Software and Programming II (SP2)
2018/19: Coursework Assignment Two

1 Introduction

- **Submission Deadline: 27 November 2018, 11:55pm GMT**
- Feedback Deadline: 18 December 2018

There are **three** coursework assignments for this module. The coursework assignments contribute to your overall module mark as follows:

- Assignment 1 accounts for 20% to the coursework mark (i.e., 5% of the overall module mark);
- Assignments 2 and 3 account for 40% to the coursework mark each (i.e., 10% of the overall module mark).

Each of the assignments is marked out of 100. The aims of this coursework are:

- To work with object-orientation, inheritance, abstract classes, and interfaces.
- To distribute attributes and method implementations suitably over an inheritance hierarchy.
- To practice using information hiding together with inheritance.
- To perform sanity checks on method and constructor parameters and to throw exceptions to indicate unsuitable parameter values to the user of your code.
- To give you further practice with Git and GitHub.

The code for this coursework is made available to you via the following GitHub Classroom invitation link:

`https://classroom.github.com/a/cQw9D7wV`

The assignment is further explained in Section 2 below. Section 3 of this document explains the marking scheme. Section 4 presents the deadlines and submission instructions. Section 5 explains the penalties for late submissions, and Section 6 how the College deals with plagiarism. Section 7 provides additional information on learning resources.

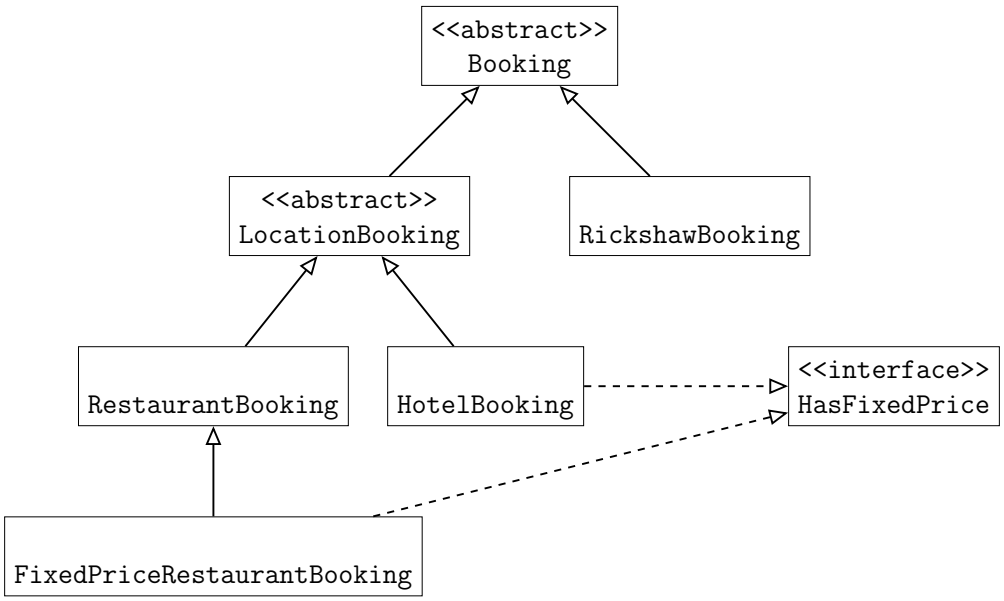

Figure 1: The inheritance hierarchy for our object-oriented data structure in this coursework

2 Description of the work

2.1 Bookings

In this coursework we are going to develop an extensible object-oriented data structure for managing bookings for hotels, restaurants, and rickshaw taxis. Here part of the code, including an inheritance hierarchy, is already given. You can access the Git repository with the initial files on GitHub via the invitation link in Section 1, similar to Coursework Assignment 1.

Your task will be to complete the classes so that they implement the desired functionalities using suitable object-oriented concepts and information hiding. This includes adding attributes, constructors, and (abstract or concrete) methods to the provided classes.

