**CSC262** Programming in JAVA - **Project**

|  |  |
| --- | --- |
| **First Name** | Jayson |
| **Last Name** | Levario |
| **Student ID#** | 040844066 |
| **Email Address** | Jaysonlevario13@aol.com |

**Robot with Graphical User Interface**

Table of Contents

[How to submit your work 1](#_Toc16700189)

[Part One – Robot Class – 35 Points 2](#_Toc16700190)

[Part Two - Graphical User Interface – 65 Points 3](#_Toc16700191)

# How to submit your work

After filling all the parts in this file, please follow the following steps.

1. Add your name and Student ID# to the first page.
2. Save the file in the original format (Docx or Doc)

(please do not convert to other file formats e.g. PDF, ZIP, RAR, …)

1. Rename the file as

*YOUR First Name YOUR Last Name -* ***Project****.docx*

**Example:**

John Smith - Project.docx

**Please do not seek any assistance when programing this project. You might be asked to explain your code in details. Contact your instructor for any clarifications.**

# Part One – Robot Class – 35 Points

In this project you will create a robot class. The robot objects move within a grid similar to the 3x3 grid shown below to pick up and carry the letters on the grid to different locations.

|  |  |  |
| --- | --- | --- |
| B |  |  |
|  |  | D |
|  | C |  |

The grid size should be declared as constants, use a 25X25 grid for our program.

The Robot class must have the following data members and methods:

1. The **Robot** class has three private data members
   1. An integer type variable for the **x** component of the location of the robot on the grid
   2. An integer type variable for the **y** component of the location of the robot on the grid
   3. A char type variable “**payload**” to hold letters as load
2. The Robot class must have the following public member functions
   1. A **default constructor** places the robot at location (0,0) and payload to character space: ‘ ‘ ( empty)
   2. A **constructor** that receives three parameters to initialize the private data members.
   3. Include a **set** and **get** method for each of the private data members.
   4. Include a member function **print()** that prints the location of the robot on the grid as well as its load.
   5. Method **pickup(int lx, int ly)** with boolean return type. This method examines the current location of the robot and if not at location (lx, ly), it should display a message “not at (lx,ly)” and return false. If the is no load (letters) at the location print “No load at this location” and return false. Otherwise, the method should examine the payload of the robot and if empty it will pick up the load at location (lx,ly) and place it in the payload and returns true. Note, the character at location (lx,ly) on the grid should be set to blank ( space)
   6. Method **dropOff(int lx, int ly)** with Boolean return type. This method examines the current location of the robot and if not at location (lx, ly), it should display a message “not at (lx,ly)” and return false. Otherwise, the method examines the payload of the robot and if full, it will drop its load at location (lx, ly) on the grid and and returns true. Note, the grid element grid[lx][ly] should be set to the character that was placed there.
   7. Method **moveRight()** with a void return type. The method moves robot one unit to the right on the grid. If the robot hits the right boundary of the grid, it should print “Right boundary reached”.
   8. Method **moveLeft()** with a void return type. The method moves robot one unit to the left on the grid. If the robot hits the left boundary of the grid, it should print “Left boundary reached”.
   9. Method **moveUp()** with a void return type. The method moves robot one unit up on the grid. If the robot hits the top boundary of the grid, it should print “Top boundary reached”.
   10. Method **moveDown()** with a void return type. The method moves robot one unit down on the grid. If the robot hits the bottom boundary of the grid, it should print “Bottom boundary reached”.
   11. Method **MoveTo(int lx, int ly)** with boolean return type. This method checks to see whether the location (lx, ly) is within the boundary of the grid, if not returns false. Otherwise, it uses the combination of moveUp(), moveDown(), moveRight(), moveLeft() methods to move the robot to this new location on the grid. Note: if the robot has to move, say 5, units to the right you must write a loop and call moveRight() five times rather that literally writing moveRight() five times.

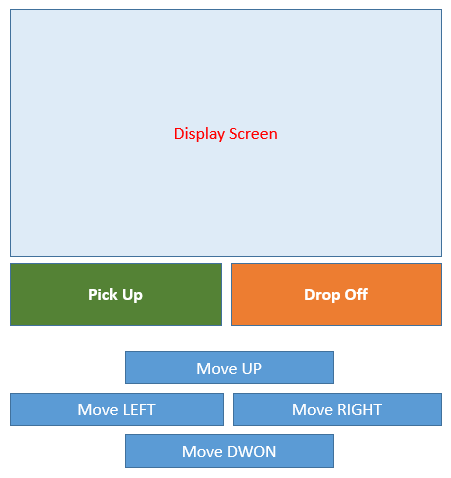
Write a nonmember function **print2D()** that reveries a 2D array of characters and prints it ( in our case 25 elements per row).

Write a main function to test the operation of the robot.

