Exercise 1:

A) What would be a very good example of an abstraction?

iii. Numbers because we use them abstractly (1, 2, 3, etc) without necessarily being concerned with the concrete objects they may be attached to (1 apple, 2 chairs, 3 TV sets).

B) What are model invariants?

iv. Constraints that must be satisfied in all states of the model.

C) What is OO inheritance?

ii. There are two sides to inheritance. On the one hand, there is abstraction specialisation (or is-a): an object of a subclass is also an object of its ancestor classes. On the other hand there is incremental definition (or extension or inheritance proper) as subclasses inherit the static and behavioural characteristics of their ancestor classes.

D) What is an abstract class?

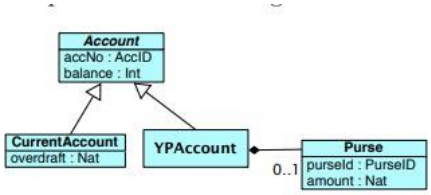
i. A class without direct instances that is used to capture generalisations that lack a direct representation in an inheritance hierarchy (for example, a class for primates as a generalisation of humans and other apes such as gorillas).

E) What are whole-part (or part-of) relations?

iii. It is a way of saying that some class (the whole) is made up of other constituent classes (the parts)

F) Consider the fragments of UML CDs given below, which describe the account products of a new bank that intends to target young people. There are normal current accounts and young-person's (YP) accounts; current accounts have an overdraft, unlike YP accounts. Unlike current accounts, YP accounts have an optional purse card that can be used to buy good in shops affiliated with universities throughout the country; purses are associated with one account only. When a YP account holder is 26 years old or when he ceases to be a student, a normal current account is created from the YP account, the balance is transferred, and the YP account and the associated purse is deleted (the amount remaining in the purse is lost). The purse's identifier allows the amount to be recovered and transferred to another purse in case the purse is lost or stolen. Which UML CD fragment complies with the requirements narrative given above?

Answer- i)



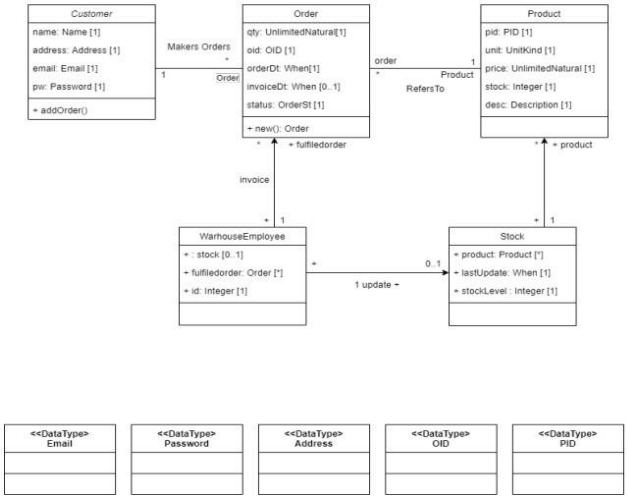
(i)

G) Please justify your answer to the previous question.

The correct UML class diagram fragment is **(i)** because it accurately models all aspects of the requirements described in the narrative. It clearly distinguishes between CurrentAccount and YPAccount as separate subclasses of a general Account class, correctly reflecting that only current accounts have an overdraft feature, while YP accounts do not. Additionally, it models the Purse as a separate class with its own purseId and amount, which is essential for supporting the recovery mechanism in case of loss or theft. The optional one-to-one association (0..1) between YPAccount and Purse correctly represents that a YP account may or may not have a purse and that each purse is linked to exactly one YP account. This setup also supports the scenario where, upon the account holder aging out or ceasing to be a student, the YP account is deleted, the balance is transferred to a new current account, and the purse is removed—while still allowing recovery via the purseId. Other diagrams either incorrectly imply that YP accounts have overdrafts, fail to represent the purse as a separate recoverable entity, or allow invalid associations.

