CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Automated Debugging

WS 2022/2023

Prof. Dr. Andreas Zeller
Paul Zhu
Marius Smytzek

Exercise 3 (10 Points)

Due: 07. December 2022

Submit your solutions as a Zip file on your status page in the CMS.

We will provide you a structure to submit your solutions where each task has a dedicated file. You can add new files and scripts if you want, but you must not delete any provided ones. You can verify whether your submission is valid by calling python3.

```
python3 verify.py
```

The output provides an overview if a required file, variable, or function is missing and if a function pattern was altered. If you do not follow this structure or change it, we cannot evaluate your submission. A non evaluable exercise will result in 0 points, so make sure to verify your work before submitting it. Note that the script does not reveal if your solutions are correct.

Exercise 3-1: Tell Me... Do You Bleed? (2 Points)

Are you familiar with the infamous Heartbleed vulnerability? If not, make yourself familiar with it by reading this article.

We have implemented a simple heartbeat in **exercise_1.py** that is vulnerable to the said bug. Your goal for this exercise is to add pre-, post-conditions, and data invariants such that the program aborts the execution when the input triggers the Heartbleed vulnerability. Use Python's `assert` keyword to introduce assertions.

For this exercise, you cannot assume that `store_data()` and `get_data()` are correct (indeed, for evaluating your submission, we will modify these functions). Hence, you should add all your assertions to the `heartbeat()` function.

In [1]:
```python
data = 'password:hjasdiebk456jhaccount:smytzek'

def store_data(payload: str):
    global data
    data = payload + data

def get_data(length: int) -> str:
    return data[:min(length, len(data) + 1)]

def heartbeat(length: int, payload: str) -> str:
    store_data(payload)
    return get_data(length)
```

Exercise 3-2: PyUnit (3 Points)

For this exercise, you should implement all standard assertion functions of a unit testing framework with the help of Python's `assert`. Implement the `TODO`'s in **exercise_2.py**. The comments in each function give you hints on what you should implement. In addition, running `python3 exercise_2.py` executes several tests verifying your functions.

In [11]:
```python
def assert_equal(a, b):
    # assert that a is equal to b
```

```
    assert True # TODO: add the assertion

def assert_not_equal(a, b):
    # assert that a is not equal to b
    assert True # TODO: add the assertion

def assert_true(a):
    # assert that the bool representation of a is True
    # you can get the bool representation by casting a to bool
    assert True # TODO: add the assertion

def assert_false(a):
    # assert that the bool representation of a is False
    # you can get the bool representation by casting a to bool.
    assert True # TODO: add the assertion

def assert_is(a, b):
    # assert that a is the same object as b
    assert True # TODO: add the assertion

def assert_is_not(a, b):
    # assert that a is not the same object as b
    assert True # TODO: add the assertion

def assert_is_none(a):
    # assert that a is None
    assert True # TODO: add the assertion

def assert_is_not_none(a):
    # assert that a is not None
    assert True # TODO: add the assertion

def assert_is_in(a, b):
    # assert that a is an element of b
    assert True # TODO: add the assertion

def assert_is_not_in(a, b):
    # assert that a is not an element of b
    assert True # TODO: add the assertion

def assert_is_instance(a, t):
    # assert that a is of type t
    assert True # TODO: add the assertion

def assert_is_not_instance(a, t):
    # assert that a is not of type t
    assert True # TODO: add the assertion
```

Exercise 3-3: I Like Trains! (5 Points)

For this exercise, you should add data invariants and some preconditions to a signalling block system. Please read the following article about signalling block systems. We designed a train network in the `TrainNetwork` class. This network consists of stations, where each station knows its tracks to other stations. A track knows the start and end station and the train that currently occupies this track. The train is `None` if there is no train on this track. The basic assumption of a signalling block system is that a train can only move to a new track if no other train blocks an adjacent track. Or in other words, the minimal distance between two trains is two stations or one track.

This network is a simple representation of a bidirectional graph, where the stations are nodes and the tracks are edges.