1. You program should define a 2D array representing the grid **char grid [26][26]** **.** Initialize all the elements of the 2D array to blank. Then, place character ‘B’ at location (10, 8) and ‘C’ at (22, 4). Use **print2D()** to print the grid
2. Create two robots R1 and R2 using the two constructors. Print the location of the robots.
3. Use “moveTo()” method to place R1 at location (23, 24) and R2 at location (15, 3). Print the location of the robots
4. Direct R1 to pick up ‘B’ at location (10, 8) and place it at location (20,20). Print the location of the robot. Print the grid.
5. Direct R2 to pick up ‘C’ at location (22, 4) and place it at location (0, 0). Print the location of the robot. Print the grid.

# Part Two - Graphical User Interface – 65 Points

Create a Graphical User Interface for simulating the robot. A sample GUI for this project may be as following.



|  |
| --- |
| Your C++ code for this HW |
| package com.company;  public class Robot {  private int x;  public int y;  char payload;  String name;  Grid grid;   public Robot() {  this.x = 0;  this.y = 0;  this.payload = ' ';  this.name = " ";  }   public Robot(int x, int y, char payload, String name)  {  this.x = x;  this.y = y;  this.payload = payload;  this.name = name;  }   public int getX() {  return x;  }   public int getY() {  return y;  }   public char getPayload() {  return payload;  }   public void setX(int x) {  this.x = x;  }   public void setY(int y) {  this.y = y;  }   public void setPayload(char payload) {  this.payload = payload;  }   public void setName(String name) { this.name = name; }   public void print() { System.*out*.println(name + " is at location (" + x + " ,"+ y + ") payload: " + payload); }   public void useGrid(Grid grid) { this.grid = grid; }   boolean pickUp(int lx, int ly)  {  if (x != lx || y != ly)  {  System.*out*.println("not at (" + lx + ", " + ly + ")");  return false;  }   //If pickUp is at robots location then we want to check if location has something to pick up  else  {  if (grid.getValueAt(lx, ly) == ' ')  {  System.*out*.println("No load at this location");  return false;  }   else  {  //Check if our payload is empty and if it is then grab letter  if (payload == ' ')  {  payload = grid.getValueAt(lx, ly);  grid.setValueAt(lx, ly, ' ');  return true;  }   //Otherwise tell the user their payload is full  else  {  System.*out*.println("Your payload is full");  return false;  }  }  }  }   boolean dropOff(int lx, int ly)  {  if (x != lx || y != ly)  {  System.*out*.println("not at (" + lx + ", " + ly + ")");  return false;  }   //if payload isn't full then it's okay to dropOff  if (payload != ' ')  {  grid.setValueAt(lx, ly, payload);  payload = ' ';  return true;  }   else  {  System.*out*.println("You don't have a payload to drop off");  return false;  }  }   public void moveRight()  {  x++;  if (x > grid.rows)  {  //this puts x back to the original position  x--;  System.*out*.println("Right boundary reached.");  }  }   public void moveLeft()  {  x--;  if (x < 0)  {  //this puts x back to the original position  x++;  System.*out*.println("Left boundary reached.");  }  }   public void moveUp()  {  y--;  if (y < 0)  {  //this puts y back to the original position  y++;  System.*out*.println("Top boundary reached.");  }  }   public void moveDown()  {  y++;  if (y > grid.cols)  {  //this puts y back to the original position  y--;  System.*out*.println("Bottom boundary reached.");  }  }   boolean moveTo(int lx, int ly)  {  //If x or y location isn't in the grid then neither value will get set  //Could still set x or y if the opposite value is invalid  if (lx < 0 || lx > grid.rows || ly < 0 || ly > grid.cols)  {  System.*out*.println("Not a valid move inside the grid");  return false;  }   int originalX = x;  int originalY = y;   if (originalX < lx)  {  for (int i = 0; i < lx - originalX; i++)  {  moveRight();  }  }   else if (originalX > lx)  {  for (int i = 0; i < originalX - lx; i++)  {  moveLeft();  }  }   if (originalY < ly)  {  for (int i = 0; i < ly - originalY; i++)  {  moveDown();  }  }   else if (originalY > ly)  {  for (int i = 0; i < originalY - ly; i++)  {  moveUp();  }  }  return true;  } }  package com.company;  public class Grid {  int rows;  int cols;  private char grid[][];   public Grid()  {  rows = 25;  cols = 25;  }   public Grid(int rows, int cols)  {  this.rows = rows;  this.cols = cols;  }   public void setValueAt(int row, int col, char val)  {  grid[row][col] = val;  }   public char getValueAt(int x, int y)  {  return grid[x][y];  }   public void init()  {  grid = new char[cols][rows];  for (int i = 0; i < rows; i++)  {  for (int j = 0; j < cols; j++)  {  grid[j][i] = ' ';  }  }  }   public void print2D()  {  for (int i = 0; i < rows; i++)  {  for (int j = 0; j < cols; j++)  {  //To visually see the grid  System.