Module Code
COM00016C

BEng, BSc, MEng and MMath Degree Examinations 2023–24

Department Computer Science

Title Software 2: Java Programming

TIME ALLOWED FIVE hours (Recommended time to complete FOUR hours)

Papers late by up to 30 minutes will be subject to a 5 mark penalty; papers later than 30 minutes will receive 0 marks.

The time allowed includes the time to download the paper and to upload the answers.

Word Limit Not Applicable

Allocation of Marks: Question 1 is worth 15%, question 2 is worth 10%, question 3 is worth 30% and question 4 is worth 25%. The remaining 20% are allocated to style, clarity, code documentation and quality of code. The code must adhere the Google Java style guide.

Instructions:

Candidates must answer **all** questions using Java OpenJDK 11 or above. Failing to do so may result in a mark of 0%. Download the paper from the VLE, in the "Assessment>SOF2 2023-24 Summer Java exam" section. Submit all your answers to the GradeScope submission point named **SOF2 2023-24 Summer Java exam**. You can find the GradeScope submission point on the "Assessment>SOF2 2023-24 Summer Java exam" page on the VLE. **DO NOT** submit a zip file, only submit the Java files containing your solutions.

If you have urgent queries regarding a suspected error in the exam, inform `cs-exams@york.ac.uk` with enough time for a response to be considered and made within the first hour of the start of the exam. Corrections or clarifications will NOT be announced after the first hour of the exam. If a question is unclear, answer the question as best you can, and note the assumptions you have made to allow you to proceed. Inform ⟨`cs-exams@york.ac.uk`⟩ about any suspected errors on the paper immediately **after** you submit.

Note on Academic Integrity

We are treating this online examination as a time-limited open assessment, and you are therefore permitted to refer to written and online materials to aid you in your answers. However, you must ensure that the work you submit is entirely your own, and for the whole time the assessment is live you must not:

- communicate with other students on the topic of this assessment.
- communicate with departmental staff on the topic of the assessment (other than to highlight an error or issue with the assessment which needs amendment or clarification).
- seek assistance with the assessment from academic support services, such as the Writing and Language Skills Centre or Maths Skills Centre, or from Disability Services (unless you have been recommended an exam support worker in a Student Support Plan).
- seek advice or contribution from any other third party, including proofreaders, friends, or family members.

We expect, and trust, that all our students will seek to maintain the integrity of the assessment, and of their award, through ensuring that these instructions are strictly followed. Where evidence of academic misconduct is evident this will be addressed in line with the Academic Misconduct Policy and if proven be penalised in line with the appropriate penalty table. Given the nature of these assessments, any collusion identified will normally be treated as cheating/breach of assessment regulations and penalised using the appropriate penalty table (see AM3.3 of the Academic Misconduct Policy).

More than 40 years ago, on 27 August 1982, the very first Fighting Fantasy Gamebook, "The Warlock of Firetop Mountain" by Steve Jackson and Ian Livingstone, was published (source: what is fighting fantasy's blog).

In each Fighting Fantasy book, the reader becomes the hero! The main text is divided into a series of consecutively numbered sections (usually around 400), each section varying from a few lines to several paragraphs long, and describe encounters, people, conversations and events. At the end of these paragraphs a choice is usually presented with a reference to which paragraph should be read next, if chosen. Choices range wildly from something simple like – 'If you wish to go north, turn to 324. Otherwise, go to 13' to much more complex alternatives. At each section, the reader must make choices that impact the story's outcome. Whether you face combat, solve riddles, or encounter traps, your decisions shape your character's fate.

The aim of the paper is to build a prototype of a Fighting Fantasy eBook.

