Software and Programming II (SP2) 2023/24: Coursework

1 Introduction

- **Submission Deadline: 17 November 2023, 14:00 UK time.**

The coursework for this module is worth 50% of the overall module mark. There are **two** parts to the coursework. Both are described in this document, and they have the same submission deadline. The coursework contributes to your overall module mark as follows:

- Part 1 accounts for 50% of the coursework mark (i.e., 25% of the overall module mark);
- Part 2 accounts for 50% of the coursework mark (i.e., 25% of the overall module mark).

The coursework is marked out of a total of 100. The code for this coursework assignment is made available to you on Moodle and on github classroom.
The parts of the coursework are further explained in Section 2 and Section 3 below. 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.

2 Description of the work — Part One: Self Storage Unit

2.1 `SelfStorageUnit` and `Item`

In this coursework part, we want to write a class `SelfStorageUnit`. Its instances can store and provide information about objects of a class `Item` that has already been written. The `Item`s stored in our `SelfStorageUnit` objects are very simple: they have a name and a weight. A `SelfStorageUnit` object can then tell us, e.g., the average weight of its current items, the number of current items, . . .

The following example:

```
SelfStorageUnit unit = new SelfStorageUnit();
unit.add(new Item("Bicycle", 9000));
unit.add(new Item("Scarf", 60));
System.out.println(unit.numberOfItems());
unit.add(new Item("Pen", 35));
System.out.println(unit.numberOfItems());
```

should print:

```
2
3
```

Here, the method `numberOfItems()` returns the number of items that have been added to the SelfStorageUnit so far. When we called `unit.numberOfItems()` for the first time, only the first two items, named "`Bicycle`" and "`Scarf`", had been added to our storage unit, so the result was 2. When we then called `unit.numberOfItems()` for the second time, the third item, named "`Pen`", had also been added to our unit, so the result was 3. Thus, the same method call on the same SelfStorageUnit object (e.g., `unit.numberOfItems()`) can have different results, depending on the state of the object.
In this coursework we do not want to analyse any `null` item references. So, the code snippet

```
SelfStorageUnit unit = new SelfStorageUnit();
unit.add(new Item("Tinned food", 400));
unit.add(null);
System.out.println(unit.numberOfItems());
```

should print:

```
1
```

It is *up to you* as the implementor of the class `SelfStorageUnit` whether the method add or the method `numberOfItems` deals with the `null` references that may occur as an argument of add. For the users of your class (who only care about the "behaviour" of its objects, i.e., what effects calling the methods on the objects have), this is an implementation detail that they need not know about. What they (mainly) care about is that your methods always give the right results.
The *public interface* of the class `SelfStorageUnit` is already present in the template code on Moodle in the form of headers for constructors and methods and documentation comments describing the desired behaviour. However, the current method *implementations* are "stubs" that allow the code to compile but not to work correctly. Thus, you will need to provide implementations for these methods that work correctly according to the documentation of the public interface of the class `SelfStorageUnit`. You will certainly also need one (or more) suitable `private` instance variables — also called fields or attributes — for the class `SelfStorageUnit`.

2.2 Coursework1Main

We are providing the file `Coursework1Main.java` in the repository. This class makes use of some of the desired functionalities of the class `SelfStorageUnit` in the `main` method. You can (and should) test your implementation of `SelfStorageUnit` by running `Coursework1Main.main`. These tests provide further clarification for the behaviour that `SelfStorageUnit` is supposed to show. It is a requirement that your implementation of `SelfStorageUnit` compiles and works with the *unmodified* `Coursework1Main.java` and `Item.java`. You should expect that we will use the original versions of these files to test your implementation of the `SelfStorageUnit` class, not the ones that you may have modified!
The file `Coursework1Main.java` also contains a comment at the end with the output that its `main` method produces with our implementation of the `SelfStorageUnit` class.
Note, however, that the tests performed by `Coursework1Main` are not meant to be exhaustive — so even if `Coursework1Main` 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 with further test cases, but also to review it before handing in your solution.

2.3 Coding Requirements

- Only one of the two constructors should explicitly initialise all the instance variables of the class. The other constructor should then just call the first constructor via `this(...)` with suitable arguments.
- All instance variables should be `private`.
- Your implementations of the methods should not modify their arguments.
- None of the methods and none of the constructors in the class `SelfStorageUnit` may print on the screen (i.e., no `System.out.println`).
- Every method (including constructors) should have Javadoc comments.

  For Part One, comments for the required methods and constructors are already provided, but if you write additional methods or constructors, you should of course document them.
- Every class you create or modify should have your name in the Javadoc comment for the class itself, using the Javadoc tag `@author`.
- Write documentation comments also for the instance variables (attributes) of your classes. These are useful not only for you at the moment, but also for programmers who may later work on your code. This may well be *you* again, a few years down the line, wondering what you were thinking back in the days when you had written that code.

