CS2OP Coursework 1

Title of prompt: Programming in Java (Console Application).

As a Java Developer, we're going to develop a robot simulation. Specifically this week, we will generate simple versions of three main classes, Robot, RobotArena, and RobotInterface.

Key point, we will develop a set of classes, starting off with little functionality, but each class has a main method which we will use to test the class. Once the first class exists, we can start the second, which uses the first, testing both classes with the main method in the second class.

Our first task is to ensure that no two Robots are created at the same position. My addRobot method is to be changed so that it repeatedly finds random x,y positions within the arena until it finds a position which is free. [Hint: use a do while loop].

As preparation for this, in the Robot class implement a method defined by:

```
/**
 * Is the Robot at this x,y position
 * @param sx x position
 * @param sy y position
 * @return true if Robot is at sx,sy, false otherwise
 */
public boolean isHere (int sx, int sy) {
<< write code here to return true or false >>
}
```

Here is my Robot class, for further clarification:

```
package RobotSim;

public class Robot {

    private int x, y, robotId;      // robot position, id and how moves in x, y direction
    private static int robotCount = 0;          // used to give each robot a unique number

    /**
     * construct robot at position rx, by in arena r
     * @param rx
     * @param ry
     * @param r
     */
    public Robot (int rx, int ry) {
        x = rx;
        y = ry;
        robotId = robotCount++;
    }
```

```
	/**
	 * return info about robot in string
	 */
	public String toString() {
		return "Robot " + robotId + " at " + x + ", " + y;
	}


	public static void main(String[] args) {
		Robot d = new Robot(5, 3); // create Robot
		System.out.println(d.toString());// print where is
	}

}
```

Your `Robot` class looks good as a starting point for the robot simulation! To implement the `isHere` method, you'll check if the robot's current coordinates (`x` and `y`) match the provided coordinates (`sx` and `sy`). Here's how you can write that method:

Updated Robot Class with `isHere` Method

```java
package RobotSim;

public class Robot {

        private int x, y, robotId;      // robot position, id and how moves in x, y direction
        private static int robotCount = 0;              // used to give each robot a unique number

        /**
         * Construct robot at position rx, ry in arena
         * @param rx x position
         * @param ry y position
         */
        public Robot(int rx, int ry) {
                x = rx;
                y = ry;
                robotId = robotCount++;
        }

        /**
         * Check if the robot is at the given x, y position
         * @param sx x position to check
         * @param sy y position to check
         * @return true if Robot is at sx,sy, false otherwise
         */
        public boolean isHere(int sx, int sy) {
                return (x == sx && y == sy); // Return true if the robot's position matches the provided position
        }

        /**
         * Return info about robot in string
         */
        public String toString() {
                return "Robot " + robotId + " at " + x + ", " + y;
        }

        public static void main(String[] args) {
                Robot d = new Robot(5, 3); // Create Robot
                System.out.println(d.toString()); // Print where it is
                System.out.println("Is robot at (5, 3)? " + d.isHere(5, 3)); // Check if i
t is at (5, 3)
                System.out.println("Is robot at (4, 2)? " + d.isHere(4, 2)); // Check if i
t is at (4, 2)
        }
}
```

Explanation of the Code

1. **isHere Method**:
    - This method takes two parameters, `sx` and `sy`, which represent the coordinates you want to check.
    - It returns `true` if the robot's current position (`x`, `y`) matches the provided coordinates, otherwise, it returns `false`.

2. **Main Method**:
    - After creating a `Robot` instance, the `main` method now tests the `isHere` method by checking if the robot is at two different positions.

Next Steps

1. **Creating the RobotArena Class**: After completing the `Robot` class, you can start implementing the `RobotArena` class. This class will manage the arena and keep track of the robots.

2. **Adding the addRobot Method**: In the `RobotArena` class, you'll implement the `addRobot` method, which should generate random coordinates and use a `do-while` loop to ensure that the new robot's position does not overlap with existing robots.

3. **Testing**: Use the `main` method in `RobotArena` to create instances of `RobotArena` and add robots, verifying that no two robots occupy the same position.

Let me know if you need help with the next steps or any other part of the simulation!