Your goal is to implement the `repOK()` methods for the `Track`, `Station`, and `TrainNetwork` class that checks whether all data is valid. Please orientate yourself on the following points:

- There cannot be two trains with the same `train_id` in the same network.
- A station can only have a track if the station is either the start or end of the track.
- The signalling blocking system must hold before and after each operation in the network.

In addition to data invariants, you must verify some preconditions so that the state will not corrupt:

- A train can only move to a track if no other train occupies the track.
- A train can only move from and to a track part of the network.
- A train can only move if it is part of the network.
- A train can only move to an adjacent track.

If you are unsure, we have implemented several tests to help verify your assertions. Please run `python3 exercise_3.py` to execute the tests.

We will also provide some tips for you:

- You only need to add code at the TODO locations.
- You are allowed to add new tests to verify your results.
- You are allowed to share your tests with other students as long as you do not share any part of your solution.

In [15]:
```
from typing import Optional, List

class Train:

    def __init__(self, train_id: int):
        self.train_id = train_id


class Track:

    def repOK(self):
        # TODO: implement the function that verifies whether
        # this track is in a valid state
        assert True

    def __init__(self, start, end, train: Optional[Train] = None):
        self.start = start
        self.end = end
        self.train = train
        self.repOK()

    def set_train(self, train: Train):
        self.repOK()
        # TODO: add preconditions
        self.train = train
        self.repOK()

    def remove_train(self):
        self.repOK()
        self.train = None
        self.repOK()


class Station:

    def repOK(self):
        # TODO: implement the function that verifies whether
        # this station is in a valid state
        assert True

    def __init__(self, tracks: Optional[List[Track]] = None):
        self.tracks = tracks if tracks else []
        self.repOK()


    def add_track(self, track: Track):
        self.repOK()
        self.tracks.append(track)
        self.repOK()


class TrainNetwork:
```

```
    def repOK(self):
        # TODO: implement the function that verifies whether
        # this train network is in a valid state
        assert True

    def __init__(self, stations: Optional[List[Station]] = None,
                 tracks: Optional[List[Track]] = None,
                 trains: Optional[List[Train]] = None):
        self.stations = stations if stations else []
        self.tracks = tracks if tracks else []
        self.trains = trains if trains else []
        self.repOK()

    def add_station(self, station: Station):
        self.repOK()
        self.stations.append(station)
        self.repOK()

    def add_track(self, track: Track):
        self.repOK()
        self.tracks.append(track)
        self.repOK()

    def add_train(self, train: Train):
        self.repOK()
        self.trains.append(train)
        self.repOK()

    def move_train(self, train: Train, from_track: Track, to_track: Track):
        self.repOK()
        # TODO: add preconditions
        from_track.remove_train()
        to_track.set_train(train)
        self.repOK()
```

As an example, the following code

```
# Trains
train_0 = Train(0)
train_1 = Train(1)

# Stations
station_0 = Station()
station_1 = Station()
station_2 = Station()
station_3 = Station()
station_4 = Station()

# Tracks
track_0 = Track(station_0, station_1, train_0)
# add track to start and end station
station_0.add_track(track_0)
station_1.add_track(track_0)

track_1 = create_track(station_1, station_2)
station_1.add_track(track_1)
station_2.add_track(track_1)

track_2 = create_track(station_1, station_2)
station_1.add_track(track_2)
station_2.add_track(track_2)

track_3 = create_track(station_2, station_3)
station_2.add_track(track_3)
station_3.add_track(track_3)

track_4 = create_track(station_3, station_4, train_1)
station_3.add_track(track_4)
station_4.add_track(track_4)

track_5 = create_track(station_4, station_0)
```

```
station_4.add_track(track_5)
station_0.add_track(track_5)


TrainNetwork(
    stations=[station_0, station_1, station_2, station_3, station_4],
    tracks=[track_0, track_1, track_2, track_3, track_4, track_5],
    trains=[train_0, train_1]
)
```

creates this train network: