

Course: CSC220.01

Student: Mark Kim

Instructor: Duc Ta

Assignment Number: 00

Due Date & Time: 06-18-2019 at 11:55 PM

Assignment 00

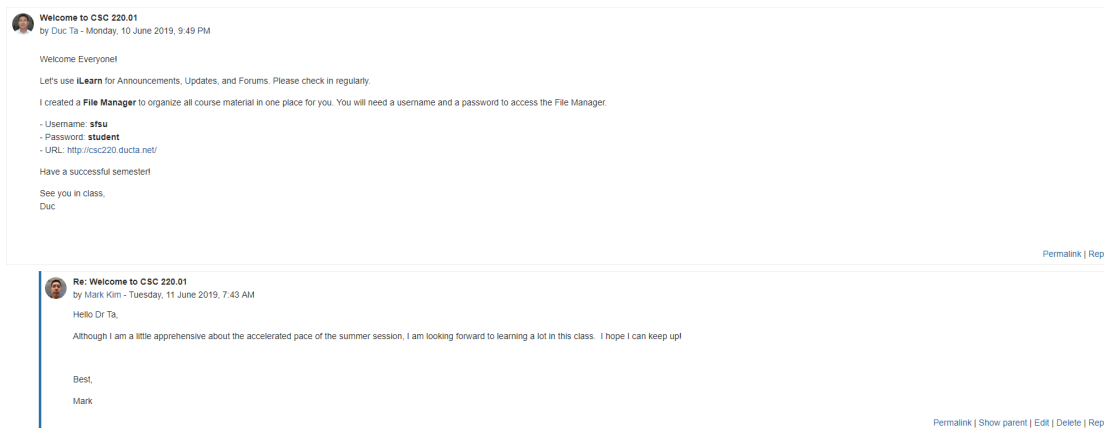
PART A

1.

ASMT 0 Discussions

Welcome to CSC 220.01

Display replies in nested form



The screenshot shows a discussion forum interface. At the top, there's a header for 'ASMT 0 Discussions' and 'Welcome to CSC 220.01'. Below this, a post by 'Duc Ta' dated 'Monday, 10 June 2019, 9:49 PM' is visible. The post content includes a welcome message, instructions to use iLearn, and details about a File Manager. A reply by 'Mark Kim' dated 'Tuesday, 11 June 2019, 7:43 AM' is shown below, expressing excitement for the summer session. The interface includes a 'Permalink | Reply' link at the bottom right of the reply.

Welcome to CSC 220.01
by Duc Ta - Monday, 10 June 2019, 9:49 PM

Welcome Everyone!

Let's use **iLearn** for Announcements, Updates, and Forums. Please check in regularly.

I created a **File Manager** to organize all course material in one place for you. You will need a username and a password to access the File Manager.

- Username: **sfsu**
- Password: **student**
- URL: <http://csc220.ducta.net/>

Have a successful semester!

See you in class,
Duc

[Permalink](#) | [Reply](#)

Re: Welcome to CSC 220.01
by Mark Kim - Tuesday, 11 June 2019, 7:43 AM

Hello Dr Ta,

Although I am a little apprehensive about the accelerated pace of the summer session, I am looking forward to learning a lot in this class. I hope I can keep up!

