Consultation restricted to the "Alloy quick reference". Duration: 50 minutes.

Student name:____**Number**____________

1. **[11.2 points]** For each question, mark with a cross (X) the correct answer.
Each **correct answer** is graded **1.6 points**. Each **incorrect answer** is graded **-0.4 points**.

a) Pixels always have three components: red, green and blue. A possible translation to Alloy would be:

- ☐ `sig Pixel { }     sig Red, Green, Blue extends Pixel { }`
- ☐ `enum Component { Red, Green, Blue }     sig Pixel { components: Component -> Int }`
- ☐ `sig Pixel { Red: Int, Green: Int, Blue: Int }`
- ☐ `All the previous answers are correct`

b) In an University, a School has several Departments, but a Department belongs to just one school. How to translate that in Alloy?

- ☐ `sig University { schools: some School, deps: Department some -> one schools }`
- ☐ `sig University { schools: some School, deps: schools -> Department }`
- ☐ `sig University { schools: some School, deps: schools some -> one Department }`
- ☐ `sig University { schools: some School, deps: schools 1..* -> 1 Department }`

c) What is the converse (~R) of the binary relation `R = {(a,b),(b,c),(c,b)}`?

- ☐ `~R = {(a,b),(b,a),(b,c),(c,b)}`
- ☐ `~R = {(a,b),(b,c),(c,b),(a,a),(b,b),(c,c)}`
- ☐ `~R = {(b,a),(c,b),(b,c)}`
- ☐ `~R = {(b,c),(c,b)}`

d) Given `R1={(a,a),(a,b),(b,c)}` and `R2={(a),( c)}` what is the value of the restriction `R1 :> R2`?

- ☐ `R1 :> R2 = {(a,a)}`
- ☐ `R1 :> R2 = {(a,a,a),(a,a,c)}`
- ☐ `R1 :> R2 = {(a,a),(b,c)}`
- ☐ `R1 :> R2 = {(a,a),(a,b)}`

e) Given `R1={(a,b),(b,b)}`, `R2={(a,a)}` and `R3={(b,a)}` what is the value of `(R1 ++ R2) + R3`?

- ☐ `{(a,b),(b,b),(a,a),(b,a)}`
- ☐ `{(a,a),(b,b),(b,a)}`
- ☐ `{(a,a),(b,a)}`
- ☐ `None of the previous answers is correct`

f) Consider a graph definition where each node has a set of adjacent nodes: `sig Node { adjacent: set Node }`. A graph is connected if there is a path from every node to any other node. How to express that constraint in Alloy?

- ☐ `fact connected { iden in ^adjacent }`
- ☐ `fact connected { all disj n1, n2: Node | n2 in n1.^adjacent }`
- ☐ `fact connected { all n1: Node | n1 in n1.*adjacent }`
- ☐ `All the previous answers are correct`

g) Given `sig Exam{grades: Student->lone Int}`, how can we obtain the pairs (exam, student) that received a certain grade?

- ☐ `fun results[g: Int]: Exam->Student { g <: grades }`
- ☐ `fun results[g: Int]: Exam->Student { grades <: g }`
- ☐ `fun results[g: Int]: Exam->Student { grades :> g }`
- ☐ `None of the previous answers are correct`

2. [**8.8 points**] Fill in the empty blocks.

```
sig Account {}

abstract sig Transaction { amount: Int }

0.2 sig Deposit, Withdrawal  extends Transaction   -- A transaction is either a deposit
                                                      or a withdrawal

sig Client {
0.4  accounts:  some  Account, -- a client can access several accounts (1 or more)
0.4  withdrawPrivileges:  set  Account, -- but can't withdraw from all of them (0..*)
0.6  balance: Account  set  ->  lone  Int -- the amount each account currently has
0.6  transactions: Account  one  ->  set  Transaction -- a list of all account movements
}

pred invariants[c: Client] {
  -- the balance of an account should never be lower than 0
0.6    all n : Account.balance | n >= 0
  -- a client can only withdraw from accounts she has access to
0.6    withdrawPriviliges in accounts
  -- a client only has balance from accounts she has access to
0.6    balance.Int in accounts
}


-- transaction t withdraws quantity q from account a of client c,
-- resulting in a new state c'
0.2 pred withdraw[c, c': Client, a: Account, qty: Int, t:  Transaction  ] {
   -- pre-conditions (without using predicate invariants)
1.4      (TODO)
   -- post-conditions (without using predicate invariants)
1.6    (TODO)
}

-- gives the total balance of a client c
fun totalBalance[c: Client] : Int {
0.8    sum a: c.accounts | c.balance[a]
}

assert withdraw_preserves_invariants {
  all c, c': Client, account: Account, qty: Int, t: Transaction |
    -- if one withdraws from a consistent client
0.4    (invariants[c]  and  withdraw[c,c',a, qy, t]  ) =>
    -- one ends up with a new consistent client state
0.4      (  invariants[c']  )
}

check withdraw_preserves_invariants
```

Good luck!