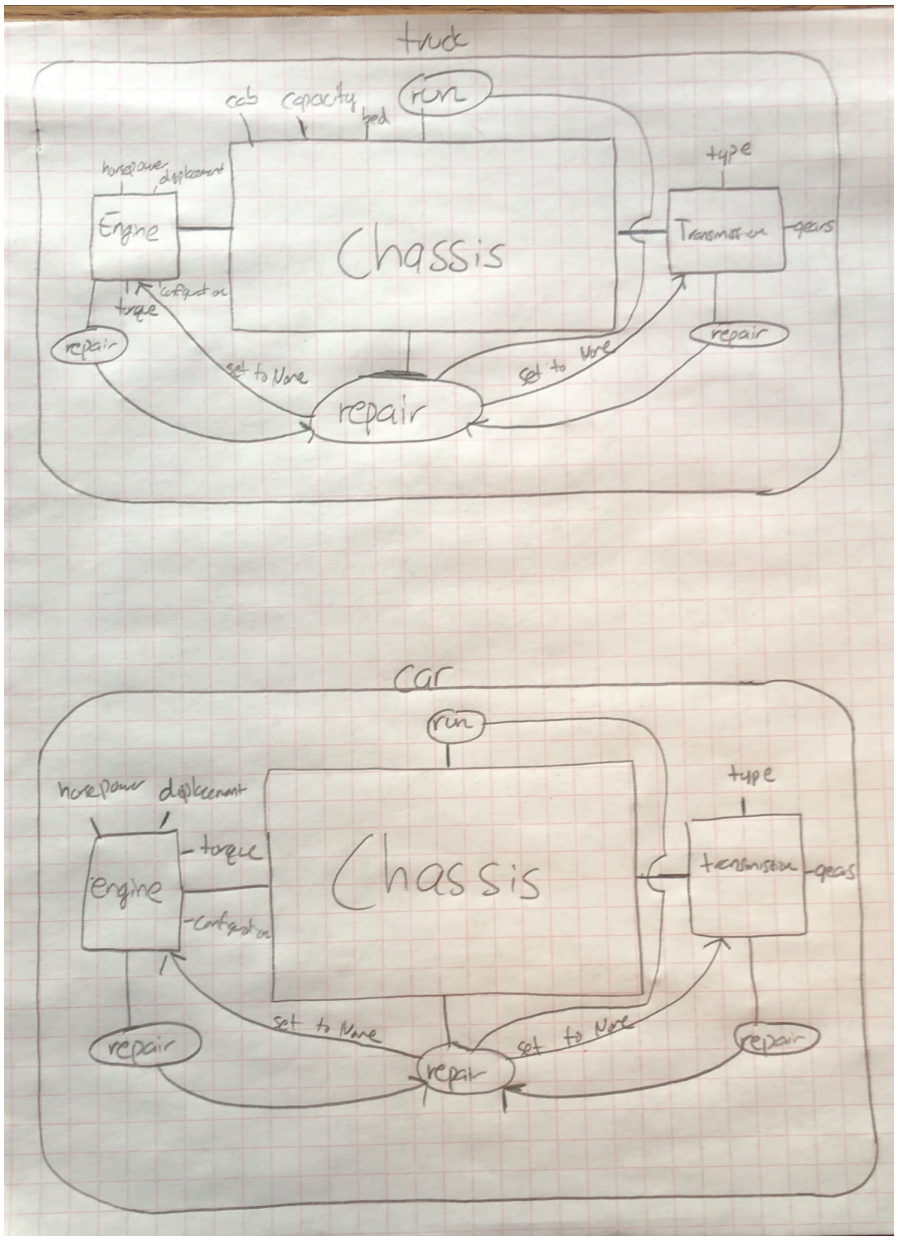
1) A generic object that can cause an exception within its state is an engine or transmission breaking within a car. To model this using class inheritance, I created four classes. Two classes were for the engine and transmission, and another class was called the Chassis. This would represent the chassis of the vehicle. The chassis contained both the engine and the transmission classes as attributes. Lastly, there was a car and truck class. These were child classes of the chassis. To cause a planned exception to occur, there was a repair method for the transmission and engine. It would randomly choose whether or not the transmission or engine would need to be repaired. To see if either needed to be repaired, a run method of the chassis needed to be executed. This would in turn see if any of the two parts were broken. I defined broken as the two objects becoming set to None.
2)



3) Some kinds of tests that I could use for this program are input validity and proper communication between all the attributes and methods of the program. To test the input data type to be correct, I can create some truck and car instances with incorrect data types and see if an exception occurs. To test to see if all the methods and attributes are working correctly, I can print out the name of the instance of the truck/car class. This is because the string method for each will display all the important attribute information required. The act of printing them will call upon their values, and if issues are going to occur, it is going to be there. Attached are screenshots of the test of the program.

