School of Science

Programming1 COSC 1073, S2, 2019

Assignment Part B: ASCIIBot Robot Implementation

✾	**Assessment Type:** Individual assignment; no group work. Submit online via Canvas→Assignments→Assignment Part B. Marks are awarded for meeting requirements as closely as possible according to section 2 and the supplied rubric. Clarifications/updates may be made via announcements/relevant discussion forums.
📅	**Due date:** Due 06:00 PM Fri. 18th October 2019. Late submissions are handled as per usual RMIT regulations - 10% deduction (2 marks) per day. You are only allowed to have 5 late days maximum unless special consideration has been granted.
⌄⌄	**Weighting:** 20 marks (20% of your final semester grade)

1. Overview

This specification builds upon your part A solution however it can be done without a working part A since there is no error checking or collision detection and the provided ASCIIBot skeleton code responds to manual keyboard control in the same way as the part A `RobotImpl`. Nevertheless, if necessary you should seek help in the tutelab/consultation to finish assignment part 1 since it is a core requirement of this course that you should be able to work with basic control structures. Furthermore, you will require a solid grasp of basic java programming to effectively use classes and interfaces in assignment part 1!

NOTE: The supplied `Robot P1 Part B/` eclipse Project is dependent on the `Robot P1/` project via the build path so this project should be available and open in the workspace. You should use your own `Robot P1/` solution project however the project will also work with the supplied startup project for part A.

2. Assessment Criteria

As well as functional correctness (ASCIIBot behaviour matches video requirements) you will also be assessed based on the following code quality requirements:

- Follows the provided design in terms of class structure.
- Uses meaningful / descriptive identifiers (eg variable and method names).
- Demonstrates understanding of local variables versus class attributes and prefer local scope where possible.
- Demonstrates the use of defined constants in Control.java (Rather than using magic numbers)
- Avoids code repetition. (THIS IS THE MOST IMPORTANT ONE!)
- Well encapsulated (use of private methods where appropriate etc.)
- High Cohesion/Low Coupling
- Appropriate use of comments (but remember that easily understandable code is better than a comment).
- Include a comment at the top of `ASCIIBot.java` class stating your name and student number.

3. Learning Outcomes

This assessment is relevant to the following Learning Outcomes:

This assignment addresses the following learning outcomes from the Course Guide

CLO 1: Solve simple algorithmic computing problems using basic control structures and Object-Oriented Techniques.
CLO 2: Design and implement computer programs based on analysing and modelling requirements.
CLO 3: Identify and apply basic features of an Object-Oriented programming language through the use of standard Java (Java SE) language constructs and APIs.
CLO 4: Identify and apply good programming style based on established standards, practices and coding guidelines.
CLO 5: Devise and apply strategies to test the developed software.

Ok so now you are going to do your own complete Robot program! You are no longer going to use the supplied `RobotImpl` but are instead going to write your own implementation of the `Robot` interfaces called the `ASCIIBot`. Well ok not from scratch, since we have given you a fair bit of code to get you started :)

Don't panic you don't have to deal with real graphics for this assignment since we are going old school and are using the Lanterna terminal emulator package to allow you to do basic ASCII drawing. The `Robot P1 Part B/` Eclipse project includes the `lanterna-3.0.1.jar` file and includes code which shows you how to initialise and draw characters at a specific x,y (col/row) co-ordinate. You do not need to, but if you are interested you can find more information about other extra functionality of Lanterna here:

http://mabe02.github.io/lanterna/apidocs/3.0/
https://mvnrepository.com/artifact/com.googlecode.lanterna/lanterna/3.0.1

NOTE: The primary specification is the supplied video "PartB Solution.mp4" which shows the full behaviour you should reproduce.

You have been given skeleton code that provides all of the required classes and interfaces.

The only new interface beyond assignment part A is the following Drawable interface.

```
public interface Drawable
{
        public abstract void draw(SwingTerminalFrame terminalFrame);
}
```

Along with the other provided classes, this gives you a basic design where each of the components of the robot environment is able to draw itself to the terminal by implementing the `draw()` methods. You may also find it useful to include an **extra** abstract class as follows (this is how we did it and it reduces some code duplication):

```
public abstract class AbstractItem implements Drawable
```

If you do this then `Bar`, `Block` and even the `Arm` class can extend `AbstractItem` instead of implementing the `Drawable` interface directly.

Skeleton code for all of these classes is provided in the supplied *Robot P1 Part B* eclipse project. It also manually draws a single 'bar' to show you Lanterna code in action since this is the only part of the assignment that is not specified by standard Java and described in the standard Java API docs.

In addition to drawing the various components (bars, blocks and arm segments) your `ASCIIBot` class will need to keep track of the current position of all bars and blocks as it responds to commands from the `Robot` interface.

