```
open util/time
open util/integer

sig Duration{
         start: one Time,
         end: one Time
}{
    gt[end,start]
}

sig Position{}

sig  User{
         username: one String,
         position: Position lone -> lone Time
}

sig Visitor{}

sig QRcode{}

sig QRreader{
          scanned: QRcode lone-> lone Time,
          acceptableQRcodes:  QRcode -> Time
}

sig QueueBoard{
    show:  Int lone -> lone Time
}

sig TicketPrinter{
         toPrint: PhysicalTicket lone -> lone Time
}

abstract sig Ticket{
         store: one Store,
         qrcode: one QRcode,
         number: one Int,
         permanenceTime: one Duration,
         checkInTime: lone Time,
         lineUpTime: one Time,
         ticketStatus: TicketStatus lone -> lone Time
}{
         number > 0
    gt[checkInTime, lineUpTime]
    #ticketStatus > 0 and #ticketStatus < 5
}

abstract sig TicketStatus{}
    one sig Released extends TicketStatus{}
    one sig Current extends TicketStatus{}
    one sig CheckedIn extends TicketStatus{}
    one sig Expired extends TicketStatus{}

sig InstantTicket extends Ticket{
         user: one User,
         waitingTime: one Duration,
         travelTime: one Duration
} {
    gte[travelTime.start,waitingTime.start]
    gte[travelTime.end,waitingTime.end] <=> travelTime.start = waitingTime.start
    waitingTime.start = lineUpTime
}

sig BookedTicket extends Ticket{
         reservation: one Time,
         user: one User
}{
    gt[reservation, lineUpTime]
}

sig PhysicalTicket extends Ticket{
         visitor: one Visitor,
         waitingTime: one Duration
}{
    waitingTime.start = lineUpTime
    gt[permanenceTime.start,waitingTime.end]
}

sig Store{
          name: one String,
          address: one String,
          defaultPermanenceTime: set Duration,
          maximumPeopleCapacity: one Int,
          qrReader: some QRreader,
          queueBoard: some QueueBoard,
          ticketPrinter: some TicketPrinter
}{
          maximumPeopleCapacity > 0
    #defaultPermanenceTime>2
}




fact defaultPermanenceTimeSameLenghtInAStore{
    all s:Store |
        (all pt: s.defaultPermanenceTime| pt.end = pt.start.next)
        or
        (all pt: s.defaultPermanenceTime| pt.end = pt.start.next.next)
        or
        (all pt: s.defaultPermanenceTime| pt.end = pt.start.next.next.next)
}

fact usernameUnique{
    no disjoint u1,u2: User | u1.username=u2.username
}

fact storeNameAddressUnique{
    no disjoint s1,s2: Store | s1.name=s2.name and s1.address=s2.address
}

fact QRcodeIsUnique{
    no disjoint t1,t2: Ticket | t1.qrcode=t2.qrcode
}

fact oneInstantTicketPerUser{
    no disjoint t1,t2: InstantTicket | t1.checkInTime=none and  t2.checkInTime=none
    and t1.user=t2.user
}

fact oneBookedTicketPerUser{
    no disjoint t1,t2: BookedTicket | t1.checkInTime=none and  t2.checkInTime=none
    and t1.user=t2.user
}

fact noQRcodeWithoutATicket{
    all qc: QRcode| one tck: Ticket | qc=tck.qrcode
}

fact noVisitorWithoutTicket{
    all v: Visitor | some pt:PhysicalTicket| pt.visitor=v
}

fact noPositionWithoutUser{
    all p: Position | some u: User, t:Time | u.position.t=p
}

fact ticketStateChart{
    --These first 3 lines are only to generate a world in which is possible to see
    all the possible states
    some tck: Ticket | one t': Time | tck.ticketStatus.t' = Current
    some tck: Ticket | one t': Time | tck.ticketStatus.t' = CheckedIn
    some tck: Ticket | one t': Time | tck.ticketStatus.t' = Expired
    all tck: Ticket | all t: TicketStatus.(tck.ticketStatus) |
        --Once a Ticket is "Expired"...
        (tck.ticketStatus.t = Expired =>
            ((all t':TicketStatus.(tck.ticketStatus) | t!=t' and gte[t',t] =>
            tck.ticketStatus.t' != CheckedIn and tck.ticketStatus.t' != Current)
            and
            (one t'': TicketStatus.(tck.ticketStatus) | tck.ticketStatus.t'' =
            Current and t''=t.prev.prev.prev)))
        and
        --Once a Ticket is "Current"...