```
PS C:\Users\kyleg\OneDrive\Desktop\CS162> pytest -s test_project_7.py
============================================================================
platform win32 -- Python 3.10.6, pytest-8.2.0, pluggy-1.5.0
rootdir: C:\Users\kyleg\OneDrive\Desktop\CS162
plugins: anyio-3.6.1
collected 7 items

test_project_7.py Your vehicle needs to be repaired
Engine needs to be fixed
Press ENTER to fix engine
Engine fixed
Your vehicle has been started correctly

Year: 1994
Brand: Toyota
Capacity: 5
Model: Car

Engine Specifications:
Engine Type: petrol
Engine Configuration: I4
Engine Displacement: 2.4
Engine Horsepower: 105.0
Engine Torque: 95.0

Transmission Specifications:
Transmission Type: manual
Number of Gears: 5



Year: 1994
Brand: Toyota
Capacity: 5
Model: Car
```

```
Engine Configuration: I4
Engine Displacement: 2.4
Engine Horsepower: 105.0
Engine Torque: 95.0

Transmission Specifications:
Transmission Type: manual
Number of Gears: 5


.Your vehicle has been started correctly

Year: 2005
Brand: Chevy
Capacity: 6
Model: Truck
Cab size: crew

Engine Specifications:
Engine Type: diesel
Engine Configuration: V8
Engine Displacement: 6.7
Engine Horsepower: 600.0
Engine Torque: 700.0

Transmission Specifications:
Transmission Type: automatic
Number of Gears: 8
```

```
.FFFFF

================================================================================ FAILURES ================================================================================
_____________________________________________________________________ test_car_wrong_param_creation _____________________________________________________________________

    def test_car_wrong_param_creation():
        """Will fail due to 'five' not being an integer"""
>       my_car = Car(capacity='five', mass=2000,
                     drag_coef=0.3, year=1994, brand='Toyota',
                     engine=Engine(type='petrol', config='I4', displ=2.4, hprs=105, torq=95),
                     tranny=Tranny(type='manual', gears=5))

test_project_7.py:30:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

self = <project_7.Car object at 0x0000024418F99F00>, capacity = 'five', mass = 2000, drag_coef = 0.3, year = 1994, brand = 'Toyota'
engine = <project_7.Engine object at 0x0000024418F99960>, tranny = <project_7.Tranny object at 0x0000024418F997B0>

    def __init__(self, capacity: int, mass,
                 drag_coef, year, brand, engine, tranny):
        super().__init__(mass, drag_coef, year, brand, engine, tranny)
>       self.cap = abs(int(capacity))
E       ValueError: invalid literal for int() with base 10: 'five'

project_7.py:241: ValueError
_____________________________________________________________________ test_truck_wrong_param_1_creation _____________________________________________________________________

    def test_truck_wrong_param_1_creation():
        """Will fail due to 'shorts' not being a valid bed type"""
>       my_truck = Truck(bed_type='shorts', cab_type='crew', mass=4000,
                     drag_coef=0.3, year=2005, brand='Chevy',
                     engine=Engine(type='diesel', config='V8', displ=6.7, hprs=600, torq=700),
                     tranny=Tranny(type='automatic', gears=8))
```

```
test_project_7.py:39:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

self = <project_7.Truck object at 0x0000024419036D10>, bed_type = 'shorts', cab_type = 'crew', mass = 4000, drag_coef = 0.3, year = 2005, brand = 'Chevy'
engine = <project_7.Engine object at 0x0000024419036DD0>, tranny = <project_7.Tranny object at 0x0000024419036E00>

    def __init__(self, bed_type: str, cab_type: str, mass,
                 drag_coef, year, brand, engine, tranny):
        super().__init__(mass, drag_coef, year, brand, engine, tranny)
        bed_types=['short','mid','long']
        cab_types=['regular','extended','crew']
        capacities = {
            'regular' : 3,
            'extended' : 5,
            'crew' : 6}
        self.bed_type = str(bed_type)
        self.cab_type = str(cab_type)
        #want to raise errors on purpose because of project specification of unhandled exception
        if self.bed_type not in bed_types:
>           raise ValueError(f"{self.bed_type} is not a valid choice")
E           ValueError: shorts is not a valid choice

project_7.py:211: ValueError
_____________________________________________________________________ test_truck_wrong_param_2_creation _____________________________________________________________________

    def test_truck_wrong_param_2_creation():
        """Will fail due to 'crews' not being a valid cab type"""
>       my_truck = Truck(bed_type='short', cab_type='crews', mass=4000,
                     drag_coef=0.3, year=2005, brand='Chevy',
