Assignment Report for Assignment 05

Course and Section	CSC215.28
Assignment Name	Assignment 05
Due Date and Time	11-01-2024 @ 11:55 PM
First Name and Last Name	Miguel Antonio Logarta
SFSU Email Account	92306283@sfsu.edu
First Name and Last Name of Teammate	N/A
SFSU Email Account of Teammate	N/A

PART A

Question Description and Analysis:

This part of the assignment asks me to write about Y. Daniel Liang's Class Design Guidelines. I need to write half a page explaining the guideline, write code, and write half a page of how I applied the guideline.

Answer:

In Y. Daniel Liang's Class Design Guidelines, his first guideline talks about **Cohesion**. There are two points:

1. A class should describe a single entity, and all the class operations should logically fit together to support a coherent purpose.
2. A single entity with many responsibilities can be broken into several classes to separate the responsibilities.

From my understanding of the first point, a class is supposed to have a clear purpose. If a class has too many unrelated responsibilities, it can confuse the next person who will use the class. The data and methods of the class should have a strong relationship and purpose with each other.

It should all fit together in a way that makes sense intuitively. For example, a Horse class may run(), beMounted(), and neigh(), but it doesn't make sense for them to moo(). Another example would be a printer that can also wash your dishes for some reason. The printer has responsibilities that are not related to its main purpose. When the purpose of a class is not coherent, it has low cohesion.

The second point accompanies the first point. You should split a class into multiple subclasses to separate responsibilities and concerns. Splitting your code into several classes makes it more modular, increasing readability and cohesion. Code with high cohesion is easier to maintain since the logic is nicely separated into places where it makes sense.

Here is a code example of Tetris:

```
public class Tetris {
    int score;
   int timeLeft;
   int fallSpeed;
   String[] tetrisBag;

   Tetris() {
      score = 0;
      timeLeft= 1000;
      fallSpeed = 1;
      tetrisBag = new String[] {“S”, “Z”, “L”, “J”, “T”, “I”};
      …
   }

   void handleKeyPresses() {
     …
  }

   void displayGrid() {
      …
   }

   void moveBlocks() {
      …
```

```
    }

    void handleCollision() {
        …
    }

    void getNextBlock() {
        …
    }

    void shuffleTetrisBag() {
        …
    }
}
```

Right now the Tetris class has too many responsibilities. It is handling the behavior of the key presses, handling the rendering behavior of the game, controlling the order of the pieces as they appear in the Tetris bag, and so much more. It would be better if the code is split into multiple classes.

```
public class TetrisGame {
    GameState state;
    GUI gui;
    TetrisBag bag;

    TetrisGame() {
        state = new GameState();
        gui = new GUI();
        TetrisBag = new TetrisBag();
    }

    run() {
        while (true) {
            // Input handling
            handleKeyPresses();

            // Update state
            handleCollisions();

            if (blockPlaced) {
```

```
            piece = TetrisBag.getNextPiece();
        }

        // Render game
        gui.displayGrid();
    }
  }
}

public class GameState {
  int score;
  int timeLeft;
  int fallSpeed;

  GameState() {
    ...
  }

  public int getScore() { ... }
  public int setScore() { ... }
  public int getFallSpeed() { ... }
  public int setFallSpeed() { ... }
}

public class GUI {
  displayGrid() {
    ...
  }
}

public class TetrisBag {
  private String[] tetrisBag;

  public String[] shuffleBag() {
    tetrisBag = new String[] {“S”, “Z”, “L”, “J”, “T”, “I”};
    // Shuffle a new deck
    ...
  }

  public String getNextPiece() {
    if (tetrisBag.length == 0) {
      this.tetrisBag = shuffleBag();
    }

    // Remove newest block from the bag, then return it
    ...
```

```
    }
}
```

By using the cohesion guideline, we can group related logic into separate classes to make the code easier to maintain. For example, anything related to rendering is put into the GUI class. The GUI now has the displayGrid() method. We also moved the game's state into its own separate class called GameState. We can use this class to manipulate the score, falling speed, and time remaining. We also moved the piece shuffling logic into a TetrisBag class. Now whenever we need to shuffle our Tetris pieces or get a new block, we can use this class to retrieve it. By splitting the code into classes, it makes our run() method shorter. The run method handles the key presses, does some updates to the game's state (which is not shown), and then calls GUI to render the screen.

