Computer/Engineering Programming 1 – Practical Class 5

Aims and Objectives

This laboratory has been designed to help you

- to learn how to construct program code which is executed conditionally (to make use of selection through if-statements) and
- to learn how to use both fixed repetition (a fixed number of repetitions of actions) and conditional repetition (repetition of actions while a condition is true).

Getting Started

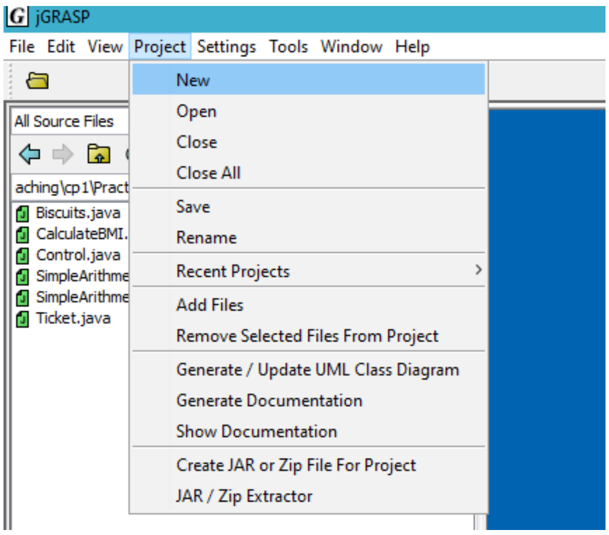
1. Create a directory called `prac5` (in your cp1 directory) and make it your current working directory.
2. Copy all the Java files contained in the Week 5 area on FLO to your current working directory (`prac5`).
3. Ensure that the *image folder* is in the same folder as the java files, see Figure 2

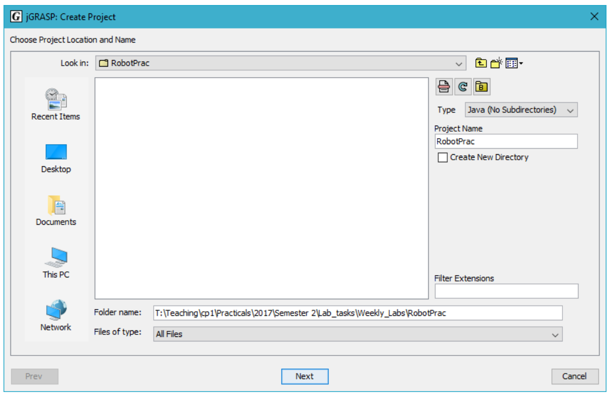
```
RobotWorld (folder)
  |
  .java files
  |
  images (folder)
    |
    diagonal.gif
```

Figure 2: Folder Structure after copying from FLO

Create a project by following the following steps:

1. From the control panel menu, Project > New will bring up a “new project” dialog. From there you choose the project filename and other properties.

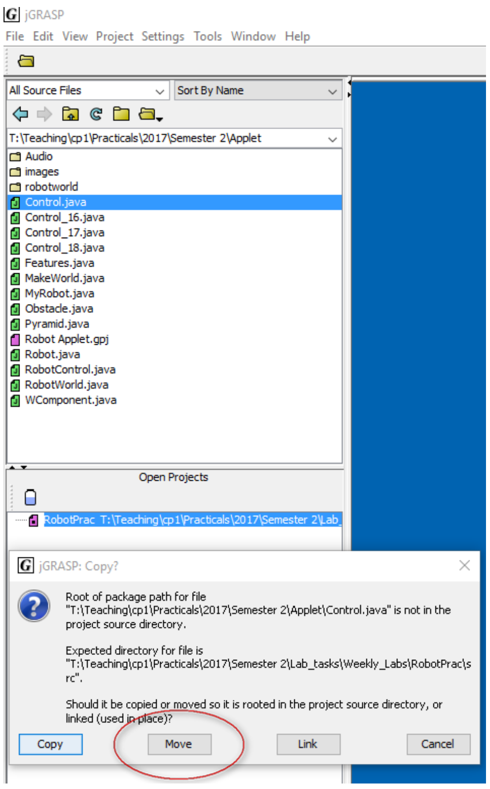




Look in: RobotPrac
Type: Java (No Subdirectories)
Project Name: RobotPrac
☐ Create New Directory
Filter Extensions:
Folder name: T:\Teaching\cp1\Practicals\2017\Semester 2\Lab_tasks\Weekly_Labs\RobotPrac
Files of type: All Files

2. Confirm the Project Directory and Project File name and press “Create”

3. Move/Copy the files from the RobotWorld folder to the project directory. The simplest way to add files to a project is to drag them from the browse window to the project window.

