Assignment 3 - Solution

Exercise 1

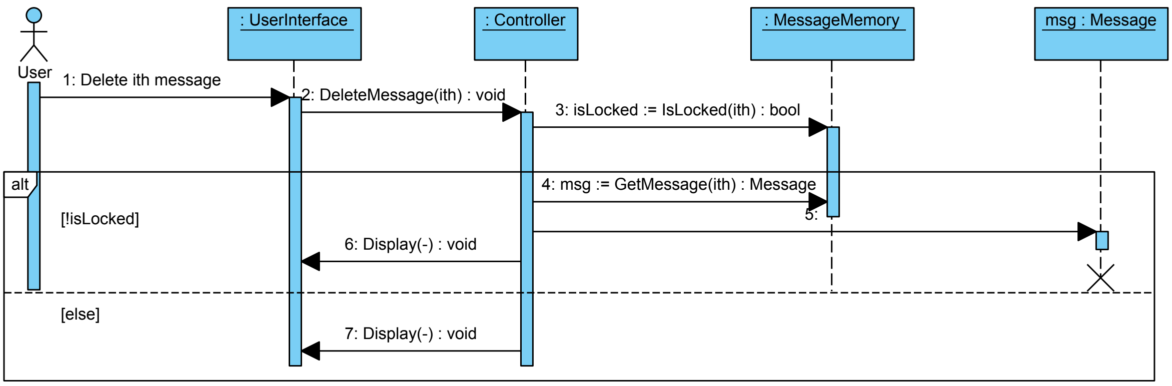
1. The corresponding code for the method `PlayMessage` is given below:

```
public void PlayMessage(int index){
  boolean isSufficient = battery.isSufficientlyHigh();
  if (isSufficient) {
    Message msg = messageMemory.GetMessage(index);

    //it is not specified how to iterate over the blocks in msg
    //we just assume that the method GetAudioBlock returns null
    //when the index is out of range

    AudioBlock block;
    int blockIndex = 0;
    while ( (block = msg.GetAudioBlock(blockIndex++)) != null) {
      speaker.PlayAudioBlock(block);
    }
    userInterface.Display("message played");
  } else {
    userInterface.Display("battery low");
  }
}
```

2. Both use cases can be represented in the following sequence diagram:



Exercise 2

```
open util/boolean

abstract sig Student{
  id: one ID,
  major: one Major,
  university: lone University,
  isLegal: Bool,
  classmates: set Student,
}

sig Graduate extends Student{}
sig Undergraduate extends Student{}
sig ID{}
sig Major{}
sig University{}

fact unique_ids {
  all disj s, t: Student | s.id != t.id
}

fact no_id_without_student {
  all i: ID | one s: Student | s.id = i
}

fact legal_in_university {
  all s: Student | (s.university != none) iff (s.isLegal = True)
}

fact classmates_have_same_major_and_uni {
  all disj s, t: Student | (s.major = t.major and s.university = t.university and
  (s in Undergraduate and t in Undergraduate
   or s in Graduate and t in Graduate)) iff (s in t.classmates)
}

fact no_self_classmate {
  all s: Student | s not in s.classmates
}

pred show {}

run show for 2 University, 3 Major, 3 Student, 3 ID
```

Exercise 3

Properties of binary relations File: properties_sol.als

```
Properties of Binary Relations
(solution by Martin Ouimet)

1. The non-empty property can be removed.
   Relation satisfying the remaining properties:
        univ = {}
        r = {}

2. The transitive property can be removed.
   Relation satisfying the remaining properties:
        univ = {a,b}
        r = {(a,b), (b,a)}

3. The irreflexive property can be removed.
   Relation satisfying the remaining properties:
        univ = {a,b}
        r = {(a,a),(b,b)}

No other individual property can be removed such that the remaining properties
are satisfied.
```

