14. Inheritance

Ján Dugáček

March 24, 2019

Table of Contents

Motivation

- If there are more classes that solve the same problem in different ways, it's useful to have a class both can be converted to

```
StringSaver asStr;
FileSaver asFile("backup.dat");
saveStuff(asStr);
saveStuff(asFile);
```

Composition

```
class Report : public std::string {
public:
        unsigned int timestamp_ = 0;
};
```

- Class `Report` has all the behaviour of `std::string`, but it also has a public integer attribute `timestamp_`
- It can be implicity converted into a `std::string` when taken as a reference, but only the `std::string` part is taken when it's *copied* into a `std::string`
- `public` keyword means that the parent class' properties don't change, if there was `private` instead, they would all be private and accessible only to the one child class

Composition #2

```
class Report : public std::string {
public:
        unsigned int timestamp_ = 0;
        Report(const std::string& msg) :
        std::string(msg)
        {
                timestamp_ = time(nullptr);
        }
        std::string toString() {
                return std::to_string(timestamp_)
                                + ":" + *this;
        }
};
```

- Child class' constructor calls parent class' constructor
- Child class can use parent's methods as its own

Composition #3

```
class bigClass : public std::string, public complex,
        public std::array<int, 12> {
};
```

- It's possible to inherit from any number of parent classes
- The object can be converted to any of the parent classes and has methods of all the parent classes
- It all works with `struct` as well, but not with `union`
- If some methods have the same names, it uses the current type's method, if there isn't one and more parents have such methods, it has to be specified like `std::string::size()`

Exercise

1. Create a class just like `std::vector` that checks its boundaries when accessed via the square brace operator
2. Create a class just like `std::vector`, that has a method to sort the elements
3. Create a class that is just like `std::string`, but also has a method to replace a character by a number
4. Create a complex number class by inheriting from `std::pair<float,float>`

Modification

- A class must be written to be modifiable this way
- This kind of modification makes function calls take longer
- Standard's classes aren't designed to be modified this way, other classes are meant to be composed of them and thus they should be as fast as possible
- Many libraries, however, are designed to have their classes extended this way, some can't even be made to work without being finished this way

Modification #2

```
class parent {
        virtual int a() {
                std::cout << "Not implemented" << std::endl;
                return 0;
        {
};
class child : public parent {
        virtual int a() {
                return 9;
        }
};
```

- The way to allow a method to be modified by children is to declare the method virtual
- An object's virtual method will be called instead of the parent class' virtual method even if it's used as a reference to a parent's class

Modification #3

```
class parent {
        virtual int a() {
                std::cout << "Not implemented" << std::endl;
                return 0;
        {
        int value() {
                return this->a();
        }
};
class child : public parent {
        virtual int a() {
                return 9;
        }
};
```

- A parent's method can call the child's virtual method only by using `this->method()`
- Careful, some people who used Java before will write `this->something` only to mark something is a member

Modification #4

```
class parent {
        virtual int a() = 0;
        int value() {
                return a();
        }
};
class child : public parent {
        virtual int a() {
                return 9;
        }
};
```

- The parent doesn't even have to define the virtual method, only to declare it
- In that case, its methods will call the child's virtual methods
- Such undefined methods are called *abstract methods*, classes that contain them are called *purely abstract classes* and they cannot be created, only their children that add these methods can
- A class whose all methods are abstract and has no attributes is called *interface*, but that word is often used to refer to any shared parent class

Modification #5: Remarks

- Calling virtual methods requires the program to check the object's type in runtime
- Checking a variable's type in runtime generally isn't a good approach in statically typed languages, but it's the only way to solve some issues efficiently, so virtual methods were created to do it in a controlled and error-resilient way
- This is a very powerful way to have classes share any amount of functionality without duplicating code
- If a class inherits from more parents that inherit from the same parent class, they normally each contain a copy of the parent class; this isn't always intended and can be avoided by making the attributes virtual as well, using the so called *virtual inheritance*, which will not be further described here
- Destructors must be virtual, calling a parent class' destructor only would not destroy the child's contents

Exercise

1. Create a group of classes that contain a number, either integer or floating point, but have a common `write` method that outputs the number as a string
2. Create a named vector class that works both as a string and as a vector, but `push_back()` uses the `push_back()` method of `std::vector`
3. Create a group of classes with one method that either appends at the back of a string, back of a file or writes into `std::cout` depending on the actual type, but all can be used by the same function
4. Create a class that appends string messages at the end of a block that can be retrieved and allows other methods to inherit from it to change its separator from newline to whatever else

Homework

- Write a class and its descendants that represent parts of a formula and have a common method to be computed, so that you could compose functions consisting of some numeric constants, variables and some basic operations of your choice
- `std::shared_ptr` will be very useful here
- You have two weeks to do it