However it does **NOT** need to:

- Do any control of its own (this is still done by the `RobotControl` class) or manually using the same key bindings as assignment part A
- Do any error checking. i.e. if invalid commands are given your code will either ignore them or drop blocks in the wrong position, morph through objects etc. i.e. we will test with correct control code.

Suggestions on How to Proceed

To get started you should experiment and familiarise yourself with the Lanterna `setCursorPosition()` and `putCharacter()` methods. They are straight forward i.e. move to a col (x) and row (y) position and then draw a character. An example is provided in the `ASCIIBot.demoDraw()` method. NOTE: Screen co-ordinates are 0,0 is top left so you need to do some translation!

Next you should implement the `draw()` method for the `Block` and `Bar` classes so that they can be drawn at a fixed arbitrary position (no need to worry about moving them yet). Use the colors and number the bar size as shown in the video.

Next do the same for the `Arm` class. This one is a bit more complicated since drawing the arm involves drawing all three segments. In fact you can choose to have a separate class for each arm but for simplicity a single class works fine.

Once the three robot environment components (`Arm`, `Bar`, `Block`) are able to draw themselves, the final and most challenging piece of the puzzle is to keep track of the internal state of the arm, bars and blocks and draw them in the correct position.

i.e. first handle the `init()` case when the bars and block arrays are passed in and draw them at the correct starting position. This is also the right place to initialise all of the data structures that will be used to keep track of positioning in the robot environment when you perform subsequent moves.

Finally for each movement command (`up()`, `down()`, `pick()` `drop()` etc .) keep track of the positional changes (using appropriate variables/data structures) and redraw the entire screen.

You will run your `ASCIIBot` using the provided `main()` method:

```java
public static void main(String[] args)
{
        new RobotControl().control(new ASCIIBot(),null, null);
}
```

This simply creates a new instance of the `RobotControl()` class from part A and executes the control method which will then call `init()` and execute commands on your ASCIIBot.

NOTE: There is no special trick to the animation/movement. But rather every time a robot movement command is issued, you redraw everything at the updated co-ordinates. If you are clever you can do this polymorphically with `draw()`!

The final requirement is to draw the numbering for the three `Arm` segments noting that since only a single digit is shown, the value of '+' is drawn for any value > 9 for any given segment.

IMPORTANT: As with Part A do not change any of the provided interfaces such as `Control`, `Robot` or `RobotMovement` since we will rely on these for testing using our own `RobotControl` implementation. Additionally Part B is compatible with your Part A `RobotControl` implementation.

Summary

Write Object-Oriented code using classes, interfaces, loops, conditionals and methods to reproduce the behaviour shown in the video **PartB Solution.mp4** and elaborated above. NOTE: All of the robot colors, sizes etc. follow part A for consistency. To obtain full marks you must implement the provided methods in the provided classes. If these are unused then you are doing something wrong so please seek help!

Manual Robot Control (Testing Only No Marks)

For testing purposes you can control the robot arm manually using the following keys. This will help you determine when collisions occur etc. before or while writing control code.

Robot Command	Key(s)
up()	Page Up .or [
down()	Page Down or]
extend()	Right Arrow
contract()	Left Arrow
raise()	Up arrow
lower()	Down arrow
pick()	Home or P
drop()	End or D
Speed up (1)	NumPad +
Slow down (1)	NumPad -
Speed up (5)	NumPad *
Slow down (5)	NumPad /

5. Referencing and third party code exclusion

- You are free to refer to textbooks and notes, and discuss the design issues (and associated general solutions) with your fellow students or on Canvas; however, the assignment should be your OWN INDIVIDUAL WORK and is NOT a group assignment.
- You may also use other references, but since you will only be assessed on your own work you should NOT use any third-party packages or code (i.e. not written by you) in your work.

6. Submission format

The source code for this assignment (i.e. complete compiled **Eclipse project**[1]) should be submitted as a .zip file by the due date. You can use the Eclipse option export->general->archive.

IMPORTANT: SUBMISSIONS WHICH DO NOT IN ANY WAY ADHERE TO THE SPECIFICATION AND PROVIDED CODE WILL RECEIVE A **ZERO MARK**

---

[1] You can develop your system using any IDE but will have to create an Eclipse project using your source code files for submission purposes.

7. Academic integrity and plagiarism (standard RMIT warning)

NOTE: Any discussion of referencing below in the standard RMIT policy is generic and superseded by the third-party code exclusion in section 5.

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods,
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites.

If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.
RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the University website.

8. Assessment declaration

Assignment Project Exam Help

When you submit work electronically, you agree to the assessment declaration.

https://powcoder.com

Add WeChat powcoder