Figure 1 shows a graphical sketch of the inheritance hierarchy between the classes and interfaces. A box with `<<interface>>` stands for an interface, a box with `<<abstract>>` stands for an abstract class, and a box with neither of the two entries stands for a concrete class. The arrows stand for the "extends" or "implements" relation. For example, the abstract class `LocationBooking` extends the abstract class `Booking`, and the concrete class `HotelBooking` implements the interface `HasFixedPrice`.

Moreover, we provide a class `Coursework2Main`, which you can use to test your code.

2.2 Given Classes and Interfaces

This subsection provides some background on the classes and interfaces that are given and should not be modified (that is, the code that you write in the other classes should work with the *original* classes from this subsection).

2.2.1 `public abstract class Booking`

The abstract class `Booking` provides the basis of our inheritance hierarchy. Every object of class `Booking` (and hence its subclasses) stores information about the name for which the

booking was made and the point in time for which the booking was made. We can query a booking for this information. Moreover, we can query a booking for the number of persons for whom the booking was made.

The class `Booking` provides you with the following constructors and methods.

Constructor:

`public Booking(String name, Date date)`
Constructs a new booking with a name and a date according to the parameters:

- `name` – name used for the booking; must not be null
- `date` – the point in time for the booking; must not be null and must not be in the past

Methods:

`public String getName()` Returns the name of the booking.

`public Date getDate()` Returns the point in time for which the booking was made.

`public abstract int getNumberOfPersons()` Returns the number of persons for whom the booking has been made (at least 0). *(The method is abstract, so you will still need to implement it in suitable subclasses.)*

2.2.2 `public interface HasFixedPrice`

Classes implementing this interface provide methods that returns a fixed price as an `int` in pence or as a formatted string. This interface also provides convenience functionality to generate a string representation from an amount of money given in pence.

Methods:

`int getFixedPriceInPence()` Returns a fixed price in pence for this object.

`static String computeFormattedPrice(int priceInPence)` Provides a string representation of an amount in pence.

- `priceInPence` – a price in pence

`default String getFormattedPrice()` Returns a formatted string representation of the fixed price of this object.

2.2.3 `public class Coursework2Main`

We are providing the file `Coursework2Main.java` in the repository. This class makes use of some of the desired functionalities of our data structure in its `main` method. You can (and should) test your implementation by running `main`. This provides further clarification for the desired behaviour of the objects. It is a requirement that your implementation compiles with the *unmodified* `Coursework2Main.java` (i.e., all the methods and constructors used by `Coursework2Main.java` should also be present). The file `Coursework2Main.java`

contains a comment at the end with the output that its `main` method produces with our implementation.

Note, however, that the tests performed by `Coursework2Main` are not meant to be exhaustive — so even if `Coursework2Main` has the desired outputs, this does not automatically mean that your implementation is necessarily correct for all purposes. Thus, it is a good idea to not only test your code (also with further test cases), but also to review it before handing in your solution.

2.3 Classes that You Will Need to Modify

This subsection is about the classes that you will need to modify in this coursework.

We consider concrete classes (`RickshawBooking`, `HotelBooking`, `RestaurantBooking`, and `FixedPriceRestaurantBooking`) for specific kinds of bookings that we wish to create. Additionally, our inheritance hierarchy contains an abstract class (`LocationBooking`) that allows us to further *share commonalities* among classes, such as instance variables and (concrete or abstract) methods.

The concrete classes should provide the following constructors and methods to their users. Methods that are required by implemented interfaces or abstract superclasses are not mentioned explicitly.

The implementation of a method may also be inherited from a superclass where suitable. Then you do not even need to mention that method in the subclass that is supposed to provide the method. Note that your code should not all go into each of the concrete classes!

2.3.1 `public class RickshawBooking extends Booking`

A RickshawBooking is a booking for travelling from a start point to a destination point via a rickshaw taxi. In our setting, a rickshaw always transports exactly one passenger.

Constructor:

`public RickshawBooking(String name, Date date, String from, String to)`
Constructs a RickshawBooking object according to the parameters.