1 (15 marks) The class Hero

The first step is to build a class `Hero`, representing the hero of the story (that is the player/reader of the eBook). In addition, you **must** provide the `Javadoc` for the entire class `Hero`.
The class `Hero` must be in the package `book` and has the following **package visible** attributes:

- `String name` the name of the hero,
- `int maxAbility` the maximum ability the hero can have during the game, which must be greater or equal to 4,
- `int maxHealth` the maximum health the hero can have during the game, which must be greater or equal to 4,
- `int ability` the current ability of the player, which must be between 1 and `maxAbility`,
- `int health` the current heath of the hero, which must be between 0 and `maxHealth`. Note that a `health` of 0 means the hero is dead and therefore the game is lost.

(i) [5 marks] implement the class `Hero` with all its attributes. In addition, the class has one constructor:

`public Hero(String name, int maxAbility, int maxHealth)` that initialises the corresponding attributes. In addition, the `health` attribute should be initialised to `maxHealth`, and the attribute `ability` should be initialised to `maxAbility - 3`. The constructor must throw an `IllegalArgumentException` if the constraints for `maxAbility` and `maxHealth` are not satisfied.

(ii) [4 marks] Implement the following accessor methods: `getName`, `getMaxAbility`, `getMaxHealth`, `getAbility`, and `getHealth`.

(iii) [3 marks] Implement the following two methods that modify the attribute `ability`:

- `public void gainAbility(int gain)` that adds `gain` to the hero's current ability,
- `public void loseAbility(int loss)` that substracts `loss` from the hero's current ability.

Note that the constraints defined in the definition of the attribute `ability` must be satisfied after calling these methods.

(iv) [3 marks] Implement the following two methods that modify the attribute `health`:

- `public void receiveInjury(int seriousness)` that reduces the current health of the hero by `seriousness`,
- `public void addHealthBonus(int bonus)` that modifies the hero's current health by `bonus`. The `bonus` could be positive (health potion) or negative (poison).

Note that the constraints defined in the definition of the attribute `health` must be satisfied after calling these methods.

2 (10 marks) Exceptions

The project also needs some specific exceptions that we define in this question. Both classes must be in the package `book`.

(i) [5 marks] Implement the class `InvalidSectionReferenceException`. The exception is an **unchecked exception** that has a package visible attribute `int sectionReference`. The class has only one constructor `public InvalidSectionReferenceException(String message, int sectionReference)`, initialising the corresponding attributes.

(ii) [5 marks] Implement the class `DuplicateSectionReferenceException`. The exception is a **checked exception** that has a package visible attribute `int sectionReference`. The class has only one constructor `public DuplicateSectionReferenceException(String message, int sectionReference)`, initialising the corresponding attributes.

3 (30 marks) Abstract Classes and Inheritance

In this question, we will implement the classes needed to represent the different types of sections in a book. In our design, the superclass of all book sections is the abstract class `AbstractBookSection` as shown in Figure 1.

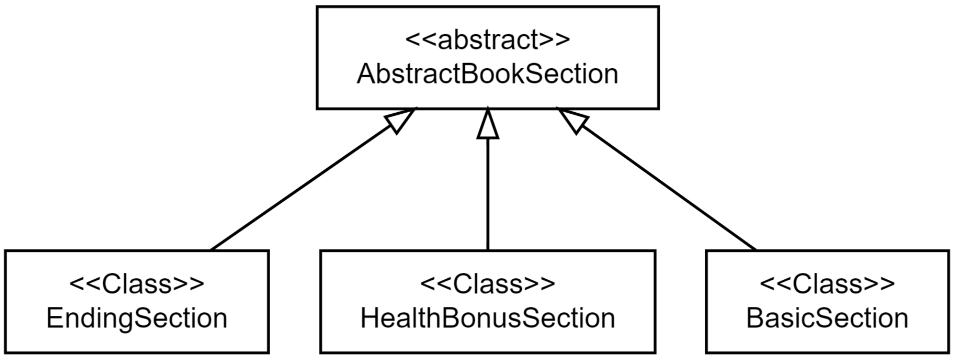

Figure 1: The class diagram for the sections of the eBook.

