3 hours

THE UNIVERSITY OF MANCHESTER

Introduction to Programming II

Instructions for candidates

Answer ALL questions and upload your solutions to Gradescope. The examination is divided into TWO Sections and is worth a total of 40 marks:

- Section 1 is worth 25 marks (62.5%)
- Section 2 is worth 15 marks (37.5%)

You should write all of your code in a single package named **`kingdom`**.

Uploading your responses to Gradescope

You should upload a zip file containing your `kingdom` package directory. For example, if the exam requires you to write a class named `Item` in the `kingdom` package, and you've been working on the file:

```
S:\kingdom\Item.java
```

Then you should upload a zip of the **`kingdom`** directory to Gradescope.

You can upload your solutions as many times as you wish during the exam. Each time you upload a solution your mark/test output will be updated.

Your final mark will be determined by the final solution that you upload to Gradescope. You **MUST** upload a solution to Gradescope before the exam ends in order to receive a non-zero mark.

Additional permitted materials

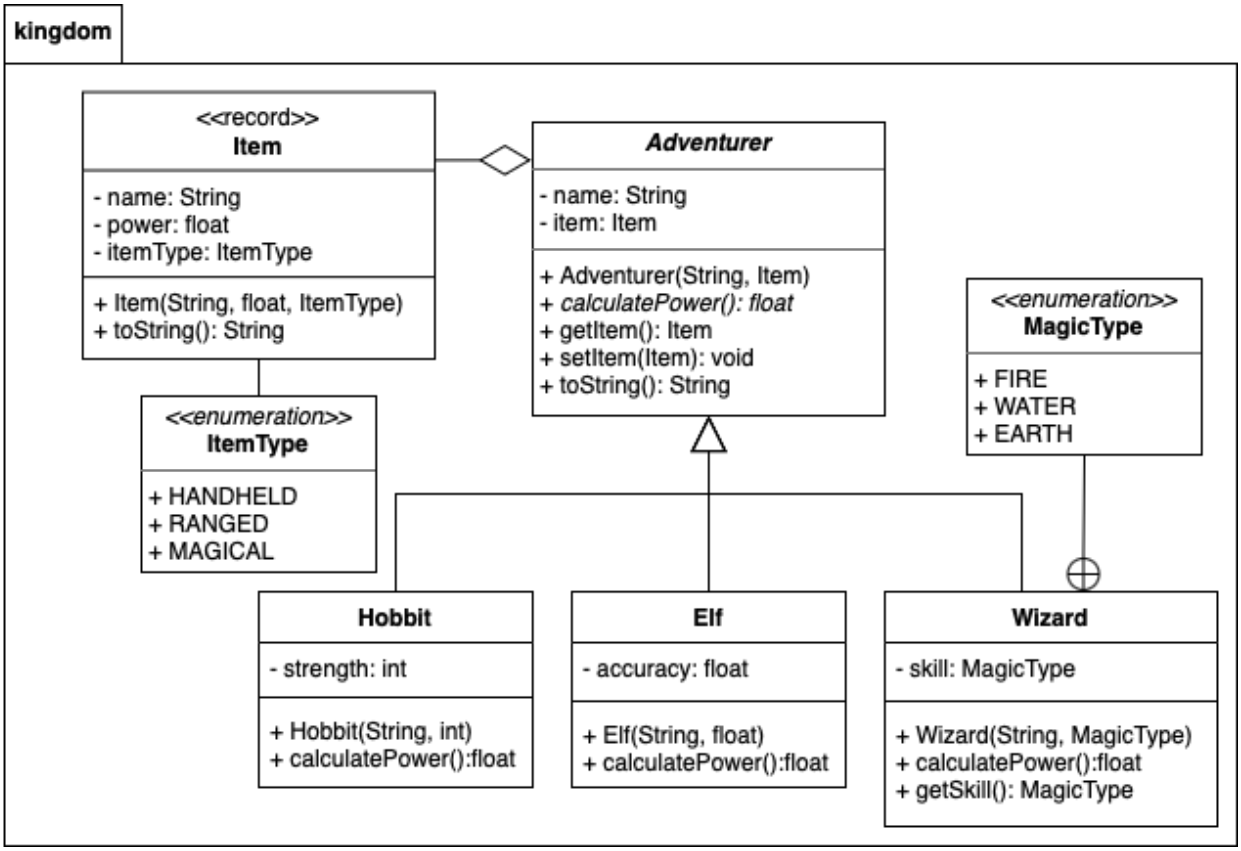
Answer this exam on the computer provided.

The use of calculators is NOT permitted in this exam.

Access to the Java API (and JavaFX APIs where appropriate) will be possible in the exam. Other websites will NOT be available during this exam.

Section 1. Answer ALL PARTS of ALL FOUR questions in this section.

The following UML diagram represents entities involved in a fictional kingdom. All entities are contained in a single package named **`kingdom`**.



Question 1

(a) Implement the `Item` and `ItemType` entities, respecting their specification as `record` and `enum` types respectively. The `toString` method of `Item` should return a string in the following format `"<ItemType>: <name> with <power> power"` (e.g. `"HANDHELD: stick with 50 power"`).