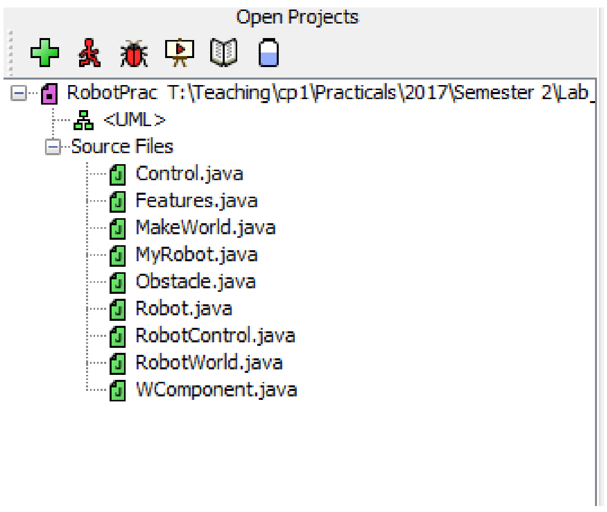


Root of package path for file "T:\Teaching\cp1\Practicals\2017\Semester 2\Applet\Control.java" is not in the project source directory.

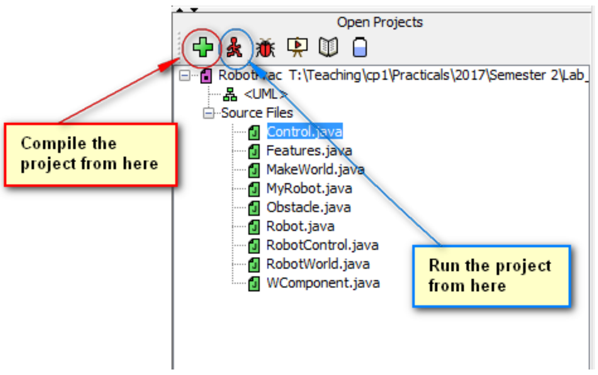
Expected directory for file is "T:\Teaching\cp1\Practicals\2017\Semester 2\Lab_tasks\Weekly_Labs\RobotPrac\src".

Should it be copied or moved so it is rooted in the project source directory, or linked (used in place)?

when you have finished moving all of the files the Project Window will look similar to the image below



4. Ensure that the images folder is in the same folder as the java files which is now (`RobotPrac`) (see Figure 2 above for a similar structure)
5. Open the files you need to edit from the Project Window
6. To compile and run the program use the compile and run buttons from the Projects Window



Task 1

For this task, rather than using RCL to control the robot, you will use Java code. The code you write will become part of the whole application through the modification of the `class MyRobot`. It has a method called `go` that the controlling method of the application invokes for you.

On completion of this task you will have the ability to construct a **while loop** which performs a fixed number of iterations and which calls a method on each iteration.

There will be several checkpoints which require `Control.java` to be modified. After you have completed each checkpoint and had it recorded, do the following:

1. Select *Save As* from the *File* menu and save the file (`Control.java`) with a new name (perhaps ending in the check point number as in `Control_16.java`).
2. Select *Open* from the *File* menu and reopen `Control.java`.
3. Continue working on `Control.java`

Do not change the name of the class `Control` - the class name and file name will be different for the previous version but since we will not be using Run Application this is not a problem. The names of the `.class` file created by the compiler **always depends on the name of the class** and not on the name of the file it is contained in. To test an earlier version just compile the saved file, for example, the file `Control_16.java` – this will create a new version of `Control.class`.

The file `Control.java` contains code which forms part of an application. It can be compiled in the normal way but should be run by running the application. If you need to see the program execute a second time then you will need to quit the current application and run it again.

Each time you make changes to `Control.java` you will need to

1. save the file (select *Save* from the *File* menu),
2. re-compile,
3. and restart the application

In the description which follows, *North* refers to the top of the application window, *South* to the bottom, *East* to the right edge and *West* to the left edge.

