```
/*
                    Travlendar+
                    ---Alloy project---

                    Paolo Fumagalli
                    Pietro Grotti
                    Marco Gullo
*/

module TravlendarPlus
open util/boolean

                    // SYSTEM

// Encloses all the users and the external actors
one sig System {
        users: set User,
        externals: set Actor
}

                    // EXTERNAL ACTORS

// Represents an external actor
abstract sig Actor {}

// All the kinds of external actors:
// >
sig PublicTransports extends Actor {
        service: one Public, // This relation links the actor with the service provided
        issued: some PublicTransportTicket // This relation defines the origin of the
emitted tickets
}

sig CarSharingService extends Actor {
        service: one CarSharing
}

sig BikeSharingService extends Actor {
        service: one BikeSharing
```

```
}

sig TrainService extends Actor {
	service: one Train,
	issued: some TrainTicket
}

sig TaxiService extends Actor {
	service: one Taxi
}
// <

	// ...facts

// All the actors considered belong to the system
fact actorsInSystem {
	all a: Actor | a in System.externals
}

			// TRANSPORT

// Represents the generic mean of transport
abstract sig Transport {}

// The transports which don't need an external actor in the system (they always exist and they are unique)
one sig Car, Walking extends Transport {}

// The transports which DO need an external actor in the system (they exist with an arity of 1 only if their provider exists)
lone sig Public, CarSharing, BikeSharing, Train, Taxi extends Transport {}

	// ...facts

// The actors and their services exist only in pairs
fact noTransportWithoutService {
	all t: Public | one s: PublicTransports | t in s.service
	all t: CarSharing | one s: CarSharingService | t in s.service
	all t: BikeSharing | one s: BikeSharingService | t in s.service
```

```
        all t: Train | one s: TrainService | t in s.service
        all t: Taxi | one s: TaxiService | t in s.service
}

                    // USER

// Represents a user in the system
abstract sig User {}

// A user registered in the system
sig Registered extends User {
        schedule: one Schedule, // Personal schedule containing the events
        dailyPath: one DailyPath, // Personal list of paths
        settings: one Settings, // Personal settings
        tickets: some Ticket, // Bought tickets
        balance: one Balance // Personal balance
}

// This signature represents the balance of the user without specifying it
sig Balance {}

// A user who doesn't own an account
sig Guest extends User {}

        // ...facts

// The considered users are inside the system
fact usersInSystem {
        all u: User | u in System.users
}

// A "balance" belongs to a user
fact balanceOfUser {
        all b: Balance | one u: Registered | b in u.balance
}

                    // DAY TIME
                    // A discrete representation of the day
```

```
// Represent a discrete time of the day
abstract sig DayTime {}

// Times of the day considered
one sig Morning, Lunch, Afternoon, Evening, Night extends DayTime {}

                    // DISTANCE

// Represents a generic distance
abstract sig Distance {}

// Kinds of distance
one sig Short, Medium, Long extends Distance {}

                    // USER SETTINGS

// The personal settings
sig Settings {
        refuses: some Transport, // defines the means of transport a user doesn't
want to use
        constraints: some TimeConstraint, // defines some day times a user
doesn't want to use a kind of transport
        lunchConstraint: one Bool, // defines if the user wants to establisha   lunch
break
        walkConstraint: lone Bool, // defines if the user wants to walk only for
short paths
        bikeConstraint: lone Bool // defines if the user wants to use the bike only
for short paths
}

        // ...facts

// Any setting belongs to one user
fact uniqueSettings {
        all s: Settings | one u: User | s in u.settings
}

// A user can't define a constraint on biking or walking if he refuses them
fact walkAndBikeConstraints {
```

```
	all s: Settings | Walking in s.refuses => #s.walkConstraint = 0 and
BikeSharing in s.refuses => #s.bikeConstraint = 0
}

		// TIME CONSTRAINTS
		// Defines the day times the user wants to avoid a kind of
transport

// Constraint used in personal settings (see Settings)
sig TimeConstraint {
	transport: one Transport,
	time: some DayTime // denied day times
}

	// ...facts

// Every constraint belongs to some settings
fact constraintsForSettings {
	all c: TimeConstraint | one s: Settings | c in s.constraints
}

// A user can't define a constraint on mean of transports he refused
fact noConstraintsOnRefuses {
	all s: Settings | all t: Transport | t in s.refuses => t not in
s.constraints.transport
}

// A user can't define multiple contraints on the same mean of transport
fact noDifferentConstraintsOnTheSameTransport {
	all c1, c2: TimeConstraint | all t: Transport | all s: Settings | (c1 != c2
and c1 in s.constraints and c2 in s.constraints and t in c1.transport) => t not in
c2.transport
}

		// SCHEDULE AND DAILY PATH

// Gathers all the events belonging to a user
sig Schedule {
	events: set Event
```

```
}

// Gathers all the paths suggested for the events in a schedule
sig DailyPath {
	paths: set Path
}

	// ...facts

// Daily paths belong to one user
fact DailyPathForUsers {
	all d: DailyPath | one u: Registered | d in u.dailyPath
}

// Schedules belong to one user
fact ScheduleForUsers {
	all s: Schedule | one u: Registered | s in u.schedule
}

			// EVENT

// An event defined by a user in the system
sig Event {
	path: lone Path, // The main path suggested
