ShowPedia
Please don't interrupt my binge watching with silly questions

Object Oriented Programming
2nd Project, version 0.9.1 – 2019-05-10 @ 20h30
Contact: mgoul@fct.unl.pt

```
Modified text is presented in red
Modified commands or outputs are presented in green
```

Important remarks

Deadline: until 23h59 (Lisbon time) of May 31, 2019.

Team: this project is to be MADE BY GROUPS OF 2 STUDENTS.

Deliverables: Submission and acceptance of the source code to **Mooshak** (score > 0). See the course web site for further details on how to submit projects to Mooshak.

To include in the submitted code: Your submission to Mooshak should include:

- The definition and verification of invariants related to the class representing human actors and, in particular to a persons ancestors, descendants and siblings.
- Unit tests to verify the correct functioning of the list of episodes in which a given character actively participates. This implies developing JUnit tests for the methods of the interface representing a character that are related to the list of episodes in which that character participates.

Recommendations: We value the documentation of your source code, as well as the usage of the best programming style possible and, of course, the correct functioning of the project. Please carefully comment both interfaces and classes. The documentation in classes can, of course, refer to the one in interfaces, where appropriate. Please comment the methods explaining what they mean and defining preconditions for their usage. Students may and should discuss any doubts with the teaching team, as well as discuss this project with other students, but may not share their code with other colleagues. This project is to be conducted with full respect for the Code of Ethics available on the course web site.

1 Development of the application *ShowPedia*

1.1 Problem description

Every now and then you are watching TV and someone else starts asking a lot of annoying questions like "Wait, is he dating his aunt?", or "Why are you so upset? They would never kill his character! He is one of the most important ones in the whole show, right?"

The goal of this project is to develop an application that one can use to find out trivia about your favorite Tv show. Here is how it works: You will have a database with the most relevant information on each character, their relationships, and most significant events at your fingertips. You just need pass your device to whoever is distracting you with all these *naïf* questions making you lose that crazy cool Easter egg just because. Sit back and enjoy the show while the other person is distracted looking up information.

The application should support several *shows*. Each show has a *unique name*, and is composed of one or more *seasons*. Each *season* has one or more *episodes*. Apart from the *season* and *episode numbers*, an *episode* is also characterized by its *name*. The show has also a set of characters. Each *character* has a *unique name within the show*, zero or more *parents* (these may include natural parents, as well as adoptive ones, and we do not put an upper boundary to their number, just in case), zero or more children, zero or more other characters with whom that character is, or has been, romantically involved, a list of appearances (essentially, the episodes in which the character participated), and a set of famous quotes by that character. Each appearance may include several special events involving the character (and others, where appropriate). Last, but not the least, it is also possible to know who is the actor playing a particular character is. Note that the same actor can play more than one role in the same show.

Increasingly, we have virtual actors, as well. These are CGI-generated characters such as direwolves, dragons, lions and even people, just to name a few, in your favorite tv shows. We will have information about these, as well.

To make the application more usable, the user can select which show to focus on. His operations will then be performed within the scope of that show, until the user decides to change the focus to a different show.

The application allows the user to add data to it, making this a sort of personalized wiki (What I Know Is) of that user on the shows the user enjoys watching. While in this first prototype you will not have to develop a persistence module (e.g. a database) for all the data manipulated by this application, you should assume that you will end up manipulating a large set of series, each with a few seasons, a potentially large set of characters an even larger set of actors

2 Commands

In this section we present all the commands that the system must be able to interpret and execute. In the following examples, we differentiate `text written by the user` from the feedback written by the program in the console. You may assume that the user will make no mistakes when using the program other than those described in this document. In other words, you only need to take care of the error situations described here, in the exact same order as they are described.

