COMP1202 Coursework 2014/15

Key Information

Deadline: **Wednesday 10th December 2014 at 17.00** via https://handin.ecs.soton.ac.uk

Lecturers: David Millard, Mark Weal, Rikki Prince

Value: 30% of the marks available for this module.

This coursework is to be completed individually.

Please note: Any alterations to the coursework and answers to frequently asked questions will be submitted to the course wiki:
https://secure.ecs.soton.ac.uk/student/wiki/w/COMP1202/Coursework

Coursework is the continuously assessed part of the examination, and is a required part of the degree assessment. Assessed work must be submitted as specified below. You are reminded to read all of the following instructions carefully.

Coursework Aims

This coursework allows you to demonstrate that you:

- Understand how to construct simple classes and implement methods.
- Are able to take simple pseudo-code descriptions and construct working Java code from them.
- Can write a working program to perform a complex task.
- Have an understanding of object oriented programming.
- Can correctly use polymorphism and I/O.
- Can write code that it is understandable and conforms to good coding practice.

Contacts

General programming queries related to the coursework should be made to your demonstrator in the timetabled labs.

Queries about the coursework specification should be made to Mark Weal (mjw@ecs.soton.ac.uk).

Any issues that may affect your ability to complete the coursework should be made known to Mark Weal (mjw@ecs.soton.ac.uk) or David Millard (dem@ecs.soton.ac.uk), ideally before the submission deadline.

ECS Hospital

Specification

The aim of this coursework is to construct a simple simulation of a hospital. This hospital will admit patients with particular illnesses and they will be treated by a team of specialist doctors and surgeons to make them better.

You are not required to have any knowledge of hospitals or medicine in order to complete this coursework and no medical accuracy is claimed in the representation of illnesses and medical practices presented here.

The hospital may bear some superficial similarities to a real hospital but is grossly simplified and in most cases likely to be quite different to how a real hospital might work.

Your task is to implement the specification as written.

No marks will be awarded for deviating from the specification in order to increase the realism of the hospital in fact it may well cost you marks and make it much harder to use the test harnesses provided.

How ECS Hospital works

For this coursework you are expected to follow the specification of the hospital, doctors and patients as set out below. This will not correspond exactly to a real hospital or doctors in reality but we have chosen particular aspects for you to model **that help you to demonstrate your Java programming**.

There are a number of people, buildings and procedures that contribute to this simulation. For our purposes these include:

Patients: Patients will arrive at the hospital with a particular illness. This illness needs to be treated by the doctors. Once it has been treated and the patient has recovered they can be discharged from the hospital.

a **Hospital:** For our purposes a hospital has a ward for patients and a number of operating theatres. Patients are admitted to a ward, treated, sometimes in an operating theatre, put back in a ward to recover and then discharged from the hospital.

Doctors : The hospital will be staffed by a number of doctors. The doctors may have particular specialisms that let them treat particular illnesses. Some illnesses can only be treated by doctors with the correct specialism.

HospitalAdministrator: The hospital administrator is in charge of admitting and discharging patients from the hospital and deciding which patients get treated in the operating theatre.

The next sections will take you through the construction of the various people and buildings you require. You are recommended to follow this sequence of development as it will allow you to slowly add more functionality and complexity to your simulation. The first steps will provide more detailed information to get you started. It is important that you conform to the specification given. If methods are written in red then they are expected to appear that way for the test harness and our marking harnesses to work properly.

Part 1 – Modelling People

The first class you will need to create is an abstract class that represents a *Person*.

The *Person* class will be the basis for your *Patient* and *Doctor* classes.

The first class to create is the *Person* class. This is an abstract class that defines the basic properties and methods that all the different people classes will use. The properties that you will need to define are:

- `gender` – this will define whether the person is male or female.
- `age` – This says how old the person is in years.
- `health` – initially this can be defined as a String. Later on, we will create a health class that is used to represent the state of a person's health. To start with, the default value can be "healthy".

You can add a constructor to the *Person* class that creates a person with default values.

The abstract *Person* class will also need to define some methods for use by the *Hospital* and *Doctors*. These methods can be overridden by the other specific sub-classes. The methods you need to create are *getGender()* (returns a `char`), *setAge(`int age`), getAge(), getHealth()* and the abstract method *aDayPasses()*,which will have a return type of `boolean`. *aDayPasses()* will be called on each Person in the hospital once each day and will contain all the code that enables the person to do what they need to do and to change over time. Principally, for *Doctors* it will allow them to make people better, and for *Patients* it will allow them to recover from their illnesses.

You should now have a *Person* class. As we progress you may need to, and choose to add additional methods to your abstract class.

Part 2 – Modelling Patients

Patients are derived from the abstract class *Person*.

Initially, all you need to do is create a class *Patient* that inherits from class *Person*.

You will need to implement the aDayPasses() method, but for now you can just leave this empty.

Part 3 – Modelling the Hospital

Our *Hospital* class is where all our *Patients* are treated and our *Doctors* work. You should use an `ArrayList` to represent the beds in the *Hospital*. For the purposes of our simulation we will allow our Hospital to have no more than 50 beds in it. The *Hospital* also has a number of operating theatres, that can also be modelled using an `ArrayList` . Our hospital will have 4 theatres initially.

The *Hospital* will need a number of access methods. These should include:

- *admitPatient (`Patient`)* – this adds a new patient to a free bed in the *Hospital* if there is one available returning the bed number, or -1 if there is no free bed.
- *getPatient(`int`)* – this returns the *Patient* in bed n, or null if the bed n doesn't have a patient.
- *size()* – this returns the number of *Patients* in the *Hospital*.
- *dischargePatient(`int`)* – this removes the Patient from bed n, effectively sending them home. Usually used when they have recovered.

We also need some mechanisms for moving patients to operating theatres for their operations. We will assume that their bed is kept for them whilst they are in the operating theatre, but the two methods we will need to implement are:

- *isTheatreFree(`int`)* – this returns true if the operating theatre specified is free, or false otherwise.
- *prepForTheatre(`int`,`Patient`)* – This assigns the specified patient to the particular operating theatre.
- *takeForRecovery(`int`)* – This removes the patient from the specified operating theatre. They are now effectively back in their bed on the ward.
- *aDayPasses()* –this method is called as part of the simulation to trigger a new day in the *Hospital*.

You should now have a *Hospital* class that you can add *Patients* to.

You should now move on and create some actual *Patients* to add to our *Hospital*. You can test this using a main method in your Hospital class to create new patients and add them to

the hospital. You can even try moving some of them to an operating theatre and back again to test this works. You are advised to get all of this working before attempting to extend your functionality.

Part 4 – Modelling Health

When we modelled the health of a person initially, we used a simple String. For our hospital to function more smoothly we need to add some complexity to this and we will do this by replacing the String object with our own class called *Health*.

Health is a standard Java class and has the following properties

- `healthState` – this will be healthy, sick or recovering. You can use the values 0,1 and 2 to model this simply.
- `recoveryTime` – this will be 0 if the patient is ill or healthy, or set to a value in days if the patient is recovering. This value will be decremented each day whilst the patient is recovering until it reaches zero, at which point the patients healthState will change from recovering to healthy.
- `Illness` – this defines what illness the patient has. The illness will dictate which doctors can treat them, how long it takes for them to recover, and whether they need to be treated in one of the operating theatres. Table 1. Below shows the different types of illnesses. The ID number should be used to identify the illness.

Illness	ID number	Recovery	Can be treated by	Requires Theatre
Djkstra's syndrome (considered harmful)	1	5 days	Any Doctor	No
Java Flu	2	3 days	Any Doctor	No
Deadline Panic Attacks	3	1 day	Any Doctor	No
Polymorphic Cist	4	2-4 days	Any Surgeon	Yes
Semicolon Missing	5	5-8 days	Organ Surgeon	Yes
Trapped Exception	6	6-8 days	Organ Surgeon	Yes
Tim Berners Knee	7	4-6 days	Limb Surgeon	Yes
Coder's Elbow	8	2-3 days	Limb Surgeon	Yes

Table 1: Types of illness and relevant information

You will need to define a number of access methods for the Health class to enable these properties to be retrieved and set. At the minimum these probably should include

setHealthState(), *getHealthState()*, *setRecoveryTime()*, *getRecoveryTime()*, *setIllness()* and *getIllness(),* all taking the appropriate parameters and returning the appropriate parameters.