- `name` – name used for the booking; must not be null
- `date` – the point in time for the booking; must not be null and must not be in the past
- `from` – the start point of the rickshaw travel; must not be null
- `to` – the destination point of the rickshaw travel; must not be null

Methods:

`public String getName()` Returns the name of the booking.

`public Date getDate()` Returns the point in time for which the booking was made.

`public String getFrom()` Returns the start point.

`public String getTo()` Returns the destination point.

2.3.2 `public class HotelBooking extends LocationBooking implements HasFixedPrice`

Constructor:

```
public HotelBooking(String name, Date date, String location,
                    int totalPriceInPence, int singleRooms, int doubleRooms)
```

Constructs a HotelBooking object according to the parameters. Note that at least one room must be booked. For the number of persons, we assume that a single room is for one person and a double room is for two persons.

- `name` – name used for the booking; must not be null
- `date` – the point in time for the booking; must not be null and must not be in the past
- `location` – the address of the hotel; must not be null
- `totalPriceInPence` – the total price in pence for the booking
- `singleRooms` – the number of booked single rooms, must be at least 0
- `doubleRooms` – the number of booked double rooms, must be at least 0

Methods:

`public String getName()` Returns the name of the booking.

`public Date getDate()` Returns the point in time for which the booking was made.

`public String getLocation()` Returns the location of this booking.

`public int getSingleRooms()` Returns the number of booked single rooms.

`public int getDoubleRooms()` Returns the number of booked double rooms.

2.3.3 `public class RestaurantBooking extends LocationBooking`

Constructor:

```
public RestaurantBooking(String name, Date date, String location,
                                                 int numberOfPersons)
```

Constructs a new RestaurantBooking according to the parameters.

- `name` – name used for the booking; must not be null
- `date` – the point in time for the booking; must not be null and must not be in the past
- `location` – the address of the restaurant; must not be null
- `numberOfPersons` – the number of persons for whom the booking is made; at least 1

Methods:

`public String getName()` Returns the name of the booking.

`public Date getDate()` Returns the point in time for which the booking was made.

`public String getLocation()` Returns the location of this booking.

2.3.4 `public class FixedPriceRestaurantBooking extends LocationBooking implements HasFixedPrice`

Constructor:

`public FixedPriceRestaurantBooking(String name, Date date, String location, int numberOfPersons, int pricePerPersonInPence)`

Constructs a new FixedPriceRestaurantBooking according to the parameters.

- `name` – name used for the booking; must not be null
- `date` – the point in time for the booking; must not be null and must not be in the past
- `location` – the address of the restaurant; must not be null
- `numberOfPersons` – the number of persons for whom the booking is made; at least 1
- `pricePerPersonInPence` – the price per person in pence

Methods:

`public String getName()` Returns the name of the booking.

`public Date getDate()` Returns the point in time for which the booking was made.

`public String getLocation()` Returns the location of this booking.

2.4 Your Mission

Your mission for this coursework is as follows:

1. *Abstract Methods.*

   Think about extendability of your code. Should *all* concrete classes that extend a certain abstract class (here: LocationBooking) have that method, also those that might only be written later? (It is entirely plausible that a future release of our software might include classes for, say, a ReceptionBooking that extends LocationBooking.)

   If so, then it could be a good idea to add the method to the abstract class. Then we could use that method also with a reference of the type of that abstract class.

   If we would need further information from specific subclasses to *implement* the method, we should label it as abstract: this means that concrete subclasses will need to provide an implementation, but in the current abstract class we need not worry about what the implementation will look like.

   An example is the abstract method

```
public abstract int getNumberOfPersons()
```

in the class Booking. We make it *available* in the class Booking because we want to be able to query all Bookings for their number of persons. And it is *abstract* because in the class Booking itself, we do not have enough information to implement the method. Subclasses with more information about how to compute the total charge can then implement the method.