                     engine=Engine(type='diesel', config='V8', displ=6.7, hprs=600, torq=700),
                     tranny=Tranny(type='automatic', gears=8))

test_project_7.py:48:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
test_project_7.py:48:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

self = <project_7.Truck object at 0x000002441903F790>, bed_type = 'short', cab_type = 'crews', mass = 4000, drag_coef = 0.3, year = 2005, brand = 'Chevy'
engine = <project_7.Engine object at 0x000002441903F910>, tranny = <project_7.Tranny object at 0x000002441903F850>

    def __init__(self, bed_type: str, cab_type: str, mass,
                 drag_coef, year, brand, engine, tranny):
        super().__init__(mass, drag_coef, year, brand, engine, tranny)
        bed_types=['short','mid','long']
        cab_types=['regular','extended','crew']
        capacities = {
            'regular' : 3,
            'extended' : 5,
            'crew' : 6}
        self.bed_type = str(bed_type)
        self.cab_type = str(cab_type)
        #want to raise errors on purpose because of project specification of unhandled exception
        if self.bed_type not in bed_types:
            raise ValueError(f"{self.bed_type} is not a valid choice")
        elif self.cab_type not in cab_types:
>           raise ValueError(f"{self.cab_type} is not a valid choice")
E           ValueError: crews is not a valid choice

project_7.py:213: ValueError
____________________________________________________________________________ test_truck_params _____________________________________________________________________________

    def test_truck_params():
        """test fails due to `repair()` should only be used in `run()` method, not by itself"""
        my_truck = Truck(bed_type='short', cab_type='crew', mass=4000,
                drag_coef=0.3, year=2005, brand='Chevy',
                engine=Engine(type='diesel', config='V8', displ=6.7, hprs=600, torq=700),
                tranny=Tranny(type='automatic', gears=8))
        my_truck.repair()
>       my_truck.run()

test_project_7.py:63:
                                                                             Ln 7, Col 24   Spaces: 4   UTF-8   CRLF   {}
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

self = <project_7.Truck object at 0x0000024418F97E20>

    def repair(self):
        """repair method for actually working on the chassis
        """
        #makes copies of the good parts
        self.good_eng = copy.deepcopy(self.Engine)
        self.good_tranny = copy.deepcopy(self.Tranny)
        #sees which parts needs to be repaired
>       repair_eng = self.Engine.repair()
E       AttributeError: 'NoneType' object has no attribute 'repair'

project_7.py:130: AttributeError
_____________________________________________________________________ test_car_params ____

    def test_car_params():
        """test fails due to `repair()` should only be used in `run()` method, not by itself"""
        my_car = Car(capacity=5, mass=2000,
                     drag_coef=0.3, year=1994, brand='Toyota',
                     engine=Engine(type='petrol', config='I4', displ=2.4, hprs=105, torq=95),
                     tranny=Tranny(type='manual', gears=5))
        my_car.repair()
>       my_car.run()

test_project_7.py:73:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
project_7.py:148: in run
    self.repair()
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

self = <project_7.Car object at 0x00000244190346A0>

    def repair(self):
        """repair method for actually working on the chassis
        """
```

```
self = <project_7.Car object at 0x0000024419034 6A0>

    def repair(self):
        """repair method for actually working on the chassis
        """
        #makes copies of the good parts
        self.good_eng = copy.deepcopy(self.Engine)
        self.good_tranny = copy.deepcopy(self.Tranny)
        #sees which parts needs to be repaired
>       repair_eng = self.Engine.repair()
E       AttributeError: 'NoneType' object has no attribute 'repair'
```

```
project_7.py:130: AttributeError
=========================== short test summary info ============================
FAILED test_project_7.py::test_car_wrong_param_creation - ValueError: invalid literal for int() with base 10: 'five'
FAILED test_project_7.py::test_truck_wrong_param_1_creation - ValueError: 'shorts' is not a valid choice
FAILED test_project_7.py::test_truck_wrong_param_2_creation - ValueError: 'crews' is not a valid choice
FAILED test_project_7.py::test_truck_params - AttributeError: 'NoneType' object has no attribute 'repair'
FAILED test_project_7.py::test_car_params - AttributeError: 'NoneType' object has no attribute 'repair'
========================== 5 failed, 2 passed in 2.12s ==========================
```