        (tck.ticketStatus.t = Current =>
            ((all t': TicketStatus.(tck.ticketStatus) | t!=t' and gte[t',t] =>
            tck.ticketStatus.t' != Released)
            and
            (one t'': TicketStatus.(tck.ticketStatus) | tck.ticketStatus.t'' =
            Released)))
        and
        --Once a Ticket is "CheckedIn"...
        (tck.ticketStatus.t = CheckedIn =>
            ((all t': TicketStatus.(tck.ticketStatus) | t!=t' and gte[t',t] =>
            tck.ticketStatus.t' != Current and tck.ticketStatus.t' != Released)
            and
            (one t'': TicketStatus.(tck.ticketStatus) | tck.ticketStatus.t'' =
            Current)))
}

fact lineUpTimeIsReleasedStatusTime{
    all tck: Ticket|
        tck.lineUpTime = Released.(tck.ticketStatus)
}

fact checkInTimeIsCheckedInStatusTime{
    all tck: Ticket|
        tck.checkInTime = CheckedIn.(tck.ticketStatus)
}

fact noDevicewithoutStore{
    (all qr:QRreader | one s:Store | qr in s.qrReader)
    and
    (all qb:QueueBoard | one s:Store | qb in s.queueBoard)
    and
    (all tp:TicketPrinter |  one s:Store | tp in s.ticketPrinter)
}

fact permancenceTimeIsDefaultPermanenceTimeForVisitor{
    all pt: PhysicalTicket | one d: Duration|
        pt.permanenceTime = d and (d in pt.store.defaultPermanenceTime)
}

fact oneCurrentTicketForStoreAtTime{
    no disjoint tck1,tck2: Ticket|one t:Time | tck1.ticketStatus.t=Current and
    tck2.ticketStatus.t=Current and tck1.store=tck2.store
}

fact oneUserPositionAtLiningUpInstantTicketTime{
    all tck:InstantTicket | one p:Position|
        tck.lineUpTime in p.(tck.user.position)
}

fact ticketPermanenceTimeCorrespondencesIfCheckedIn{
    all tck: Ticket |
        CheckedIn.(tck.ticketStatus)!=none =>
            tck.checkInTime = tck.permanenceTime.start
}

fact ticketScannedBeforeCheckedIn{
    all tck: Ticket |
        CheckedIn.(tck.ticketStatus) != none =>
            (one qr: QRreader | (qr in tck.store.qrReader) and
            (tck.qrcode)->(CheckedIn.(tck.ticketStatus).prev) in qr.scanned)
}

fact ticketWaitingTimeTravelTimeCorrsepondences{
    all tck: PhysicalTicket |
        Current.(tck.ticketStatus)!=none =>
            (tck.waitingTime.end=Current.(tck.ticketStatus))
        else
            (gt[tck.waitingTime.end, Released.(tck.ticketStatus)])
    all tck: InstantTicket |
            Current.(tck.ticketStatus)!=none =>
                (gte[tck.waitingTime.end, tck.travelTime.end] =>
                    (tck.waitingTime.end=Current.(tck.ticketStatus))
                else
                    (tck.travelTime.end=Current.(tck.ticketStatus)))
            else
                (gt[tck.waitingTime.end, Released.(tck.ticketStatus)] and
                gt[tck.travelTime.end, Released.(tck.ticketStatus)])
}

fact bookedTicketBecomeCurrent{
    all tck: BookedTicket|
        tck.reservation = Current.(tck.ticketStatus)
```

```
}

fact physicalTicketStoreEqualTicketPrinterStore{
    all tp: TicketPrinter, tck: tp.toPrint.Time | tp in tck.store.ticketPrinter
}

fact queueBoardShowTheCurrentTicketNumberOfStore{
    (all tck: Ticket, t:Time |
        tck.ticketStatus.t = Current => (all qb: tck.store.queueBoard| qb.show.t =
        tck.number))
    and
    (all t:Time, s: Store | (all tck: store.s| tck.ticketStatus.t != Current) <=>
    s.queueBoard.show.t  = none)
}

fact oneTicketPrinterPrintPhysicalTicketAtReleasingTime{
    (all tck: PhysicalTicket|
        one tp: TicketPrinter|
            (tp in tck.store.ticketPrinter) and (tck->(Released.(tck.ticketStatus))
            in tp.toPrint) and (all tp':TicketPrinter| tp!=tp' => tck.(tp'.toPrint)
            = none))
    (all tp: TicketPrinter |