Exercise 2:

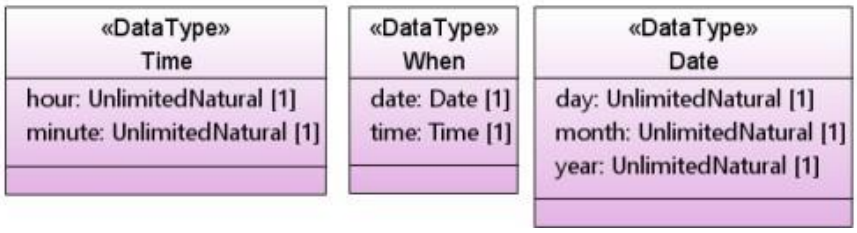
Exercise 2. Consider the following UML class diagram for the OrderOnStock system discussed in Week 4. Define the relevant invariants for the attributes `qty` and `price` of classes `Order` and *Product*, respectively.



The quantity (qty) in an order should always be more than zero because you can't really order nothing or a negative amount. For the product's price, it also has to be greater than zero since products can't be free or have a negative price. These rules keep the system's data sensible and make sure orders and invoices are handled properly.

Exercise 3:

Exercise 3. Consider the data types When, Date and Time given in the UML class diagram (CD) below, a fragment of the model of SimpleClinic. Import the class diagram of SimpleClinic into Draw.io (available on Moodle under week 4) and define the required invariants for these data types.



Time:
The Time data type makes sure that the hour is between 0 and 23, representing a full day in 24-hour format, and the minute is between 0 and 59, covering all valid minutes in an hour. This keeps the time values realistic and consistent.

Date:
The Date data type ensures the month is a number between 1 and 12, and the year is positive. The day must fit the specific month and year, taking into account months with 30 or 31 days and also adjusting for leap years, when February has 29 days instead of 28.

When:
The When data type combines both Date and Time, so it requires that the date and time values are both valid. This means the date must be a real calendar date, and the time must follow the rules for hours and minutes described above.

Exercise 4:

Exercise 4. Modify SimpleClinic's CD shown in Fig. 1, which you have imported into Draw.io in Exercise 3, to meet the revised requirements given in Table 1. Please consider the following: (a) both nurses and doctors can see a patient (R6), (b) the surgery's walk-in service (R9) and (c) the visits are registered in the reception by a nurse (R9).

