

COMP 1771 – Fall 2008

HW#3

Instructor:

Prof. Matt Rutherford – mjr@cs.du.edu

Due by:

10:00am on October 14, 2008

Homework Policies

As stated in the class syllabus:

Homework assignments and labs may be discussed in a general fashion with other students. You should not discuss specific solutions or code. You should NEVER copy assignments that have been written by another student or allow another student to copy your assignments. If any of your work includes ideas or quotes from a book, paper, or web site, you should clearly cite the original source.

Your code should be emailed as attachments to me at mjr@cs.du.edu before 10am on October 14, 2008. Please use the subject: "COMP 1771 Homework #3".

Part 1: Initial Project Implementation

The first part of this homework is to implement the first 5 or 6 classes for your lab project. I would like you to pick the lowest level classes that will implement the core logic of your game. For each class that you implement, say `MyClass`, I want you to create a class that tests the implementation, say `MyClassTest`. The test classes will have a main function and instantiate and manipulate the class being tested.

For each class that you implement please use this basic process:

1. Decide on the data members and constructors that make sense for the class
2. Add accessor and mutator methods for whichever data members need them
3. Decide on the other public methods that might be necessary

Please keep these specific requirements in mind when doing your implementation:

- Non-final data members must be private
- You must implement the parts of the classes that perform the logic, do not turn in a bunch of classes with just setters and getters.
- It is premature to implement classes that are involved with the graphical rendering of your games; if you want to do this now, that is fine, but I don't want any of the classes you turn in to be involved with the graphical output.
- If you don't think you will have 5 or 6 non-grapical classes, please talk with me.

Part II: Implement DateTime Class

For this part, you will implement a class called `DateTime` that is capable of storing the year, month, day, hour, minute, second and performing operations on the time to produce new `DateTime` objects. The most important aspect of `DateTime`'s implementation is that it never stores an invalid date/time. This implies that it contains enough logic to deal with the different number of days in months, leap years, etc.

The class is immutable (there are no mutator methods), and it should have the following public methods:

```
public class DateTime
{
    public DateTime(int year, int month, int day) { ... }

    public DateTime(int year, int month, int day, int hour, int minute) { ... }

    public DateTime(int year, int month, int day, int hour, int minute, int second) { ... }

    public int getYear() { ... }
    public int getMonth() { ... }
    public int getDay() { ... }
    public int getHour() { ... }
    public int getMinute() { ... }
    public int getSecond() { ... }

    public DateTime addYear(int n) { ... }
    public DateTime addMonth(int n) { ... }
    public DateTime addDay(int n) { ... }
    public DateTime addHour(int n) { ... }
    public DateTime addMinute(int n) { ... }
    public DateTime addSecond(int n) { ... }

    public int compareTo(Object that);

    public boolean equals(Object that);

    public String toString();
}
```

The constructors and accessor methods should be obvious.

The `add*()` methods all return a new object that represents the time of the original object plus the specified number of units. NOTE: the add methods should handle negative values of `n` properly.

The `compareTo()` method will return a negative integer if `this` object is before `that` object, zero if the two objects represent the same exact date-time value, and a positive integer if `this` is after `that`.

The `equals()` method should return `true` if the two objects represent the same date-time value, `false` otherwise.

The `toString()` method should return a string of the form:

2008-10-07 20:10:12

You should implement a test class `DateTimeTest` that has a main function and that tests your code.

Extra Credit (15 possible points)

For extra credit, you should implement the `DateTime` class without using separate data members for each of the components. Your class should have only a single data member.

Grading

Module	Percentage
Part 1: Project Classes	60%
Part 2: DateTime implementation	40%