	alternative: lone Path, // The alternative path suggested
	time: one DayTime, // The time of the day the event occurs
	distance: one Distance // The distance form the previous event or the
expected user's location
}

	// ...facts

// Every event belongs to a schedule
fact eventsAssociation {
	all e: Event | one s: Schedule | e in s.events
}

// If the user set a lunch break, deny any event at that time and deny events at
long distances in the afternoon
```

```
fact eventsAtLunch {
	all u: Registered | all e: Event | (u.settings.lunchConstraint = True and e in u.schedule.events) => (Lunch not in e.time and (Long in e.distance => Afternoon not in e.time))
}

		// PATH

// A path suggested to go and take part to an event
sig Path {
	transport: one Transport, // The mean of transport
	accepted: one Bool, // Accepted by the user
	inTime: one Bool // Defines if the user will be able to be in time following this path
}

	// ...facts

// A path belongs to one event, and it can't be the main path and the alternative at the same time
fact pathUnicity {
	all p: Path | one e: Event | p in e.path or p in e.alternative
	all p: Path | all e: Event | all u: Registered | ((p in e.path or p in e.alternative) and e in u.schedule.events) => p in u.dailyPath.paths
	all p: Path | all e: Event | (p in e.path => p not in e.alternative) and (p in e.alternative => p not in e.path)
}

// An alternative is suggested only if another path was suggested before
fact alternativesAfterPaths {
	all e: Event | #e.path = 0 => #e.alternative = 0
}

// An alternative path must consider a different mean of transport from the main path's one
fact alternativesSuggestDifferentTransports {
	all e: Event | all t: Transport | t in e.path.transport => t not in e.alternative.transport
}
```

```
// Paths can't contain a refused mean of transport
fact dontSuggestRefusedTransports {
	all u: Registered | all t: Transport | all e: Event | all p: Path | (e in
u.schedule.events and (p in e.path or p in e.alternative) and t in
u.settings.refuses) => t not in p.transport
}

// A path must be by train or car if the distance is "Long"
fact trainOrCarForLongDistances {
	all e: Event | all p: Path | ((p in e.path or p in e.alternative) and Long in
e.distance) => (Train in p.transport or Car in p.transport)
}

// If the user set a walk constrain, consider "Walking" only for short distances
fact walkConstraint {
	all p: Path | all e: Event | all u: Registered | (e in u.schedule.events and
(p in e.path or p in e.alternative) and u.settings.walkConstraint = True and
Walking in p.transport) => Short in e.distance
}

// If the user set a bike constrain, consider "Bike" only for short distances
fact bikeConstraint {
	all p: Path | all e: Event | all u: Registered | (e in u.schedule.events and
(p in e.path or p in e.alternative) and u.settings.bikeConstraint = True and
BikeSharing in p.transport) => Short in e.distance
}

// Don't violate user's time contraints in the suggested paths
fact timeConstraint {
	all u: Registered | all e: Event | all p: Path | all c: TimeConstraint | all d:
DayTime | all t: Transport | (t in c.transport and d in c.time and c in
u.settings.constraints
		and e in u.schedule.events and (p in e.path or p in e.alternative)
and d in e.time) =>
			t not in p.transport
}

// An alternative is suggested only if the main path is refused or if the user is late
```

```
fact suggestAlternativesIfLateOrRefused {
	all p: Path | all e: Event | (p in e.path and p.accepted = True and
p.inTime = True) => #e.alternative = 0
}

		// TICKETS

// Represent a ticket bought by a user
abstract sig Ticket {}

// Public transport ticket (bus, undergrount...)
sig PublicTransportTicket extends Ticket {}

// Represent a generic train ticket
abstract sig TrainTicket extends Ticket {}

// The kinds of train tickets
sig SingleTrainTicket, DailyTrainTicket, MonthlyTrainTicket extends TrainTicket
{}

	// ...facts

// Every ticket in the system belongs to one user
fact ticketsForUsers {
	all t: Ticket | one u: Registered | t in u.tickets
}

// A ticket can be bought only if there's an external actor providing it
fact availableTickets {
	all t: TrainTicket | one s: TrainService | t in s.issued
	all t: PublicTransportTicket | one s: PublicTransports | t in s.issued
}

// A user who bought a monthly ticket or a daily ticket can't buy other train
tickets of the same type
fact oneTrainTicket {
	all u: Registered | all d: DailyTrainTicket | all t: TrainTicket | t!=d and d
in u.tickets => t not in u.tickets
	all u: Registered | all m: MonthlyTrainTicket | all t: TrainTicket | t!=m and
```

```
m in u.tickets => t not in u.tickets
}

                    // COMMANDS AND PREDICATES

// There isn't any event in the system
pred noEvents {
        #Event = 0
}

// There are events in the system
pred eventsExisting {
        #Event > 2
}

// There isn't any registered user in the system
pred noRegisteredUsers {
        #Registered = 0
}

// There isn't any external acotr in the system
pred noExternalActors {
        #PublicTransports = 0 and #BikeSharingService = 0 and
#CarSharingService = 0 and #TrainService = 0 and #TaxiService = 0
}

// The system suggests acceptable alternative paths for some events
pred alternativesExisting {
        some e: Event | #e.alternative = 1
        all p: Path | all e: Event | p in e.alternative => p.accepted = True and
p.inTime = True
}

// Some users bought some tickets
pred ticketsExisting {
        #TrainTicket > 0 and #PublicTransportTicket > 0
        #Actor = 2
        #Registered = 1
        #Path > 0
```

```
}

//      run noEvents
//      run eventsExisting
//      run noRegisteredUsers
//      run noExternalActors
//      run alternativesExisting
//      run ticketsExisting
        run {}
```