Class Relations - Polymorphism

1. Content

2. Objective

At the end of these exercises, you should be able to:

- Make use of **polymorphism through inheritance**.
- Know the constructor and destructor sequence calling when inheritance is involved.
- Explain the meaning of the virtual and override keywords.
- Declare **pure virtual functions**.
- Implement a camera
- Draw flipped images

We advise you to **make your own summary of topics** that are new to you.

3. Exercises

Give your **projects** the same **name** as mentioned in the title of the exercise, e.g. **PolymorphismBasics**. Other names will be rejected.

Always adapt the **window title**: it should mention the name of the project, your name, first name and group.

Create a blank solution with name **W04** in your **1DAExx_04_name_firstname** folder.

3.1. PolymorphismBasics

3.1.1. Create the project

Add a new project with name **PolymorphismBasics** to the W04 solution and delete the generated file **PolymorphismBasics.cpp**.

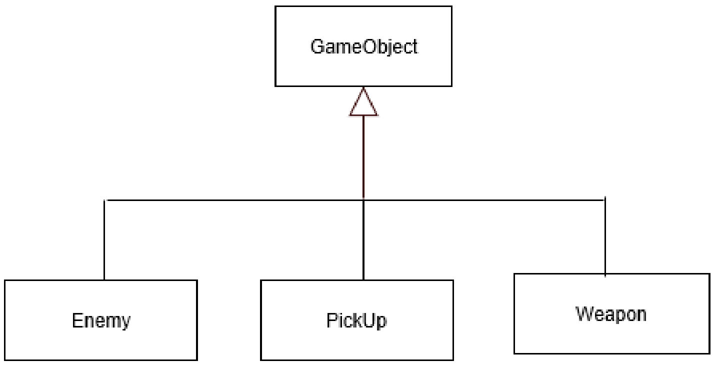
Copy and add the **given code files** – located in the folder CodeAndResources/01_PolymorphismBasics - to the project.

Have a look at the given code. In this project, the gamer can create 3 different types of game objects (enemy, pickup and weapon) and ask a report of the created game objects.

Build and test the application.

3.1.2. Examine the code

Have a look at the classes. Notice that the inheritance tree as depicted in the image below is implemented.



Some tracing has been added to the constructors and destructors of these classes.

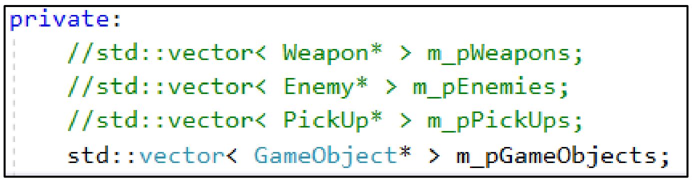
Run the application and notice the sequence in which the constructors and destructors are called when objects that inherit from a base class are created/deleted. Also note that no memory leaks are reported.

3.1.3. Introduce polymorphism

Three vectors are defined in the Game class.

```
private:
    std::vector< Weapon* > m_pWeapons;
    std::vector< Enemy* > m_pEnemies;
    std::vector< PickUp* > m_pPickUps;
```

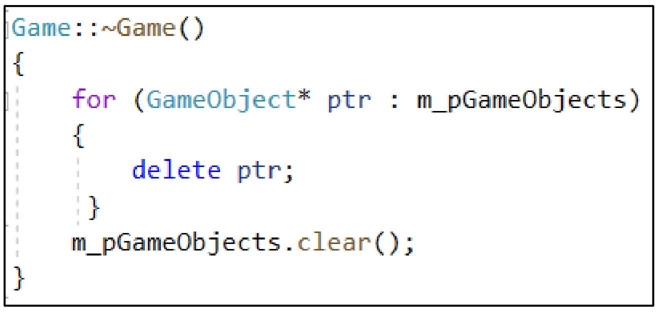
Now use **polymorphism through inheritance** so that you can hold the pointers to the 3 different kind of GameObjects in 1 instead of 3 vectors.



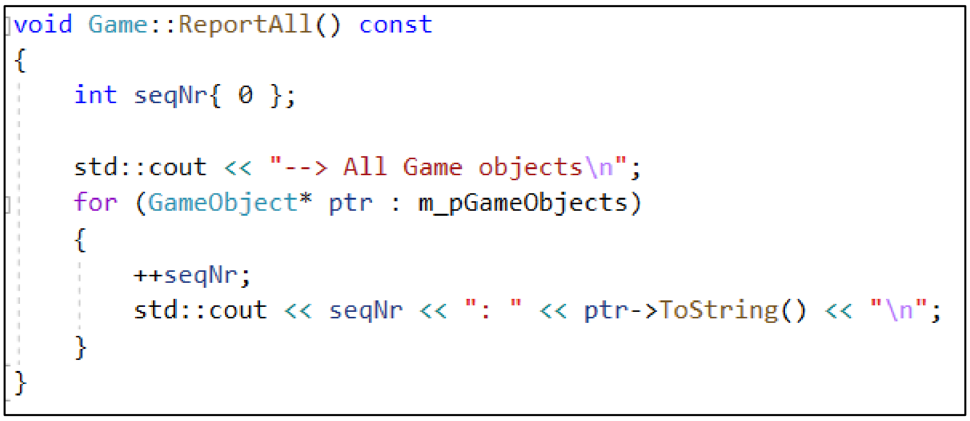
```
private:
    //std::vector< Weapon* > m_pWeapons;
    //std::vector< Enemy* > m_pEnemies;
    //std::vector< PickUp* > m_pPickUps;
    std::vector< GameObject* > m_pGameObjects;
```

Adapt the Add… methods to this new situation.

