to constructor
- RPGClass: second argument passed to constructor
- life: 50
- agility: 2
- strength: 2
- wit: 2

Add an **attack** method that takes a string as argument, and prints the following (whatever the argument):

```
[name]: Rrrrrrrrr....
```

Of course, [name] must be replaced by your character’s name.

Here is an example in which `TestCharacter` is an implementation of our abstract class which doesn’t change any attributes:

```
public class Example {
    public static void main(String[] args) {
        Character perso = new TestCharacter("Jean-Luc");

        System.out.println(perso.getName());
        System.out.println(perso.getLife());
        System.out.println(perso.getAgility());
        System.out.println(perso.getStrength());
        System.out.println(perso.getWit());
        System.out.println(perso.getRPGClass());
        perso.attack("my weapon");
    }
}
```

Terminal

```
~/Algebra X Epitech> java Example
Jean-Luc
50
2
2
2
SomethingSetByTestCharacter
Jean-Luc: Rrrrrrrrr....
```

EXERCISE 02

Files to hand in: ./Character.(java|cpp)
./Warrior.(java|cpp)
./Mage.(java|cpp)

Create the `Warrior` class as well as a `Mage` class, which *extends* the `Character` class.
Modify each class's attributes as follows:

Warrior
- RPGClass: "Warrior"
- life: 100
- strength: 10
- agility: 8
- wit: 3

Mage
- RPGClass: "Mage"
- life: 70
- strength: 3
- agility: 10
- wit: 10

These two classes must each implement the **attack** method.
Its parameter defines the weapon used to attack.

The Warrior can attack with a "hammer" or a "sword".
If anything else is passed as parameter, he doesn't attack.
The "Warrior" class's **attack** method must display:

```
[name]: Rrrrrrrrr....
[name]: I'll crush you with my [weapon]!
```

The Mage can attack with "magic" or with a "wand".
If anything else is passed as parameter, he doesn't attack.
The "Mage" class's **attack** method must display:

```
[name]: Rrrrrrrrr....
[name]: Feel the power of my [weapon]!
```

Of course, [name] must be replaced by your character's name, and [weapon] by your character's weapon.

Our characters are proud and they like to announce themselves on the battlefield.
You will make sure that when creating a "Warrior" or "Mage" object, a message is written in the following format:

```
[name]: My name will go down in history!
```

for a Warrior, and

```
[name]: May the gods be with me.
```

for a Mage.
Here is an example:

```
public class Example {
    public static void main(String[] args) {
        Character warrior = new Warrior("Jean-Luc");
        Character mage    = new Mage("Robert");

        warrior.attack("hammer");
        mage.attack("magic");
```

```
        }
    }
```

Terminal

```
~/Algebra X Epitech> java Example
Jean-Luc: My name will go down in history!
Robert: May the gods be with me.
Jean-Luc: Rrrrrrrrr....
Jean-Luc: I'll crush you with my hammer!
Robert: Rrrrrrrrr....
Robert: Feel the power of my magic!
```

Exercise 03

Files to hand in: ./Character.(java|cpp)
./Warrior.(java|cpp)
./Mage.(java|cpp)
./Movable.(java|cpp)

We now have characters who can be Mages or Warriors.
They can attack, fair enough, but they still cannot move! This is bothersome…
In order to add this behavior to our classes, we are going to create an interface called `Movable` that contains the following methods: **moveRight**, **moveLeft**, **moveForward**, and **moveBack**.

This interface must be implemented by the `Character` classes.

These methods must display the following messages, respectively:

```
[name]: moves right
[name]: moves left
[name]: moves forward
[name]: moves back
```

Exercise 04

Files to hand in: ./Character.(java|cpp)
./Warrior.(java|cpp)
./Mage.(java|cpp)
./Movable.(java|cpp)

Paralysis is over!
Our characters can now move, but, being so proud, they want more!
Our friend Warrior refuses to be compared to a small and skinny Mage.
While the Warrior moves in a bold and virile manner, the Mage moves delicately!

To satisfy our boorish Warrior, implement overrides for the **Movable** methods inherited by `Character`.
Your methods must display the following messages that correspond to the class that overrides them:
for a warrior:

```
[name]: moves right like a bad boy.
[name]: moves left like a bad boy.
[name]: moves back like a bad boy.
[name]: moves forward like a bad boy.
```

for a mage:

```
[name]: moves right furtively.
[name]: moves left furtively.
[name]: moves back furtively.
[name]: moves forward furtively.
```

Here is an example:

```
    public class Example {
        public static void main(String[] args) {
            Warrior warrior = new Warrior("Jean-Luc");
            Mage mage       = new Mage("Robert");

            warrior.moveRight();
            warrior.moveLeft();
            warrior.moveBack();
            warrior.moveForward();
            mage.moveRight();
            mage.moveLeft();
            mage.moveBack();
            mage.moveForward();
        }
    }
```

Terminal

```
~/Algebra X Epitech> java Example
Jean-Luc: My name will go down in history!
Robert: May the gods be with me.
Jean-Luc: moves right like a bad boy.
Jean-Luc: moves left like a bad boy.
Jean-Luc: moves back like a bad boy.
Jean-Luc: moves forward like a bad boy.
Robert: moves right furtively.
Robert: moves left furtively.
Robert: moves back furtively.
Robert: moves forward furtively.
```

EXERCISE 05

Files to hand in: ./Character.(java|cpp)
./Warrior.(java|cpp)
./Mage.(java|cpp)
./Movable.(java|cpp)

Our characters are now customized to talk, walk and attack.
Yet, they still can't unsheathe their weapons!
Being able to attack is nice, but attacking while the weapon is still in its sheath is going to be difficult…

You will agree that, whether Warrior or Mage, the character will draw his weapon the same way.
This is why, you will make sure that the `Character` class implements the **unsheathe** method so that both "Warrior" and "Mage" inherit from it.
However, you will also make sure that the **unsheathe** method cannot be overrided by "Warrior" and "Mage".

This method must display the following text when called:

```
[name]: unsheathes his weapon.
```

EXERCISE 06

Files to hand in: ./Character.(java|cpp)
./Warrior.(java|cpp)
./Mage.(java|cpp)
./CharacterFactory.(java|cpp)
./Movable.(java|cpp)

We want to be able to create any kind of `Character` from a single place. So let's use the factory design pattern and create a `CharacterFactory`.
This class must have a `create` method which takes two strings as a parameters (a character type and a name) and returns a new instance of the corresponding character type with the given name.

- If the parameter is "warrior", it returns a `Warrior`;
- If it is "mage", it returns a `Mage`;
- Else it returns `null`

The `create` method must be case-insensitive.