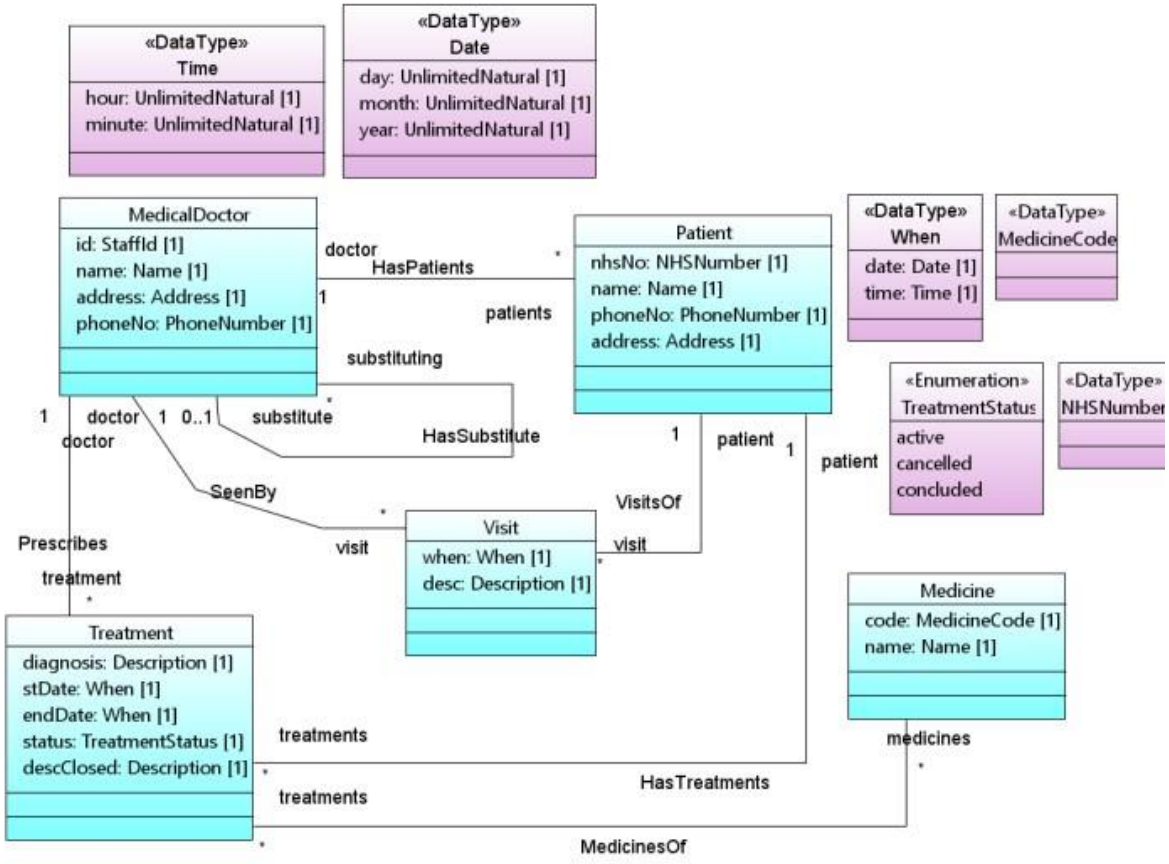


Figure 1: UML CD of SimpleClinic (trivial datatypes are omitted)

Requirements:

R1 SimpleClinic is a medical information system that supports small surgeries.

R2 SimpleClinic records the assignment of medical doctors —or general practitioners (GPs) — to patients and the GP's optional substitute doctor.

R3 The system shall record the following data of GPs: staff id, medical licence number, name, address, email and contact telephone number.

R4 The system shall record the following data of nurses: staff id, name, address, email and contact telephone number.

R5 The system shall record the following data of patients: NHS number, name, address, email and contact telephone number.

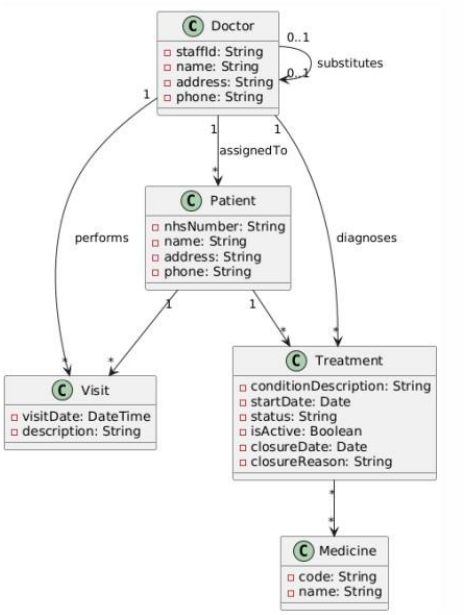
R6 The system shall keep a record of the visits of patients to the clinic with the following information: the nurse or GP who has seen the patient, when the visit has taken place, and a small description of what the visited consisted of.

R7 The system shall keep track of treatments that are administered to patients; a treatment includes a description of the diagnosed medical condition, the doctor that made the diagnostic and started the treatment, the date when the treatment started, the prescribed medicines, an information on whether the treatment is active, cancelled or concluded, a description on why the treatment has been closed (cancelled or concluded) and the date when it has been closed. Only GPs may start treatments.