2. *Concrete Methods.*

   Think about suitable places in the inheritance hierarchy where you could implement the methods. For those methods that are common for several classes both for their header and for their implementation, it is often a good idea to put that implementation into a common superclass. Then implement your methods.

   An example is the method

   ```
   public String getName()
   ```

   in the class Booking, which is available in all subclasses and which can already be implemented in the superclass Booking.

3. *Overriding.*

   You may also need to **override** some methods inherited from superclasses.

4. *Attributes (aka instance variables).*

   Think about what attributes are needed for each for the classes. Keep an eye on commonalities, and represent common attributes in abstract superclasses where suitable.

   Keep in mind the principles of information hiding! This means that in general it is a good idea to make your attributes *private* unless you have a really good reason not to do so. You can still give public or protected access to the attributes in a "controlled way" via getters or setters.

   An example are the attributes in the class Booking.

5. *Constructors.*

   Write a suitable constructor for the abstract class LocationBooking that calls a superclass constructor with suitable arguments and initialises the attributes for the present class. Then write constructors of the concrete classes that call their superclass constructors with suitable arguments and initialise the attributes for the present class.

   An example is the constructor of the abstract class Booking (which implicitly calls the superclass constructor of the class Object with 0 arguments).

6. *Parameter checking.*

   Whenever you take a parameter in a constructor or method where certain parameter values do not make sense (e.g., `null`, or negative values for a number of units; see also the constructor and method descriptions above), detect this at a suitable point in the code and throw an `IllegalArgumentException` with a suitable message to inform the user of your class. For an example, see the constructor of the class Booking.

   Note that a superclass constructor that you may be invoking may already be doing part of the work for you!

2.5 Coding Requirements

- All instance variables should be `private`.
- Whenever you override a (concrete or abstract) method from a (direct or indirect) superclass or an interface, always mark this with an `@Override` annotation (see the `toString()` method in Booking for an example).
- Every method (including constructors) should have full Javadoc comments. (However, in this coursework you do not need to write Javadoc comments for methods that you override from a superclass or implement from an interface.) In BlueJ, the "example class" that you get when you create a new class also provides example Javadoc comments. You can use Eclipse's `Source` → `Generate Element Comment` to get a suitable template with `@param` entries for all method parameters and with `@return` for methods that have a non-void return type.
- **Code style**: One aspect for Java projects is the use of the `this.` prefix before the name of an instance variable or method. **In this coursework**, please use the following style: Always write `this.` for calls to instance methods and accesses to instance variables (so in your instance methods and constructors you would always write `this.foo()` and `this.bar` instead of `foo()` and `bar`; see also the code style used in the class `Booking`).

  The rationale behind this particular code style is that it makes it immediately clear to other programmers when an access is to a (possibly inherited) *instance* method or an *instance* variable.

  (Programmers often spend significantly more time on reading *code, often by other developers, than on* writing *code. This is why many large software projects impose such style guidelines on their contributors so that the code looks uniform and is easy to read for other project members. This requirement is supposed to give you practice for such a setting.)*
- Every class you modify should have your name in the Javadoc comment for the class itself, using the Javadoc tag `@author`.
- Write documentation comments also for the attributes of your classes. (See the class Booking for an example.)

  The other programmers who will later have to modify your code need to know what they may assume about the values of the attributes, and at the same time also what they must guarantee themselves so that the code in the class will still work correctly after *their* new method has run. (See also the requirements for Coursework Assignment 1 for further discussion.)
- The code template for this coursework is made available to you as a Git repository on GitHub, via an invitation link for GitHub Classroom.

  The idea for the workflow is similar to Coursework Assignment 1:

  1. First use a web browser to follow the invitation link for the coursework assignment that is available on the Moodle page of the module. Use your GitHub account to log in.

2. Then *clone* the Git repository from the GitHub server that GitHub will create for you. Initially it will contain the following files: `README.md`, `Coursework2Main.java`, and the files for our inheritance hierarchy.
3. Your task is to enter your name in `README.md` (this makes it easy for us to see whose code we are marking) and to edit the Java source code files according to the requirements of the coursework (i.e., replacing a number of `// TO DO` and dummy implementations of methods with actual code).
4. You must also enter the following Academic Declaration into `README.md` for your submission (see also Section 6):

   > "I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software."

