Programming Practice (PRP), Coursework Exercise 4 (Part A), Mark Scheme (Full Version with Notes)

1 Overview

1. The main focus of this assignment was to test the coordination of a large number of classes in order to solve a problem.
2. **This mark scheme only relates to Problem A (28 marks). The mark scheme from Problem B will be released in due course.**
3. The maximum mark that a student can achieve if their code does not compile is 16 / 40 (40%).
4. Students who provide their solutions procedurally should receive a maximum mark of 16 / 40 (40%). Procedural solutions do not use objects at all, solving everything in the main method; create objects but do not use them; or, in the context of this assignment, create objects only to store information *without* calling any methods on those objects to solve the task.
5. Students who do not include the access point address and the access point password they were asked to receive a maximum mark of 27 / 40 (67%).

2 Mark Scheme

For 0 - 16 marks: 0% - 40%, a passing grade

1. Correctly decomposing the problem into a set of classes relevant to the problem (**4 marks total**).

- The only requirement, at this low level, is to have *at least* four classes: `NetworkDevice` (**1 mark**), `Channel` (**1 mark**), `Packet` (**1 mark**) and `Network` (potentially containing main) (**1 mark**). Although there are references made to `Client` and `AccessPoint` at this point in the mark scheme, actual marks for this will be awarded later.

2. Using these classes to both store and provide access to all information relevant to the problem. This information should be of an appropriate type (**8 marks total**).

   - Required fields:
     - `NetworkDevice` (these fields may be replicated in subclasses `Client` and `AccessPoint`. Do not penalise this for now.)
       * Address, of type String or a special object (**1 mark**)
       * Channel (the current Channel in use), of type Channel (You can be lenient with this mark, if the student has achieved the same thing in another reasonable way) (**1 mark**)
       * Key, of type String **or** integer (**1 mark**)
     - `Channel`
       * Number, of type integer (**1 mark**)
       * Traffic, of type `List<Packet>` (**1 mark**)
     - `Packet`
       * Source Address, of type String or a special object (**1 mark**)
       * Destination Address, of type String or a special object (**1 mark**)
     - `Network`
       * A **map** linking `NetworkDevice`s to `Channel`s. (**1 mark**).
       * *Separate maps for `Clients` and `AccessPoints` are fine. There may also be a list of `Channels`, but there are no additional marks available for this, as this information can be derived from the map. Moreover, there may be a list of `Clients` and `AccessPoints` separately. Again, because this information can be derived from the map, no additional marks are available for this.*

3. Correctly encapsulating all information (**2 marks total**).

   - Fields are either **private** or **protected**. If they are public, they are both final and static. (**1 mark**).
   - All information is set to private fields using methods (could be via a constructor, but this is not essential for these marks). If fields are protected, they *must* only be set from inside the class hierarchy, not from outside (i.e. students should not write <`Object`>.<`Protected Field`> anywhere in their code) (**1 mark**).

- **Important:** There should be **no** way to retrieve the `Channel` field directly from a `NetworkDevice` (e.g. no 'getChannel' method). **Deduct one mark if this is not the case**.

4. Taking the appropriate steps to ensure that all information that is required by each class is always present (e.g. every `NetworkDevice` has an address) (**2 marks total**).

   - A constructor inside `NetworkDevice` that ensures the presence of `address` (other parameters in the constructor are fine) (**1 mark**).
   - A constructor inside `Packet` that ensures the presence of `sourceAddress` and `destinationAddress` (other parameters in the constructor are fine) (**1 mark**).

For 16 - 20 marks 40% - 50%

All of the above, and:

1. Maximising efficiency through abstraction (i.e. collecting the common features of all `NetworkDevice`s) (**2 marks total**).

   (a) Two subclasses of `NetworkDevice`: `Client` and `AccessPoint`. A subclass of `Packet`: `HandshakePacket`. (**1 mark**).

   (b) A field inside `Client` listing the `AccessPoint` to which it is currently connected. A field inside `AccessPoint` listing `authorisedClients` (as either client objects or String addresses). A `key` field inside `HandshakePacket` (**1 mark**).