Commands are case insensitive. For example, the `exit` command may be written using any combination of upper and lowercase characters, such as `EXIT`, `exit`, `Exit`, `exIT`, and so on. In the examples provided in this document, the symbol ↵ denotes a change of line. Whenever a command is expected from the user, the program should present a *prompt* denoted by the symbol >, which is always followed by a white space. The user input is written right after that white space, in the same line.

If the user introduces an unknown command, the program must write in the console the message Unknown command. Type help to see available commands. For example, the non existing command `some random command` would have the following effect:

```
> some random command↵
Unknown command. Type help to see available commands.↵
```

Several commands have arguments. You may assume that the user will only write arguments of the correct type. However, some of those arguments may have an incorrect value. For that reason, we need to test each argument exactly by the order specified in this document. Arguments will be denoted **`with this style`**, in their description, for easier identification.

2.1 `exit` command

Terminates the execution of the program. This command does not require any arguments. The following scenario illustrates its usage.

```
> exit↵
Bye!↵
```

This command always succeeds. When executed, it terminates the program execution.

2.2 `help` command

Shows the available commands. This command does not require any arguments. The following scenario illustrates its usage.

```
> help↵
currentShow - show the current show↵
addShow - add a new show↵
switchToShow - change the context to a particular show↵
addSeason - add a new season to the current show↵
addEpisode - add a new episode to a particular season of the current show↵
addCharacter - add a new character to a show↵
addRelationship - add a family relationship between characters↵
addRomance - add a romantic relationship between characters↵
addEvent - add a significant event involving at least one character↵
addQuote - add a new quote to a character↵
seasonsOutline - outline the contents of the selected seasons for the selected show↵
characterResume - outline the main information on a specific character↵
howAreTheseTwoRelated - find out if and how two characters may be related↵
famousQuotes - find out which character(s) said a particular quote↵
alsoAppearsOn - which other shows and roles is the same actor on?↵
mostRomantic - find out who is at least as romantic as X↵
kingOfCGI - find out which company has earned more revenue with their CGI virtual actors↵
help - shows the available commands↵
exit - terminates the execution of the program↵
```

This command always succeeds. When executed, it shows the available commands.

2.3 `currentShow` command

Show which is the current show. The command does not require any arguments. It prints the show name, followed by the number of available seasons and total episodes count.

```
> currentShow↵
Friends. Seasons: 10 Episodes: 236↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).

The following example illustrates an interactive session where this error message is presented.

```
> currentShow↵
No show is selected!↵
```

2.4 addShow command

Add a new show. The command receives as arguments the show's **unique name**. In case of success, the new show starts with a first season but no episodes and no characters. You will need to add those later. The current show becomes the newly added one. In general, we expect a very large number of shows.

```
> addShow↵
The Big Bang Theory↵
The Big Bang Theory created.↵
```

The following may go wrong:

- (1) If there is already a show with the same name, the error message is (Show already exists!).

The following example illustrates an interactive session where this error message would be generated. The problem with the input is highlighted in **red**.

```
> addShow↵
The Big Bang Theory↵
The Big Bang Theory created.↵
> addShow↵
After Life↵
After Life created.↵
> addShow↵
The Big Bang Theory↵
Show already exists!↵
```

2.5 switchToShow command

Changes the context to a particular show. The command receives 1 parameter: the **show name**. The application changes its current show to the one which was just mentioned:

```
> currentShow↵
After Life. Seasons: 1 Episodes: 6↵
> switchToShow↵
Downton Abbey↵
Downton Abbey. Seasons: 6 Episodes: 52↵
```

The following may go wrong:

- If the show to switch to does not exist, the program presents the message Unknown show!.

```
switchToShow↵
IT Crowd (US remake)↵
Unknown show!↵
```

2.6 `addSeason` command

Add a new season to the current show. The command receives no parameters. Seasons are added with a new serial number (the successor of the last season). For every show, the first season is labeled season 1, and so on. In general, there is a low number of seasons.

```
> currentShow↵
The Good Fight. Seasons: 2 Episodes: 23↵
> addSeason↵
The Good Fight. Seasons: 3 Episodes: 23↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).