Your task is to modify the file `Control.java` so that the robot moves 5 steps to the North. It is initially facing North so all it need do is move 5 steps (see below for how to move the robot a step). Although the file only contains part of the code for the program, you **do not** need to refer to any of the remaining code in order to complete the task – this is typically the case!

The contents of `Control.java` is as follows:

```
class Control extends RobotControl {

   MyRobot robbie;

   void go() {
      robbie = new MyRobot(50,80);
   // type your statements after this line.
   }
}
```

The program will "automatically" create a `Control` object and invoke the `go()` method so statements included in its body will be executed. The line:

`robbie = new MyRobot(50,80);`

creates a new robot which can be controlled through calling methods of the object referred to by `robbie`. The statement:

`robbie.moveOneStep();`

will cause the robot to move one step. The are a number of other methods which can be used to control the robot which will be described as part of the next task.

You should now be in a position to add code to the file `Control.java` so that the **robot moves 10 steps to the North**.

Checkpoint 16

Show the program source code and the output to a demonstrator

Task 2

Quit the application.

Save your current version of the file `Control.java` as `Control_17.java` and re-open `Control.java`.

The robot moves in the direction it is facing. That is, a call to `moveOneStep()` will move it in the direction it is currently facing. The statement:

```
robbie.rotate90Right();
```

will cause the robot to rotate 90 degrees (a quarter of a turn) to the right. For example, if it is facing North then after the above statement is executed it will be facing East and calls to `moveOneStep` will move the robot to the East.

Modify `Control.java` so that the robot moves 10 steps north, rotates 90 degrees to the right and then moves 30 steps. The robot should stop not far from the top right-hand (North Eastern) corner of its world (the green-sided rectangle).

Checkpoint 17

Show the program source code and the output to a demonstrator

Task 3

Quit the application.

Save your current version of the file `Control.java` as `Control_17.java` and re-open `Control.java`.

The robot cannot move it it runs in an obstruction (a green wall, for example). Calls to `moveOneStep` will have no effect. It is possible to detect if the robot cannot move (it is blocked) with a call to a robot's `blocked` method. It returns `true` if the robot cannot move forward and `false` otherwise. Since the method returns a `boolean` value it can be used as the condition in a *while*-statement or an *if*-statement. For example,

```
  while (!robbie.blocked())
    ... code to move the robot here ....
```

Modify `Control.java` so that the robot

1. moves until it runs into the top (North) wall,
2. turns left 90 degrees,
3. moves 10 steps forward,
4. turns left 90 degrees,
5. moves until it hits the bottom (South) wall,
6. turns left 90 degrees,
7. moves 20 steps forward,
8. turns left 90 degrees, and then
9. moves until it runs into the top (North) wall.

Checkpoint 18

Show the program source code and the output to a demonstrator

Task 4

Quit the application.

Save your current version of the file `Control.java` as `Control_18.java` and re-open `Control.java`.

The robot also responds to the following methods:

- rotate45Right() : rotate right 45 degrees
- rotate90Left() : rotate left 90 degrees
- rotate45Left() : rotate left 45 degrees
- dropSquare() : leave a small square at the robot's current position

A line can be drawn by "dropping" a square before each step.

Modify the program so that the robot draws a house consisting of a rectangle for the base, a triangle for the roof and a rectangular front door. Include a leafless tree next to the house which consists of a vertical trunk and horizontal branches which get smaller towards the top of the tree.

__________________ **Checkpoint 19** __________________
Show the program source code and the output to a demonstrator

Task 5

1. For this task you will need to implement a dialog box driven user interaction session. You will need to prompt the user to supply 2 strings that will be used for a calculation – either addition or subtraction (Skeleton code is provided in Figure 3).
2. The screenshots (see Figure 4) on the next page indicate an example run through the application.
3. An input dialog should be used to capture a string and store this within a local variable `firstString`, which should be converted to an `int`.
4. Use a second input dialog to capture the second string and store it in a local variable `secondString`, which should be converted to an `int`.
5. A confirm dialog should be used to allow the user to decide whether they want to perform an addition or subtraction of the two inputs. When creating a confirm dialog we can define what buttons appear on the dialog, see below:

