Semester 1 2018
Programming 1 (COSC1073)
PART A Specification – Robot Simulation

Overview

This semester you have a single assignment topic in two parts (Part A and B)

> **Assignment Part A (15%) due 9:00AM Mon. 23rd April 2018**
> **Assignment Part B (20%) due 9:00AM Mon. 21st May 2018**

This specification provides an overview of the problem and supplied code and also specifies the requirements for the Part A submission.

Make sure you CAREFULLY WATCH THE ASSIGNMENT RELATED DEMO VIDEOS on Canvas which show the default startup behaviour (when no code is added) as well as the correct output of a working solution with a diverse data set (mixed `blockHeights[]/barHeights[]` arrays). See section *Description of the Supplied Videos* later in this document for more details.

Next, don't panic about the apparent complexity! The code to do the graphical drawing of the robot environment is provided and the use, and understanding of, the supplied interfaces is not required in detail until assignment B.

The possible operations on the robot are specified by the supplied `Robot` and `RobotMovement` interfaces (`Robot.java` and `RobotMovement.java`). Some additional `CONSTANTS` you should use in your code are specified in the `Control.java` interface.

IMPORTANT: DO NOT CHANGE ANY OF THESE INTERFACES!

PART A Requirements (due 9:00AM Mon. 23rd April 2018).

To complete this assignment you will use the supplied eclipse project `Robot P1/`. It is already set up to execute a simple arm movement loop which you will build upon to create the full solution.

NOTE: The primary requirements specifications are the supplied videos which show the EXACT behaviour you should reproduce.

However, some points worth noting about the displayed behaviour:

INITIALISATION:

After the `init()` method is called with valid array parameters:

- Blocks are placed in order (bottom to top) from the supplied (hard coded) `blockHeights` array, into the far right source column (column 10) (`Control.SRC_COLUMN`).
- Bars are placed in order from the supplied (hard coded) `barHeights` array from left to right (from `Control.FIRST_BAR_COLUMN` to a maximum of `Control.LAST_BAR_COLUMN`).
- **This initialisation is done AUTOMATICALLY by the supplied** `RobotImpl.jar`

BLOCK PLACMENT:

You must reproduce this behaviour by calling methods on the Robot. Further hints are given in the section HOW TO PROCEED below.

- Blocks are *picked* from the source column from top to bottom until there are no blocks remaining. You will need to control the robot movement to do this.
- Blocks are *dropped* (placed) as follows:
  Yellow (Size 1) blocks are placed on the top of column 1 (Dest 1).
  Red (Size 2) blocks are placed on the top of column 2 (Dest 2).
  Blue (Size 3) blocks are placed on top of bars starting at the left most bar (`Control.FIRST_BAR_COLUMN`) and progressing to the right most bar (`Control.LAST_BAR_COLUMN`).
- Blocks are lowered to the drop position using the `raise()/lower()` methods to move Arm3.
- If less bars than columns are supplied, blocks can still be placed on the empty column and should still be laid out as described above from `Control.FIRST_BAR_COLUMN` to `Control.LAST_BAR_COLUMN`

HEIGHT OPTIMISATION (ARM1 movement):

Again, you must reproduce this behaviour by calling appropriate methods on the Robot.

- Arm 2 (the horizontal arm controlled by `extend()/contract()`) should always be at the lowest height to clear any obstacles (i.e. the top of any column). This is achieved by using the `up()/down()` methods on Arm1.
- This is set before moving to make a pick (or after any drop) so that the arm can JUST clear any obstacles as it moves to the source column.
- The height is then rechecked as soon as you make a pick taking into account the picked block and the additional clearance it needs as it moves to the drop destination.

HOW TO PROCEED

Your task is to write code using loops and selection/conditionals, arrays and methods to solve the problem, thereby writing an algorithm. You will also need to create variables/data structures to keep track of the position of bars, blocks and the arm segments so you can move and pick/drop as required. Arrays and primitive variables are sufficient for this purpose and you MUST NOT use higher level Collections (such as ArrayList) since we want you to get experience and demonstrate competency using arrays. This assignment does not require any of the Object-Oriented concepts that will covered in assignment part B.

The simplest way to solve this problem is to build the behaviour with small methods, passing parameters as necessary to avoid code repetition. *If you try to do this with a single method, the loop nesting will get complex and you* ***will*** *lose marks for code quality! (see Code Quality Assessment section below).* For example, *rather than nesting loops*, place one loop in a method (as with the supplied `extendToWidth()`) and call that single method in a loop .. much easier and cleaner!

The possible *robot arm movement* operations are specified and described by the supplied `RobotMovement` interface (`RobotMovement.java`). Additionally, `Control.java` contains some constants you can use to avoid hard coding values and ensure correct operation. You should not need any constants beyond those already provided.

As previously stated, the supplied `RobotControl.java` provides the example method:

```
private void extendToWidth(int width)
```

This method uses the `extend()` method from `RobotMovement.java` and serves as an example you can follow to keep your methods small, cohesive and useful.

Suggestions

In order to solve the problem in an incremental manner, it can be useful to start with a simple configuration such as the following (4 bars of height 7 with 2 blocks of height 3):

```
        this.barHeights = new int[] { 7, 7, 7, 7 };
        this.blockHeights = new int[] { 3, 3 };
```

This example is simple since you only have two blocks of a fixed size to place on a known bar size. You can even start with size one or two blocks and no bars if you are feeling cautious!

You can then increasingly add more and diverse block sizes and bars.

You can also start with the arm always at the maximum height which simplifies your calculations. The *height optimisation* can be added separately once you have finished the block placement.

Description of the Supplied Videos

The supplied video **`RobotPartASupplied.mp4`** shows the default behaviour of the robot with the supplied code project once it is correctly imported and executed from eclipse.

The supplied video **`RobotPartASolution.mp4`** uses the following data set which has a mix of block and bar sizes. If you can solve this without hard coding you should be good to go ☺

```
this.barHeights = new int[] { 7, 3, 1, 7, 5, 3, 2 };
this.blockHeights = new int[] { 3, 1, 2, 3, 1, 1, 1 };
```

NOTE: Do NOT hardcode a solution to these exact data sets. A proper algorithm will work for ALL different (legal) combinations and we WILL test with a different combinations! If you are using "magic numbers" or constants other than those defined in `Control.java` this is not a good sign so please seek help in class.

Summary

Write code using loops, conditionals, arrays and methods in `RobotControl.java` to reproduce the behaviour shown in the supplied videos and described in this specification.

Code Quality Assessment

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

- Use meaningful / descriptive identifiers (eg variable and method names).
- Demonstrate understanding of local variables versus class attributes and prefer local scope where possible.
- Demonstrate the use of defined constants in `Control.java` (Rather than using *magic numbers*)
- Avoid code repetition. (THIS IS THE MOST IMPORTANT ONE!)
- Write small private methods to avoid code other than one or more method calls going in the `RobotControl.control()` method.
- Appropriate use of comments (but remember that easily understandable code is better than a comment).
- Include a comment at the top of `RobotControl.java` class stating your name and student number.

Submission Instructions

- You are free to refer to textbooks and notes, and discuss the design issues (and associated general solutions) with your fellow students; however, the assignment should be **your own individual work**.
- 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.

The source code for this assignment (i.e. complete compiled **Eclipse project**) should be submitted as a .zip file by the due date. You can either zip up the project folder or use the Eclipse option `export->general->archive`. You will be provided with specific submission instructions on Canvas before the deadline.

See course guide for information about late penalties and special consideration.

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 /