2. Setting up a `Network` and its associated `NetworkDevice`s (without handshaking) (**2 marks total**).

   (a) Creating a `Client` and an `AccessPoint` object in preparation for adding the `AccessPoint` to a `Network` and in preparation for handshaking the `Client` with the `AccessPoint`. (**1 mark**).

   (b) A method inside `Network` that allows an `AccessPoint` to be added to a `Network`, achieved with the following (**1 mark total**):

      i. Finding a channel upon which an `AccessPoint` should communicate, and joining the `AccessPoint` to that channel.

      ii. Recording the relationship between the `AccessPoint` and the `Channel` it has just been joined to in the map storing a relationship between `NetworkDevice`s and `Channel`s.

For 20 - 24 marks 50% - 60%

All of the above, and implementing each stage of the handshake:

1. Assuming one `Client` that handshakes with one `AccessPoint` **and** students have added a `key` field to their `NetworkDevice` class (**4 marks total**):

   (a) Joining the `Client` to the `AccessPoint`'s `Channel`. Adding a `HandshakePacket` to the `Channel` now shared by both devices. The source address of the `HandshakePacket` should be the `Client` and the destination address should be the `AccessPoint`. The `HandshakePacket` should contain the `Client`'s `key`. The best solutions will do all of this inside `Client` (**1 mark**).

   (b) Reading through all the `Packet`s in the `Channel`, determining if

      i. The destination address of the packet matches the `AccessPoint`.
      ii. The type of the packet is `HandshakePacket` (`instanceof`).
      iii. The `key` in the packet matches the `key` stored by the `AccessPoint`

      Storing the `Client` who sent the `HandshakePacket` as an authorised client inside `AccessPoint` (this may also happen once the handshake is successful). Adding a `HandshakePacket` to the `Channel` where the source address is the `AccessPoint` and the destination address is the `Client`. The packet should contain the `AccessPoint`'s `key`. The best solutions will do all of this inside `AccessPoint` (**1 mark**).

   (c) Reading through all the `Packet`s in the `Channel`, determining if

      i. The destination address of the packet matches the `Client`.
      ii. The type of the packet is `HandshakePacket` (`instanceof`).
      iii. The `key` in the packet matches the `key` stored by the `Client`.

      Storing the `AccessPoint` as the `Client`'s current access point (this may also happen once the handshake is successful). The best solutions will do all of this inside `Client` (**1 mark**).

   (d) After the above has occurred we consider the handshake complete. After a complete handshake the following should happen (**1 mark total**):

      i. If the handshake is successful, the relationship between the `Client` and the `Channel` is recorded in the map storing a relationship between `NetworkDevice`s and `Channel`s.
      ii. The `AccessPoint` lists the `Client` as authorised (if it hasn't already) and the `Client` lists the `AccessPoint` as its current connection (if it hasn't already).

iii. If the handshake is unsuccessful, the `Channel` field inside `Client` is set to `null`.

(**1 mark**)

For 24 - 28 marks 60% - 70%

All of the above, and creating network activity (**4 marks total**):

1. Creating client communication activity, by placing a normal `Packet` into the `Channel`, where the source address is the `Client` and the destination address is the `AccessPoint`. The best solutions will do this inside `Client` (**1 mark**).

2. Creating access point communication by reading through all the `Packet`s in a `Channel` and responding the those where the destination address is the `AccessPoint`. An `AccessPoint` should **only** respond to `Packet`s from authorised `Client`s in order to get this mark. The best solutions will do this inside `AccessPoint` (**1 mark**).

3. Creating network activity, which involves all `Client`s engaging in communication activity, *then* all `AccessPoint`s engaging in communication activity. This obviously needs to occur in this order for the communication to work. This activity should happen over a number of iterations (where the number of iterations is not important). Using `Thread.sleep()` is good, but omitting it should not lose a mark (**1 mark**).

4. Clearing all channels before each burst of network activity by going through each `Channel` stored by the `Network`, and access the list of `traffic` held inside the `Channel` in order to clear it (**1 mark**).