At this stage you should now be able to create patients with a range of illnesses and admit them to your hospital.

Part 5 – Modelling Doctors

To model the various types of *Doctors* you will need for your *Hospital* (*Doctor, Surgeon, OrganSurgeon* and *LimbSurgeon*) you will need to continue to use inheritance.

The following simple diagram Fig 1. shows you how the classes that you will create are related to each other. As you can see, both *Patient* and *Doctor* inherit from *Person*. *Surgeon* is a subclass of *Doctor*, and *LimbSurgeon* and *OrganSurgeon* are subclasses of *Surgeon*.

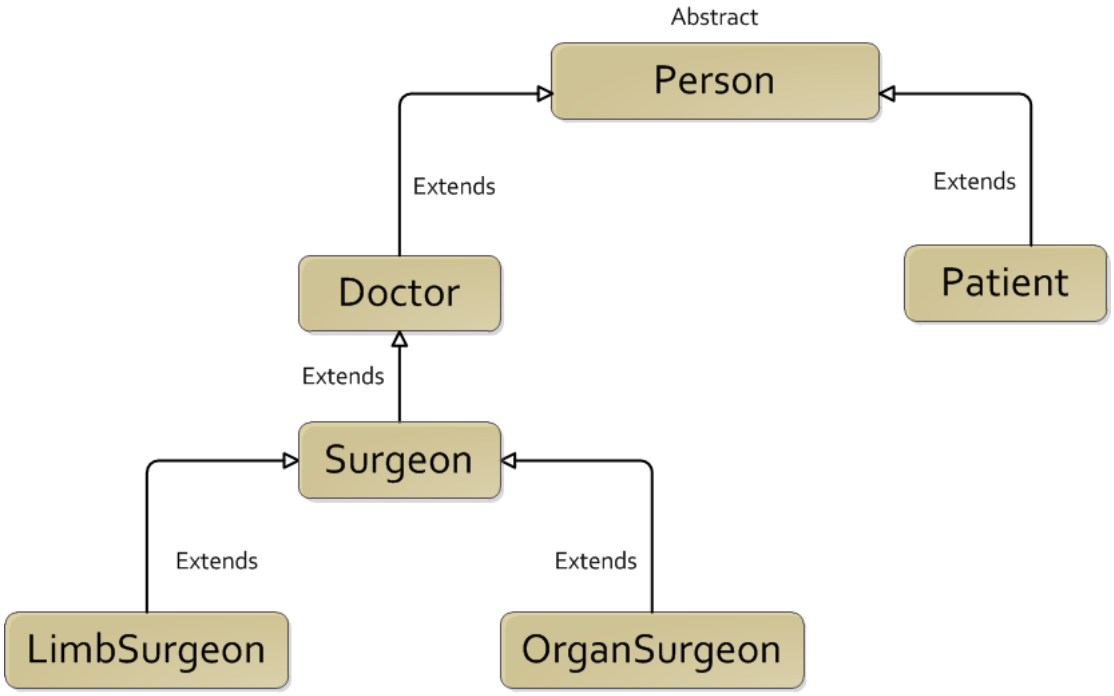

Fig 1: Class diagram for Person classes.

The first class to create is the *Doctor* class. This is a class that defines the basic properties and methods that all the different doctor classes will use. The properties that you will need to define are:

- `specialism` – the type of the Doctor (The following Type IDs are recommended. 1=Doctor, 2=Surgeon, 3=LimbSurgeon, 4=OrganSurgeon)
- `assignedPatient` – Doctors can be assigned a patient to be treated.

