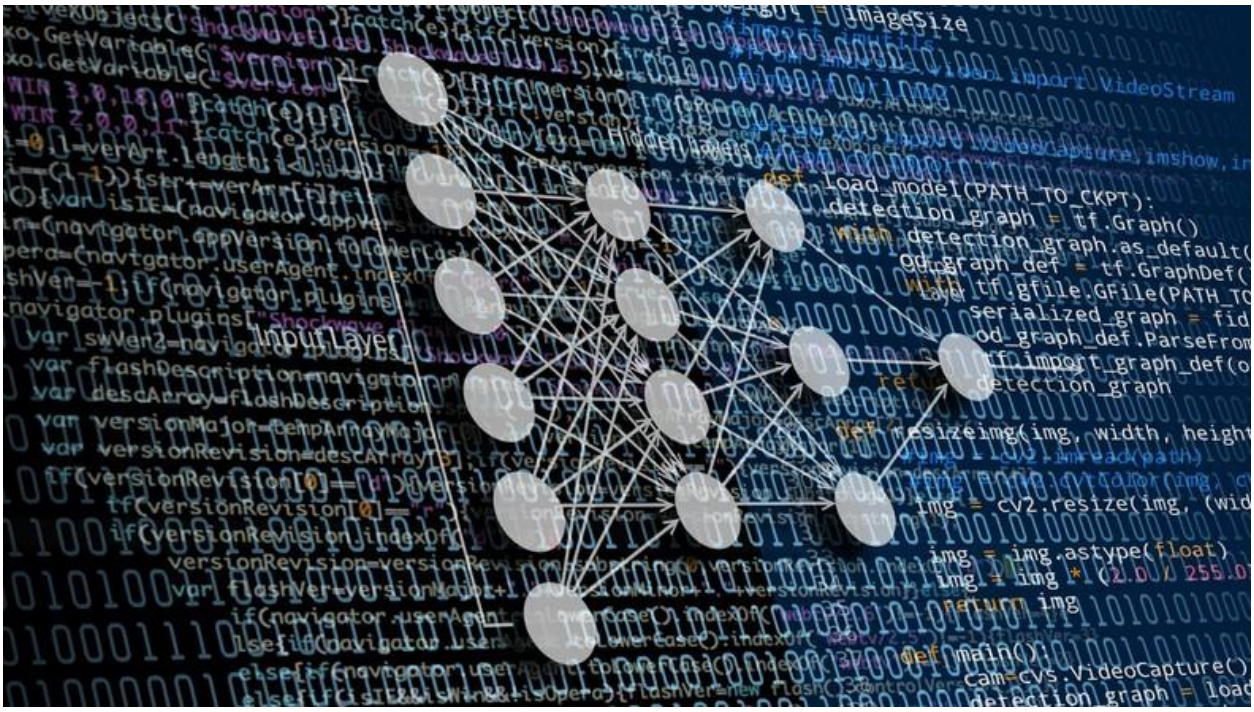
Assignment-1-trains



Coded by:

Tygo Geervliet

Student Number: 500897270

Santosh Kakkar

Student Number: 500904843

Seven relevant methods

Train

method 1: Attaching a Wagon Sequence to the rear of the Train

```
21 usages   Tygo +1
public boolean attachToRear(Wagon wagon) {
    if (wagon == null) {
        return false;
    }

    // Check if the wagon is already part of this train
    if (findWagonById(wagon.id) != null) {
        return false;
    }

    // Detach the wagon
    if (wagon.hasPreviousWagon()) {
        wagon.getPreviousWagon().setNextWagon(null);
        wagon.setPreviousWagon(null);
    }

    // Check if there's capacity to attach the wagon sequence
    int totalWagons = getNumberOfWagons();
    int wagonsToAttach = wagon.getSequenceLength();
    if (totalWagons + wagonsToAttach > this.engine.getMaxWagons()) {
        return false; // Not enough capacity to attach the wagons
    }

    // Attach the wagon (or sequence) to the rear of the train
    if (this.firstWagon == null) {
        this.firstWagon = wagon;
    } else {
        Wagon lastWagon = this.firstWagon.getLastWagonAttached();
        lastWagon.setNextWagon(wagon);
        wagon.setPreviousWagon(lastWagon);
    }

    return true;
}
```

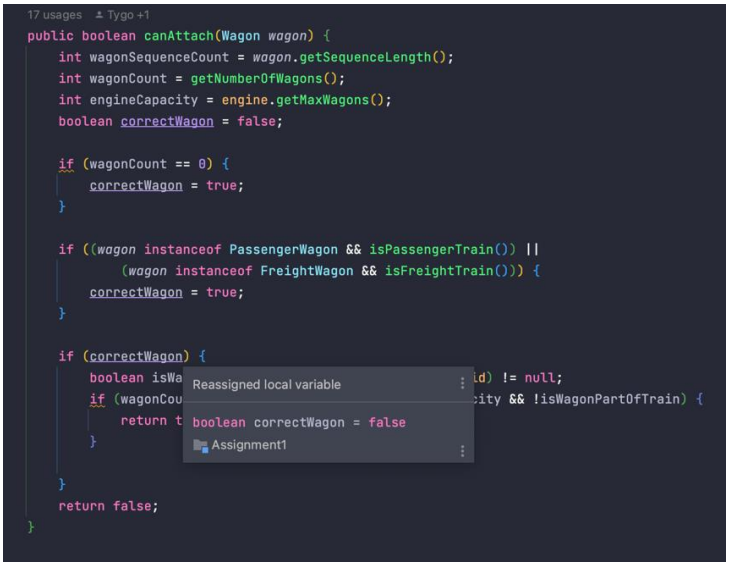
In the Train class, the **attachToRear** method attaches a sequence of wagons to the rear of the train. It checks if the train can accommodate the wagons based on type and engine capacity before attaching them.