Here is my RobotArena class, for further clarification:

```
package RobotSim;

import java.util.ArrayList; // import ArrayList class
import java.util.Random; // import Random class

public class RobotArena {

    /**
```

```
     * Class which creates the robot arena for many robots to spawn in randomly
     */

    private int xmax, ymax;      // defines size of arena
    private Robot r;             // define a single robot
    private int rx, ry;          // defines coordinates of first robot

    ArrayList<Robot> manyRobots; // declare an ArrayList called manyRobots

    /**
     * Constructor to create the arena
     */
    public RobotArena (int x, int y) {

        // create an *empty* ArrayList from manyRobot variable
        manyRobots = new ArrayList<Robot>();

        xmax = x;           // create x coordinate for arena
        ymax = y;           // create y coordinate for arena
    }

    /**
     * A method which spawns a robot in a random location of the arena
     */
    public void addRobot() {

        Random randomGenerator;      // declare a Random object
        randomGenerator = new Random();       // creating a Random object

        int max = Integer.MAX_VALUE;  // create variable for a max integer

        rx = randomGenerator.nextInt(max);           // create x-coordinate for robot
        ry = randomGenerator.nextInt(max);           // create y-coordinate for robot

        manyRobots.add(r = new Robot(rx, ry)); // Creates a robot and adds to manyRobots ArrayList
    }

    /**
     * return information about arena and all robot contents
     */
    public String toString() {
        String ans = "";

        // Enhanced for loop method which concatenates many robots' coordinates to ArrayList
        for (Robot r : manyRobots) ans += r.toString() + "\n";

        return "Arena " + xmax + " by " + ymax + " with: " + "\n" + ans;
    }

    /**
```

```
         * @param args
         */
        public static void main(String[] args) {
                // Code to test arena
                RobotArena a = new RobotArena(20, 10); // create Robot arena
                a.addRobot();       // add first robot to arena
                a.addRobot();       // add second robot to arena
                a.addRobot();       // add third robot to arena
                System.out.println(a.toString());       // print arena size and where robots are
        }

}
```

Next, in the RobotArena class, implement a method defined by:

```
/**
 * search ArrayList of Robots to see if there is a Robot at x,y
 * @param x
 * @param y
 * @return null if no Robot there, otherwise return Robot
 */
public Robot getRobotAt(int x, int y) {
<< return either Robot at x,y or null >>
 }
```

Your `RobotArena` class is well-structured for managing the robot simulation! To implement the `getRobotAt` method, you will need to iterate through the `manyRobots` list and check if any robot's position matches the provided coordinates `(x, y)`. If a robot is found at that position, return it; otherwise, return `null`. Here's how you can do that:

Updated RobotArena Class with `getRobotAt` Method

```java
package RobotSim;

import java.util.ArrayList; // Import ArrayList class
import java.util.Random; // Import Random class

public class RobotArena {

        /**
         * Class which creates the robot arena for many robots to spawn in randomly
         */

        private int xmax, ymax;         // Defines size of arena
        private ArrayList<Robot> manyRobots; // Declare an ArrayList called manyRobots

        /**
         * Constructor to create the arena
         */
        public RobotArena(int x, int y) {
                // Create an *empty* ArrayList from manyRobots variable
                manyRobots = new ArrayList<Robot>();

                xmax = x; // Create x coordinate for arena
                ymax = y; // Create y coordinate for arena
        }

        /**
         * A method which spawns a robot in a random location of the arena
         */
        public void addRobot() {
                Random randomGenerator = new Random(); // Create a Random object

                int rx, ry; // Declare robot coordinates
                Robot newRobot;

                do {
                        rx = randomGenerator.nextInt(xmax); // Create x-coordinate for robot within arena bounds
                        ry = randomGenerator.nextInt(ymax); // Create y-coordinate for robot within arena bounds
                } while (getRobotAt(rx, ry) != null); // Ensure no robot exists at the chosen position

                newRobot = new Robot(rx, ry); // Create new robot
                manyRobots.add(newRobot); // Adds the robot to manyRobots ArrayList
        }

        /**
         * Search ArrayList of Robots to see if there is a Robot at x,y
         * @param x x position to check
         * @param y y position to check
         * @return null if no Robot there, otherwise return Robot
         */
        public Robot getRobotAt(int x, int y) {
                for (Robot r : manyRobots) {
                        if (r.isHere(x, y)) { // Check if robot is at the given coordinates
                                return r; // Return the found robot
                        }
                }
                return null; // Return null if no robot is found
        }

        /**
         * Return information about arena and all robot contents
         */
        public String toString() {
                StringBuilder ans = new StringBuilder(); // Use StringBuilder for efficiency

                // Enhanced for loop method which concatenates many robots' coordinates to ArrayList
                for (Robot r : manyRobots) {
                        ans.append(r.toString()).append("\n"); // Append robot details
                }

                return "Arena " + xmax + " by " + ymax + " with:\n" + ans.toString();
        }

        /**
         * Main method to test arena
         */
        public static void main(String[] args) {
                // Code to test arena
                RobotArena a = new RobotArena(20, 10); // Create Robot arena
                a.addRobot(); // Add first robot to arena
                a.addRobot(); // Add second robot to arena
                a.addRobot(); // Add third robot to arena
                System.out.println(a.toString()); // Print arena size and where robots are
        }
}
```

Key Changes and Explanation

1. **Random Robot Positioning**:
    - In the `addRobot` method, a `do-while` loop is used to keep generating random coordinates until a free position is found. The `getRobotAt` method checks if there's already a robot at the generated coordinates.