  (Remember — in this module we are training for "skyscraper-sized" software projects. So, even if your coursework might not consist of thousands of classes and might not be developed and maintained over several decades, **we want you to write code as if it were the case***.)*

  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.

For example, it does not make sense for a `length` attribute to have a negative value. So, your instance methods can assume that `length` is not negative. But then, your (or someone else's!) instance methods in this class must also make sure that `length` is *still* non-negative after they have run (which is not a problem if they don't modify `length`).

- Your source code should be properly formatted.

- 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, follow a consistent style: either *always* write `this.` when possible for method calls and accesses to instance variables (so in your instance methods you would always write `this.foo()` and `this.bar`), or alternatively write `this.` only when this is *unavoidable*, because a local variable or formal parameter "shadows" an instance variable (then you would always try to write `foo()` and bar to access these instance methods and variables in the same class).

  (*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.*)

- Reminder — use methods (and auxiliary method if required). Do not cram everything into one or two methods; instead, 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.

  For the methods in this assignment, their length is probably not a problematic issue. However, do keep an eye on the other methods that you are writing for this coursework — perhaps one of them already does part of the work that you need in another one? Then you could just call that method instead of writing down its code twice. So instead of re-implementing a method of `SelfStorageUnit` (or `Item`, or ...) as part of another method, you should just *call* that method whenever it can do part of the work for your method.

- The code template for this coursework part is made available to you as a Git repository on GitHub, via an invitation link for GitHub Classroom.

  1. First you follow the invitation link for the coursework that is available on the Moodle page of the module.
  2. Then *clone* the Git repository from the GitHub server that GitHub will create for you. Initially it will contain the following three files: `README.md`, `SelfStorageUnit.java`, and `Coursework1Main.java`.
  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 `SelfStorageUnit.java` 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:
`https://www.bbk.ac.uk/downloads/registry/policies-2021-22/assessment-offences-policy.pdf`

A submission without the declaration will get 0 marks.

5. Whenever you have made a change that can "stand on its own", say, "Implemented numberOfItems() method", 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.

Entering your name in `README.md` (using a text editor or IDE), then doing a *commit* of your change to the file into the local repository, and finally doing a *push* of your local repository to the GitHub server would be an excellent way to start your coursework activities.

You can benefit from the GitHub server also to synchronise between, e.g., the Birkbeck lab machines and your own computer. You *push* the state of your local repository in the lab to the GitHub server before you go home; later, you can *pull* your changes to the repository on your home computer (and vice versa).

Use meaningful commit messages (e.g., "Implemented numberOfItems() method"), and do not forget to *push* your changes to the GitHub server! For marking, we plan to *clone* your repositories from the GitHub server shortly after the submission deadline.

We **additionally** require you to upload to *Moodle* a zip file that contains the folder with your modified `README.md` and a folder containing your working copy and your local Git repository (which should be identical to your repository on GitHub Classroom). One reason is that the time of the upload will tell us if you would like your code to be considered for the regular (uncapped) deadline or for the late (capped) deadline two weeks later. (There is also the (unlikely) case that the GitHub servers have a data loss — then Moodle would provide us with an alternative way of accessing the submission version of your code and your local repository.)

In the end, please submit your repository with your commit history.

(Version control systems like Git and platforms like GitHub are standard in collaborative *software development, and industry is keen on developers who are familiar with these tools and concepts. This is why we giving Git a central role in the SP2 coursework — committing code changes and version control systems are important aspects of practical software development. Similar to programming languages, version control systems also share concepts, so the Git practice will be beneficial even if your company uses, e.g., Mercurial or more classic "centralised" systems like Subversion, CVS, . . .)*

2.4 Hints

- Think about the most suitable internal data representation for your `SelfStorageUnit` implementation. There are many correct choices, but some of them can make implementing the methods *significantly* easier than others. Take into consideration that when we construct our `SelfStorageUnit`, we do not know how many `Items` will be added to it in the future.

  (What Java classes have we already seen in SP2 for containers that can store an unbounded number of elements?)

- In case you are considering to extend (i.e., write a subclass of) a class from the Java API that may already have some of the needed functionality: this solution is very likely to be "more trouble than it's worth". Rather think about whether you can have an instance variable of that class to which you can "delegate" some of the `SelfStorageUnit` method calls.

- In this assignment, we are analysing objects of the class `Item`. The class `Item` provides you with a number of useful methods that you can call on `Item` objects.