The following example illustrates an interactive session where this error message is presented.

```
> currentShow↵
No show is selected!↵
> addSeason↵
No show is selected!↵
```

2.7 `addEpisode` command

Add a new episode to a particular season of the show. The command receives a **`season number`**, and an **`episode title`**. The new episode is added to the corresponding season, in the first available serial number. For example, consider episode 4 just added to season **`8`** of the series *Game of Thrones*, entitled **`"The Last of the Starks"`**.

```
> currentShow↵
Game of Thrones. Seasons: 8 Episodes: 70↵
> addEpisode↵
8 The Last of the Starks↵
Game of Thrones S8, Ep4: The Last of the Starks.↵
> currentShow↵
Game of Thrones. Seasons: 8 Episodes: 71↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).
- (2) If the referenced season does not exist, the error message is (Unknown season!)

The following example illustrates an interactive session where this error message is presented.

```
> currentShow↵
No show is selected!↵
> addEpisode↵
1 Pilot↵
No show is selected!↵
> switchToShow↵
1986↵
1986. Seasons: 1 Episodes: 13↵
> addEpisode↵
2 Tom Cruise e o Homem de Gelo↵
Unknown season!↵
```

2.8 **addCharacter** command

Add a new character to the show. There are two kinds of actors to consider: `Real` actors and `Virtual` actors. Real actors are characterized by **`character name`**, **`actor name`**, and **`fee by episode`**. The output includes the name of the show, the name of the character, and how many shows the actor playing that role has participated in.

Virtual actors are actors that do not really exist. Frequently, they are developed by a computer-generated imagery (CGI) team. Virtual actors are characterized by **`character name`**, **`company name`** (the name of the company producing the special effects that provides the virtual character) and the **`virtual character cost per season`**, which is a fixed value per season in which the virtual actor is used. Contrary to real actors, virtual actors only participate in one show. The output when creating a virtual actor includes the name of the show and the name of the character (<character> is now part of <show>. This is a virtual actor.).

The command adds that character to the current show. If the actor is new, it is also added to the collection of actors known by the application. Otherwise, that actor's information is updated to reflect her participation in the current show.

In the following example, an actress gets her first role. Before she was no one. From now on, she will be known as Arya Stark. Then, a virtual actor gets a role for the same show.

```
> addCharacter↵
real↵
Arya Stark↵
Maisie Williams↵
150↵
Arya Stark is now part of Game of Thrones. This is Maisie Williams role 1.↵
virtual↵
Nymeria↵
CGI Inc.↵
1000↵
Nymeria is now part of Game of Thrones. This is a virtual actor.↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).
- (2) If the actor kind is unknown (known kinds are **`real`** and **`virtual`**), the error message is (Unknown actor category!).
- (3) If there is already a character with that name in the show, the error message is (Duplicate character names are not allowed!).

- (4) If the fee for the actor is a negative integer, the error message is (Slavery is long gone and this is outrageous!).

The following example illustrates an interactive session where these error messages are presented.

```
> currentShow↵
No show is selected!↵
> addCharacter↵
real↵
Superman↵
Henry Cavill↵
100↵
No show is selected!↵
> switchToShow↵
Game of Thrones↵
Game of Thrones. Seasons: 8 Episodes: 71↵
> addCharacter↵
zombie↵
Lady Stoneheart↵
Michelle Farley↵
100↵
Unknown actor category!↵
> addCharacter↵
real↵
Arya Stark↵
Maisie Williams↵
100↵
Duplicate character names are not allowed!↵
> addCharacter↵
real↵
Lannister soldier↵
Ed Sheeran↵
-10↵
Slavery is long gone and this is outrageous!↵
```