PART B

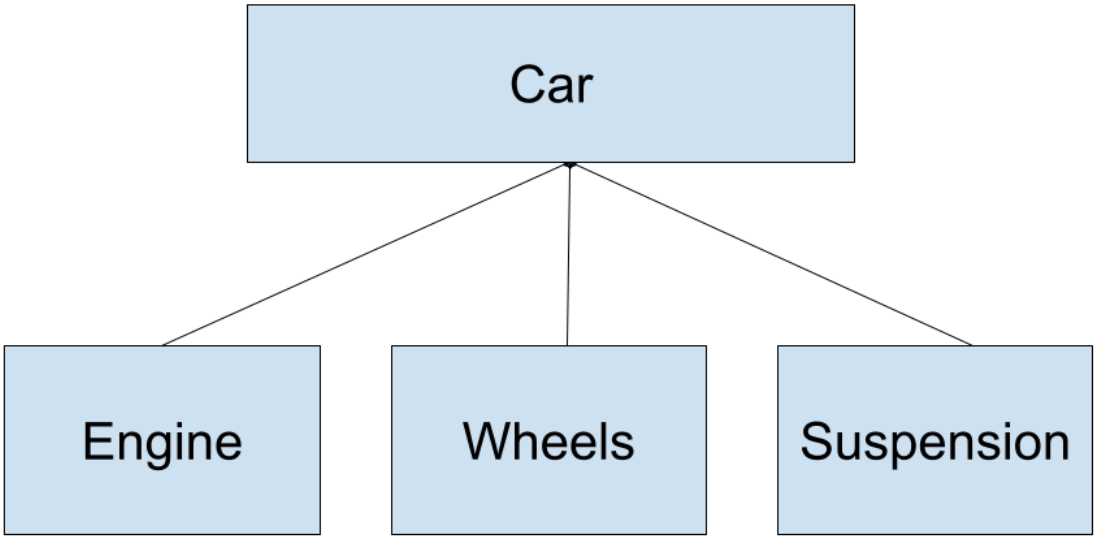
Question Description and Analysis:

This part of the assignment asks me to create some code that shows a HAS-A relationship.

Answer:

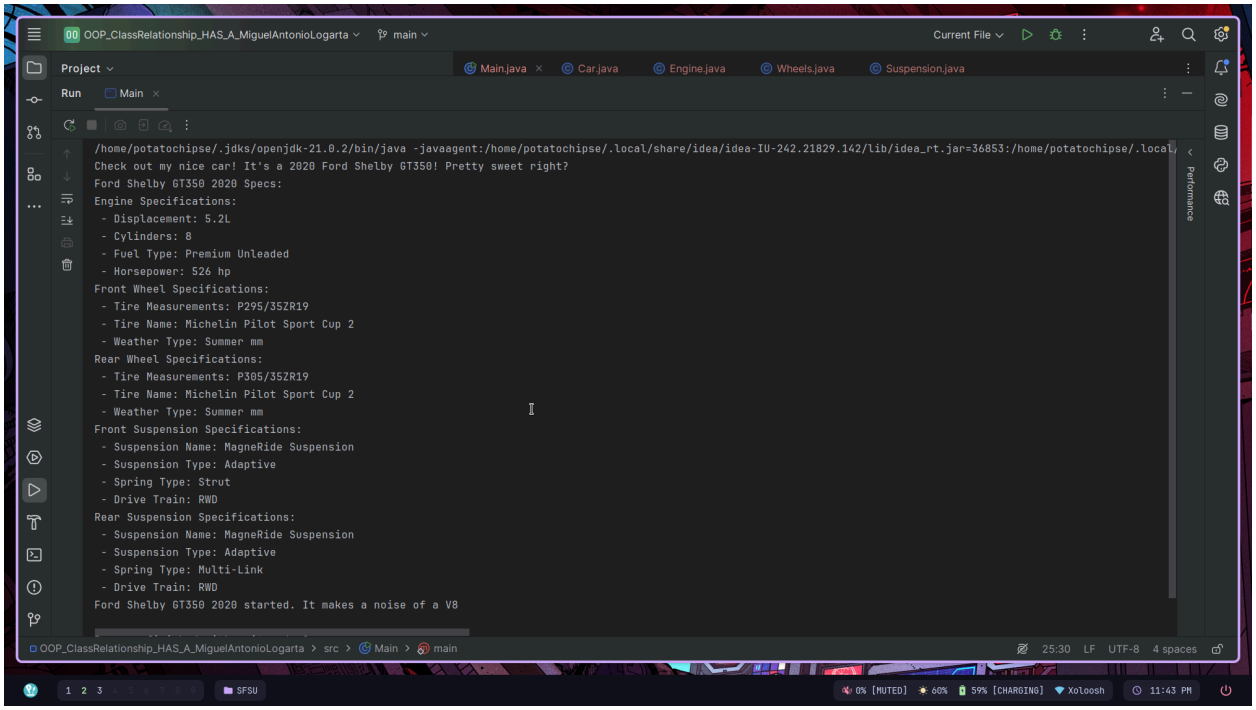
The question is asking me to create code that demonstrates my knowledge of HAS-A relationships. I have to create 1 driver class and 4 user-defined data types that will be used by this driver class. They should all have data fields inside of them. The driver class will be composed of these user-defined data types. I should also use them inside the main method. For my program, I also have to generate a UML diagram that shows the relationships between the classes in my program. For this program, I will be demonstrating a HAS-A relationship using a Car class. A Car is composed of an Engine, Wheels, and Suspension. For my class, the Car has 1