   `showConfirmDialog(null, message, title, optionType)`

   - **`message`** is the object (*String*) that is displayed as the prompt to the user.
   - **`title`** is the *String* that is displayed across the top of the dialog box.
   - **`optionType`** is an *integer* that represents the different types of button layouts for the confirm dialog:
     – `DEFAULT_OPTION` displays the default set of buttons for the confirm dialog
     – `YES_NO_OPTION` displays a Yes and No button
     – `YES_NO_CANCEL_OPTION` displays a Yes, No and Cancel button

  - `OK_CANCEL_OPTION` display an OK and Cancel button

6. Capture the user input with the confirm dialog and if they **select Yes** then complete the *addition* of the two inputs. If they **select No** then complete the *subtraction*.
7. Output the result in a message dialog and then prompt the user to determine if they want to perform another calculation.
8. You should use a **do loop** to ensure the application will continue to run until the user quits.

```
/*
 * Class to create a simple dialog to either
 * add or subtract two integers
*/

import javax.swing.JOptionPane;

public class SimpleArithmeticDialog {

   public static void main(String[] args) {
      // Your code goes here
   }
}
```

Figure 3: Skeleton code for the dialog application

Checkpoint 20

Show the program source code and the output to a demonstrator

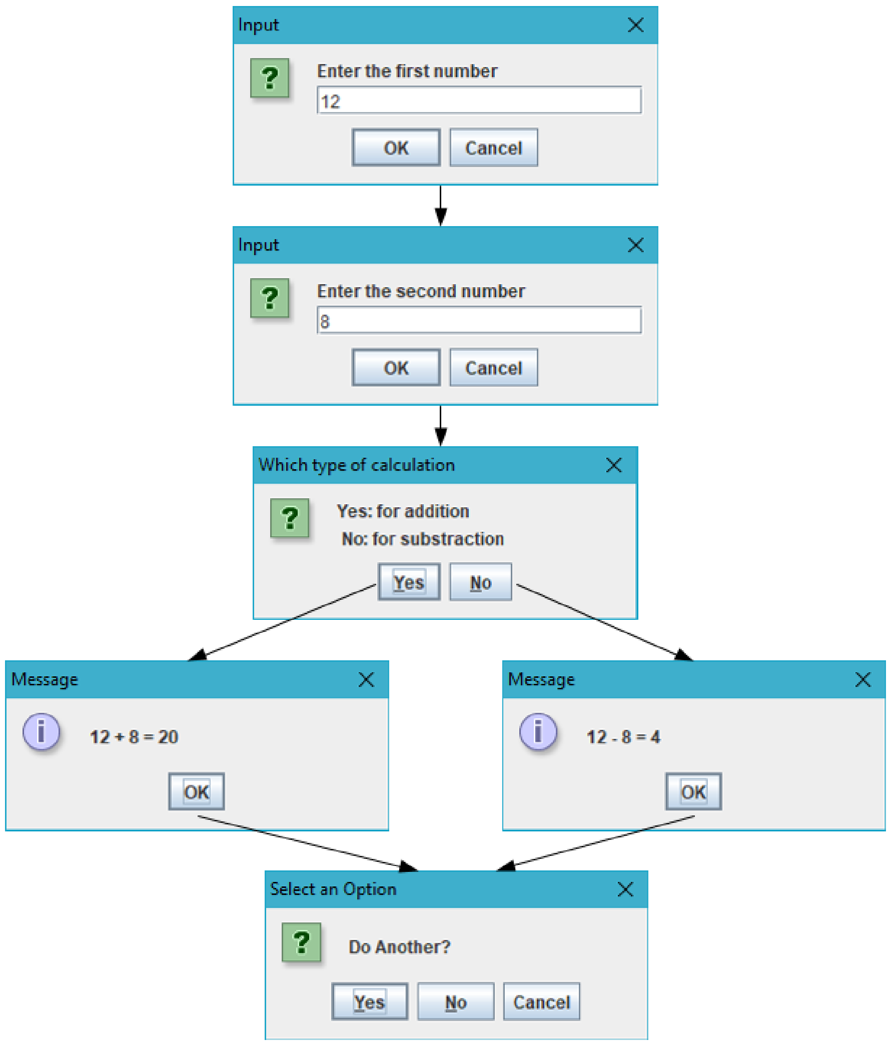


Figure 4: A sample run of both the 'addition' and 'subtraction' calculations