İTÜ Computer Engineering Department
BLG252E Object Oriented Programming
2nd Homework

Due Date: April 23, 2018 23.59 PM

In this assignment, you will design and implement a C++ program with object-oriented programming approach, which simulates mutation of species in danger of extinct.

Assume you are a biologist and very concerned about species living in River Trent where a nuclear plant located very close. There is a fish species called Grayling (*Thymallus thymallus*) that you are specifically interested in and want to analyze biological -nges in their life cycle in that area. After analyzing you get the scenario as follows: There are **3** different variations of this fish species exist in this area because of mutation all of them are different from base species in various rates. **Variation Grayling-1** has less difference from original form which has an increase of **30%** mutation probability(MPI) per year because of the nuclear plant close to their living area. They have **2** offspring per birth. Once a **Grayling-1** has **60%** mutation probability(MP) it becomes/turn into **Grayling-2** form which has an increase of **40%** mutation probability per year and **1** offspring per birth. Whenever **Grayling-2** has **80%** mutation probability it becomes **Grayling-3** form which has an increase of **50%** mutation probability per year. **Grayling-3** cannot reproduce, instead it dies in the phase of giving birth. Whenever a **Grayling** (for all variations) has **100%** of mutation probability it dies. A **Grayling-1** fish dies at age of **5** where a **Grayling-2** dies at age **4** and a **Grayling-3** fish dies at age **3**. After mutation mutation probability becomes **Zero(0)**. A new born fish is in the variation form of its parents.

You want to simulate a small population of Graylings with Object-Oriented programing approach using C++.

You are expected to design a base class (natural form) as an abstract class and other variations inherited from proper classes **considering inheritance and polymorphism aspects**.

A Grayling fish should have

Attributes

- a **short** for age years
- a **string** for name
- a **char** for gender (giving birth or not)
- a **bool** to indicate whether alive (or not)
- a **bool** to indicate whether mutant (or not)
- a **string** for name of Grayling form that the current fish **mutated to**
- a **double** for mutation probability increasing **(MPI)** rate per year (30%, 40% or 50% according to variations/forms)
- a **double** for mutation probability **(MP)** that current fish have (0 at the creation)
- a **double** for the mutation probability that the current fish **mutate at**

Methods (for all Grayling variations)

- a **default constructor** that creates a dead Grayling
- a **constructor** that you can pass values to so as to establish its gender, age, and name etc. ; it will default to alive and not mutant.
- a **copy constructor** that will create a copy of an existing object.
- a **print() virtual** function that will output to the screen the attributes of that species in a nice, easy to read format.
- a **givebirth() virtual** function that will make it reproduce.
- a **aging() virtual** function that will increase the age of fish and make changes accordingly

Your program's main() function should bet the same as follows: Three functions that accept all kind of fish forms.

```
void show(Grayling *g) {

        g->print();
}
void reproduce(Grayling *g) {
        g->givebirth();
}
void aging(Grayling *g) {
        g->aging();
}

int main() {

        Grayling1 G1_1('f',"G1_1"); //Create a Grayling1 with gender and name attributes
        Grayling1 G1_2(G1_1,"f","G1_2_CC");     // Copy Constructor *_CC
        Grayling2 G2_1('m',"G2_1");
        Grayling3 G3_1('f',"G3_1");

        show(&G1_1);
        show(&G1_2);
        show(&G2_1);
        show(&G3_1);

        aging(&G1_2);

        aging(&G1_1);
        aging(&G1_1);
        aging(&G1_1);
        reproduce(&G1_1);
        aging(&G1_1);

        aging(&G2_1);
        aging(&G2_1);
        aging(&G2_1);
        reproduce(&G2_1);
        aging(&G2_1);

        aging(&G3_1);
        aging(&G3_1);
        aging(&G3_1);
        reproduce(&G3_1);
        aging(&G3_1);

        getch();
        return 0;
    }
```

Each fish starts from age **0** and at each call for **aging()** method their age increase by **1** and some of other attributes changes accordingly. If needed the fish mutate (and **MP** becomes **0**) and **motateto** attribute becomes **"Grayling X"** name of the form that the fish mutated to. If the time has come it dies (**alive** becomes **0**). Mutate and die options checked in aging method. You can write additional private methods to handle this options. Assume a Grayling fish has born with mutation probability **0**. After it get aged its MP increase by **30% of 100% and it becomes MP=0+30=30 not ~~MP=0*30=0~~ , (MP=MP+30 NOT ~~MP=MP*1.3~~).** You can figure out the scenario from output window below.