2.9 `addRelationship` command

Add a new family relationship between characters. The command adds a new family relationship between characters. In a family relationship, the first element is the parent, the second one is the child. Note that through adoption it is perfectly possible for a character to have more than 2 known parents. For example when Catelyn Stark adopted Jon Snow, she became his fourth parent (his 2 biological parents, Ned Stark and, finally, Catelyn Stark). The output message of a new successful relationship established is (<**Character**> has now <**n**> parent(s).).

```
> addRelationship↵
Catelyn Stark↵
Jon Snow↵
Catelyn Stark has now 6 kids. Jon Snow has now 4 parent(s).↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).

- (2) It is not possible to establish a relationship with the same character. The error message is (<character> cannot be parent and child at the same time!)
- (3) If at least one of the characters does not exist, the relationship is not created (Who is <character>?).

The following example illustrates an interactive session where these error messages are presented.

```
> addRelationship↵
No show is selected!↵
> switchToShow↵
Will and Grace↵
> addRelationship↵
Bobbi Adler↵
Bobbi Adler↵
Bobbi Adler cannot be parent and child at the same time!↵
> addRelationship↵
Bobbi Adler↵
Princess Leia↵
Who is Princess Leia?↵
```

2.10 `addRomance` command

Add a romantic relationship between characters. In a romantic relationship, we add two elements, establishing a relationship among them. For the sake of flexibility, it is possible for a character to be "coupled" with more than one character at a time. The output message of a new successful relationship established is (<**Character1**> and <**Character2**> are now a couple.). For example, the following commands would put Cersei Lannister in a relationship with her husband (Robert Baratheon) and her brother (Jamie Lannister).

```
> couple↵
Cersei Lannister↵
Robert Baratheon↵
Cersei Lannister and Robert Baratheon are now a couple.↵
> addRelationship↵
couple↵
Cersei Lannister↵
Jamie Lannister↵
Cersei Lannister and Jamie Lannister are now a couple.↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).
- (2) It is not possible to establish a relationship with the same character. The error message is (<character> cannot be in a single person romantic relationship!)
- (3) If at least one of the characters does not exist, the relationship is not created (Who is <character>?).

The following example illustrates an interactive session where these error messages are presented.

```
> addRelationship↵
No show is selected!↵
> switchToShow↵
The Big Bang Theory↵
> addRelationship↵
Sheldon Cooper↵
Sheldon Cooper↵
Sheldon Cooper cannot be in a single person romantic relationship!↵
> addRelationship↵
Sheldon Cooper↵
Paige↵
Who is Paige?↵
```

2.11 `addEvent` command

Add a significant event involving at least one character. The command receives a **`description of the event`**, the **`season number`**, the **`episode number`**, the **`number of involved characters`**, and the **`list of involved characters`**. It associates all the involved characters with the event, and the event with all the involved characters.

```
> addEvent↵
Jamie pushes Bran to a deadly fall.↵
1 1 2↵
Jamie Lannister↵
Bran Stark↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).
- (2) If the season does not exist (<Show> does not have season <season number>!).
- (3) If the episode does not exist in that season (<Show> S<Season number> does not have episode <Episode number<!).
- (4) If at least one of the characters does not exist, the event is not created (Who is <character>?).

The following example illustrates an interactive session where some of these error messages are presented.

```
> addEvent↵
Cersei and the Mountain fall in love.↵
9 3 2↵
Cersei Lannister↵
The Mountain↵
Game of Thrones does not have season 9!↵
> addEvent↵
April 1st spoiler:  Arya dies in season 8 episode 7.↵
8 7 1↵
Arya Stark↵
Game of Thrones S8 does not have episode 7!↵
> addEvent↵
Christina Aguilera drinks a Starbucks coffee.↵
8 4 1↵
Christina Aguilera↵
Who is Christina Aguilera?↵
```

2.12 `addQuote` command

Add a new quote to a character. The command receives the information on the **`season`**, **`episode`**, **`character`** and a **`quote`** and adds that quote to the character.

