Ministry of Higher Education
Colleges of Applied Sciences
IT Department

Course: SFDV4001 – Object-Oriented Programming & User Interface
Lab: 3 – Class and Inheritance

1. Understanding a Class

Observe the code listing 1.1 that shows the class definition and implementation of fraction.

- Is Fraction a canonical class? Explain your answer.
- Which parts of the code is about the instantiation of an object?
- Which parts of the code do maintain the notion of encapsulation?
- Explain the reason for using **`const`** as part of the method signature in **`toDouble().`**
- Extend the program so that it can calculate the result of the multiplication of f1 and f2.

Code listing 1.1

```
#include <iostream>
#include <string>
using namespace std;
class Fraction {
  private:
    int numerator;
    int denominator;
  public:
    Fraction(int n, int d) : numerator(n), denominator(d) {}
    Fraction(const Fraction& other) :
       numerator(other.numerator), denominator(other.denominator) {}
    Fraction& operator=(const Fraction& other) {
            numerator = other.numerator;
            denominator = other.denominator;
            return *this;
    }
    double toDouble() const {return 1.0 * numerator / denominator;}
};
int main() {
    Fraction f1(1,5), f2(1,2);
    cout << "\nf1: " << f1.toDouble() << endl;
    cout << "\nf2: " << f2.toDouble() << endl;
    cout << "\nf1*f2=";
    return 0;
}
```

2. Virtual Function

Observe the code listing 2.1, and answer the following two questions.

- What is the benefit of using a virtual function?
- What will happen if the print method at Base class is not a virtual function?

Code listing 2.1

```
#include <iostream>
using namespace std;

class Base {
public:
        virtual void print() { cout << "Base class" << endl;}
};

class Derived : public Base {
public:
        virtual void print()  {cout << "Derived class" << endl;}
};

int main() {
  Base b;
  Derived d;
  b.print();
  d.print();

  Base *b1 = new Derived;
  b1->print();
}
```

3. Abstract Function

Observe the code listing 3.1, and

- Solve and explain the compile error.
- What is the benefit of using an abstract method?

Code listing 3.1

```
#include <iostream>
using namespace std;

class Base {
public:
    virtual void print() = 0;
};

class Derived : public Base {
public:
        virtual void print()  {cout << "Derived class" << endl;}
};

int main() {
  Base b;

  Derived d;
  d.print();

  Base *b1 = new Derived;
  b1->print();
}
```

4. Parent's Constructor

Observe the code listing 4.1, and explain the difference between `D d1(2)` and `D d2`?

Code listing 4.1

```
#include <iostream>
using namespace std;

class C {
public:
        C() {cout << "C default\n";}
        C(int num) {cout << "C(" << num << ")\n";}
};

class D : public C {
public:
        D() {cout << "D default\n";}
        D(int num): C(num) {cout << "D(" << num << ")\n";}
};

int main(int argc, char *argv[])
{
   cout << "D d1(2);---------------------------\n";
   D d1(2);

   cout << "D d2;------------------------------\n";
   D d2;
}
```