4.
4.1 Signatures
This is the sig for every object that we described in class diags WITH SOME CONTRAINTS

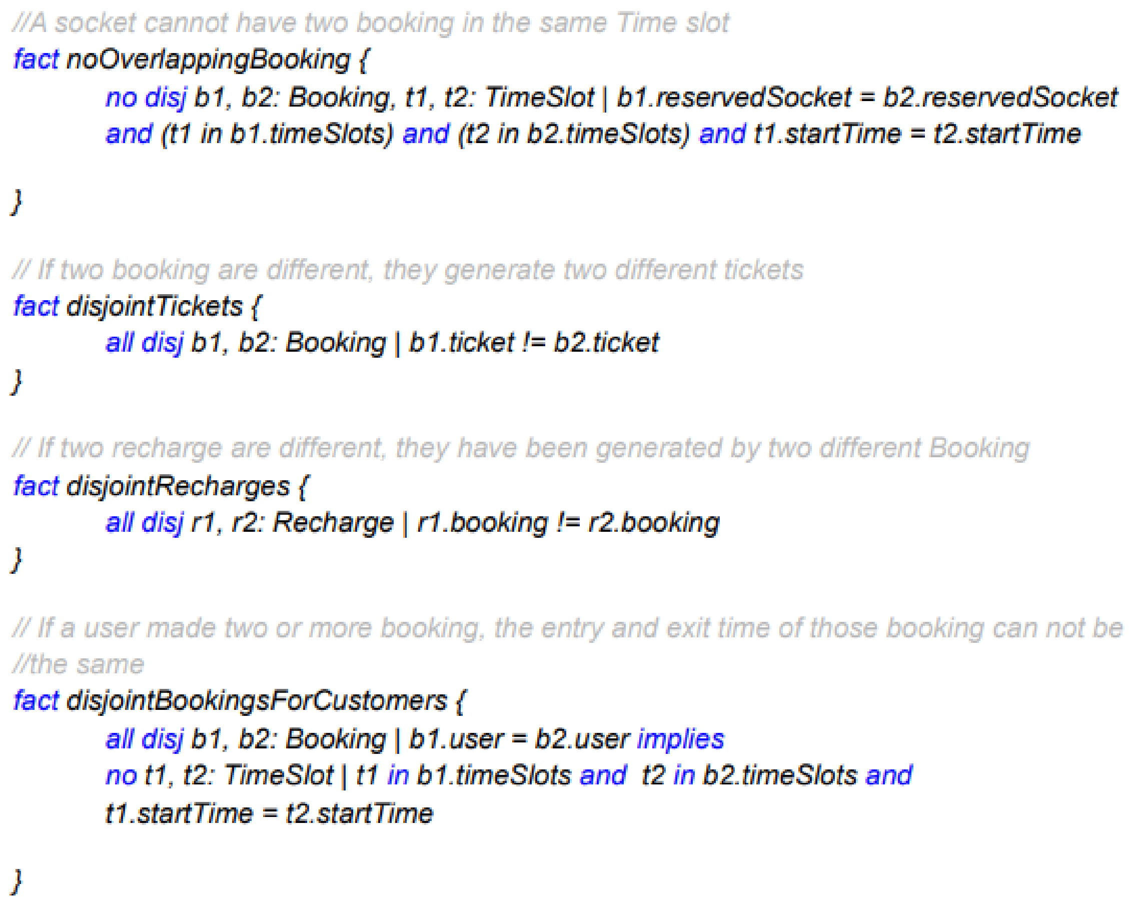
```
// Basic User Signature
abstract sig User {
   id: one String,
   name: one String,
   surname: one String,
   email: one String,
   phoneNumber: one String
}

// Differentiating between Educator and Student
sig Educator extends User {
   department: one String,
   createdTournaments: set Tournament,
   createdBattles: set Battle,
   createdBadges: set Badge
}

sig Student extends User {
   grade: one Int,
   memberOf: set Group
}

// Group of Students
sig Group {
   id: one String,
   size: one Int,
   members: some Student,
   score: one Int,
   joinedTournaments: set Tournament
}

// Badge and Requisite
sig Badge {
   id: one String,
   name: one String,
   description: one String,
   requisites: set Requisite,
   obtainedBy: set Student
}

sig Requisite {
   id: one String,
   description: one String,
   achievedBy: set Student
```

```
}

// Tournament and Battle
sig Tournament {
   id: one String,
   startDate: one Date,
   endDate: one Date,
   participatingGroups: set Group,
   battles: set Battle
}

sig Battle {
   id: one String,
   startDate: one Date,
   endDate: one Date,
   participatingGroups: set Group
}

// Submission for Battles
sig Submission {
   id: one String,
   githubLink: one String,
   group: one Group,
   battle: one Battle,
   date: one Date
}

// Date Signature for scheduling
sig Date {
   year: Int,
   month: Int,
   day: Int
}

// Constraints
fact {
   // Educators and Students are distinct
   no Educator & Student

   // Each group has members and their number should match the group's size
   all g: Group | #g.members = g.size

   // A battle belongs to only one tournament
   all b: Battle | one t: Tournament | b in t.battles

   // Submission's group must be part of the battle's participating groups
   all s: Submission | s.group in s.battle.participatingGroups
```

```
    // Badges are awarded to students who have met all requisites
    all b: Badge, s: Student | s in b.obtainedBy iff all r: b.requisites | s in r.achievedBy
}

// Date consistency
fact {
    all d: Date | d.year > 0 and d.month > 0 and d.month <= 12 and d.day > 0 and d.day <= 31
}
```

