Adapter Design Pattern
in flight.java

```
package flight.reservation.flight;

import flight.reservation.Airport;
import flight.reservation.plane.Helicopter;
import flight.reservation.plane.PassengerDrone;
import flight.reservation.plane.PassengerPlane;

import java.util.Arrays;

public class Flight {

   private int number;
   private Airport departure;
   private Airport arrival;
   protected Object aircraft;

   public Flight(int number, Airport departure, Airport arrival, Object
aircraft) throws IllegalArgumentException {
       this.number = number;
       this.departure = departure;
       this.arrival = arrival;
       this.aircraft = adaptAircraft(aircraft);
       checkValidity();
   }
private Object adaptAircraft(Object aircraft){
       if(aircraft instanceof PassengerPlane|| aircraft instanceof Helicopter){
           return aircraft;
       }
       else if(aircraft instanceof PassengerDrone){
           return new PassengerPlaneAdapter((PassengerDrone)aircraft);
       }else{
           throw new IllegalArgumentException(String.format("Aircraft not
recognizable"));
       }
}
   private void checkValidity() throws IllegalArgumentException {
       if (!isAircraftValid(departure) || !isAircraftValid(arrival)) {
           throw new IllegalArgumentException("Selected aircraft is not valid
for the selected route.");
       }
   }

   private boolean isAircraftValid(Airport airport) {
       return Arrays.stream(airport.getAllowedAircrafts()).anyMatch(x -> {
```

```
            String model;
            if (this.aircraft instanceof PassengerPlane) {
                model = ((PassengerPlane) this.aircraft).model;

            } else {
                throw new IllegalArgumentException(String.format("Aircraft is
not recognized"));
            }
            return x.equals(model);
        });
    }

    public Object getAircraft() {
        return aircraft;
    }

    public int getNumber() {
        return number;
    }

    public Airport getDeparture() {
        return departure;
    }

    public Airport getArrival() {
        return arrival;
    }

    @Override
    public String toString() {
        return aircraft.toString() + "-" + number + "-" + departure.getCode() +
"/" + arrival.getCode();
    }

}
class PassengerPlaneAdapter implements PassengerPlane {
    private final PassengerDrone passengerDrone;

    public PassengerPlaneAdapter(PassengerDrone passengerDrone) {
        this.passengerDrone = passengerDrone;
    }

    @Override
    public String getModel() {
        return "HypaHype";
    }

    @Override
    public int getPassengerCapacity() {
```

```
        return passengerDrone.getPassengerCapacity();
    }

    @Override
    public int getMaxAltitude() {
        return passengerDrone.getMaxAltitude();
    }

    @Override
    public int getWingspan() {
        return passengerDrone.getWingspan();
    }
```

create an adapter class that converts the PassengerDrone and Helicopter classes to the PassengerPlane interface, which is expected by the Flight class.

We introduced a new method adaptAircraft to the updated Flight class, which takes an Object parameter and returns an adapted aircraft. The adaptAircraft method verifies the type of the input aircraft and returns a new object of the same type. If the input aircraft is a PassengerDrone, a new PassengerPlaneAdapter object is created, which converts the PassengerDrone to the PassengerPlane interface. If the input aircraft is a PassengerPlane or Helicopter, the input object is returned directly.

In the modified code, the following changes have been made:

- The Flight Constructor now takes a passengerPlane object instead of an Object.
- The isAircraftValid method has been modified to handle Passengerplaneadapter,which returns the string”HypaHype” as model.
- Added an adapter class PassengerPlaneadapter that adapts the PassengerDrone class to the PassengerPlane.

```
private Object adaptAircraft(Object aircraft){
        if(aircraft instanceof PassengerPlane|| aircraft instanceof Helicopter){
            return aircraft;
        }
        else if(aircraft instanceof PassengerDrone){
            return new PassengerPlaneAdapter((PassengerDrone)aircraft);
        }else{
            throw new IllegalArgumentException(String.format("Aircraft not recognizable"));
        }
}
    1 usage   Sidx-sys
    private void checkValidity() throws IllegalArgumentException {
        if (!isAircraftValid(departure) || !isAircraftValid(arrival)) {
            throw new IllegalArgumentException("Selected aircraft is not valid for the selected route.");
        }
    }

    2 usages   Sidx-sys *
    private boolean isAircraftValid(Airport airport) {
        return Arrays.stream(airport.getAllowedAircrafts()).anyMatch(x -> {
            String model;
            if (this.aircraft instanceof PassengerPlane) {
                model = ((PassengerPlane) this.aircraft).model;

            } else {
                throw new IllegalArgumentException(String.format("Aircraft is not recognized"));
            }
            return x.equals(model);
        });
    }
```

```
class PassengerPlaneAdapter implements PassengerPlane {
    4 usages
    private final PassengerDrone passengerDrone;

    1 usage   new *
    public PassengerPlaneAdapter(PassengerDrone passengerDrone) {
        this.passengerDrone = passengerDrone;
    }

    no usages   new *
    @Override
    public String getModel() {
        return "HypaHype";
    }
```