CSE2050 - Programming in a Second Language
Assignment 5: Inheritance

November 15, 2017

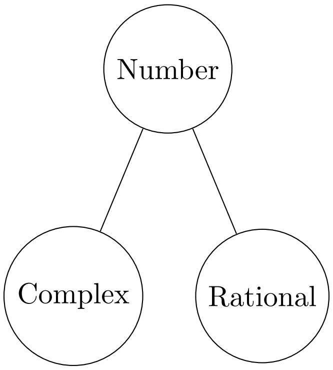
1 Submission

- Due Date: **November 28th, 11:59pm**
- Submit on **Canvas**.

2 Description

In this assignment, we will use inheritance to extend a base (abstract) class *Number* to two derived classes: *Complex*, and *Rational*.

- The base class is an abstract class (contains pure virtual methods, like *toString()*, as shown below)
- The derived classes overload some basic arithmetic operations (+ - * etc..)
- The derived classes override methods from the base class, like *print()*
- The derived classes will also provide their own unique methods.



3 Implementation details

3.1 Base class (Number)

This is the parent class, and it is an *abstract* class.

- It does not contain any data fields.

- It contains pure virtual methods to be overridden by its children:

  - **string toString()** returns the string representation of the number, for example, "2/3", or "(3 - 2i)" (without quotes).
  - **void print()** prints the string representation on the standard output device (cout).

- Your code might look like this:

```
class Number {
public:
        virtual string toString() =0;
        virtual void print() =0;

        Number() {
        ...
        }

        virtual ˜Number() {
        ...
        }
        ...
};

class Rational: public Number {
...
};

class Complex: public Number {
...
};

int main() {
...
}
```

3.2 Derived classes

The derived classes *Complex*, and *Rational* overload some of the arithmetic operators, discussed below.

3.2.1 class Complex

This class represents a *complex number*. It inherits from the base class *Number*. In addition to defining the methods mentioned above, it also includes the following fields:

- **a** represents the real part of the number.

- **b** represents the imaginary part of the number.

Use the representation $a + bi$ for both *toString()* and *print()* methods (where a is the real part). Using ***operator overloading***, implement methods for the following operators:

- Addition: operator+()
- Subtraction: operator-()
- Multiplication: operator*()
- Assignment: operator=()

3.2.2 class Fraction

This class represents a *fractional number.* It inherits from the base class *Number.* In addition to defining the methods mentioned above, it also includes the following fields:

- **a** represents the numerator.
- **b** represents the denominator.

Use the representation **a / b** for the toString() method, simplify all rationals wherever it is possible. Using *operator overloading*, implement methods for the following operators:

- Addition: operator+()
- Subtraction: operator-()
- Multiplication: operator*()
- Assignment: operator=()

3.3 main()

In your main method, define and initialize all the required objects to calculate the following expressions, and print the output to the standard output device (cout). Print the result of each expression on a separate line in the same order as below.

3.3.1 Complex numbers calculations

- $(3 - 4i) \times (5 + 2i)$
- $(4 + 3i) - (1 - 3i)$
- $(-2 + 7i) + (1 - 2i)$

3.3.2 Fraction numbers calculations

- $\frac{1}{3} \times 4$
- $\frac{2}{3} + \frac{2}{5}$
- $\frac{3}{4} - \frac{1}{4}$

4 Suggested extensions

You may try to extend your code to provide the following (no extra points will be rewarded):

- Use try-catch blocks in the code to catch potential issues (like division by zero, see below)
- Extend the code to do all arithmetic operations, including division.
- Extend the code to calculate more complex expressions, like:
  - ⋆ $\frac{1}{3} \times (4 + \frac{2}{5})$
  - ⋆ $(-2 + 7i) - (1 - 2i) \times (3 - 4i)$

5 Rubric

Criterion	Possible points	Excellent (max. points)	Satisfactory (partial points)	Unsatisfactory (no points)
Delivery	15 %	on time		late submission
Compiles	5 %	compiles on code01.fit.edu with no errors		does not compile
Runs	15 %	runs on code01.fit.edu with no run-time errors (missing files,..etc)		does not execute
Operator overloading	15%	implemented correctly	partially implemented	not implemented
Inheritance and overriding	15%	implemented correctly	partially implemented	not implemented
Test cases	35%	passes all test cases (correct output in the correct format)	passes some test cases	fails all test cases, or has an endless loop.