Refactoring navigation expressions File: distribution_sol.als

```
Distributivity of Join
(solution by Michael Craig)

Proof for part a) was not necessary.

a) Distributivity of join over union holds: for a set s and relations p and q,
   suppose (A0) is in s.(p+q). Then for some AX,  (AX) is in s and (AX,A0) is
   in p or q, or both. Then (A0) is in  either s.p or s.q, or both, so it is
   certainly in s.p + s.q.

   Now assume (A0) is in s.p + s.q. Then for some (AX) in s, (AX,A0) is in
   either p or q. Thus (AX,A0) is certainly in p+q, so that s.(p+q) must
   contain (A0).

   In other words, for any atom A0, A0 is in s.(p+q) iff it is in s.p + s.q.

b) Distributivity of join over difference does not hold: consider the set s =
   {(A0,A1)} and the relations p = {(A0,A1)} and q = {(A1,A1)}. Then s.(p-q) =
   s.({(A0,A1)}) = {(A1)}, but s.p - s.q = {(A1)} - {(A1)} = {}.

c) Distributivity of join over intersection does not hold: consider the set s =
   {(A0),(A1)} and the relations p = {(A0,A0)} and q = {(A1,A0)}. Then s.(p&q)
   = s.({}) = {}, but s.p & s.q = {(A0)} & {(A0)} = {(A0)}.
```

Doris Day's song File: everybody_sol.als

```
/*
A song by Doris Day goes:
Everybody loves my baby but my baby don't love nobody but me
David Gries has pointed out that, from a strictly logical point of view,
this implies 'I am my baby'.
Check this, by formalizing the song as some constraints,
and Gries's inference as an assertion.
Then modify the constraints to express what Doris Day probably meant,
and show that the assertion now has a counterexample.
*/

sig Person {
        loves: set Person
}

one sig Me extends Person {}

pred my_baby[b: Person] {
        (all p: Person | b in p.loves) and b.loves = Me
}

assert song {
        all p: Person| my_baby[p] implies Me = p
}

//run my_baby for 5

check song for 5
```

Barber paradox File: barber_sol.als

```
/* (a) Use the analyzer to show that the model is indeed inconsistent,
 * at least for villages of small sizes.
 */
/*
sig Man {shaves: set Man}
one sig Barber extends Man {}
*/

/* (b) Some feminists have noted that the paradox disappears if the existence
 *     of women is acknowledged. Make a new version of the model that
 *     classifies villagers into men (who need to be shaved) and women (who
 *     don't), and show that there is now a solution.
 */
/*
abstract sig Person {shaves: set Man}
sig Man, Woman extends Person{}
one sig Barber in Person {} // must be 'in' not 'extends':
*/

/* (c) A more drastic solution, noted by Edsger Dijkstra, is to allow the
 *     possibility of there being no barber. Modify the original model
 *     accordingly, and show that there is now a solution.
 */
/*
sig Man {shaves: set Man}
lone sig Barber extends Man {}
*/

/* (d) Finally, make a variant of the original model that allows for multiple
 *     barbers. Show that there is again a solution.
 */
/*
sig Man {shaves: set Man}
some sig Barber extends Man {}
*/

fact {
  Barber.shaves = {m: Man | m not in m.shaves}
}

run { }
```

Modelling the Tube File: tube_sol.als

```
sig Station {}

sig JubileeStation in Station {
  jubilee: set JubileeStation
}

sig CentralStation in Station {
  central: set CentralStation
}

sig CircleStation in Station {
  circle: set CircleStation
}

one sig Stanmore, BakerStreet, Epping extends Station {}


fact {
  // write the corresponding constraint below each statement

  // a) named stations are on exactly the lines as shown in graphic
  Stanmore in (JubileeStation - CentralStation) - CircleStation
  BakerStreet in (JubileeStation & CircleStation) - CentralStation
  Epping in (CentralStation - JubileeStation) - CircleStation

  // b) no station (including those unnamed) is on all three lines
  no (JubileeStation & CentralStation & CircleStation)

  // c) the Circle line forms a circle
  all s: CircleStation {
    one s.circle
    CircleStation in s.^circle
  }

  // d) Jubilee is a straight line starting at Stanmore
  JubileeStation in Stanmore.*jubilee
  all s: JubileeStation {
    lone s.jubilee
    s not in s.^jubilee
  }

  // e) there's a station between Stanmore and BakerStreet
```

```
  let reach = ^jubilee | some Stanmore.reach & reach.BakerStreet

  // f) it is possible to travel from BakerStreet to Epping
  Epping in BakerStreet.^(jubilee + central + circle)
}

pred show {}
run show for 6
```