The expected output from given **main()** function above that you are expected to provide by your program.

```
- Grayling 1 -> Age: 0 Nname: G1_1 G:f MPI: 30 MP 0 Mutate at: 60
- Grayling 1 -> Age: 0 Nname: G1_2_CC G:f MPI: 30 MP 0 Mutate at: 60
- Grayling 2 -> Age: 0 Nname: G2_1 G:m MPI: 40 MP 0 Mutate at: 80
- Grayling 3 -> Age: 0 Nname: G3_1 G:f MPI: 50 MP 0 Mutate at: 100

- AGING: name: G1_2_CC ->  Age: 1 MPI: 30 MP: 30

- AGING: name: G1_1 ->  Age: 1 MPI: 30 MP: 30
- AGING: name: G1_1 ->  Age: 2 MPI: 30 MP: 60
- Mutated TO: Grayling2
- AGING: name: G1_1 -> Grayling2 Age: 3 MPI: 40 MP: 40
- G1_1 gave birth to 1 offsprings!
- AGING: name: G1_1 -> Grayling2 Age: 4 MPI: 40 MP: 80
- G1_1 is dead because of AGING!

- AGING: name: G2_1 ->  Age: 1 MPI: 40 MP: 40
- AGING: name: G2_1 ->  Age: 2 MPI: 40 MP: 80
- Mutated TO: Grayling3
- AGING: name: G2_1 -> Grayling3 Age: 3 MPI: 50 MP: 50
- G2_1 is dead because of AGING!
- Trying to GIVE BIRTH  BUT  G2_1 is not alive
- Trying to AGING  BUT  G2_1 is not alive

- AGING: name: G3_1 ->  Age: 1 MPI: 50 MP: 50
- AGING: name: G3_1 ->  Age: 2 MPI: 50 MP: 100
- G3_1 is dead because of HIGH MUTATION RATE!
- Trying to AGING  BUT  G3_1 is not alive
- Trying to GIVE BIRTH  BUT  G3_1 is not alive and No ability to give birth!
- Trying to AGING  BUT  G3_1 is not alive
```

Make sure that there is no memory leak in your code.

- You may want to implement additional **private** methods.
- Be careful with the methods/attributes that are supposed to be constant, static, **private**/public.
- You can add getters/setters when they are necessary.
- Use comments wherever necessary in your code.
- Your program should compile and run on Linux environment using g++ (version 4.8.5 or later). You can test your program on ITU's Linux Server using SSH protocol. Include all necessary header files to your code. Do not use precompiled header files and Windows specific header files and functions.

Submission Notes

- You should also **write a report** to explain reasons why you need to use private/public/static/constant variables/methods in your rogram. Please make it possible to understand what you did in your program?
- After that, you should compress all files into an archive file named **"<your_student_number>.zip"**. **Do NOT** include any executable or project files in the archive file. You should only submit necessary files.
- Submissions are made through the Ninova system and have a strict deadline. Assignments submitted after the deadline **will NOT** be accepted. If you send your homework via e-mail, you **will NOT** get any points. Don't wait until the last minute. Upload whatever you have, you can always overwrite it afterwards.
- This is not a group assignment and getting involved in any kind of cheating is subject **to disciplinary actions.** Your homework **SHOULD NOT** include any copy-paste material (from the Internet or from someone else's paper/thesis/project). Check the "Academic honesty" section in the syllabus.
- For any questions about the assignment, contact **Cumali Türkmenoğlu** via e-mail (**turkmenogluc@itu.edu.tr**).