[2 marks]

(b) Implement the `Adventurer` class (i.e. all attributes, methods and the constructor). The constructor should set the two private attributes of `Adventurer` in accordance with the passed parameters.
You should assume that `name` Strings will be given in the form `"<firstname> <lastname>"`, e.g. `"Thorin Clubhand"`, and the `toString` method will return

the name in this same format.

You should also assume that passing `null` as the second parameter is the correct way to create an **`Adventurer`** who does not currently hold an **`Item`**.

[2 marks]

(c) Implement the subclasses of the **`Adventurer`** class, and the **`MagicType`** enum. Note that:

- The **`MagicType`** `enum` is defined within the **`Wizard`** class.
- Each constructor takes a parameter that matches the instance variable unique to that class, and should initialise the variable to the specified value.
- Each constructor takes one `String` parameter corresponding to the adventurer's name, plus one additional parameter corresponding to their class-specific attribute (i.e., `strength`, `accuracy`, or `skill`). The constructors of all subclasses of `Adventurer` should ensure that when the adventurer is instantiated their `item` should be set to null.
- The constructor in the **`Elf`** class should enforce that `accuracy` is a `float` between `0` and `100` (inclusive). A value that is to high or low should be replaced by the maximum or minimum values respectively (i.e., a call to `new Elf(-7)` should result in the new instance having an accuracy of zero, and a call to `new Elf(700)` should result in the new instance having an accuracy of 100).
- The **`Wizard`** class provides a getter method (`getSkill`) to return the `skill` variable.
- The `calculatePower` method in each of the above classes will differ for each class as following:
  - For **`Hobbit`** – the `power` of their `item` should be multiplied by their `strength`.
  - For **`Elf`** – the `accuracy` of their use of the `item` is a percentage value, and the result is the application of this percentage to the `power` of their `item`. E.g. if an `item` has `50` `power` and `accuracy` is `50`, then `calculatePower` will return `25`.

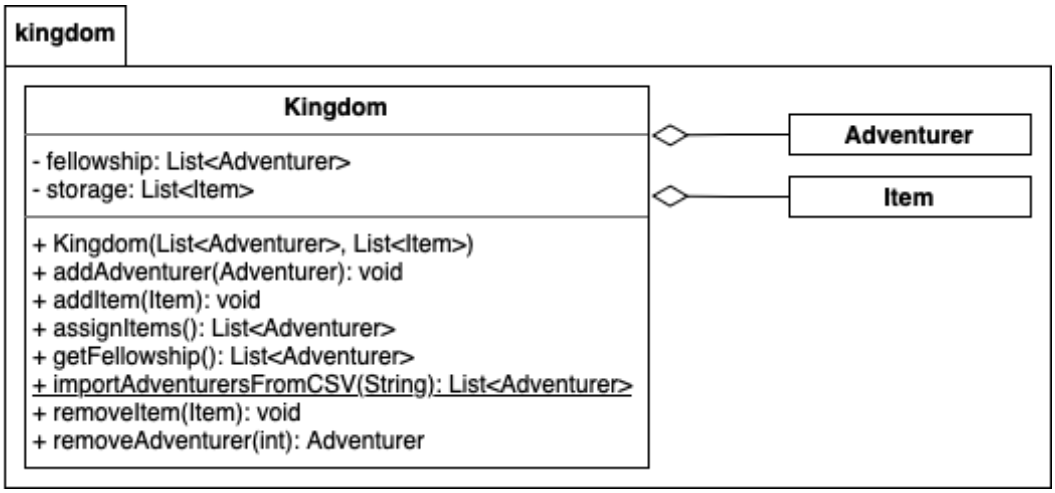
- For `Wizard` – the power of the item will be multiplied by `1` if they use `MagicType.WATER`, by `2` if they use `MagicType.EARTH`, and by `3` if they use `MagicType.FIRE`.

In all cases, the `power` of an adventurer's `item` should be treated as `0` if they do not have an item.

[3 marks]

Question 2

The following UML represents the kingdom itself, and its relationships to classes in the previous UML diagram. Note that where an ancestor type is specified in the UML (in this case, `List`), this exact type must be used (a different type may be used for instantiation).



(a) Implement the class signature, attributes, constructor and `getFellowship` method of the `Kingdom` class.

[1 mark]

(b) Implement the `addAdventurer`, `addItem`, `removeItem`, and `removeAdventurer` methods in the `Kingdom` class.

Note that:

- `addItem` should `append` the specified `Item` to `storage`, whilst `removeItem` removes the specified `Item`.

- `addAdventurer` appends the specified **`Adventurer`** to the end of `fellowship`, whilst `removeAdventurer` removes and returns the **`Adventurer`** at the specified index.

