Name:

Singleton

CSC-490, Winter 2020

1. (2pts) What is the intent of the Singleton?

2. Consider the following implementation of the Singleton:

Singleton.h

```
class Singleton {
private:
  static Singleton* _instance;
  int _data;
protected:
  Singleton();
public:
  static Singleton* getInstance();
  int getData();
  void setData(int);
};
```

Singleton.cpp

```
#include "Singleton.h"

Singleton::Singleton() {}

Singleton* Singleton::_instance = nullptr;

Singleton* Singleton::getInstance() {
  if(_instance == nullptr) {
    _instance = new Singleton();
  }
  return _instance;
}
//data getter and setter below...
```

(a) (3pts) Draw the class diagram for the program.

(b) (3pts) Assume we are working within a cpp file which has `#include "Singleton.h"`. Which of the following lines of code will produce a compile time error? Why?

```
Singleton s1;

Singleton* s2 = new Singleton();

Singleton* s3 = Singleton::getInstance();
```

(c) (2pts) Why is the data member `_instance` declared static?

last updated: January 15, 2020

(d) (4pts) How many times can the constructor of the Singleton class be executed in a program? Why?

(e) (4pts) What will the following program print to the terminal?

Main.cpp

```
#include "Singleton.h"
#include <iostream>

int main() {
  Singleton* s = Singleton::getInstance();
  s->setData(7);
  std::cout << s->getData() << "\n";
  Singleton* t = Singleton::getInstance();
  t->setData(12);
  std::cout << t->getData() << "\n";
  std::cout << s->getData() << "\n";
}
```

3. Is the Singleton an *Anti-pattern*?

   (a) (2pts) What is an anti-pattern?

   (b) (2pts) Why do many people consider the Singleton to be an anti-pattern?

   (c) (6pts) List as many reasons as you can think of (or find in any resource) as to why global states/variables are evil. Include something about testing software.