SIMPY-Refer here https://simpy.readthedocs.io/en/latest/simpy_intro/index.html

Example 1 Carwash

In [11]:
```
"""
Carwash example.

Covers:

- Waiting for other processes
- Resources: Resource

Scenario:
  A carwash has a limited number of washing machines and defines
  a washing processes that takes some (random) time.

  Car processes arrive at the carwash at a random time. If one washing
  machine is available, they start the washing process and wait for it
  to finish. If not, they wait until they can use one.

"""

import itertools
import random

import simpy

# fmt: off
RANDOM_SEED = 42
NUM_MACHINES = 2  # Number of machines in the carwash
WASHTIME = 5      # Minutes it takes to clean a car
T_INTER = 7       # Create a car every ~7 minutes
SIM_TIME = 20     # Simulation time in minutes
# fmt: on


class Carwash:
    """A carwash has a limited number of machines (``NUM_MACHINES``) to
    clean cars in parallel.

    Cars have to request one of the machines. When they got one, they
    can start the washing processes and wait for it to finish (which
    takes ``washtime`` minutes).

    """

    def __init__(self, env, num_machines, washtime):
        self.env = env
        self.machine = simpy.Resource(env, num_machines)
        self.washtime = washtime

    def wash(self, car):
        """The washing processes. It takes a ``car`` processes and tries
        to clean it."""
        yield self.env.timeout(self.washtime)
        pct_dirt = random.randint(50, 99)
        print(f"Carwash removed {pct_dirt}% of {car}'s dirt.")
```

```
def car(env, name, cw):
    """The car process (each car has a ``name``) arrives at the carwash
    (``cw``) and requests a cleaning machine.

    It then starts the washing process, waits for it to finish and
    leaves to never come back ...

    """
    print(f'{name} arrives at the carwash at {env.now:.2f}.')
    with cw.machine.request() as request:
        yield request

        print(f'{name} enters the carwash at {env.now:.2f}.')
        yield env.process(cw.wash(name))

        print(f'{name} leaves the carwash at {env.now:.2f}.')


def setup(env, num_machines, washtime, t_inter):
    """Create a carwash, a number of initial cars and keep creating cars
    approx. every ``t_inter`` minutes."""
    # Create the carwash
    carwash = Carwash(env, num_machines, washtime)

    car_count = itertools.count()

    # Create 4 initial cars
    for _ in range(4):
        env.process(car(env, f'Car {next(car_count)}', carwash))

    # Create more cars while the simulation is running
    while True:
        yield env.timeout(random.randint(t_inter - 2, t_inter + 2))
        env.process(car(env, f'Car {next(car_count)}', carwash))


# Setup and start the simulation
print('Carwash')
print('Check out http://youtu.be/fXXmeP9TvBg while simulating ... ;-)')
random.seed(RANDOM_SEED)  # This helps to reproduce the results

# Create an environment and start the setup process
env = simpy.Environment()
env.process(setup(env, NUM_MACHINES, WASHTIME, T_INTER))

# Execute!
env.run(until=SIM_TIME)
```

```
Carwash
Check out http://youtu.be/fXXmeP9TvBg while simulating ... ;-)
Car 0 arrives at the carwash at 0.00.
Car 1 arrives at the carwash at 0.00.
Car 2 arrives at the carwash at 0.00.
Car 3 arrives at the carwash at 0.00.
Car 0 enters the carwash at 0.00.
Car 1 enters the carwash at 0.00.
Car 4 arrives at the carwash at 5.00.
Carwash removed 97% of Car 0's dirt.
Carwash removed 67% of Car 1's dirt.
Car 0 leaves the carwash at 5.00.
Car 1 leaves the carwash at 5.00.
Car 2 enters the carwash at 5.00.
Car 3 enters the carwash at 5.00.
Car 5 arrives at the carwash at 10.00.
Carwash removed 64% of Car 2's dirt.
Carwash removed 58% of Car 3's dirt.
Car 2 leaves the carwash at 10.00.
Car 3 leaves the carwash at 10.00.
Car 4 enters the carwash at 10.00.
Car 5 enters the carwash at 10.00.
Carwash removed 97% of Car 4's dirt.
Carwash removed 56% of Car 5's dirt.
Car 4 leaves the carwash at 15.00.
Car 5 leaves the carwash at 15.00.
Car 6 arrives at the carwash at 16.00.
Car 6 enters the carwash at 16.00.
```