        all t: PhysicalTicket.(tp.toPrint) | tp.toPrint.t.ticketStatus.t = Released)
}

fact whenQRcodeAreAcceptableQRcodes{
    all tck: Ticket, t:Time | all qr: QRreader|
        (gte[t,Current.(tck.ticketStatus)] and (qr in tck.store.qrReader) and
        (CheckedIn.(tck.ticketStatus)!= none => gt[CheckedIn.(tck.ticketStatus),t])
        and (Expired.(tck.ticketStatus)!=none => gt[Expired.(tck.ticketStatus),t]))
        <=> (tck.qrcode->t in qr.acceptableQRcodes)
}
---------------------------------------------Pred
pred hasUserAValidInstantTicket[u:User,t:Time]{
    one tck: InstantTicket | tck.user=u and tck.ticketStatus.t != Expired
}

pred hasUserAValidBookedTicket[u:User,t:Time]{
    one tck: BookedTicket | tck.user=u and tck.ticketStatus.t != Expired
}

pred hasVisitorAValidPhysicalTicket[v:Visitor,t:Time]{
    one tck: PhysicalTicket | tck.visitor=v and tck.ticketStatus.t != Expired
}

pred hasUserAValidInstantTicketForTheStore[u:User,t:Time,s:Store]{
    one tck: InstantTicket| tck.user=u and tck.ticketStatus.t != Expired and
    tck.store=s
}

pred hasUserAValidBookedTicketForTheStore[u:User,t:Time,s:Store]{
    one tck: BookedTicket| tck.user=u and tck.ticketStatus.t != Expired and
    tck.store=s
}

pred userMakeAInstantTicketReservation[u:User,s:Store,t:Time,tck: InstantTicket]{
    //preconditions
    not hasUserAValidInstantTicket[u,t]
    not hasUserAValidBookedTicketForTheStore[u,t,s]
    //postconditions
    tck.user=u
    tck.store=s
    hasUserAValidInstantTicket[u,t.next]
    not hasUserAValidBookedTicketForTheStore[u,t.next,s]
    tck.ticketStatus.(t.next) = Released
}

pred userMakeABookedTicketReservation[u:User,s:Store,t:Time,tck: BookedTicket]{
    //preconditions
    not hasUserAValidBookedTicket[u,t]
    not hasUserAValidInstantTicketForTheStore[u,t,s]
    //postconditions
    tck.user=u
    tck.store=s
    hasUserAValidBookedTicket[u,t.next]
    not hasUserAValidInstantTicketForTheStore[u,t.next,s]
    tck.ticketStatus.(t.next) = Released
}

pred
visitorMakeAPhysicalTicketReservation[v:Visitor,s:Store,t:Time,tck:PhysicalTicket]{
    //postconditions
    tck.visitor=v
    tck.store=s
    tck.ticketStatus.(t.next) = Released
}

pred userChekInWithInstantTicket[u:User,s:Store,t:Time,tck:InstantTicket]{
    //preconditions
    hasUserAValidInstantTicketForTheStore[u,t,s]
    not hasUserAValidBookedTicketForTheStore[u,t,s]
    tck.store=s
    tck.user=u
    gt[t,Current.(tck.ticketStatus)]
    //postconditions
    CheckedIn.(tck.ticketStatus)=t.next
}

pred userChekInWithBookedTicket[u:User,s:Store,t:Time,tck:BookedTicket]{
    //preconditions
    hasUserAValidBookedTicketForTheStore[u,t,s]
    not hasUserAValidInstantTicketForTheStore[u,t,s]
    tck.store=s
    tck.user=u
    gt[t,Current.(tck.ticketStatus)]
    //postconditions
    CheckedIn.(tck.ticketStatus)=t.next
}

pred visitorChekInWithPhysicalTicket[v:Visitor,s:Store,t:Time,tck:PhysicalTicket]{
    //preconditions
    hasVisitorAValidPhysicalTicket[v,t]
    tck.store=s
    tck.visitor=v
    gt[t,Current.(tck.ticketStatus)]
    //postconditions
    CheckedIn.(tck.ticketStatus)=t.next
}
```

```

pred show{
    #BookedTicket>0
    #InstantTicket>0
    #PhysicalTicket>0
}

run show for 6 but 10 int, exactly 2 String
run userMakeAInstantTicketReservation for 6 but 10 int, exactly 2 String
run userMakeABookedTicketReservation for 6 but 10 int, exactly 2 String
run visitorMakeAPhysicalTicketReservation for 6 but 10 int, exactly 2 String
run userChekInWithInstantTicket for 6 but 10 int, exactly 2 String
run userChekInWithBookedTicket for 6 but 10 int, exactly 2 String
run visitorChekInWithPhysicalTicket for 6 but 10 int, exactly 2 String

```