IFT3911 - Devoir #1 - Contraintes OCL

Hiver 2026

Équipe X-ray :

Trung Nguyen, 20238006

Dawson Zhang, 20268585

Tien Tran, 20246669

Université de Montréal

Canada, QC

11 Février 2026

Commentaire sur les Contraintes OCL : Nous n'avons pas supprimé les redondances, car nous avons souhaité respecter la structure de l'énoncé. Ainsi, l'ordre des contraintes OCL dans notre document correspond à l'ordre textuel dans lequel elles apparaissent dans l'énoncé, de haut en bas.

1 Admin

1.1 Flight

F1 – Unicité des identifiants d'aéroport

```
context Airport inv:
    Airport.allInstances()->forAll(a1, a2 | a1 <> a2 implies
    a1.id <> a2.id)
    and
    self.id.length = 3
    and
    self.id.matches('[A-Za-z]{3}')
```

F2 – Format et unicité des identifiants de vol

```
context Flight inv:
    self.id.substring(1,2).matches('[A-Za-z]{2}')
    and
    self.id.substring(3, self.id.size()).matches('[0-9]')
    and
    self.Company.Flight -> forAll(f1, f2 |
        f1.id.substring(1,2) = f2.id.substring(1,2))
    and
    self.Company.Flight -> isUnique(f | f.id.substring(3, f.id.size()))
    and
    Flight.allInstances()->forAll(f1,f2 |
        f1 <> f2 and f1.Company <> f2.Company implies
        f1.id.substring(1,2) <> f2.id.substring(1,2))
```

Commentaire sur les règles 3 et 4 : lorsque l'on écrit self.Company.Flight, on considère ici l'ensemble de tous les vols appartenant à la même compagnie, ce qui constitue donc une collection.

F3 – Différence entre aéroport de départ et de destination

```
context Flight inv:
    self.departure <> self.destination
```

Commentaire : les attributs departure et destination sont des instances de la classe Airport dans notre diagramme de classes.

F4 – Prix uniforme des sièges par section

```
context Flight inv:
    self.Plane.sectionPlane.Siege -> forAll(s1, s2 |
        s1.section = s2.section implies s1.price = s2.price)
```

1.2 Cruise

C1 – Unicité des identifiants de port

```
context Port inv:
    Port.allInstances()->forAll(a1, a2 | a1 <> a2 implies
    a1.id <> a2.id)
    and
    self.id.length = 3
    and
    self.id.matches('[A-Za-z]{3}')
```

C2 – Durée maximale du voyage

```
context Cruise inv:
    self.arrivalTime - self.departureTime <= 21
```

C3 – Chemin fermé

```
context Cruise inv:
    self.path->notEmpty()
    and
    self.path->first() = self.path->last()
```

C4 – Absence de chevauchement de croisières

```
context Cruise inv:
    Cruise.allInstances()->forAll(c1,c2 | c1.Boat = c2.Boat
    implies not (c1.departureTime < c2.arrivalTime
                 and c2.departureTime < c1.arrivalTime))
```

C5 – Prix uniforme des sièges par section pour Cruise

```
context Cruise inv:
    self.Boat.sectionBoat.Siege -> forAll(s1, s2 |
        s1.section = s2.section implies s1.price = s2.price)
```

Commentaire : Dans notre modèle de classes, les sièges et les cabines sont représentés par une seule classe, Siege.

1.3 Train

T1 – Unicité des identifiants de station

```
context Station inv:
    Station.allInstances()->forAll(st1, st2 | st1 <> st2 implies
    st1.id <> st2.id)
    and
    self.id.length = 3
    and
    self.id.matches('[A-Za-z]{3}')
```

2 Client

2.1 Reservation

R1 – Possibilité de réservation d'un siège d'avion/train

```
context Client::makeReservation(seat: Seat) : int
    pre: seat.occupy = 0
    post: seat.occupy = 1
```

R2 – Les réservations payées marquent le siège comme occupé

```
context Client:makePayment(Reservation): Boolean
    pre: Reservation.reservedSeat.occupied = 1 and
    currentTime.diffHours(r.startTimeReservation) < 24
    post: Reservation.isPaid = 1 and Reservation.reservedSeat =
    Reservation.client.myReservations->
    select(x | x = Reservation).reservedSeat
```

Commentaire : La précondition vérifie que le siège de la réservation passée en argument a bien été reservée et que la reservation n'a pas expiré (on suppose qu'il s'agit d'une réservation appartenant au client du contexte, puisqu'il n'aurait pas accès à d'autres réservations dans notre système) . La postcondition vérifie que la réservation a été payée et que le siège a été attribué au client du contexte (on vérifie que le siège associé à la réservation est celui attribué au client).

R3 – Possibilité de réservation d'une cabine de bateau

```
context Client::makeReservation(seat: Seat) : int
    pre: seat.occupy = 0
    post: seat.occupy = 1
```