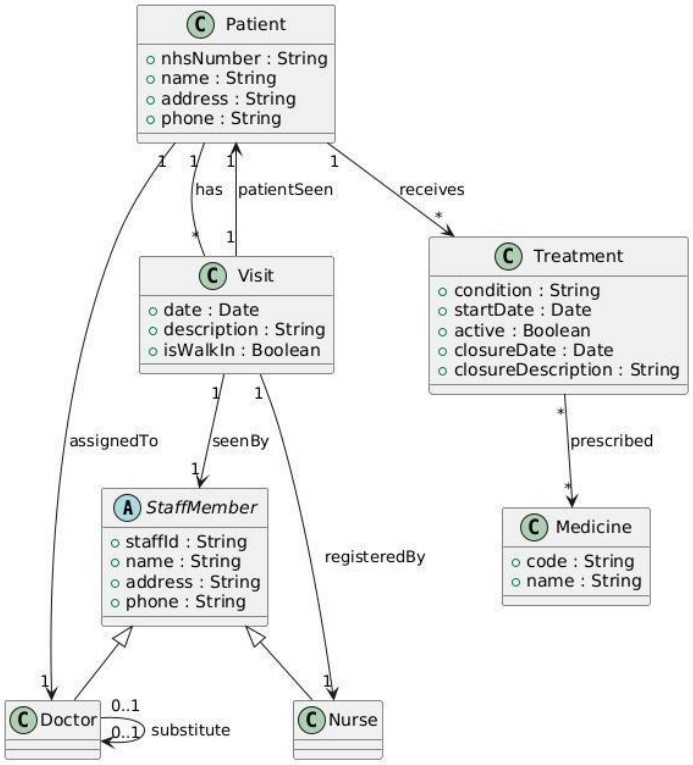
R8 The system shall keep a list of medicines that can be prescribed by GPs; each medicine has a code and a name.

R9 The system shall provide support for the surgery's walk-in morning service. Visits are registered by a reception nurse to be seen by a doctor or nurse on a first-come, first-served basis on the morning they come to the surgery.

Before:



After:



Exercise 5:

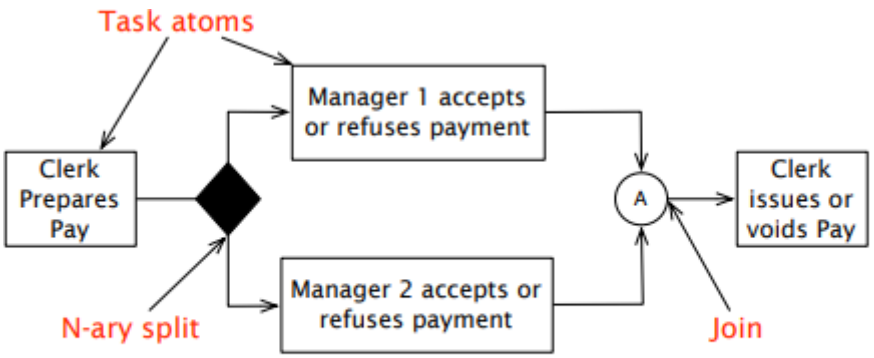


Figure 2: An example workflow

Exercise 5. A software company considers building a workflow system that aims to streamline business processes of institutions. You have been asked to build an initial domain model of the system they have in mind. The requirements narrative is as follows:

A workflow has a name and description, and is made out of task instructions, which are to be executed in sequence, and conditions, which represent the global state of the workflow (an example workflow is given in Fig. 2). A condition has a name and a boolean evaluation indicating whether it is true or false at any given time during the execution of a workflow instance. There are the following task instructions: (i) a task atom, which has a name and a description and a list of conditions; (ii) a n-ary split to introduce branching so that tasks are executed simultaneously; (iii) a join, which expresses the convergence of parallel branches; and (iv) a XOR-slit, which breaks the workflow into two alternative branches, depending on the evaluation of some condition expression. Condition expressions combine atomic conditions (the condition above) using the binary boolean operators and and or, and the unary boolean operator not.

Answer-