Example 2 Gas Station Refueling

In [12]:

```
"""
Gas Station Refueling example

Covers:

- Resources: Resource
- Resources: Container
- Waiting for other processes

Scenario:
  A gas station has a limited number of gas pumps that share a common
  fuel reservoir. Cars randomly arrive at the gas station, request one
  of the fuel pumps and start refueling from that reservoir.

  A gas station control process observes the gas station's fuel level
  and calls a tank truck for refueling if the station's level drops
  below a threshold.

"""

import itertools
import random

import simpy

# fmt: off
RANDOM_SEED = 42
STATION_TANK_SIZE = 200    # Size of the gas station tank (liters)
THRESHOLD = 25             # Station tank minimum level (% of full)
```

```python
CAR_TANK_SIZE = 50          # Size of car fuel tanks (liters)
CAR_TANK_LEVEL = [5, 25]    # Min/max levels of car fuel tanks (liters)
REFUELING_SPEED = 2         # Rate of refuelling car fuel tank (liters / second)
TANK_TRUCK_TIME = 300       # Time it takes tank truck to arrive (seconds)
T_INTER = [30, 300]         # Interval between car arrivals [min, max] (seconds)
SIM_TIME = 1000             # Simulation time (seconds)
# fmt: on


def car(name, env, gas_station, station_tank):
    """A car arrives at the gas station for refueling.

    It requests one of the gas station's fuel pumps and tries to get the
    desired amount of fuel from it. If the station's fuel tank is
    depleted, the car has to wait for the tank truck to arrive.

    """
    car_tank_level = random.randint(*CAR_TANK_LEVEL)
    print(f'{env.now:6.1f} s: {name} arrived at gas station')
    with gas_station.request() as req:
        # Request one of the gas pumps
        yield req

        # Get the required amount of fuel
        fuel_required = CAR_TANK_SIZE - car_tank_level
        yield station_tank.get(fuel_required)

        # The "actual" refueling process takes some time
        yield env.timeout(fuel_required / REFUELING_SPEED)

        print(f'{env.now:6.1f} s: {name} refueled with {fuel_required:.1f}L')


def gas_station_control(env, station_tank):
    """Periodically check the level of the gas station tank and call the tank
    truck if the level falls below a threshold."""
    while True:
        if station_tank.level / station_tank.capacity * 100 < THRESHOLD:
            # We need to call the tank truck now!
            print(f'{env.now:6.1f} s: Calling tank truck')
            # Wait for the tank truck to arrive and refuel the station tank
            yield env.process(tank_truck(env, station_tank))

        yield env.timeout(10)  # Check every 10 seconds


def tank_truck(env, station_tank):
    """Arrives at the gas station after a certain delay and refuels it."""
    yield env.timeout(TANK_TRUCK_TIME)
    amount = station_tank.capacity - station_tank.level
    station_tank.put(amount)
    print(
        f'{env.now:6.1f} s: Tank truck arrived and refuelled station with {amount:.1f}L'
    )


def car_generator(env, gas_station, station_tank):
    """Generate new cars that arrive at the gas station."""
    for i in itertools.count():
        yield env.timeout(random.randint(*T_INTER))
```

```
        env.process(car(f'Car {i}', env, gas_station, station_tank))


# Setup and start the simulation
print('Gas Station refuelling')
random.seed(RANDOM_SEED)

# Create environment and start processes
env = simpy.Environment()
gas_station = simpy.Resource(env, 2)
station_tank = simpy.Container(env, STATION_TANK_SIZE, init=STATION_TANK_SIZE)
env.process(gas_station_control(env, station_tank))
env.process(car_generator(env, gas_station, station_tank))

# Execute!
env.run(until=SIM_TIME)
```

```
Gas Station refuelling
  87.0 s: Car 0 arrived at gas station
 105.5 s: Car 0 refueled with 37.0L
 129.0 s: Car 1 arrived at gas station
 148.0 s: Car 1 refueled with 38.0L
 284.0 s: Car 2 arrived at gas station
 305.0 s: Car 2 refueled with 42.0L
 385.0 s: Car 3 arrived at gas station
 398.5 s: Car 3 refueled with 27.0L
 459.0 s: Car 4 arrived at gas station
 460.0 s: Calling tank truck
 481.0 s: Car 4 refueled with 44.0L
 705.0 s: Car 5 arrived at gas station
 750.0 s: Car 6 arrived at gas station
 760.0 s: Tank truck arrived and refuelled station with 188.0L
 779.0 s: Car 6 refueled with 38.0L
 781.5 s: Car 5 refueled with 43.0L
 891.0 s: Car 7 arrived at gas station
 904.0 s: Car 7 refueled with 26.0L
```

