SCALE FOR PROJECT PISCINE CPP (/PROJECTS/PISCINE-CPP) / DAY 04 (/PROJECTS/42-PISCINE-C-FORMATION-PISCINE-CPP-DAY-04)

Introduction

The subject of this project is rather vague and leaves a lot to the user's choice. This is INTENDED. The questions in this grading scale, however, are very focused and concentrate on what we think is the core of each exercise, what we want you to grasp. So we would like you to do the same : You can and should tolerate moderate deviations in filenames, function names, etc ... as long as the exercise basically works as intended. Of course, in case the student you are grading really strayed too far, you should not grade the exercise in question at all. We leave it to your good judgement to determine what constitutes "straying too far".

The usual obvious rules apply : Only grade what's on the git repository of the student, don't be a dick, and basically be the grader you would like to have grading you.

Do NOT stop grading when an exercise is wrong.

Guidelines

You must compile with clang++, with -Wall -Wextra -Werror

Any of these means you must not grade the exercise in question:
- A function is implemented in a header (except in a template)
- A Makefile compiles without flags and/or with something other than clang++
- A class is not in Coplien's form

Any of these means that you must flag the project as Cheat:
- Use of a "C" function (*alloc, *printf, free)
- Use of a function not allowed in the subject
- Use of "using namespace" or "friend" (Unless explictly allowed in the subject)
- Use of an external library, or C++11 features (Unless explictly allowed in the subject)

Attachments

Subject (https://cdn.intra.42.fr/pdf/pdf/2731/d04.en.pdf)

ex00

As usual, there has to be a main function that contains enough tests to prove the program works as required. If there isn't, do not grade this exercise. If any non-interface class is not in Coplien's form, do not grade this exercise.

Destructor chaining

The destructors in Victim and derived are virtuals

✓ Yes ✗ No

Easy subclass

There is a Peon class that inherits publicly from Victim.
It has the correct outputs.

✓ Yes ✗ No

Victim

There is a Victim class.
It has a name.
The required outputs on construction and destruction are present.
The required overload of operator << to ostream is present and works correctly.

✓ Yes ✗ No

Thorough testing

There are tests in the main with derived classes other than Peon, and everything works well with them.

✓ Yes ✗ No

I want sheeps !

The Victim can getPolymorphed() const, with the correct output.
The Sorcerer can polymorph(Victim const &) const.

✓ Yes ✗ No

Sorcerer

There is a Sorcerer class.
It has a name and a title.
It has a constructor with name and title.
It cannot be instanciated without parameters.
That means either the default constructor must be private, or it must be declared but non-implemented, to comply with Coplien's form.
The required outputs on construction and destruction are present.
The required overload of operator << to ostream is present and works correctly.

✓ Yes ✕ No

ex01

As usual, there has to be a main function that contains enough tests to prove the program works as required. If there isn't, do not grade this exercise. If any non-interface class is not in Coplien's form, do not grade this exercise.

Character

There is a Character class.
It has the attributes required by the subject : name, AP, pointer to AWeapon.
It has the required AP behavior : 40 on start, lose X AP on attack depending on the weapon, and recover 10 with recoverAP up to maximum of 40. attack(...) fails if there isn't enough AP.

✓ Yes ✕ No

Concrete weapons

There are concrete PlasmaRifle and PowerFirst weapons. (So, they inherit from AWeapon)
They have the attributes and attack() outputs specified by the subject.

✓ Yes ✕ No

Utility and output

The equip() and attack() functions work as required.
The << overload works as required.

✓ Yes ✕ No

Destructor chaining 2

The destructors in AWeapon and its derived classes are virtual

✓ Yes ✕ No

Thorough testing

There are tests in the main with more derived weapons and more derived enemies.

✓ Yes ✕ No

Destructor chaining AGAIN

The destructors in Enemy and its derived classes are virtual

✓ Yes ✕ No

Concrete enemies

There are concrete SuperMutant and RadScorpion enemies (That inherit from Enemy, obviously)
They have the required attributes.
The SuperMutant has the required overload of takeDamage() and it works as required.

✓ Yes ✕ No

Enemy

There is an Enemy class.
It has the attributes required by the subject : type, number of HP
Its member functions are implemented coherently.
It has the required check in takeDamage to prevent going under 0 HP

✓ Yes ✕ No

Weapon

There is an AWeapon class.
It is abstract (attack() must be a pure virtual function).
It has the attributes required by the subject : name, damage, AP cost.
Its member functions are implemented coherently.

✓ Yes ✗ No

ex02

As usual, there has to be a main function that contains enough tests to prove the program works as required. If there isn't, do not grade this exercise. If any non-interface class is not in Coplien's form, do not grade this exercise.

Interfaces

The ISquad and ISpaceMarine interfaces are present and are exactly
like the ones in the subject.

✓ Yes ✗ No

Concrete squad

"The Squad class is present and inherits from ISquad
Its member functions work as required.
Its destructor destroys the contained units.

✓ Yes ✗ No

Concrete units

The TacticalMarine and AssaultTerminator classes are present and
inherit from ISpaceMarine
Their member functions work as required.

✓ Yes ✗ No

Assignment and copy

The copy and assignation behaviours of the Squad are as the subject
required. That means deep copy, and upon assignation, exiting units must be
destroyed before they are replaced.

✓ Yes ✗ No

ex03

As usual, there has to be a main function that contains enough tests to prove the program works as required. If there isn't, do not grade this exercise. If any non-interface class is not in Coplien's form, do not grade this exercise.

Interfaces

The ICharacter and IMateriaSource interfaces are present and are exactly like in the subject.

✓ Yes ✗ No

Source

The MateriaSource class is present and implements IMateriaSource.
The member functions work as intended.

✓ Yes ✗ No

Concrete materia

There are concrete Ice and Cure classes that inherit from AMateria
Their clone() method is correctly implemented.
Their outputs are correct.

✓ Yes ✗ No

Character

The Character class is present and implements ICharacter.
It has an inventory of 4 materias.
The member functions are implemented as the subject requires.

✓ Yes ✗ No

Materia base

There is an AMateria class. It has a type.
It's abstract (clone is pure).
The XP system is implemented as the subject requires.

✓ Yes ✗ No

Assignation and copy

The copy and assignation of a Character are implemented as required
(= deep copy, very much like the previous exercise)

Yes No

ex04

As usual, there has to be a main function that contains enough tests to prove the program works as required. If there isn't, do not grade this exercise. If any non-interface class is not in Coplien's form, do not grade this exercise.

DD's patcher !

The mine/beMined dispatch mechanism works as required.
In theory, there should be a beMined(StripMiner *) and a
beMined(DeepCoreMiner*), and the mine() method should call beMined passing
\"this\" as parameter, which would dispatch the call to a method that
depends on the type of the asteroid (subtype polymorphism) and the type of
the laser (adhoc polymorphism). Basically the double-dispatcher design
pattern, just a bit dumber.
Now the clever bit : If the student tries to pass off a technique that
uses typeid, dynamic_cast, the names of the lasers/asteroids, etc ... to
select the output, MARK THE WHOLE PROJECT AS CHEAT and leave it at that,
because it is EXPLICTLY forbidden by the subject.

Yes No

Basics

The IAsteroid and IMiningLaser interfaces are present.
Concrete Asteroids and MiningLasers are implemented.

Yes No

Ratings

Don't forget to check the flag corresponding to the defense

Ok

Empty work Incomplete work No author file Invalid compilation

Norme Cheat Crash Forbidden function

Conclusion

Leave a comment on this evaluation

Preview!!!