Objektorientierte Sprachen

Sommersemester 2021 - Vierte Prüfung

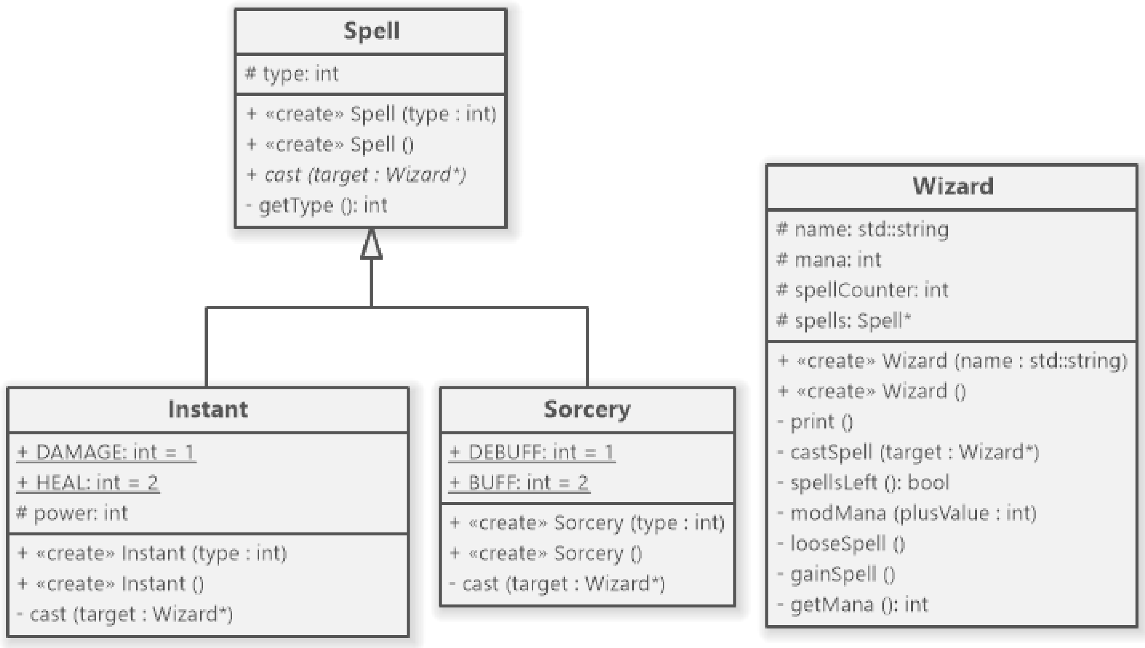
Tasks and submission

Implement a program that allows two wizards to cast spells in alternating order until there are no spells left.
You are given a basic code structure which contains a main-function as well as the header and CPP-files for the abstract class Spell. Expand this structure by implementing additional features for main-function and the abstract class and creating two new classes, which inherit from the abstract class. Additionally define and implement the wizard-class.

Put your code files into a ZIP-archive and upload that archive on Moodle. You cannot hand in any submissions afterwards, even if your submission is faulty or incomplete. The upload is possible until 19:20.

Comments in the code explain the basic code structure, on the following pages you find a class diagram as well as the program's description.

Class diagram



Summary

- At first two wizard objects are created. The constructor is called with the wizard's name as parameter. **Implement the constructor for the class wizard.**
- The wizard's constructor creates seven random spells for the wizard, these are either instant- or sorcery-objects. Both classes inherit from the class spell. **Define the purely virtual function cast for the class Spell, define an inheritance for the two child classes and implement their constructors and cast functions.**
- Afterwards the wizards cast spells until both wizards have no spells remaining. Once a spell has been cast, it cannot be cast again. **Implement the castSpell function for the wizard and an additional function spellsLeft which checks if the wizard has spells remaining.**
- **Implement a print function** to print a wizard's current status.
- During castSpell's execution exceptions may occur. **Implement error handling in the main function.**
- In the end all memory should be freed. **Implement the wizard's destructor.**

Tasks

main-function

- Implement error handling with exceptions for the castSpell-call. Catch invalid_argument-exceptions and generic objects.

Abstract class Spell

Define the following **purely virtual** function for the abstract class:

- **cast(Wizard* target):** The function does not return any value.

Class Instant

Define the class Instant, inheriting from Spell, separated into Header- and CPP-files. There are two types of Instant-objects, damaging and healing. This class has an additional integer-value power. Implement the following functions for the class:

- **Constructor(int type):** The power-value is set to a random value between 1 and 5. The object's type is set to the passed parameter and marks if it is either a damaging or healing spell.
- **cast(Wizard* target):** If the spell is damaging the parameter looses mana equal to the spell's power. If the spell is healing the parameter gains mana equal to the spell's power.

Class Sorcery

Define the class Sorcery, inheriting from Spell, separated into Header- and CPP-files. There are two types of Sorcery-objects, debuffs and buffs. Implement the following functions for the class:

- **Constructor(int type):** The object's type is set to the passed parameter and marks if it is either a debuffing or buffing spell.
- **cast(Wizard* target):** If the spell is debuffing the parameter looses it's next spell. If the spell is buffing and the parameter has less than seven spells, the parameter gains an additional spell. Take care of the memory.

Class Wizard

Define the class Wizard separated into Header- and CPP-files. The class has a Spell-Pointer-Array, fitting seven spells. This array is used to access the Instant- and Sorcery-objects in a polymorphic way. Additionally every wizard has a name (string) and a mana-value (integer). Implement the following functions for the class:

- **Constructor(string name):** The constructor is called with the wizard's name as parameter and sets it accordingly. Then the mana-value is set to 20 and seven random spells are created for the spell-array (50:50 chance to be either an instant or sorcery and 50:50 chance for the spell to be damaging/healing or debuffing/buffing).
- **Destructor:** Frees the memory, allocated for the spell-array by the constructor.
- **print():** Prints the wizard's name and mana-value.
- **castSpell(Wizard* target):** If the parameter is null an invalid_argument-Exception is thrown.
  The wizard casts the next spell at the end of the spell-array. Afterwards this spell cannot be used again. Damaging and debuffing spells use the passed parameter as target, healing and buffing spells use the casting wizard as target.
- **spellsLeft():** This function checks if there are any spells remaining for the wizard.
- You can implement additional functions you might need for the execution of spells.

Grading aspects

Aspect	Grading
Spell	15%
Instant	7.5%
Sorcery	7.5%
Wizard	20%
Error handling with Exceptions	15%
Inheritance, usage of virtual and pure virtual	25%
Memory management	10%
Total	100%

Compiler warnings, memory errors and bad coding style can result in a deduction of points.

Grading

$\geq$ 88%	Excellent
$\geq$ 75% und $<$ 88%	Good
$\geq$ 63% und $<$ 75%	Satisfactory
$\geq$ 50% und $<$ 63%	Sufficient
$<$ 50%	Unsatisfactory