```
> addQuote↵
1 1↵
Jamie Lannister↵
The things I do for love.↵
Quote added.↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).
- (2) If the season does not exist (<Show> does not have season <season number>!).
- (3) If the episode does not exist in that season (<Show> S<Season number> does not have episode <Episode number>!).
- (4) If the character does not exist, the quote is not created (Who is <character>?).

In the following example, some of these problems are illustrated:

```
> addQuote↵
14 3↵
Sheldon Cooper↵
Bazinga!↵
The Big Bang Theory does not have season 14!↵
> addQuote↵
8 34↵
Sheldon Cooper↵
Bazinga!↵
The Big Bang Theory S8 does not have episode 34!↵
> addQuote↵
8 12↵
Sheldon Bloopper↵
Bazinga!↵
Who is Sheldon Blooper?↵
```

2.13 seasonsOutline command

Outline the contents of the selected seasons for the selected show. The command receives two parameters: the **starting season** and the **ending season** (e.g. if the starting season is 3 and the ending season is 5, the command will outline seasons 3, 4 and 5). For each season, the command presents, episode by episode, the events registered in the application, by order of insertion in the application. In the following example, you can see the outline for the series *After Life*. For the sake of this example, assume that nothing important happened in episode 5.

```
> seasonsOutline↵
1 1↵
After Life↵
S1 Ep1:↵
Tony tries to adapt to life after his wife dies.↵
Tambury Gazette, a free newspaper company, hires Sandy.↵
S1 Ep2:↵
Tony feels like he has nothing to lose.↵
Tony babysits George.↵
Tony begins to somewhat get along with Sandy.↵
S1 Ep3:↵
Tony hires Roxy to complete a task.↵
Tony gets in a really bad mood after visiting a stand-up gig.↵
S1 Ep4:↵
Matt organizes Tony a blind date.↵
Tony goes to a disaster blind date.↵
S1 Ep5:↵
S1 EP6:↵
Tony acknowledges those around him in an effort to not be depressed.↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).
- (2) If the seasons interval is invalid, the error message is (Invalid seasons interval!).

The following example illustrates an interactive session where some of these error messages are presented. Assume the selected show has only 3 seasons.

```
> seasonsOutline↵
1 8↵
Invalid seasons interval!↵
> seasonsOutline↵
0 2↵
Invalid seasons interval!↵
> seasonsOutline↵
3 1↵
Invalid seasons interval!↵
```

2.14 `characterResume` command

Outline the main information on a specific character. The command receives as parameter the `name of the character`. It presents some biographic remarks (parents, kids, siblings, and romantic relationships), followed by the main events related to that character. The biographic remarks are presented by order of introduction in the system. When a particular category in the biographic remarks is empty (e.g. the character has no kids) this is denoted by (None.).

```
> characterResume↵
Viserys Targaryen↵
Parents:↵
Kids: None.↵
Siblings: Daenarys Targaryen↵
Romantic relationships: None.↵
S1 Ep1:↵
Viserys Targaryen sells his sister to the Dothraki warlord Khal Drogo.↵
Khal Drogo offers Viserys an army in exchange for his sweet little syster Daenarys.↵
Viserys wins the Brother of the year award.↵
S1 Ep3:↵
Daenerys learns that she is pregnant and begins to stand up to Viserys.↵
S1 Ep4:↵
A frustrated Viserys clashes with his newly empowered sister.↵
S1 Ep6:↵
Viserys becomes enraged with Drogo for not honoring his promise and threatens to kill Daenerys unborn child.↵
Drogo kills Viserys by pouring molten gold on his head.↵
```

2.15 `howAreTheseTwoRelated` command

Find out if and how two characters may be related. This command determines if and how one character is an ancestor, or descendant, of another. The command receives two arguments (the names of the `first` and `second` characters). The command outputs the sequence starting with the ancestor and ending with the descendant. The order in which the two arguments are given to the command is arbitrary. In other words, we can have the ancestor as either the `first` or the `second` character. The following examples are taken from the show Modern Family, where we will find out the relationship between Jay Prichett and his grandson, Luke Dunphy. Note that for all practical matters we are considering here the extended notion of parent here. Therefore, Gloria Prichett is Luke Dunphy's grandmother, because she is the stepmother of Luke's mother.

