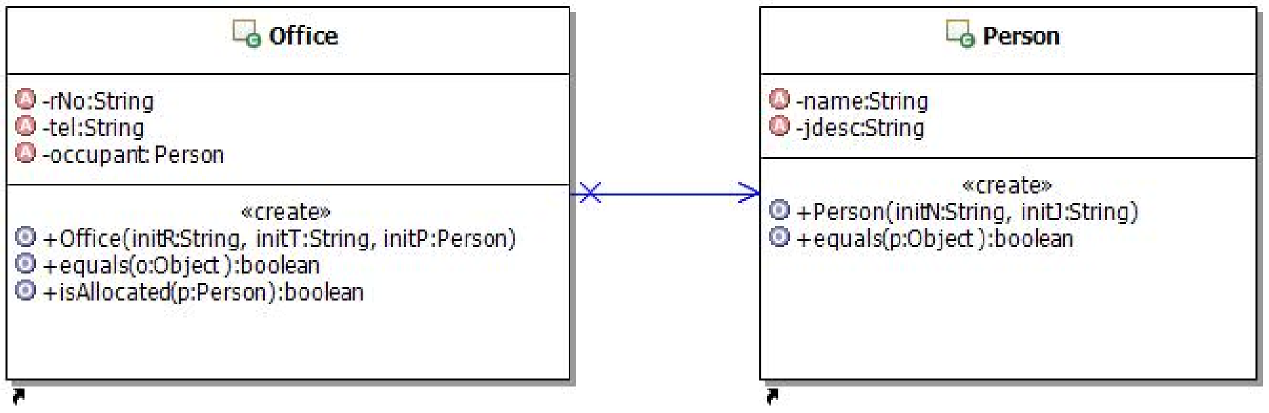
University of Stirling
Computing Science and Mathematics

CSCU9P5 Software Engineering I **Autumn 2018**

Tutorial 2 Sample Answers

Q 1. The class diagram could be (including some later parts):



Office
-rNo:String
-tel:String
-occupant: Person
«create»
+Office(initR:String, initT:String, initP:Person)
+equals(o:Object):boolean
+isAllocated(p:Person):boolean
Person
-name:String
-jdesc:String
«create»
+Person(initN:String, initJ:String)
+equals(p:Object):boolean

Q 2. Constructors: There is no fixed/correct answer. For Office: We might consider that no constructor is required/wanted – in which case some "set" methods will **definitely be required** in order to set the attribute values. Alternatively, we might choose a constructor with a parameter for every attribute (as above and below). Or perhaps we might choose just to initialise the number, because in some natural sense that is all that is associated with creating and office – it has no occupant no telephone number at that point! A similar discussion could be had about a constructor for Person.

Q 3. It is always worth considering whether "get" and/or "set" methods are required for each attribute: The decision depends on whether we anticipate, at this state of the development, needing them later (e.g. we might anticipate needing to put a different person into an office, but do we anticipate changing an Office's room number or telephone number?). Also useful to consider whether "equals" methods would be useful (overriding the Object superclass) – in this problem we are told that we need them. And also useful to consider whether "toStrings" would be useful – not asked for here, but they could be added.

Q 4, Q 6. Java code corresponding to this ***could*** be: (but other variants are equally valid!)

```
public class Office {
    private String rNo;
    private String tel;
    private Person occupant;

    public Office(String initR, String initT, Person initP) {
        rNo = initR; tel = initT;
        occupant = initP;
    } // constructor

    public boolean equals(Object o) {
        if (o != null && rNo != null)
            return rNo.equals(((Office)o).rNo);
        else
            return false;
    } // equals

    public boolean isAllocated(Person p) {    // FOR Q7
        if (occupant != null)
            return occupant.equals(p);
        else
            return false;
    } //isAllocated
} // Office

public class Person {
    private String name;
    private String jdesc;

    public Person(String initN, String initJ) {
        name = initN;
        jdesc = initJ;
    } // Person

    public boolean equals(Object p) {
        if (p != null)
            return name.equals(((Person)p).name);
        else
            return false;
    } // equals
} // Person
```

Q 5. The experience of trying this exercise is supposed to suggest, Yes, it is easier to go via the class diagram.

Q 6. See above.

Q 7. isAllocated: See code above

Q 8. No, a Person does not know to which office they have been allocated. To implement this: You would need to add an `Office` attribute to class `Person` and methods to allocate an office and determine the current office.

Q 9. Several occupants: The association between Office and Person would have, say, 0..* as multiplicity at the Person end (with optional, implicit, 1 at the Office end). The implementation of Office would need a list of Persons (eg an array) as an attribute rather than just one Person, and isAllocated would need to search the list for the given Person. Note: that this also represents a decision (possibly only implicitly made) that there is only one telephone number for the whole office – if all occupants have their own number then more changes are required...