4.2 Facts

In the following are stated in Alloy the facts that must hold for the domain modeled in order to maintain consistency with the real world



```
//A socket cannot have two booking in the same Time slot
fact noOverlappingBooking {
        no disj b1, b2: Booking, t1, t2: TimeSlot | b1.reservedSocket = b2.reservedSocket
        and (t1 in b1.timeSlots) and (t2 in b2.timeSlots) and t1.startTime = t2.startTime

}

// If two booking are different, they generate two different tickets
fact disjointTickets {
        all disj b1, b2: Booking | b1.ticket != b2.ticket
}

// If two recharge are different, they have been generated by two different Booking
fact disjointRecharges {
        all disj r1, r2: Recharge | r1.booking != r2.booking
}

// If a user made two or more booking, the entry and exit time of those booking can not be
//the same
fact disjointBookingsForCustomers {
        all disj b1, b2: Booking | b1.user = b2.user implies
        no t1, t2: TimeSlot | t1 in b1.timeSlots and  t2 in b2.timeSlots and
        t1.startTime = t2.startTime

}
```

Alloy facts for the CodeKataBattle (CKB) Platform

```
// A student cannot be part of two different groups at the same time
fact noOverlappingGroupMembership {
  all s: Student | lone g: Group | s in g.members
}
```

```
// An educator cannot create two tournaments with the same name
fact uniqueTournamentNames {
  all disj t1, t2: Tournament | t1.name != t2.name
}

// A group cannot join two battles in the same tournament at the same time
fact noOverlappingBattlesInTournament {
  all g: Group, t: Tournament |
    let battlesInTournament = g.joinedTournaments & t.battles |
    all disj b1, b2: battlesInTournament |
      !(b1.startDate < b2.endDate and b2.startDate < b1.endDate)
}

// A group cannot submit more than one solution for a battle
fact uniqueSubmissionPerGroupPerBattle {
  all b: Battle, g: Group | lone s: Submission | s.battle = b and s.group = g
}

// A badge can only be obtained by a student if all requisites are met
fact badgeRequisitesMet {
  all b: Badge | all s: Student |
    s in b.obtainedBy iff (all r: b.requisites | s in r.achievedBy)
}

// A tournament cannot have battles that overlap in time
fact noOverlappingBattles {
  all disj b1, b2: Battle |
    !(b1.startDate < b2.endDate and b2.startDate < b1.endDate)
}

// A student can only be awarded a badge once
fact uniqueBadgePerStudent {
  all s: Student | all disj b1, b2: Badge |
    not (s in b1.obtainedBy and s in b2.obtainedBy and b1 = b2)
}

// Educators cannot score the same group twice for the same battle
fact uniqueScoringPerGroupPerBattle {
  all e: Educator, g: Group, b: Battle |
    lone score: Int | (g, b, score) in e.scores
}

// A student's grade can only be modified by an educator
fact gradeModificationByEducator {
  all s: Student | all g: s.grade | some e: Educator | e.modifiedGrades[g] = s
}
```

In these facts:

- We assume that each student belongs to at most one group at a time.
- Tournaments created by educators must have unique names to avoid confusion.
- Groups are restricted from joining overlapping battles within the same tournament.
- A group is limited to one submission per battle to prevent duplicate entries.
- Badges are awarded to students only when all associated requisites are fulfilled.
- Battles within the scope of the platform are scheduled to prevent time conflicts.
- Badges are uniquely awarded to each student, preventing duplication of achievements.
- Educators provide a unique score for each group in a battle.
- Only educators can modify a student's grade, ensuring proper academic administration.

Paste images of the diagrams generated by the tool.