And that these loops – to delete the game objects, report all the game objects – should work correctly:



```
Game::~Game()
{
    for (GameObject* ptr : m_pGameObjects)
    {
        delete ptr;
    }
    m_pGameObjects.clear();
}
```



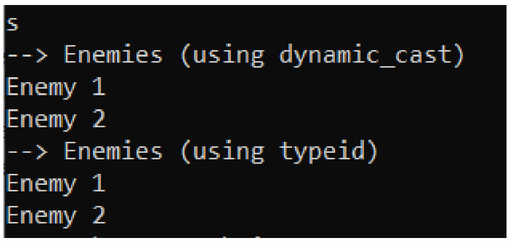
```
void Game::ReportAll() const
{
    int seqNr{ 0 };

    std::cout << "--> All Game objects\n";
    for (GameObject* ptr : m_pGameObjects)
    {
        ++seqNr;
        std::cout << seqNr << ": " << ptr->ToString() << "\n";
    }
}
```

The function **ReportEnemies** also needs a modification to the new situation.

Try the 2 possibilities:

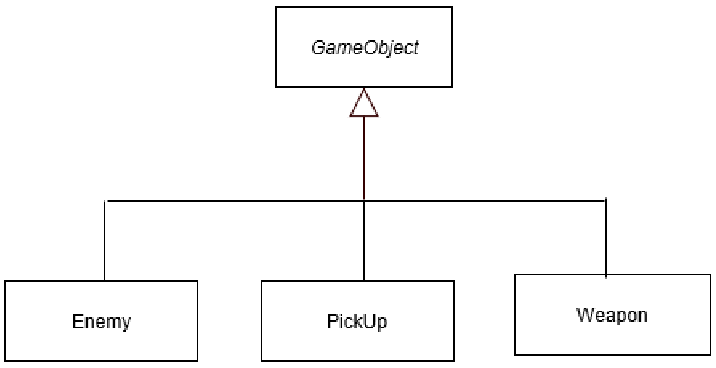
- Using dynamic cast
- Using typeid



```
s
--> Enemies (using dynamic_cast)
Enemy 1
Enemy 2
--> Enemies (using typeid)
Enemy 1
Enemy 2
```

Build, run and verify that the game behaves correctly and that **all the destructors are called** when quitting the game.

After this change the GameObject class has become an abstract class. In UML abstract classes and methods are indicated in **italic** (UML abstract classes and methods).



3.2. PolymorphismShapes

3.2.1. General

In this project you will make the same shapes application as in previous lab but using polymorphism. At the end of this exercise, the application should have the same functionality.

3.2.2. Create the project

Add a new project with name **PolymorphismShapes** to the W04 solution and delete the generated file **PolymorphismShapes.cpp**.

Copy all the **class files** and **main.cpp** file from the previous week's project InheritanceShapes into the folder of this project and add them to this Visual studio project. Also copy the **Resources** folder.

Adapt the project name in the window title.

3.2.3. Add polymorphism

Overwrite the files **Game.cpp** and **h** by the given ones – located in the folder CodeAndResources/02_PolymorphismShapes - and have a look at the code, notice that:

- only **one vector of DaeShape pointers** is defined.
- only **one loop** is used in MoveShapes, DrawShapes and DeleteShapes.

When you build the project, you get an error. Adapt the code in your shape classes (not in the game class) to solve this problem. Build and run to verify that it has the same functionality. Also verify that no memory leaks are reported when closing the application.

Before proceeding, ask your lab teacher to **verify your code**.

3.3. MiniGame

Goto the Minigame assignment and implement part 3.4.

In this exercise you'll:

- Add animations to the avatar
- Add a camera,
- Read the level vertices from a Svg file

4. Game project

Continue working on your game. **After these labs you can implement all the functionality requested for the milestone**.

- Create the svg file of your level (if needed) and load it into your game using the SvgLoader.
- Draw the state diagram of your avatar
- Move the avatar through the level
- Add appropriate animations to the avatar
- Let the camera follow the avatar
- Define the other game object classes (pickups, enemies, …) and create and manage the objects of these classes.
- Implement the interaction between the game objects.
- Meanwhile using polymorphism, composition… whenever it is appropriate
- …

5. Submission instructions

You have to upload the compressed folder *1DAExx_04_name_firstname.* This folder contains 3 solution folders:

- W04
- MiniGame
- Name_firstname_GameName which also contains your up-to-date report file.

Don’t forget to clean the projects before closing them in Visual Studio.

6. References

6.1. UML

6.1.1. UML abstract classes and methods

http://www.oxfordmathcenter.com/drupal7/node/35

6.2. SVG file

6.2.1. Path element

http://www.w3.org/TR/SVG/paths.html#Introduction