engine, a set of front wheels, a set of back wheels, a set of front suspension, and a set of rear suspension.



My program compiles and everything works as planned. In the future, I could create a new Factory class that generates different types of cars depending on what the client needs. For example, right now the program can only create a 2020 Ford Shelby GT350. In the future, I can make a class that can create any car such as a Porsche 911 or Mazda Miata for example.

Screenshots of Outputs and Explanation:



PART C

Question Description and Analysis:

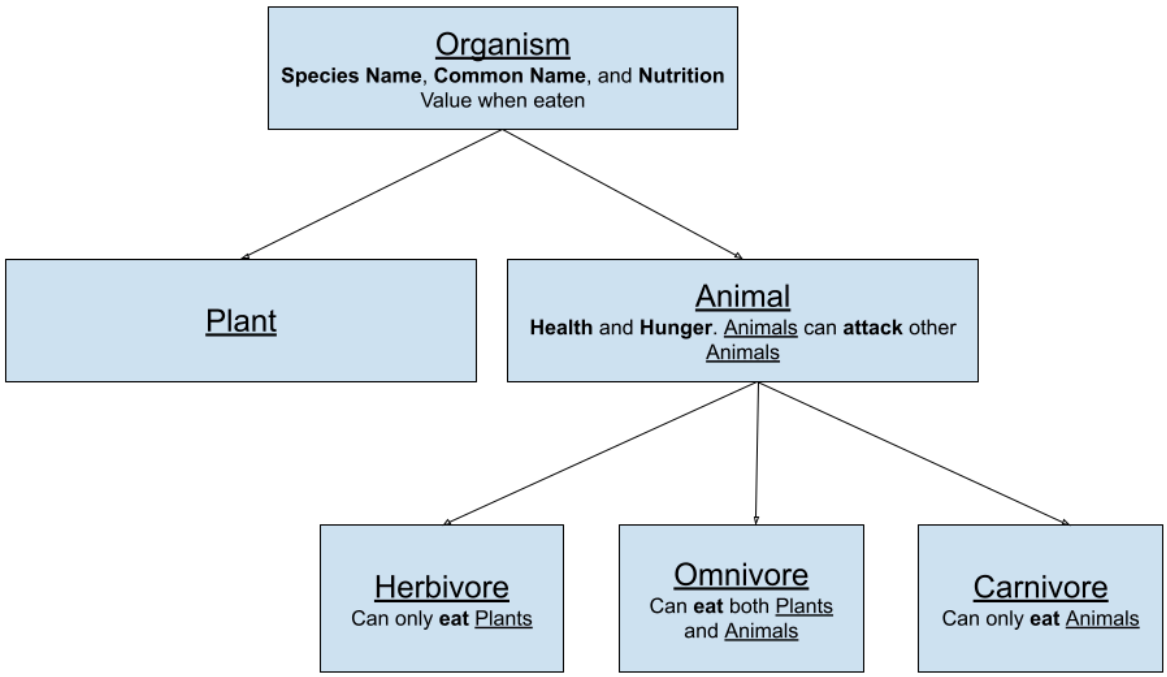
This part of the assignment asks me to create some code that shows an IS-A relationship.

Answer:

The question is asking me to create code that demonstrates my knowledge of IS-A relationships. I have to create 1 driver class and 3 user-defined data types are related to this driver class. The inheritance level has to be 3 layers deep. They should all have data fields inside of them. The driver class will be composed of these user-defined data types. I should also use them inside the

main method. For my program, I also have to generate a UML diagram that shows the relationships between the classes in my program. For this program, I will create an Animal class. Under that Animal class, I will add an additional 3 classes that inherit from Animal. The classes are Herbivore, Omnivore, and Carnivore. I will create 3 instances from each class such as deer, lions, and humans.

