Семинар 10 по ООП

Полиморфизъм

Me: *explains polymorphism*

Friend: So the subclass the same thing as the superclass?

Me:

Що е полиморфизъм

- Един от 4 те принципа на ООП.
- Полиморфизъм е свойството на един обект да се държи като друг. Идеята е да можем да работим с един тип, но да постигаме различно поведение спрямо вътрешните особености на обекта.

Видове полиморфизъм

- Ad hoc Полиморфизъм – Method override
- Параметричен полиморфизъм - Шаблони
- Подтипов полиморфизъм – Нещото на което ще се спрем сега

Подтипов полиморфизъм

- Подтиповият полиморфизъм е вид полиморфизъм, при който един обект от подтип на даден базов тип може да се използва като инстанция на базовия тип, като продължава да си има поведението на подтипа.

Liskov Substitution Principle

- Всеки обект от подтип, трябва да спазва сходен интервейс с базовия тип, за да може на всяко място, където се иска инстанция на базов тип да се сложи, някой подтип.

Статично и динамично свързване

- Както видяхме предния път само наследяване не ни е достатъчно. Когато кастнахме едно животно към тип Animal, можехме да го направим, но при функцията sound се викаше тази по подразбиране за Animal. Значи за да постигнем подтипов полиморфизъм ни трябва още нещо. Това е така защото до сега използвахме статично свързване, където това кой метод се извиква се решава по време на компилация и съответно се извиква на базовия клас.

Трябва ни динамично свързване

- Динамичното свързване избира методът, който да се извика по време на изпълнение на програмата, като се има предвид вътрешния тип на инстанцията. Понеже в C++ стандартното свързване е статично, ние трябва да окажем, че искаме динамично свързване чрез виртуални методи.

Виртуални методи

- Ключова дума virtual
- Виртуалността се наследява т.е е достатъчно да сложим virtual само в класа, който се намира най-горе в йерархията
- Override и final

Важни забележки

- В C++ динамичното свързване работи само за указатели от базовия тип
- Деструкторът също е функция. Като такава той също може да е виртуален и като се прави подтипов полиморфизъм даже трябва да е.

Време за пример

“Виртуален конструктор за копиране”

- Познахте нормалния копи конструктор ще копира родителска инстанция, т.е трябва да направим нещо, което копира каквото е във инстанцията.
- За тази цел ще напишем виртуална функция clone(), която връща указател от базовият тип, който сочи към копие на инстанцията на наследения тип.

Време за пример

Въпроси?

Задачи

- Направете клас кола, който има име, конски сили и метод identify(). Направете наследници SportsCar – освен нормалните неща има и maxSpeed и съответно това се показва в identify, и NormalCar – има comfortIndex, който се показва в identify().
- Направете масив от коли, който може да съдържа различните типове коли.

- Да се реализира клас Character, който да представлява герой в игра. Всеки герой трябва да има име, точки атака и точки живот. Да се реализират следните методи:
- deal_damage_to(Character& other) const - нанася щети по подадения герой;
- take_damage(double points) - поема щети;
- heal(double points) - възвръща точки живот;
- compare_name(const char* name) const - проверява дали героят има същото име като подаденото
- print() const - извежда подробна информация за героя в следния формат:
- `Name: {name}`
- `Type: Ordinary Character`
- `HP: {health_points}`
- `DMG: {damage_points}`
- Да се реализира клас Knight, наследник на клас Character, който представлява рицар в игра. За разлика от обикновения герой, рицарят блокира 25% от нанесените му щети и възвръща с 5% повече точки живот. Освен това при извеждането на информация за рицаря, да се извежда Knight вместо Ordinary Character.
- Да се реализира клас Archer, наследник на клас Character, който представлява стрелец в игра. За разлика от обикновения герой, стрелецът нанася 33% повече щети, но поема 15% повече щети. Освен това при извеждането на информация за стрелеца, да се извежда Archer вместо Ordinary Character.
- Да се реализира клас Game, който представлява игра, в която има списък от герои, които се бият помежду си. Да се реализират следните методи:
- add(Character* character) – добавя герой в списъка;
- battle(const char* attacker, const char* target) – героят с име {attacker} напада героя с име {target}. Ако някой от героите не е част от списъка да се изведе подходящо съобщение. Ако точките живот на героя с име {target} паднат под 0, героят да се премахне от списъка и да се изведе подходящо съобщение;
- heal(double amount) – всички герои в списъка възвръщат {amount} точки живот;
- print() const – извежда подробна информация за оцелелите до момента герои.