For our prototype, we will be creating three types of book sections: `BasicSection` that allows a reader to select a series of options, `EndingSection` that represents the end of the book, and `HealthBonusSection` that affects the health of the hero, and may offer several options to the reader for its next move. A book can have multiple ending sections, some representing winning the game, and other indicating the hero is dead and the game is lost.

All three classes are subclasses of `AbstractBookSection`. **All** classes described in this question **must** be in the package `book`. In addition, you **must** provide the `Javadoc` for the classes `AbstractBookSection` and `BasicSection`.

(i) [9 marks] Nine marks are allocated to the implementation decisions you make for this question. More specifically, which methods should be abstract, which methods should be overridden, use of convenience methods/constructors if needed, access modifiers used for these convenience methods, and reduction of code duplication.

Continued.

Implement the **abstract class** `AbstractBookSection`. The class has two public constants:
`int GAME_WON` which is equal to 0 and represents winning the game, and
`int GAME_LOST` which is equal to -1 and represents losing the game.

The class has the following **package visible** attributes:

- `int sectionNumber` representing the section number, which must be greater or equal to 1,
- `String text` contains the main text of the section,
- `Set<Integer> referencedBookSections` the set of choices presented to the reader by that section. That is, the numbers of the sections to go to depending on the player's choices. Note that the set is empty for ending sections.

In addition, the class has the following public methods:

- `int getSectionNumber()` that returns the section's number,
- `String getText()` that returns the section's text,
- `Set<Integer> getReferencedBookSections()` that returns the choices of sections' numbers available to the reader to choose from for their next move,
- `boolean addBookSectionReference(int sectionReference)` that adds a section's number to the attribute `referencedBookSections`. The method must throw an `InvalidSectionReferenceException` if the reference is less or equal to 0. If `sectionReference` is already in the set the method must return false, otherwise the method adds the `sectionReference` to `referencedBookSections` and then returns true.
- `boolean isEndingSection()` returns true if the section is an ending section, false otherwise.
- `int executeBookSection(Hero)` executes the section. The execution of the section depends on the type of section, for example adding health points to `hero` for a `HealthBonusSection`. The method returns the reference of the next section to read depending on the execution outcome. For example `GAME_LOST` if the hero dies, or 23 if the player chose the option to go to section 23. Note that in that case, the section number 23 must be in the set `referencedBookSections`.

Implement all these methods. Note that some of these methods might be abstract methods, and others not. Implement them accordingly to avoid code duplication.

(ii) [8 marks] Implement the class `BasicSection`. The class has two public constructors:

- `BasicSection(int number, String text, Set<Integer> sections)` where `number` is the section's number, `text` is the text of the section, and `sections` is the set of section numbers that the player can choose from for its next step, that is, the `referencedBookSections`. The constructor must initialise the appropriate instance attributes.
  The constructor must throw an `InvalidSectionReferenceException` if `number` or any values in `sections` are less or equal to 0.

- `BasicSection(int number, String text)` where `number` is the section's number, `text` is the text of the section. The `referencedBookSections` must be initialised to an empty set.
  The constructor must throw an `InvalidSectionReferenceException` if `number` is less or equal to 0.

For this class, the method `int executeBookSection(Hero)` returns randomly one of the sections' numbers in `referencedBookSections`. If `referencedBookSections` is empty, the method must throw an `IllegalStateException`.

Implement any abstract methods from the superclass `AbstractBookSection`. Override the superclass' methods as needed.

(iii) [8 marks] Implement the class `HealthBonusSection`. The class has one package visible attribute `int bonus` that represents how much the health of the hero is affected in that section. `bonus` can be positive (drinking a medicine) or negative (receiving a dose of poison). In addition, the class has one public constructor:

```
HealthBonusSection(int number, String text,
                   Set<Integer> sections, int bonus)
```

where `number` is the section's number, `text` is the text of the section, `bonus` is the health bonus, and `sections` is the set of sections' numbers that the player can choose from for its next step, in other words, the `referencedBookSections`. The constructor must initialise the appropriate instance attributes.
The constructor must throw an `InvalidSectionReferenceException` if `number` or any values in `sections` are less or equal to 0.