After creating my program, here is my final simplified UML diagram:



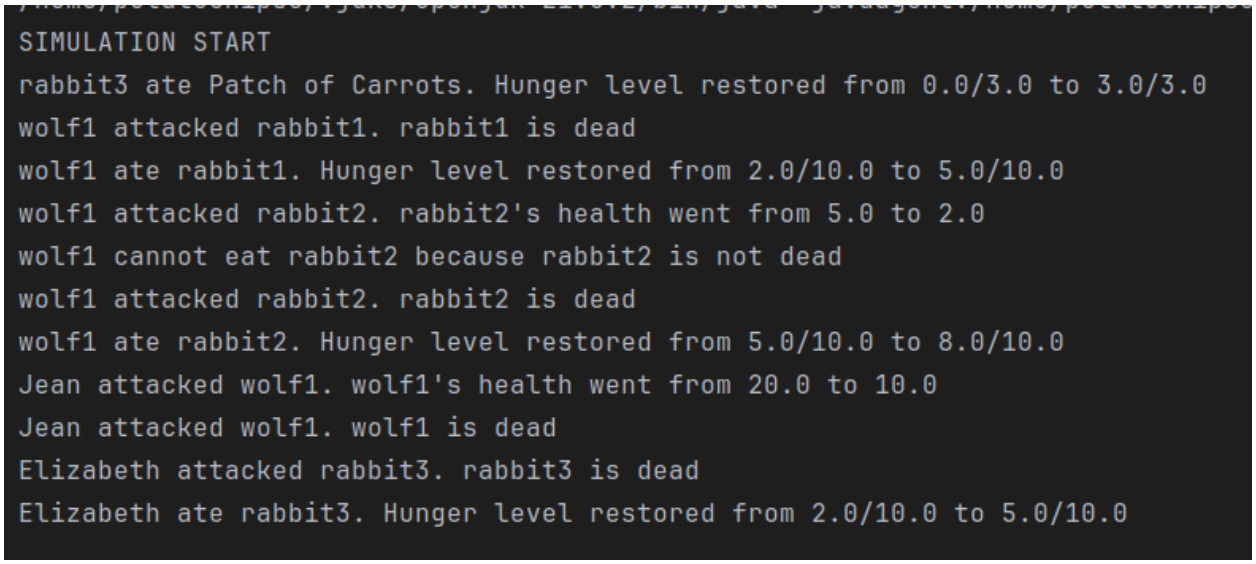
Originally my plan was to just have an Animal class with Herbivore, Omnivore, and Carnivore subclasses. However, I realized that Herbivores needed an eat() method that takes in a plant instead of another Animal. I decided to add an Organism class and a Plant class since it made

sense to put them in intuitively. Organisms all have names and a nutrition level since all Organisms can be consumed one way or another. Animals can attack each other, either for defense or for hunting. Once an animal is dead, they can be consumed. Plants do not need to be attacked to be consumed.

What's great about this inheritance tree is that I can add more species with different behaviors. For example, I can now make wolves, humans, elephants, rabbits, deer, and so much more.

Screenshots of Outputs and Explanation:

These screenshots show what I accomplished…



```
Here is the status of all the organisms:
Status of rabbit1:
Species: Oryctolagus cuniculus (Wild European Rabbit)
Health: 0.0/5.0 (Dead)
Hunger: 2.0/3.0 (Slightly Hungry)

Status of rabbit2:
Species: Oryctolagus cuniculus (Wild European Rabbit)
Health: 0.0/5.0 (Dead)
Hunger: 2.0/3.0 (Slightly Hungry)

Status of rabbit3:
Species: Oryctolagus cuniculus (Wild European Rabbit)
Health: 0.0/5.0 (Dead)
Hunger: 3.0/3.0 (Full)

Status of wolf1:
Species: Canis lupus (Grey Wolf)
Health: 0.0/20.0 (Dead)
Hunger: 8.0/10.0 (Full)

Status of Jean:
Species: Homo Sapiens (Human)
Health: 20.0/20.0 (Healthy)
Hunger: 2.0/10.0 (Hungry)

Status of Elizabeth:
Species: Homo Sapiens (Human)
Health: 20.0/20.0 (Healthy)
Hunger: 5.0/10.0 (Slightly Hungry)
```