  We are providing you with the implementation of the class `Item`. Instead of looking at the source code, you can also read up on the methods of the class `Item` in its *API documentation*. It is available in the repository in the file

  `doc/Item.html`

  You do not need to scrutinise the whole documentation of the class `Item` — just try to find method names that look potentially helpful for your task and then read up on what these particular methods do. You will most probably not need all methods in the class `Item`.

  (Reading the API documentation of other people's code that we do not want to modify, but only use *— instead of looking into the actual implementation — is fairly common, particularly when we are dealing with large code bases. In this coursework, we even have the source code of the class* `Item`*. However, in general the source code of the implementation may not even be* available *to us, e.g., because the authors of the code have not shared it with us. This may happen in particular in commercial settings. Then other programmers will just read the API documentation, which* is *available in most cases.)*

- Keep an eye on the way the methods are supposed to deal with `null` as an actual parameter (or as an *entry* of an array that is an actual parameter).

2.5 Marking Scheme: Part One

For this coursework part, we aim to award marks based on *automated testing*. This means that we will compile your file `SelfStorageUnit.java` together with the original `Item.java`, and we will run snippets of code similar to the ones in `Coursework1Main`. In particular, this means that your file `SelfStorageUnit.java` must compile together with the unchanged `Item.java`.
We will test all public constructors and methods that are present in the file `SelfStorageUnit.java` on Moodle.
There are 50 marks for Part One of the coursework. 30 marks will be allocated according to the proportion of tests passed. (i.e., 30 times the number of passed tests, divided by the total number of tests.)

$$30 \cdot \text{number of passed tests}/\text{number of tests}$$

To see whether you are "on the right track", we recommend you to run method `main` of class `Coursework1Main` and to check whether your outputs are identical to the ones in the comment at the bottom of the file. However, we have not provided all of the tests that will be used for marking. You will need to ensure that your code works correctly according to the specification, in addition to passing the tests that we have provided. in `Coursework1Main.java`.
The remaining 20 marks for Part One are awarded as follows:

- 5 marks for the quality of your design, coding style, and comments.)
- 15 marks for evidence that you have regularly committed compiling code to your github classroom repository at appropriate points during the development of your solution.

3 Description of the work — Part Two: Cards

3.1 Overview

This coursework part provides two Java classes as a starting point, and consists of a number of questions which require you to extend and develop this class hierarchy. The submission process is similar to Part One:

1. The code template for this coursework part is made available to you as a Git repository on GitHub, via an invitation link for GitHub Classroom.

2. First you follow the invitation link for the coursework that is available on the Moodle page of the module.

3. Then *clone* the Git repository from the GitHub server that GitHub will create for you. Initially it will contain the following three files: `README.md`, `Card.java`, and `CardDriver.java`.

4. 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 add code according to the requirements.

5. 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:

   `https://www.bbk.ac.uk/downloads/registry/policies-2021-22/assessment-offences-policy.pdf`

   A submission without the declaration will get 0 marks.

6. Whenever you have made a change that can "stand on its own", say, "Implemented numberOfItems() method", 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.

3.2 Marking Scheme

There are 50 marks available for Part Two. 35 marks are allocated according to the marks indicated below beside text of each question. The remaining 15 marks are awarded as follows:

- 5 marks for the quality of your design, coding style, and comments.)
- 10 marks for evidence that you have committed regularly working code to your github classroom repository at appropriate points during the development of your solution.

3.3 Questions

1. Consider the following Card class (also available on Moodle):

```
package one;

/**
 * A Card object has a name and a few methods to get the name
 * and format the card.  It never expires.
 */
public class Card {
    /** Name of this card. Potentially null. */
    private String name;

    /**
     * Constructs a Card object with empty string name.
     */
    public Card() {
        this.name = "";
    }

    /**
     * Constructs a Card object with given name
     *
     * @param name1 the given name
     */
    public Card(String name1) {
        this.name = name1;
    }

    /**
     * Accessor method for the name
     *
     * @return the name
     */
    public String getName() {
        return this.name;
    }

    /**
     * Tests whether the card is expired.
     *
     * @return false, since this type of card is never expired
     */
    public boolean isExpired() {
        return false;
    }

    /**
```

```
     * Gives a String format for the card
     *
     * @return the formatted String representing this card
     */
    public String format() {
        return "Name: " + this.name;
    }

    /**
     * Returns a String representation of instance variables
     * Note the fancy way of getting the name of the class.  Here
     * we know the class name is Card, but when subclasses run this
     * method we want the subclass name.
     *
     * @return string denoting the object
     */
    @Override
    public String toString() {
        return this.getClass().getName() + "[name = " + this.name + "]";
    }

    /**
     * Returns true if two cards have the same name and class.
     * Returns false if one card is null or not same class.
     * Note the fancy way of checking the name of the class.  Here
     * we know that the class is Card, but when subclasses run this
     * method we want the subclass.
     *
     * @return true if the two objects are same type and have same
     * instance variable values
     */
    @Override
    public boolean equals(Object other) {
        if (other == null) { return false; }
        if (!this.getClass().equals(other.getClass())) { return false; }
        Card card = (Card) other;
        return this.name.equals(card.name);
    }
}
```