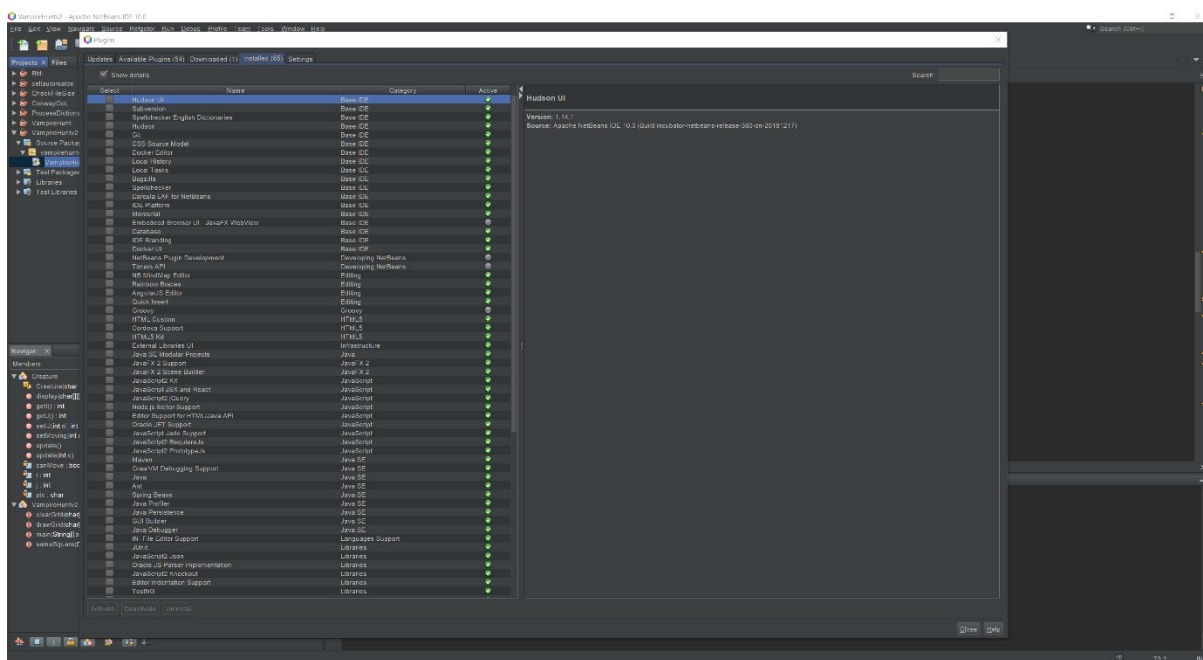
Best,
Mark

[Permalink](#) | [Show parent](#) | [Edit](#) | [Delete](#) | [Reply](#)

2. Yes, this is to confirm that I will use my SFSU email address when contacting the lecturer or graders. I will start my email subject with “CSC 220.01”, and if I do not get an answer within 24 hours, I will check if I sent my email properly and resend appropriately.
3. I have carefully read the “Guidelines for ALL Assignments” and “Assignment Report Template.”

so I would improve this ability. Some of the ways I would accomplish this is to allow the human to choose a direction from only valid moves available and introduce diagonal movement. It would be interesting to make movement a little more dynamic; instead of entirely random movement, the human could choose from directions that would put the most distance between him/her and the vampire. The area is also a bit limited, so I would likely increase the size of the field. Speaking of the field of play, I think it would be fun to introduce obstacles. I believe the generation of obstacles would be the hardest feature to implement since it would change the behavior of the game the most (i.e. provide cover from bow-fire, changes movement behavior, arrays must be generated procedurally to allow for movement, location randomization needs to be modified to account for obstacles, etc). It would be nice to have the game refresh the game board instead of regenerating line by line, but I haven't learned how to do that yet. Otherwise, I believe the rest of my modifications are possible with whatever knowledge I have now.

PART C



1. Plugins

- a. I installed 2 plugins that just changes the appearance of Netbeans (Darcula LAF and Rainbow Braces). One changes the color of the UI to black and dark gray. The second adds colors to brackets, braces, and parentheses so that I can track their depth better. This allows me to locate nested brackets, braces, and parentheses faster and with greater accuracy.
- b. The other plugin I installed is NB MindMap Editor. This is a mindmap creation tool to help track projects, their dependencies, and workflow. These mindmaps can be exported and/or printed to assist in describing a project in a presentation (or during collaboration) or to just help organize your own thoughts. With larger projects, one can easily lose track of the many moving parts, so a visual representation is often extremely helpful. When working with others, it will allow someone else to be able to understand an overview of a project quickly without delving into the code.

2. Shortcuts

- a. Ctrl-F/H: This is a universal shortcut shared by most applications that I use often to find or replace text.
- b. Alt-Shift Left/Right/Up/Down: This shortcut allows you to shift lines of code up/down/left/right.
- c. Ctrl-PgUp/PdDown: Another navigation shortcut, this allows the user to navigate through tabs.

PART D – Class Design Guidelines – Encapsulation

Encapsulation is the concept of hiding information from the rest of the program which is a fundamental of object-oriented programming. By hiding the implementation of a class, its

contents are protected from access by other parts of the program. Access to the data is now restricted and is only made accessible to the rest of the program with getters and setters (if access is necessary). As can be seen below, access to information and methods can be restricted and access granted by getters and setters:

```
class Car {  
    private double ispeed = 0; //access only within class  
    private double fspeed = 0; //access only within class  
    public void setispeed(double s) { //setter with outside access  
        ispeed = s;  
        setfspeed(); //calls method within class  
    }  
    private void setfspeed() { //access only within class  
        fspeed = ispeed * 2;  
    }  
    public double getfspeed() { // getter with outside access  
        return fspeed;  
    }  
}
```

In the example above, access to the method `setfspeed()` is restricted to the class that contains it which allows the method `setispeed(double s)` to call it since they are both members of the same class.

There are several purposes for using encapsulation: security, flexibility, reusability, and ease of maintenance. By encapsulation, the contents of a class can be hidden from a user. This

prevents unauthorized access to the data and functions within the class. Furthermore, by restricting access to data via getters and setters, we can make data read-only or write-only as our needs dictate which improves the flexibility of the program. Separating data and methods within a class improves modularity, allowing the class to be reused throughout a program; this modularity allows one to modify the class as needs require without requiring changes to the rest of the code. Lastly, since the code is wrapped up in its own unit, the code can be easily tested and maintained without affecting the rest of the program.