   This refers to the document at:

   `http://www.bbk.ac.uk/mybirkbeck/services/rules/Assessment%20Offences.pdf`

   A submission without this declaration will get 0 marks.
5. Whenever you have made a change that can "stand on its own", say, "Implemented constructor for RickshawBooking", this is a good opportunity to *commit* the change to your local repository and also to *push* your changed local repository to the GitHub server.

   As a rule of thumb, in collaborative software development it is common to require that the code base should at least still compile after each commit.

- Your source code should be properly formatted. You can use Eclipse's `Source` → `Format` for this purpose. In BlueJ you can use `Edit` → `Auto-layout`.
- *Reminder – use (also auxiliary) methods.* Don't cram everything into one or two methods, but try to divide up the work into sensible parts with reasonable names. Every method should be short enough to see all at once on the screen. (This is probably not a problem in this coursework, also thanks to the way object orientation helps us distribute the code over several classes.)

2.6 Remarks

Some remarks to simplify the task:

- For this coursework it is not necessary to override the methods `equals(Object)` or `hashCode()` inherited from the class Object.
- The `toString()` method from the class Object has already been overridden for some classes in the inheritance hierarchy. Do not modify these implementations. (Note that these implementations are not necessarily "optimal" with respect to object-oriented principles.) You do not need to implement `toString()` in this coursework assignment.
- For simplicity, in this coursework it is not necessary to write getters or setters unless they are needed to implement the required functionalities.

3 Marking Scheme

We aim to award marks according to the following scheme.

1. Code style.
   (a) Formatting and indentation: **5 marks**
   (b) Consistent use of `this.myField` and `this.myMethod()`: **5 marks**
2. Comments.
   (a) Instance variables have suitable documentation comments (see also Coursework Assignment 1): **6 marks**
   (b) Methods have suitable documentation comments, and `@Override` is used wherever applicable: **6 marks**
3. Reasonable data representation.
   (a) Proper information hiding/encapsulation: **5 marks**
   (b) Instance variables in the "right" (most general) classes (and nowhere else): **20 marks**
4. Object construction and error checking.
   (a) Parameter checking in constructors and throwing suitable exceptions: **12 marks**
   (b) Correct calls to superclass constructors: **10 marks**
   (c) Initialisation of instance variables: **5 marks**
5. Methods, with suitable distribution over the available classes: **26 marks**

4 Deadlines and Submission Instructions

Submission is through BOTH Moodle AND your GitHub Classroom repository. In Moodle, you need to upload a zip file of the folder containing your working copy and your local Git repository with your modifications to `README.md` and the Java source code files. The GitHub Classroom repository should contain the same commits as your local Git repository.

You should upload the completed assignment on Moodle by

27 November 2018, 11:55pm GMT

(this is Moodle time, not your PC's time; in case you are planning to upload your files whilst at a remote location, make sure you check Moodle's time and take into account the time zone difference). Make sure that the files in your repository on GitHub Classroom correspond to those in your zip file uploaded to Moodle.

It is your responsibility to ensure that the files transferred from your own machines are in the correct format and that any programs execute as intended on Department's systems prior to the submission date.

Each piece of submitted work MUST also have the "Academic Declaration" in `README.md` by the author that certifies that the author has read and understood the sections of plagiarism in College's Policy on Assessment Offences; see `http://www.bbk.ac.uk/mybirkbeck/services/rules/Assessment%20Offences.pdf`. Confirm that the work is your own, with the work of others fully acknowledged. Also include a declaration giving us permission to submit your report to the plagiarism testing database that the College is using.

Reports without the Academic Declaration are not considered as completed assignments and are not marked. The Academic Declaration should read as follows:

> "I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software."