```
> howAreTheseTwoRelated↵
Jay Pritchett↵
Luke Dunphy↵
Jay Prichett; Claire Dunphy; Luke Dunphy↵
> howAreTheseTwoRelated↵
Luke Dunphy↵
Gloria Pritchett↵
Gloria Prichett; Claire Dunphy; Luke Dunphy↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).
- (2) If at least one of the characters does not exist, the relationship does not exist (Who is <character>?).
- (3) If the characters exist, but are the same, the error message is (Like... you know, they are THE SAME character! duuuuh...)

The following example illustrates an interactive session where some of these error messages are presented.

```
> howAreTheseTwoRelated↵
Jay Pritchett↵
Luke Skywalker↵
Who is Luke Skywalker?↵
> howAreTheseTwoRelated↵
Phil Dunphy↵
Phil Dunphy↵
Like... you know, they are THE SAME character! duuuuh...↵
```

2.16 famousQuotes command

Find out which character(s) said a particular quote. The command receives a **quote** and looks up in the current show who said it. More than one character may be known for the same quote. The character names are presented in a comma-separated list by alphabetical order.

```
> famousQuotes↵
Valar morghulis↵
Arya Stark, Jaqen Hghar, Melissandre↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).
- (2) If nobody said the quote, the error message is (First time I hear that!).

In the following example, an unknown quote is looked up.

```
> famousQuotes↵
Julianna Margulies↵
First time I hear that!↵
```

2.17 `alsoAppearsOn` command

Which other shows and roles does this actor appear on? This command searches for other roles played by the same actor in other shows. The command receives the **`name of the character`** and uses it to identify who the actor playing it is. Then, it searches for other roles of that actor. The list of shows should be presented in alphabetic order, one in each line, and include the current show. In the following example, we lookup where else Sophie Turner (the actress playing Sansa Stark in Game of Thrones) has participated.

```
> alsoAppearsOn↵
Sansa Stark↵
Carpool Karaoke: The Series↵
Game of Thrones↵
The Thirteenth Tale↵
```

The following may go wrong:

- (1) If there is no selected show, the error message is (No show is selected!).
- (2) If there is no character in the show with that name, the error message is (Who is <character>?).

```
> alsoAppearsOn↵
Ser Duncan the Tall↵
Who is Ser Duncan the Tall?↵
```

2.18 `mostRomantic` command

Is there anybody more romantic than X. The command receives the **`name of the actor`** that you want to use as romantic icon and lists all actors which are at least as romantic as that actor. Actors are presented by descending order of romantic relationships count. Note that this search is done in all shows and that the actors names, rather than characters names should be listed. The comparison is done following a fixed set of criteria (in case of a tie until a given criterion, the next criterion is used to discern between actors, and so on, until an order relationship is effectively established):

1. Number of romantic relationships (more is more romantic)
2. Number of shows (less is more romantic, as it means a higher average number of romantic relationships)
3. Number of shows with romantic relationships (more is more romantic, because it shows a more consistent romantic nature)
4. Alphabetic order (earlier in the alphabet is more romantic - ok, ok, this is just to have an unambiguous criterion, in case all the others fail)

In the following example we answer the question Catherine Heigl is making to her mirror every day since she lost a casting to Charlize Theron, for Snow White: *"mirror mirror on the wall, who is the most romantic of them all?"* Oddly enough, the mirror never answers back. But your application would. It turns out that Simon Baker has participated in less shows than

Katherine Heigl, but has the same number of romantic relationships giving him a better romantic ranking. And there is Morena Baccarin, too. Tough luck, Miss Heigl...