Basis / Prototype for WIPCP

In [15]:
```
import simpy
import os
import numpy as np
```

```python
from sklearn.linear_model import LinearRegression
from datetime import datetime, timedelta
import pandas as pd
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

# Regression model for station processing time prediction
def train_regression_model():
    # Training data: station number and hour of check-in (as an integer)
    X_train = np.array([[1, 10], [2, 15], [3, 20], [4, 30], [5, 45], [6, 60], [7, 75]])
    y_train = np.array([5, 7, 6, 10, 8, 9, 12])  # Example processing times (in hours)
    model = LinearRegression()
    model.fit(X_train, y_train)
    return model

# Load the regression model
regression_model = train_regression_model()

# Function to predict time completion for a station based on station number and check-in time
def predict_time_completion(station_number, indatetime):
    indatetime = indatetime.hour  # Extract the hour from check-in time
    input_data = np.array([[station_number, indatetime]])
    return regression_model.predict(input_data)[0]  #yield inference from model

# Discrete Event Simulation for manufacturing process, predicting check-in/arrival (ETA) and estimate time completion(ETC) timestamps at each station
class Product:
    def __init__(self, env, product_id, check_in_time, start_station, completion_table):
        self.env = env
        self.product_id = product_id
        self.check_in_time = check_in_time
        self.current_time = check_in_time  # Start with initial check-in time
        self.current_station = start_station
        self.completion_table = completion_table
        env.process(self.manufacturing_process())

    def manufacturing_process(self):
# Process through stations from current to station 7
        while self.current_station <= 7:
# Save the current check-in time
            current_check_in_time = self.current_time
# Predict processing time for the current station
            processing_time = predict_time_completion(self.current_station, self.current_time)
# Calculate completion time as check-in time + processing time
            completion_time = pd.Timestamp(self.current_time) + pd.Timedelta(hours=processing_time)
# Log check-in time and completion time as a tuple
            self.completion_table.at[self.product_id, f"Station {self.current_station}"] = (
                current_check_in_time.strftime('%Y-%m-%d %H:%M:%S'),
                completion_time.strftime('%Y-%m-%d %H:%M:%S')
            )
# Update current time to the completion time of this station
            self.current_time = completion_time

# Simulate the processing time at the station while executing other processes
            yield self.env.timeout(3)

# Move to the next station
            self.current_station += 1


# Simulate multiple Products based on dataset
```

```
def run_simulation(dataset):
    env = simpy.Environment()

# Create a DataFrame to store the completion times for each product
    completion_table = pd.DataFrame(columns=[f"Station {i}" for i in range(1, 8)] ,
                        index=[f"{row['Product ID']}" for _, row in dataset.iterrows()])

# Create Product instances based on the dataset
    for _, row in dataset.iterrows():
        check_in_time = pd.to_datetime(row['Check-in Time'])
        start_station = row['Current Station']
        Product(env, f"{row['Product ID']}", check_in_time, start_station, completion_table)
    env.run()
    return completion_table


# Sample dataset
data = {
        "Product ID": ['U1231AMY61089010', 'U1231AMY61129035', 'U1231AMY62070043', 'U1231AMY62120053', 'U1231AMY62129051'],
        "Check-in Time": ["2024-10-9 9:30", "2024-10-9 16:30", "2024-10-10 14:45", "2024-10-11 08:00", "2024-10-11 07:15"],
        "Current Station": [3, 2, 5, 4, 1]
    }
dataset = pd.DataFrame(data)
# Run the simulation with the dataset and display the results
projection_table = run_simulation(dataset)
projection_table
```

Out[15]:

	Station 1	Station 2	Station 3	Station 4	Station 5	Station 6	Station 7
U1231AMY61089010	NaN	NaN	(2024-10-09 09:30:00, 2024-10-09 16:42:59)	(2024-10-09 16:42:59, 2024-10-10 00:54:25)	(2024-10-10 00:54:25, 2024-10-10 10:07:28)	(2024-10-10 10:07:28, 2024-10-10 20:18:32)	(2024-10-10 20:18:32, 2024-10-11 07:27:36)
U1231AMY61129035	NaN	(2024-10-09 16:30:00, 2024-10-09 22:42:37)	(2024-10-09 22:42:37, 2024-10-10 05:53:48)	(2024-10-10 05:53:48, 2024-10-10 14:06:45)	(2024-10-10 14:06:45, 2024-10-10 23:17:51)	(2024-10-10 23:17:51, 2024-10-11 09:27:06)	(2024-10-11 09:27:06, 2024-10-11 20:37:42)
U1231AMY62070043	NaN	NaN	NaN	NaN	(2024-10-10 14:45:00, 2024-10-10 23:56:06)	(2024-10-10 23:56:06, 2024-10-11 10:05:21)	(2024-10-11 10:05:21, 2024-10-11 21:15:49)
U1231AMY62120053	NaN	NaN	NaN	(2024-10-11 08:00:00, 2024-10-11 16:12:32)	(2024-10-11 16:12:32, 2024-10-12 01:23:21)	(2024-10-12 01:23:21, 2024-10-12 11:35:40)	(2024-10-12 11:35:40, 2024-10-12 22:46:00)
U1231AMY62129051	(2024-10-11 07:15:00, 2024-10-11 12:29:28)	(2024-10-11 12:29:28, 2024-10-11 18:42:38)	(2024-10-11 18:42:38, 2024-10-12 01:54:23)	(2024-10-12 01:54:23, 2024-10-12 10:07:54)	(2024-10-12 10:07:54, 2024-10-12 19:19:33)	(2024-10-12 19:19:33, 2024-10-13 05:29:22)	(2024-10-13 05:29:22, 2024-10-13 16:40:32)