You should note that all original material is retained by the Department for reference by internal and external examiners when moderating and standardising the overall marks after the end of the module.

We aim to provide you with feedback on your solutions by **18 December 2018**.

5 Late coursework

It is our policy to accept and mark late submissions of coursework. You do not need to negotiate new deadlines, and there is no need to obtain prior consent of the module leader.

We will accept and mark late items of coursework up to and including 14 days after the normal deadline. Therefore the last day the system will accept a late submission for this module is

11 December 2018, 11:55 pm GMT

(this is Moodle time not your PC's time; in case you are planning to upload your files whilst at a remote location, make sure you check the Moodle time and take into account the time zone difference). This is the absolute cut-off deadline for coursework submission.

However, penalty applies on late submissions. Thus, the maximum mark one can get in the coursework is 40 out of 100. If you believe you have a good cause to be excused the penalty for late submission of your coursework, you must make a written request using a mitigating circumstances application form and attach any evidence. Your form should be handed in or emailed to Programme Administrator (with a carbon copy to the module leader and Programme Director) as soon as possible, ideally by the cut-off deadline. This letter/email does not need to be submitted at the same time as the coursework itself but MUST be submitted by

4 December 2018.

Even if the personal circumstances that prevented you from submitting the coursework by the last day are extreme, the Department will not accept coursework after this date. We will, naturally, be very sympathetic, and the Programme Director will be happy to discuss ways in which you can proceed with your studies, but please do not ask us to accept coursework after this date; we will not be able to as there is a College-wide procedure for

managing late submissions and extenuating circumstances in student assessment. As soon as you know that you will not be able to meet the deadline, it will be useful for you to inform the module leader. They will be able to advise you on how best to proceed. Another person to speak to, particularly if the problem is serious, is the Programme Director. You will then have the opportunity to discuss various options as to how best to continue your studies.

Further details concerning the rules and regulations with regard to all matters concerning assessment (which naturally includes coursework), you should consult College Regulations at `http://www.bbk.ac.uk/mybirkbeck/services/rules`. Please see the programme booklet for the rules governing Late Submissions and consideration of Mitigating Circumstances and the Policy for Mitigating Circumstances at the College's website `http://www.bbk.ac.uk/mybirkbeck/services/rules`.

6 Plagiarism

The College defines plagiarism as "copying a whole or substantial parts of a paper from a source text (e.g., a web site, journal article, book or encyclopaedia), without proper acknowledgement; paraphrasing of another's piece of work closely, with minor changes but with the essential meaning, form and/or progression of ideas maintained; piecing together sections of the work of others into a new whole; procuring a paper from a company or essay bank (including Internet sites); submitting another student's work, with or without that student's knowledge; submitting a paper written by someone else (e.g. a peer or relative), and passing it off as one's own; representing a piece of joint or group work as one's own".

The College considers plagiarism a serious offence, and as such it warrants disciplinary action. This is particularly important in assessed pieces of work where the plagiarism goes so far as to dishonestly claim credit for ideas that have been taken from someone else.

Each piece of submitted work MUST have an "Academic Declaration" by the student in the file `README.md` which certifies that the student has read and understood the sections of plagiarism in the College Regulation and confirms that the work is their own, with the work of others fully acknowledged. This includes a declaration giving us permission to submit coursework to a plagiarism testing database that the College is subscribed.

If you submit work without acknowledgement or reference of other students (or other people), then this is one of the most serious forms of plagiarism. When you wish to include material that is not the result of your own efforts alone, **you should make a reference to their contribution, just as if that were a published piece of work**. You should put a clear acknowledgement (either in the text itself, or as a footnote) identifying the students that you have worked with, and the contribution that they have made to your submission.

7 Useful resources

Here are some resources on plagiarism, study skills, and time management that can help you to better manage your project and avoid plagiarism.

On Plagiarism

- `https://owl.english.purdue.edu/owl/resource/589/1/`

On Study Skills

- `http://www.bbk.ac.uk/student-services/learning-development`