*out*.print(grid[j][i]);  }  System.*out*.println();  }  } }  package com.company; import java.util.\*;  public class Main {  public static void main(String[] args) {  Grid grid = new Grid(25, 25);  grid.init();  grid.setValueAt(10, 8, 'B');  grid.setValueAt(22, 4, 'C');  grid.print2D();   Robot R1 = new Robot();  R1.setName("R1");  Robot R2 = new Robot();  R2.setName("R2");  R1.useGrid(grid);  R2.useGrid(grid);   R1.moveTo(23, 25);  R2.moveTo(15, 3);  R1.print();  R2.print();   R1.moveTo(10, 8);  R1.pickUp(10, 8);  R1.moveTo(20, 20);  R1.dropOff(20,20);  grid.print2D();  R1.print();   R2.moveTo(22, 4);  R2.pickUp(22, 4);  R2.moveTo(0, 0);  R2.dropOff(0,0);  grid.print2D();  R2.print();  } }  ////////////////////////////////  package com.company; import java.awt.\*; import java.util.\*; import javafx.application.Application; import javafx.event.ActionEvent; import javafx.event.EventHandler; import javafx.fxml.FXMLLoader; import javafx.scene.Group; import javafx.scene.Parent; import javafx.scene.Scene; import javafx.scene.control.Button; import javafx.scene.control.MenuButton; import javafx.scene.layout.Pane; import javafx.scene.layout.GridPane; import javafx.scene.layout.StackPane; import javafx.scene.paint.Paint; import javafx.scene.shape.\*; import javafx.stage.Stage;  import javax.swing.\*; public class Main extends Application {  public static void main(String[] args)  {  *launch*(args);  }   @Override  public void start(Stage stage) throws Exception  {  Parent root = FXMLLoader.*load*(getClass().getResource("ProjectSceneBuilder.fxml"));  stage.setTitle("2D Grid");   stage.setScene(new Scene(root, 600, 600));  stage.show();  } }  package com.company; import javafx.event.ActionEvent; import javafx.fxml.FXML; import javafx.scene.control.Label; import javafx.scene.paint.Paint; import javafx.scene.shape.Circle;  import java.lang.ModuleLayer.Controller;  public class Robot {  public Circle circle;   @FXML  void moveUp()  {  circle.setLayoutY(circle.getLayoutY() - 1);  }   @FXML  void moveDown()  {  circle.setLayoutY(circle.getLayoutY() + 1);  }   @FXML  void moveLeft()  {  circle.setLayoutX(circle.getLayoutX() - 1);  }   @FXML  void moveRight()  {  circle.setLayoutX(circle.getLayoutX() + 1);  } }  <?xml version="1.0" encoding="UTF-8"?>  <?import javafx.scene.control.Button?> <?import javafx.scene.layout.ColumnConstraints?> <?import javafx.scene.layout.GridPane?> <?import javafx.scene.layout.Pane?> <?import javafx.scene.layout.RowConstraints?> <?import javafx.scene.shape.Circle?>  <GridPane alignment="center" hgap="10" vgap="10" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.company.Robot">  <columnConstraints>  <ColumnConstraints />  <ColumnConstraints />  </columnConstraints>  <rowConstraints>  <RowConstraints />  <RowConstraints />  </rowConstraints>  <children>  <Pane prefHeight="511.0" prefWidth="505.0" GridPane.columnIndex="1" GridPane.rowIndex="1">  <children>  <Pane layoutX="-10.0" layoutY="403.0" prefHeight="111.0" prefWidth="515.0" style="-fx-background-color: grey;">  <children>  <Button layoutX="220.0" layoutY="14.0" mnemonicParsing="false" onAction="#moveUp" prefWidth="75.0" text="UP" />  <Button layoutX="132.0" layoutY="42.0" mnemonicParsing="false" onAction="#moveLeft" prefWidth="75.0" text="LEFT" />  <Button layoutX="303.0" layoutY="42.0" mnemonicParsing="false" onAction="#moveRight" prefWidth="75.0" text="RIGHT" />  <Button layoutX="220.0" layoutY="69.0" mnemonicParsing="false" onAction="#moveDown" prefWidth="75.0" text="DOWN" />  <Button layoutX="7.0" layoutY="7.0" mnemonicParsing="false" prefWidth="75.0" text="PICK UP" />  <Button layoutX="416.0" layoutY="7.0" mnemonicParsing="false" prefWidth="85.0" text="DROP OFF" />  </children>  </Pane>  <Circle fx:id="circle" fill="DODGERBLUE" layoutX="43.0" layoutY="43.0" radius="29.0" stroke="BLACK" strokeType="INSIDE" />  </children>  </Pane>  </children> </GridPane> |

Run the code and insert the result in the following box.

|  |
| --- |
| The result of the query |
| A screenshot of a computer  Description automatically generated  A picture containing computer  Description automatically generated  A screen shot of a computer  Description automatically generated  ////////////////////////////////////    A screenshot of a cell phone  Description automatically generated |