You can test it with the following “driver class” (also available on Moodle):

```
package one;

/**
 * CardDriver is a basic driver with a simple main method.
 * It tests the Card class and subclasses
 */
public class CardDriver {
    public static void main(String[] args) {
        Card card = new Card();
        System.out.println("Card format after constructor without arguments:\n"
                + card.format());
        card = new Card("Emmett Brown");
        System.out.println("Card format after constructor with String argument:\n"
                + card.format());
    }
}
```

Note that both classes start with a declaration that they are inside the *package* `two`. After you extract the content of the zip file from Moodle, you will find a folder `cw` with a subfolder `two`. The subfolder `two` is a *Java package*, corresponding to the package declarations at the top of `Card.java` and `CardTester.java`.

Use the `Card` class as a superclass to implement a hierarchy of related classes:

Class	**Data**
`CallingCard`	Name, card number, PIN
`IDCard`	Name, ID number
`DriverLicence`	Name, ID number, expiration year

Here `CallingCard` and `IDCard` should both be subclasses of `Card`, and `DriverLicence` should be a subclass of `IDCard`.

Begin by writing declarations for each of the subclasses. For each subclass, supply private instance variables. (**6** marks)

2. Implement constructors for each of the three subclasses. Each constructor should call the superclass constructor to set the name and (possibly) the id. (**6** marks)

3. Replace the implementation of the `format` method for the three subclasses. The methods should produce a formatted description of the card details. The subclass methods should call the superclass `format` method to get the formatted name of the cardholder. (**6** marks)

4. Devise another class, `Wallet`, which contains slots for two cards, `card1` and `card2`, a method `void addCard(Card)` and a method `String formatCards()`.

   The addCard method should check whether `card1` is `null`. If so, it sets `card1` to the new card. If not, it checks `card2`. If both cards are set already, the method has no effect.

   `formatCards` should invoke the `format` method on each non-null card and produce a string with the format

   ```
   [card1|card2]
   ```

   (**6** marks)

5. Write a tester class `WalletTester` that adds two objects of different subclasses of `Card` to a `Wallet` object. Test the results of the `formatCards` methods. (**4** marks)

6. The `Card` superclass defines a method `isExpired`, which always returns `false`. This method is not appropriate for the driver licence. Supply a method header and body for `DriverLicence.isExpired()` that checks whether the driver licence is already expired (i.e., the expiration year is less than the current year). (**7** marks)

   To work with dates, you can use the methods and constants supplied in the abstract class `Calendar` from the `java.util` package which are inherited by the concrete class `GregorianCalendar` (also from `java.util`). You can create a `Calendar` as follows:

   ```
   GregorianCalendar calendar = new GregorianCalendar();
   ```

   Then, you can obtain the current year using the constant `Calendar.YEAR` and method `get` in `GregorianCalendar`. The constant indicates that the method should return the current year. By comparing the returned value with the `expYear` field in `DriverLicence`, you can determine if the card has expired. The code below will retrieve the current year:

   ```
   calendar.get(Calendar.YEAR)
   ```

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 `SelfStorageUnit.java`. The GitHub Classroom repository should contain the same commits as your local Git repository.
You should upload the completed assignment on Moodle by

17 November 2023, 14:00 UK time

(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).
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 `https://www.bbk.ac.uk/downloads/registry/policies-2021-22/assessment-offences-policy.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.

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

1 December 2023, 14:00 UK time

(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. This is described here: `https://www.bbk.ac.uk/downloads/registry/policies-2021-22/late-submission-of-coursework.pdf`
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; for the procedures, see `https://www.bbk.ac.uk/downloads/registry/policies-2021-22/covid-19-mit-circs-student-guidance.pdf`. This request does not need to be made at the same time as the coursework itself but MUST be submitted by

27 November 2023.

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 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.

Further details concerning the rules and regulations with regard to all matters concerning assessment (which naturally includes coursework), you should consult College Regulations 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://ori.hhs.gov/avoiding-plagiarism-self-plagiarism-and-other-questionable-writing-practices-guide-ethical-writing`

`https://www.bbk.ac.uk/professional-services/registry-services/policies-2021-22/plagiarism-prevention-document`

On Study Skills:

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