İskender Akyüz
150150150

BLG252E – Homework 2

Report

- The program can be compiled by using this command line:
  g++ main.cpp Grayling.h Grayling.cpp -o main
- In order to run the program, you can use this command line:
  ./main

```
class Grayling {
protected:
	short age;
	string name;
	char gender;
	bool is_alive;
	bool is_mutant;
	string mutated_to;
	double MPI;
	double MP;
	double mutate_at;
public:
	Grayling();   // Default Constructor
	Grayling(char, string);    // Constructor
	Grayling(const Grayling &, char, string);      // Copy Constructor
	virtual void print() const;
	virtual void givebirth();
	virtual void aging();
};
```

This is my base class. It has protected attributes and public methods. The reason why it has protected attributes is that they should be accessible from derived classes. It has two constructors. One of them is default constructor and it creates a dead grayling. The other one creates a living grayling which has a name and its gender is specified. It has also three virtual methods because for the derived classes, the correct function should be called for an object. The virtual functions are empty in the base class since we don't use them in the program. Also, the reason why print virtual function is constant is that there is no change in that function.

```
class Grayling1 : public Grayling {
public:
	Grayling1(char, string);   // Constructor
	Grayling1(const Grayling1 &, char, string);     // Copy Constructor
	void print() const;
	void givebirth();
	void aging();
};
```

This my first derived class. Grayling 2 and 3 are similar to this. The only difference is class names. In derived classes, I did not put any attributes since all of them are in the base class. The methods are again public here because they should be accessible from the main function. The constructors of derived classes set MPI, MP and mutate_at variables.

İskender Akyüz
150150150

```
void Grayling1::print() const {   // virtual function
       cout << "- Grayling 1 -> Age: " << age << " Name: " << name << " G: " << gender
<< " MPI: " << MPI << " MP: " << MP << " Mutate at: " << mutate_at << endl;
}
```

The print function prints the information about the grayling to the secreen. Like I said before, since we don't change anythin in it, it is an constant function.

```
void Grayling1::givebirth() {     // virtual function
       if (is_alive) {
              if (gender == 'm' || gender == 'M')
                     cout << "- Trying to GIVE BIRTH BUT " << name << " has no abiltiy
to give birth because of its gender!" << endl;
              else if (MPI == 50) {
                     cout << "- Trying to GIVE BIRTH BUT " << name << " has no ability
to give birth. It caused the death of it." << endl;
                     is_alive = false;
              }
              else {
                     cout << "- " << name << " gave birth to " << (MPI == 30 ? 2 : 1)
<< " offsprings!" << endl;
              }
       }
       else
              cout << "- Trying to GIVE BIRTH BUT " << name << " is not alive" << endl;
}
```

The givebirth function firstly checks whether or not the grayling is alive. If it is not alive, it cannot give a birth, and this message is printed to the console. If it is alive, I check its gender. İf it is a male, it cannot give a birth, and a message will be printed to the console. If it is an female, I am checking its MPI value in order to check if it is mutated to grayling3. If that is the case, since a grayling3 dies when it tries to give a birth, I kill the grayling. If it is not mutated to grayling3, then there is no problem. The grayling gives a birth according to its MPI value. The givebirth function for the Grayling 2 and 3 are similar to this. They work in a similar way with this one.

```
void Grayling1::aging() {  // virtual function
       if (is_alive) {
              age++;
              MP += MPI;
              cout << "- AGING: Name: " << name << " -> " << (is_mutant ? mutated_to :
"") << " Age: " << age << " MPI: " << MPI << " MP: " << MP << endl;
              if ((age == 5 && MPI == 30) || (age == 4 && MPI == 40) || (age == 3 &&
MPI == 50)) { // checking if it is dead
                     cout << "- " << name << " is dead because of AGING!" << endl;
                     is_alive = false;
              }
              if (is_alive && MP >= mutate_at) {       // checking mutation
                     if (MP < 80) {
                            mutated_to = "Grayling2";
                            MPI = 40;
                            MP = 0;
                            mutate_at = 80;
                     }
                     else {
                            mutated_to = "Grayling3";
                            MPI = 50;
                            MP = 0;
                            mutate_at = 100;
```

```
                    }
                    is_mutant = true;
                    cout << "- Mutated TO: " << mutated_to << endl;
            }
        }
        else
            cout << "- Trying to AGING BUT " << name << " is not alive" << endl;
}
```

Like givebirth function, aging function firstly checks if the grayling is alive or not. If it is not alive, it cannot age, and a message is printed to the console. If it is alive, its age is incremented, and its MP value increases by its MPI value. Then, I am checking if the aging caused its death. If it is still alive, I am checking if it is mutated to other forms.

The aging function for Grayling 2 and 3 work in a similar logic.