```
> mostRomantic↵
Katherine Heigl↵
Morena Baccarin 6↵
Simon Baker 5↵
Katherine Heigl 5↵
```

The following may go wrong:

- (1) If no characters with romantic relationships exist, the error message is (Love is not in the air. :-().
- (2) If the actor provided as argument does not exist, the error message is (Who is <actor>?))

In the following example, not a single actor in the application is considered as romantic as Diogo Morgado. Not even Diogo Morgado is as romantic as himself. He has 0 romantic relationships in the users database.

```
> mostRomantic↵
Diogo Morgado↵
Love is not in the air. :-(↵
```

In the following example, suppose Claudia Schiffer is not an actress.

```
> mostRomantic↵
Claudia Schiffer↵
Who is Claudia Schiffer?↵
```

2.19 `kingOfCGI` command

Which company has the highest revenue with their CGI virtual actors? This command receives no arguments. It simply presents the company with the highest revenue on CGI virtual actors. This is independent of having (or not) a particular show selected. The revenue is computed using the fee for the character per season, and the number of seasons in which the character participates. This is how to sort the candidate companies.

1. Total fees collected (more is better)
2. Number of characters by that company (less is better, as it minimizes costs)
3. Alphabetic order of the company name (just to make sure we can present a single answer)

In the following example, the company "Lucas Arts Inc." leads this particular ranking with earnings of 2300000.

```
> KingOfCGI↵
Lucas Arts Inc. 2300000↵
```

The following may go wrong:

- (1) If no virtual characters exist, the error message is (This is the real thing, this is art!) .

Here is an example:

```
> KingOfCGI↵
This is the real thing, this is art!↵
```

3 Developing this project

Your program should take the best advantage of the elements taught up until now in the Object-Oriented programming course. You should make this application as **extensible as possible**.

You can start by developing the main user interface of your program, clearly identifying which commands your application should support, their inputs and outputs, and preconditions. Then, you need to identify the entities required for implementing this system. Carefully specify the **interfaces** and **classes** that you will need. You should document their conception and development using a class diagram, as well as documenting your code adequately, with Javadoc.

It is a good idea to build a skeleton of your `Main` class, to handle data input and output, supporting the interaction with your program. In an early stage, your program will not really do much. Remember the **stable version rule**: do not try to do everything at the same time. Build your program incrementally, and test the small increments as you build the new functionalities in your new system. If necessary, create small testing programs to test your classes and interfaces.

Have a careful look at the test files, when they become available. You should start with a really bare bones system with the `help` and `exit` commands, which are good enough for checking whether your commands interpreter is working well, to begin with. Step by step, you will incrementally add functionalities to your program and test them. **Do not try to make all functionalities at the same time. It is a really bad idea.**

In this project you will handle several collections. Use the most appropriate collection for each situation. You are expected to use the Java library collections.

Last, but not the least, **do not underestimate the effort for this project**.

4 Submission to Mooshak

To submit your project to mooshak, please register in the mooshak contest POO1819-TP2 and follow the instructions that will be made available on the moodle course website.

4.1 Command syntax

For each command, the program will only produce one output. The error conditions of each command have to be checked in the exact same order as described in this document. If one of those conditions occurs, you do not need to check for the other ones, as you only present the feedback message corresponding to the first failing condition. For example, if you attempt to create a drone with an invalid range, the remaining conditions do not need to be checked. However, the program does need to consume all the remaining input parameters, even if they are to be discarded.

4.2 Tests

The Mooshak tests verify incrementally the implementation of the commands. They will be be made publicly available on Moodle. When the sample test files become available, use them to test what you already have implemented, fix it if necessary, and start submitting your partial project to mooshak. Do it from the start, even you just with the exit and help commands. Also, you will soon learn how to make unit tests. You are encouraged to use those to test your classes, too. You are only required to submit some of them, as specified earlier. So, in the end, you will probably have several tests you can verify and validate your work with. Good luck!