[2 marks]

(c) Implement the `assignItems` method of the **`Kingdom`** class. When `assignItems` is called, each **`Adventurer`** in the **`Kingdom`** who does not have an item will be assigned the first **`Item`** in the `storage` that is compatible with their class. Specifically:

- A **`Hobbit`** should be assigned the first `ItemType.HANDHELD` **`Item`**
- An **`Elf`** should be assigned the first `ItemType.RANGED` **`Item`**
- A **`Wizard`** should be assigned the first `ItemType.MAGICAL` **`Item`**

Note that:

- `assignItems` should attempt to allocate an **`Item`** to each **`Adventurer`** in the `fellowship` **in order**, and adventurers can still exist in the fellowship without items.
- `assignItems` returns all adventurers that were assigned an item.

[2 marks]

(d) Implement the `importAdventurersFromCSV` of the **`Kingdom`** class. This method should read in the file at the specified relative or fully qualified file path and return a `java.util.List` of **`Adventurer`** instances based on its content. Note that:

- You can assume that, if it exists and is readable, the file will always be a valid CSV file that describes zero or more adventurers, one per line. None of the values contain the comma character ',' and you do NOT need to handle "quoted" fields. Values may contain spaces.
- Every line of the CSV file contains three fields. The first line of the CSV contains the column headers (`Name,AdventurerType,Extra`) and should be ignored. The first field of each line thereafter contains the adventurer's name, and the second field their `AdventurerType`. The third field varies with the `AdventurerType` and reflects the additional

parameter passed to that `Adventurer` subclass's constructor (i.e. `strength` for hobbits, `accuracy` for elves and `skill` for wizard).

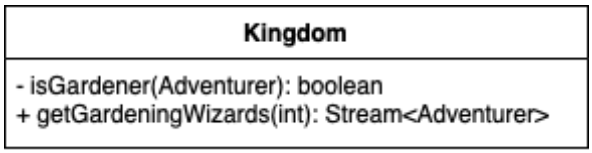
- If the `AdventurerType` contains a string that is not one of the three specified subclasses (`Hobbit`, `Elf`, `Wizard`), or if the third field contains a value that cannot be cast to the appropriate type (e.g. a float value of "51.1.1") then the constructor should throw a `java.lang.IllegalArgumentException`.
- The return value should be ordered identically to the original CSV representation.
- The method should **claim** all errors that occur from the underlying file IO operations (using the `throws` keyword).

[4 marks]

Question 3

The ruler of the kingdom wants to get all of the top wizards that perform water or earth magic to tend to his garden. The wizards can be recruited even if they do not have any items. You are tasked with finding these wizards.

The following UML describes two methods that will be used to achieve the above functionality, these methods should be added to the existing `Kingdom` class.



(a) Implement the `isGardener` method **IN THE `Kingdom` CLASS**. The method's return value should indicate whether or not the specified `Adventurer` is a `Wizard` with a `MagicType` that is either `MagicType.WATER` **OR** `MagicType.EARTH` (returns `true`) or not (returns `false`).

[1 marks]

(b) Implement the `getGardeningWizards` of the `Kingdom` class. This method should use the existing `isGardener` method **AS PART OF A STREAM**, in order to return all of the gardening wizards in the `fellowship` whose `calculatePower` return value is equal to or greater than the `int` parameter.

The Stream elements should be ordered in the same order as the `fellowship`. If no suitable wizards were found, the method should return an empty `Stream`.

For example, given a **`Kingdom`** initialised as follows:

```
Kingdom kilburn = new Kingdom();
kilburn.addAdventurer(gandalf);          // Wizard - Power is 999 FIRE
kilburn.addAdventurer(stianusWizzard);   // Wizard-  Power 350 WATER
kilburn.addAdventurer(sarahzamWizard);   // Wizard - Power 500 EARTH
kilburn.addAdventurer(bilboHobbit);      // Hobbit – Power 600
kilburn.addAdventurer(davidCopperfield); // Wizard- Power 100 WATER
```

A subsequent call to `kilburn.getGardeningWizards(350)` will return a `Stream` containing:

```
stianusWizzard
sarahzamWizard
```

[3 marks]

Question 4

The `fellowship` in the **`Kingdom`** needs feeding. In order to do that you need to add a `feedFellowship` method to **`Kingdom`** class. Note that where an ancestor type is specified in the UML (in this case, **`Queue`**), this exact type must be used (a different type may be used for instantiation).

Kingdom
+ feedFellowship(): Queue<Adventurer>

Adventurers eat based on their classes. **`Hobbit`**s will eat first, elves second, **`Wizard`**s third, and then **`Hobbit`**s eat again (hobbits always eat twice). Within each class, adventurers eat in the order they were added to the `fellowship`. The return value of `feedFellowship` is a `java.util.Queue` of adventurers in the order that they were fed.

For example, given a **`Kingdom`** initialised as follows:

```
Kingdom kilburn = new Kingdom();
kilburn.addAdventurer(stianusElf);
kilburn.addAdventurer(sarahzamWizard);
kilburn.addAdventurer(davosHobbit);
kilburn.addAdventurer(bilboHobbit);
kilburn.addAdventurer(legolasElf);
```

A subsequent call to `kilburn.feedFellowship()` should return a `java.util.Queue` in the following order:

```
davosHobbit
bilboHobbit
stianusElf
legolasElf
sarahzamWizard
davosHobbit
bilboHobbit
```

Implement the `feedFellowship` method.

[5 marks]