For this class, the method `int executeBookSection(Hero)` modifies the health of the `hero` by `bonus` and returns `GAME_LOST` if the health of the hero is 0. Otherwise, the method returns randomly one of the sections' numbers in `referencedBookSections`. If `referencedBookSections` is empty, the method must throw an `IllegalStateException`.

Implement any abstract methods from the superclass `AbstractBookSection`. Override the superclass' methods as needed.

Continued.

(iv) [5 marks] Implement the class `EndingSection`. The class has one package visible attribute `boolean winsGame` which states the section is a game winning section (`winsGame` is true) or a game losing section (`winsGame` is false).

The class has two public constructors:

- `EndingSection(int number, String text, boolean winsGame)` where `number` is the section's number, `text` is the text of the section, and `winsGame` is the outcome of the game, that is true if the player wins the game, false otherwise. The constructor must initialise the appropriate instance attributes, and the `referencedBookSections` must be initialised to an empty set.
  The constructor must throw an `InvalidSectionReferenceException` if `number` is less or equal to 0.

- `EndingSection(int number, String text)` where `number` is the section's number, `text` is the text of the section. The constructor must initialise the corresponding instance attributes, `winsGame` is set to true, and the `referencedBookSections` must be initialised to an empty set.
  The constructor must throw an `InvalidSectionReferenceException` if `number` is less or equal to 0.

For this class, the method `int executeBookSection(Hero)` returns `GAME_WON` if the attribute `winsGame` is true, `GAME_LOST` otherwise.

In addition, you must override the method `public boolean addBookSectionReference(int)` to always throw a `UnsupportedOperationException`.

Implement any abstract methods from the superclass `AbstractBookSection`. Override the superclass' methods as needed.

4 (25 marks) The GameBook Class.

The class `GameBook` is the main class of the project, representing a book with all its sections and a hero. The class contains the following **package visible** attributes:

- `Hero hero` the hero of the story,
- `AbstractBookSection start` the starting section of the book. There is only one starting section in a book, and its section number **must be 1**.
- `Map<Integer, AbstractBookSection> endings` a mapping of the ending sections with their section numbers. There might be many ending sections in a book.
- `Map<Integer, AbstractBookSection> bookSections` a mapping of all sections in the book with their section numbers. The map includes the starting section and the ending sections.

(i) [5 marks] The class `GameBook` has only one public constructor taking a single argument of type `Hero`. The constructor initialise the corresponding attribute, and in addition sets `start` to `null` and the two maps `endings` and `bookSections` to empty maps. Implement the class `GameBook` attributes and its constructor.

(ii) [6 marks] Implement the public method `void addSection(AbstractBookSection section)`. The methods adds `section` to `bookSections` if it is not already present in the map. The method must throw a `DuplicateSectionReferenceException` if another section with the same section number already exists in the map, and the two sections do not have the same text.

In addition, the section must be added to the attribute `endings` if and only if the section is an ending section (as determined by `isEndingSection()`).

(iii) [4 marks] Implement the public method `void setStartingSection(AbstractBookSection start)`. The method assigns the argument to the attribute `start`, and adds it to the `bookSections`. The method must throw an `InvalidSectionReferenceException` if the section number of `start` is not 1.

In addition, the method must throw a `DuplicateSectionReferenceException` if another section with the same section number already exists in the map `bookSections`, and the two sections do not have the same text.

(iv) [10 marks] A book is considered complete and correct if it meets the following conditions:

- it has a starting section,
- it has at least one ending section,
- all sections in the book can be reached from the starting section.

The book structure we have designed has an implicit graph, where the sections are the nodes, and the edges are formed by the `referencedBookSections`. To check the last condition, we must traverse the graph from the starting section and check that all the sections in `bookSections` can be reached.

Implement the public method `boolean checkCorrectness()` that returns true if the three conditions are satisfied, false otherwise.

End of examination paper