T6 - C++

T-CPP-600

Bootstrap

Welcome to C++

1.0

This bootstrap quickly covers everything you need to step into the world of C++ in the forms of small tasks. You may already be familiar with some concepts, so feel free to skip them (if you're sure about them!) and take your times on the ones you don't know/don't master.

+ HELLO WORLD

1. Compiling a C++ program with a `main.cpp`.
2. Create an `msg` variable of type `std::string` with the value `"Hello world"`, then display it on stdout (C++ way).
3. Display the length of the string.

stream

+ CLASS, METHOD, PROPERTY, PUBLIC/PRIVATE, STATIC

1. Build a `Cat` class (`Cat.cpp` and `Cat.h`) that displays `"cat constructor"` in its **constructor** and `"cat destructor"` in the **destructor**.

Instanciate a `Cat` object from the `main`.

2. Add a public method `jump` that displays `"jump"` and call it from the previously created object in your `main`.
3. Add a `name` parameter in the constructor and store it in a **private** property of the class.
   Modify the `jump` method, so that it displays: `"[NAME]: jump"`.
4. Create a `Mouse` class with a `crock` method, which displays `"i'm dead"`.

Instanciate a `Mouse` object in your `main`. Then add an `eatAnimal` method in the `Cat` class, which takes a `Mouse` object as a parameter and call `crock()` on it.

5. Create an `AnimalFactory` class with a **public** `buy` method that creates and returns an instance of `Cat`. `AnimalFactory` will never be instantiated.

+ POINTERS VS REFERENCES

1. Do some research on your favorite search engine :wink: ;)

+ INHERITANCE, VIRTUAL, OVERRIDE

1. Create an abstract class `AAnimal`. The `Mouse` and `Cat` classes should now inherit from `AAnimal`.
2. Create a `Lion` class that inherits from `AAnimal`.
3. The `AAnimal` class receives `name` in its constructor.
4. From your main, call the jump method of the `Lion` and `Cat` objects, but without any code duplication between these 2 classes.
5. In the `AAnimal` class, add a property `isSavage` (boolean), accessible from the child classes, but not from the outside.
6. Add **getter** and **setter** for the `isSavage` and `name` properties.

In the `eatAnimal` method of the `Cat` class, display the name of the victim, before calling `crock()`.

7. Move the `Cat` method `eatAnimal` to the `AAnimal` class. It now takes in parameter an `AAnimal` instead of a `Mouse`.
8. Add a `talk()` method of type **pure virtual** in `AAnimal`, and implement it in each child classes. It should display, accordingly: `"graouhhh"`, `"meww"`, `"crss"`.
9. Overload the `eatAnimal` method in the mouse class and display: `"I don't eat animals"`.

+ EXCEPTIONS

1. If a `Mouse` eats a `Cat` or a `Lion`, it should **throw** an exception.
2. Display an error in the main, when a `Mouse` throws an exception.

+ POLYMORPHISM

1. In `AAnimal`, add 2 `run` methods (of the same name):

```
void run() /* Displays "ruuuun!" */
void run(int distance) /* displays "running {distance} kilometers." */
```

Should you replace `{distance}` by the value of `distance`?

+ TEMPLATING/GENERIC

1. Create a simple `Pair` class, with a **constructor** that takes 2 `int` as parameter.
2. Add `min` and `max` methods that return the smallest or largest number respectively.
3. Search for **"C++ template"** on your favorite search engine.
4. Evolve this class (constructor+min+max) into a template to compare `std::string`, `int`, `float`...

The `min` and `max` methods can compare `std::string` with the strcmp function (which takes char * as parameter).

+ STL

Search **"data structures C++"** on your favorite search engine: `Vector`, `List`, `Map`, `Stack`...