The *Doctor* class will also need to define some methods for use by the *Hospital*. These methods will be overridden by the other doctor sub-classes. The methods you need to create are *getSpecialism()*, *assignPatient(Patient)*, and the methods *treatPatient()* and *aDayPasses()*, both of which have a return type of `boolean`. *aDayPasses()* will be called on

each Doctor in the hospital once each day and will contain all the code that enables the doctor to carry out its various tasks.

You should now have a *Doctor* class. As we progress you may choose to add additional methods to your class.

For now, we will implement the *treatPatient()* method in the *Doctor* class. This will be overridden when you come to create the Surgeon classes.

For a Doctor, the treatPatient() method allows them to treat any patient assigned to them that has an illness of ID 1-3. If this is the case, the Health of the patient is changed to recovering and the days to recovery is set randomly within the range specified in Table 1.

Surgeon is a subclass of *Doctor*. They also have a method operate() that can be called from within their overridden method of treatPatient(). For a general surgeon, they can treat illness 4 using the operate function providing the patient is allocated to a theatre.

LimbSurgeon and *OrganSurgeon* override the operate() method of *Surgeon* to enable them to perform more specific types of operations (illnesses ID 5-8). Again, operations can only be performed if the Patient is allocated to an operating theatre.

Doctors can only successfully treat patients that they are qualified to treat. To summarise by illness ID:

- *Doctor* can treat illnesses 1,2,3
- *Surgeon* can treat illnesses 1,2,3 and 4 (4requires operating theatre)
- *OrganSurgeon* can treat 1,2,3,4,5 and 6 (4,5,6 require operating theatre)
- *LimbSurgeon* can treat 1,2,3,4,7 and 8 (4,7,8 require operating theatre)

Part 6 – Modelling the Hospital Lifecycle

In order to run our Hospital as a simulation we are going to need a *HospitalAdministrator*. This class will have a *Hospital*, a collection of *Doctors*, and will organise the day to day running of the *Hospital*, i.e. run the simulation itself.

You should create a new class called *HospitalAdministrator*. It should have a Hospital and an ArrayList of Doctors. It should have a aDayPasses() method which is used to run the Simulation.

We are going to keep our simulation simple and try and avoid any complicated scheduling problems. No additional marks will be given for having really efficient scheduling systems. You are required to have a very simple working process, it is not necessary to make this too complicated.

A typical day at the hospital will have the following events occur.

- New patients will be admitted to the hospital and allocated to an available bed.
- The hospital administrator will assign patients to doctors based on their specialisms. The simplest way to do this is to:
    - Look at each doctor and work out what their specialism is
    - Go through the patients until you find one with a health condition that can be treated by that doctor.
    - If the doctor is a surgeon assign the patient to an empty theatre.
- Tell each Doctor to treat their patients (call aDayPasses().)
- Remove all patients from theatres.
- Let the patients recover (call aDayPasses() on each patient.)
- Discharge any patients that are now healthy

The list of events above represent the algorithm for running your hospital. This should be implemented inside the aDayPasses() method of your *HosptialAdministrator*.

In your *HospitalAdministrator* go() method you can loop round the simulation by calling aDayPasses(). You can use print statements to record what is happening in your simulation. To slow things down a little, the following code will allow you to pause for half a second between days if you choose to include it.

```
try
{
      Thread.sleep(500);
}
catch (InterruptedException e)
{
}
```

Part 7 – Reading a simulation configuration file

A good simulation will allow you to set the starting conditions for your simulation and one way of doing this is for the simulation to read in a simple configuration file. For this step you will need to use your file handling methods as well as split strings into different component parts.

Our basic configuration file will look like the example below. You may choose to extend this for your extensions, but for testing purposes your code should accept and use configuration files in this form. Each line gives information about a hospital patient or a doctor.

```
hospital:100,5
patient:M,60,1,-1
patient:M,22,3,-1
patient:F,31,4,-1
patient:F,58,8,-1
patient:M,23,0,6
doctor:M,55
```

```
doctor:F,45
limbSurgeon:M,62
organSurgeon:F,48
```

The format for the *Hospital* is

Hospital:beds,theatres

for *Patients*

patient:gender,age,condition,daystorecovery

for *Doctors* is

doctor:gender,age

Some example simulation files of varying complexity will be placed on the WIKI.

