Software and Programming II (SP2)
2024/25: Coursework Reassessment

1 Introduction

- **Submission Deadline: 15th August 2025, 14:00 by github and Moodle confirmation**

There is **one** coursework assignment for this module. This coursework assignment contributes 50% of your overall module mark.
There are a total of 100 marks for this coursework. There are two parts to the coursework, which carry equal weight (50 marks each). The code for this coursework assignment is made available to you on Moodle.

2 Part 1 — Objects and Classes: Smartshelf — 50 marks

Smartshelf

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

```
Smartshelf shelf = new Smartshelf();
shelf.add(new Item("Clock", 400));
shelf.add(new Item("Newspaper", 20));
System.out.println(shelf.numberOfItems());
shelf.add(new Item("Pen", 35));
System.out.println(shelf.numberOfItems());
```

should print:

```
2
3
```

Here, the method numberOfItems() returns the number of items that have been added to the Smartshelf so far. When we called shelf.numberOfItems() for the first time, only the first two items, named "Clock" and "Newspaper", had been added to our Smartshelf, so the result was 2. When we then called shelf.numberOfItems() for the second time, the third item, named "Pen", had also been added to our Smartshelf, so the result was 3. Thus, the same method call on the same Smartshelf object (e.g., shelf.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

```
Smartshelf shelf = new Smartshelf();
shelf.add(new Item("Soda", 400));
shelf.add(null);
System.out.println(shelf.numberOfItems());
```

should print:

```
1
```

It is *up to you* as the implementor of the class Smartshelf 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 Smartshelf 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 Smartshelf. You will certainly also need one (or more) suitable private instance variables — also called fields or attributes — for the class Smartshelf.

Coursework1Main

We are providing the file Coursework1Main.java in Moodle download. This class makes use of some of the desired functionalities of the class Smartshelf in the main method. You can (and should) test your implementation of Smartshelf by running Coursework1Main.java. These tests provide further clarification for the behaviour that Smartshelf is supposed to show. It is a requirement that your implementation of Smartshelf 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 Smartshelf 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 Smartshelf 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.

Coding Requirements (Part One and general)

- 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.
- None of the methods and none of the constructors in the class Smartshelf may print on the screen (i.e., no System.out.println).

- Every method (including constructors) should have Javadoc comments.

  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.

  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).

  Avoid either an excessively concise 'functional' style with lots of 'one-liners', or unnecessarily lengthy or redundant operations.

  Otherwise, follow the naming and style conventions I used during the module.

  (*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. 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 `Smartshelf` (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.

Marking Scheme: Part One

Marking this part of the coursework will be assisted by *automated testing*. This means that we will compile your file Smartshelf.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 Smartshelf.java must compile together with the unchanged Item.java.
We will test all public constructors and methods that are present in the file Smartshelf.java on Moodle.
There are 50 marks for this part of the assignment. 30 marks will be allocated according to the proportion of tests passed (as 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 Coursework1Main.java.

- There are 30 marks for the automated tests as above
- Ten marks for this part will be allocated for code style, formatting, and documentation.
- Ten marks for this part will be allocated for a regular and informative github commit history.

Hints - Part One and general

- Think about the most suitable internal data representation for your Smartshelf 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 Smartshelf, we do not know how many Items will be added to it in the future, and there is no limit for how many items may be added to it.

  (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 your Smartshelf class can have an instance variable of that class to which you can "delegate" some of the Smartshelf 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 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).

3 Part 2— Inheritance and Interfaces — 50 marks

This part of the coursework asks you to develop a class and interfaces to represent and work with geometric shapes. Classes **GeometricObject** and **Rectangle** are provided to you on Moodle. **Rectangle** is a subclass of **GeometricObject**.

1. Implement another class, **Circle**, that is also a subclass of **GeometricObject**. In addition to the attributes and methods that it inherits from **GeometricObject**, **Circle** should have:
   (a) a radius attribute (and associated getter and setter methods)
   (b) a getDiameter() method (the diameter is calculated as two times the radius)
   (c) a getArea() method (the area of a Circle with radius r is $= \pi r^2$)
   (d) a getPerimeter() method (the perimeter of a circle with radius r is $2\pi r$)
2. Update the **GeometricObject** class to implement the **Comparable** interface. The *compareTo* method should compare **GeometricObject** objects based on area.
3. Create a new interface named **Scalable** which contains the following method
   public abstract void scale(double factor);
   Update the **GeometricObject** class so that it also implements the *Scalable* interface. Since **GeometricObject** is an abstract class, the *scale* method does not need to be defined in this class. However, subclasses of **GeometricObject** (**Circle** and **Rectangle**) do need to override this method.
   For **Circle**,the method *scale* should multiply the *radius* by the parameter *factor*.
   For **Rectangle**, the *scale* method should multiply both the width and the height by the parameter *factor*. For both **Circle** and **Rectangle**, if *factor* ≤ 0, the values of the instance variable(s) should remain unchanged.
4. Create a new interface named **Rotatable** which contains the following method:
   public abstract void rotate();
   Update the **Rectangle** class to implement the **Rotatable** interface. Each time the method *rotate* is invoked on a **Rectangle** object, the object should be rotated by 90° (i.e. the values of the instance variables width and the height should be switched).

Marking Scheme: Part Two

- There are eight marks for each of the above coding tasks (32 in total)
- There are eight marks for code style and documentation
- The remaining ten marks are for a regular and informative github commit history

Hints and coding requirements- Part Two

The purpose of this part is to test your ability to use inheritance and interfaces. It is not necessary to check for incorrect inputs to methods, except where specified (i.e. checking if $factor \leq 0$)
Otherwise, follow the general instructions as in Part One — break your code into methods, use javadoc comments, format your code properly, and use a consistent code style.

4 Submission (both parts)

4.1 Submit using github classroom and confirm on Moodle

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 re-assessment coursework that is available on the Re-assessment tile of the Moodle page.
2. Then *clone* the Git repository from the GitHub server that GitHub will create for you. Initially this repository will be an empty folder. You should add the template files from the Moodle to this folder to get started.
3. Enter your name in $\mathrm{README.md}$ (this makes it easy for us to see whose code we are marking)
4. For Part One, add the Item.java, Smartshelf.java, and Coursework1Main.java starter files available for download on the Moodle page.
5. For Part Two add GeometricObject and Rectangle files available for download on Moodle.
6. Edit and add code as necessary to complete the requirements. Commit and push to github regularly.
7. You must also enter the following Academic Declaration into $\mathrm{README.md}$ for your submission:

> "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/student-services/exams/plagiarism-guidelines

A submission without the declaration will get 0 marks.

8. 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), 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 confirm your submission on Moodle by submitting the link to your github repository at the time when you have completed your coursework. See the Moodle re-assessment tile for this form, and ensure you complete all steps of the form (confirm, submit, save).

4.2 Deadlines

- The submission deadline is: 15th August 2025, 14:00 UK time.
- **There are no late submissions for re-assessment**.