method 2: Finding a Wagon by ID

```
9 usages  Tygo +1
public Wagon findWagonById(int wagonId) {
    Wagon currentWagon = firstWagon;

    if (currentWagon == null) {
        return null;
    }

    //the length of the sequence of wagons towards the end of its tail
    int WagonLength = currentWagon.getSequenceLength();

    for (int i = 0; i <= WagonLength; i++) {
        if (currentWagon == null) {
            return null;
        }
        if (currentWagon.getId() == wagonId) {
            return currentWagon;
        }
        currentWagon = currentWagon.getNextWagon();
    }

    return null;
}
```

The findWagonById method in the Train class allows for searching a wagon based on its ID. This is a handy method for operations involving specific wagons.

method 3: Checking if a Wagon Sequence can be Attached



```
17 usages  Tygo +1
public boolean canAttach(Wagon wagon) {
    int wagonSequenceCount = wagon.getSequenceLength();
    int wagonCount = getNumberOfWagons();
    int engineCapacity = engine.getMaxWagons();
    boolean correctWagon = false;

    if (wagonCount == 0) {
        correctWagon = true;
    }

    if ((wagon instanceof PassengerWagon && isPassengerTrain()) ||
            (wagon instanceof FreightWagon && isFreightTrain())) {
        correctWagon = true;
    }

    if (correctWagon) {
        boolean isWa
        if (wagonCou
            return t
        }

    }
    return false;
}
```


In the Train class, the canAttach method checks whether a given sequence of wagons can be attached to the train based on the type and engine capacity.

Method 4: finding the tail of a sequence

This method finds the tail of the sequence.

Loop-Invariant:

Loop-Invariant: At the start of each iteration, Tail points to a wagon within the sequence.

Justification --

Initialization: Tail starts as the given wagon, so it's within the sequence.

Maintenance: Inside the loop, Tail moves to the next wagon. Since the loop checks if a next wagon exists, Tail always stays within the sequence.

Termination: The loop stops when Tail is the last wagon in the sequence, and it's returned.

Using this loop-invariant, I can say that findTailOfSequence always returns the last wagon in the sequence.

Wagon

Method 1: detach from from front.

```
1 usage  santoshkakkar
public void reAttachTo(Wagon front) {
    //deletes former connections
    if (this.hasPreviousWagon()) {
        this.detachFront();
    }

    if (front.hasNextWagon()) {
        front.detachTail();
    }
    //links "this" behind the front
    this.nextWagon = front.getNextWagon();
    front.attachTail(this);

    //the previous wagon of the current wagon is connected to the front
    if (this.hasNextWagon()) {
        this.getNextWagon().previousWagon = front;
    }
}
```

In this method I detach the connection to the previousWagon. First, I check if it has any former connections. Then I link the main wagon behind the front. Lastly, I let the previous Wagon know that it's now attached to the "front".

Method 2: removes a wagon from the sequence.

```
public void removeFromSequence() {
    if (this.previousWagon != null) {
        this.previousWagon.nextWagon = this.nextWagon;// this skips "this" and points to the next one
    }
    if (this.nextWagon != null) {
        this.nextWagon.previousWagon = this.previousWagon;// this skips the "this" and goes to the next one
    }
    //connections of the current are detached
    this.previousWagon = null;
    this.nextWagon = null;
}
```

In this method I link the previous nextWagon from the mainWagon to the nextWagon from the mainWagon. It skips the mainWagon so it can attach to the wagon that follows up after the mainWagon.

It's also the same for the previousWagon: I link the next previousWagon from the mainWagon to the previousWagon from the mainWagon. It skips the mainWagon so it can attach to the wagon that is behind the mainWagon

Method 3: reverses the order of the sequence.

```
public Wagon reverseSequence() {
    Wagon lastWagon = this.getLastWagonAttached();
    Wagon newOrder = lastWagon; //new variable

    // loops till there are no more previousWagons
    while (lastWagon.hasPreviousWagon()) {
        Wagon beforeLastWagon = lastWagon.previousWagon;
        beforeLastWagon.removeFromSequence();
        newOrder.attachTail(beforeLastWagon);

        newOrder = beforeLastWagon;
        // stop if the beforeLastWagon and this are the same
        if (beforeLastWagon == this) {
            break;
        }
    }

    return lastWagon;
}
```

In this method I reverse the order of the sequence.

(1) Firstly, I make a loop that checks if the lastWagon of the sequence has a predecessor. (2) Then makes a variable to hold the predecessor from the lastWagon and removes it form the sequence. (3) Lastly I make a new order that starts with the lastWagon and attaches the removed sequences to the tail. Example:

1. 1-2-3-4
2. 1-2 3-4 → 1-2 4-3 → 1 2 4-3 →2-1 4-3
3. 4-3-2-1