You should modify the main method in your *HospitalAdministrator* class so that it can take a file on the command line. This will enable you to start your simulation by typing

java HospitalAdministrator myHospital.txt

When the *HospitalAdministrator* receives the configuration it should read the file a line at a time. For each line the *file* will need to identify the Class, create a new class of this type, and set the appropriate parameters. If creating a new *Patient* it should admit it to the *Hospital*, if creating a new *Doctor*, add it to the *HospitalAdministrator*. You may find you need to create specific methods or indeed a helper class, to parse the configuration file and extract the information that the *HospitalAdministrator* needs.

We are placing this code in the *HospitalAdministrator* to make it simple, so your simulation is of a *HospitalAdministrator* with a *Hospital* in it. You could create a new Class *Simulator* to perform this function if you wish, provided it is clear in the Readme.txt file you supply where the code can be found.

You should now have a simulator that runs with a *HospitalAdministrator* containing a *Hospital* which has multiple *Doctors and Patients* in it.

Exceptions

You should be trying to use exceptions in the construction of your simulator where possible. You should be catching appropriate I/O exceptions but also might consider the use of exceptions to correctly manage:

- The input of a configuration file that does not conform to the specified file format.
- Attempts to admit too many patients.
- A *Doctor* treating a *patient* they're not qualified to treat.
- A*Patient* needing surgery but there not being an available theatre.
- …

Extensions

You are free to extend your code beyond the basic simulator as described above. You are advised that your extensions should not alter the basic structures and methods described above as marking will be looking to see that you have met the specification as we have described it. If you are in any doubt about the alterations you are making please include your extended version in a separate directory.

Scheduling can become very complicated very quickly, so we have deliberately not made efficient scheduling a part of this coursework. Be warned if you attempt to do anything clever in this regard for your extension.

Some extensions that we would heartily recommend include:

- Try modifying the simulator so that *patients* that remain untreated for a long period of time are treated differently. You may need to add a property to patient for this, such as timeSinceAdmitted.
- Your Doctors might be able to treat more than one patient in a day. Modify your doctors and simulation to deal with this. Each theatre should only ever have one patient in it though.
- Perhaps *Patients* can suffer from more than one illness. How would you need to modify the *Patient* class to have multiple *Health* issues that need treating by perhaps different *Doctors*.
- You might want to extend the configuration file and classes so some of the parameters of the simulation can be set in the configuration file. This might include some of the details about recovery times for instance.
- You might want to allow the simulation to save out the current state of a simulation to a file so that it can be reloaded and restarted at another time.

15% of the marks are available for implementing an extension. Any of the above examples would be suitable, but feel free to come up with one of your own if you like. It is not necessary to attempt multiple extensions in order to gain these marks. Please describe your extension clearly in the readme file that you submit with your code.

Space Cadets

You might want to add a GUI to your simulator so that you can visualise the state of the *Hospital* at any given moment. **No marks are available for a GUI**, we put the suggestion forward simply for the challenge of it.

Marking

Marks are available for:

- Applications that compile and run
- Meeting the specification properly
- Successful completion of one or more extensions
- Good coding style
- Good comments in your source code

85% of the marks are available without attempting any extensions.

Submission

Submit **all Java source files** you have written for this coursework in a zip file **hospital.zip**. Do not submit class files. As a minimum this will be:

Person.java, Patient.java, Doctor.java, Surgeon.java, LimbSurgeon.java, OrganSurgeon.java, Hospital.java, HospitalAdministrator.java, Health.java.

Also include a text file **hospital.txt** that contains a brief listing of the parts of the coursework you have attempted, describes how to run your code including command line parameters, and finally a description of the extensions you have attempted if any. Submit your files by **Wednesday 10th December 2014 17:00** to: https://handin.ecs.soton.ac.uk

Originality of Work

Students' attention is drawn to Section IV of the University Calendar on Academic Integrity:

http://www.calendar.soton.ac.uk/sectionIV/part8a.html