İskender Akyüz
150150150

BLG252E – Homework 3

Report

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

```
template <class Type>
class genericArray{
private:
	int size;
public:
	Type *elements;
	Type total;
public:
	genericArray(int);   // Constructor
	~genericArray();     // Destructor
	Type sum();
};
```

This is my template class. It has private and public attributes, and public methods. The reason why it has public attributes is that we should be able to access them from main function as the sample program in the homework pdf. Also, it has a constructor and destructor. What its constructor does is that it takes an integer number which indicates the size, and creates a dynamic array with this size. Since the constructor function takes dynamic space from memory, the destructor class deletes the memory to prevent memory leak. Additionally, it has a method named as sum. It returns the sum of the elements in the elements array.

The type of *elements array, total, and sum method is generic so that the class can be used by multiple types. This prevents us from writing the same class for each type.

```
class Money {
private:
	int lira;
	int kurus;
public:
	Money();      // Default Constructor
	Money(int, int);     // Constructor
	// Operator overloading
	bool operator < (int);
	bool operator > (int);
	Money & operator =(Money const &obj);
	Money & operator =(int);
	Money & operator += (Money const &obj);
	Money & operator -= (Money const &obj);
	friend std::ostream& operator<<(std::ostream&, const Money&);
};
```

This is my second and last class named as Money. It represents lira and kurus amounts. It has two private attributes which are lira and kurus. They are private since they should not be accessible from the outside of the class. Also, it has multiple public methods.

Two of them are constructors. The default constructor creates an object with 0 lira and 0 kurus. On the other hand, the second constructor takes two arguments, and creates an object with these arguments.

```
Money::Money(int in_lira, int in_kurus) {
	if (in_lira < 0 || in_kurus < 0)  // if lira or kurus is less than zero
		throw "The amount of money cannot be below zero!";	// throw an
exception
	else {
		lira = in_lira;
		kurus = in_kurus;
	}
}
```

In the constructor, the arguments are checked if their values are less than zero or not. Even if one of them is less than zero, the program throws an exception and terminates.

I also wrote multiple overloading operator methods.

```
bool Money::operator<(int in_lira) {	// < operator with an integer
	if (this->lira < in_lira)
		return true;
	else
		return false;
}

bool Money::operator>(int in_lira) {	// > operator with an integer
	if (this->lira > in_lira)
		return true;
	else if (this->lira == in_lira && this->kurus > 0)
		return true;
	else
		return false;
}
```

These two overloading methods compare a Money object and an integer, and return a Boolean.

```
Money & Money::operator=(Money const & obj) {	// assign operator
	this->lira = obj.lira;
	this->kurus = obj.kurus;
	return *this;
}
```

This assign operator assigns one Money object to another. It is a standard assign operator method for the object.

```
Money & Money::operator=(int in_lira) {  // assign operator. It assigns only lira.
Kurus is assigned to 0 by default.
	this->lira = in_lira;
	this->kurus = 0;
	return *this;
}
```

However, this one assigns an integer to a Money object. The integer is assigned to the lira attribute and kurus of the object is assigned to 0.

Since we have a sum method in genericArray class, I overloaded the summation and subtraction operations for the Money objects.

```
ostream & operator<<(ostream &out, const Money &obj) {
	out << obj.lira << " lira : " << obj.kurus << " kurus";
	return out;
}
```

Finally, in order to print the Money object to the screen, I overloaded the << operator.

All of these methods are public because they should be accessible from anywhere when they are needed.

In the main function, objects and other variables are created in the try block in order to catch any possible error in the program. After initialization of the objects, the sum of the generic arrays is calculated and assigned to the total variable. After each summation and assigning, the program checks if the total amount of money in the array is less or greater than 100. If it is greater than 100, a bonus is added to the total. If it is less than 100, a wage cut is applied. If it is equal to 100, then nothing happens to the total value.