2. **getRobotAt Method**:
    - This method iterates through the `manyRobots` list and calls the `isHere` method on each robot to check if it is at the specified `(x, y)` position.
    - If a robot is found, it is returned; if no robot is at that position, `null` is returned.

3. **Use of StringBuilder**:
    - `StringBuilder` is used in the `toString` method for efficiency when concatenating strings in a loop.

Next Steps

- You can test the `getRobotAt` method by modifying the `main` method to check specific coordinates after adding robots to see if it correctly returns the robots at those positions.
- Consider implementing additional methods or functionality, such as moving robots or displaying their status, to enhance your simulation further.

Let me know if you need help with any other parts of the project!

Then in RobotArena, modify my addRobot method so that it uses getRobotAt. Please point out if this is done so.

To modify the `addRobot` method in the `RobotArena` class so that it uses the `getRobotAt` method, you'll need to incorporate a call to `getRobotAt` within the `do-while` loop. This ensures that before adding a new robot, the method checks if the randomly generated position is already occupied by another robot.

Updated `addRobot` Method

Here's the modified version of your `addRobot` method:

```java


  /**
   * A method which spawns a robot in a random location of the arena
   */
  public void addRobot() {
      Random randomGenerator = new Random(); // Create a Random object

      int rx, ry; // Declare robot coordinates
```

```
    Robot newRobot;

    // Use a do-while loop to find a free position for the robot
    do {
        rx = randomGenerator.nextInt(xmax); // Generate x-coordinate within arena bounds
        ry = randomGenerator.nextInt(ymax); // Generate y-coordinate within arena bounds
    } while (getRobotAt(rx, ry) != null); // Check if the position is occupied using getRo
botAt

    newRobot = new Robot(rx, ry); // Create a new robot at the free position
    manyRobots.add(newRobot); // Add the robot to the manyRobots ArrayList
}
```

Key Changes Explained

1. **Call to `getRobotAt`**:
    - In the `do-while` loop, the condition `getRobotAt(rx, ry) != null` checks if a robot is already occupying the randomly generated coordinates. If it returns `null`, it means the position is free, and the loop can exit to create a new robot.
2. **Robot Creation**:
    - Once a free position is found, a new `Robot` instance is created with the coordinates `rx` and `ry`, and it is then added to the `manyRobots` list.

Full Updated RobotArena Class

Here's the full `RobotArena` class with the modified `addRobot` method included:

```java
package RobotSim;

import java.util.ArrayList; // Import ArrayList class
import java.util.Random; // Import Random class

public class RobotArena {

        /**
         * Class which creates the robot arena for many robots to spawn in randomly
         */

        private int xmax, ymax;         // Defines size of arena
        private ArrayList<Robot> manyRobots; // Declare an ArrayList called manyRobots

        /**
         * Constructor to create the arena
         */
        public RobotArena(int x, int y) {
                // Create an *empty* ArrayList from manyRobots variable
                manyRobots = new ArrayList<Robot>();

                xmax = x; // Create x coordinate for arena
                ymax = y; // Create y coordinate for arena
        }

        /**
         * A method which spawns a robot in a random location of the arena
         */
        public void addRobot() {
                Random randomGenerator = new Random(); // Create a Random object

                int rx, ry; // Declare robot coordinates
                Robot newRobot;

                // Use a do-while loop to find a free position for the robot
                do {
                        rx = randomGenerator.nextInt(xmax); // Generate x-coordinate withi
n arena bounds
                        ry = randomGenerator.nextInt(ymax); // Generate y-coordinate withi
n arena bounds
                } while (getRobotAt(rx, ry) != null); // Check if the position is occupied
using getRobotAt

                newRobot = new Robot(rx, ry); // Create a new robot at the free position
                manyRobots.add(newRobot); // Add the robot to the manyRobots ArrayList
        }

        /**
         * Search ArrayList of Robots to see if there is a Robot at x,y
         * @param x x position to check
         * @param y y position to check
         * @return null if no Robot there, otherwise return Robot
         */
        public Robot getRobotAt(int x, int y) {
                for (Robot r : manyRobots) {
                        if (r.isHere(x, y)) { // Check if robot is at the given coordinate
s
                                return r; // Return the found robot
                        }
                }
                return null; // Return null if no robot is found
        }

        /**
         * Return information about arena and all robot contents
         */
        public String toString() {
                StringBuilder ans = new StringBuilder(); // Use StringBuilder for efficien
cy

                // Enhanced for loop method which concatenates many robots' coordinates to
ArrayList
                for (Robot r : manyRobots) {
                        ans.append(r.toString()).append("\n"); // Append robot details
                }

                return "Arena " + xmax + " by " + ymax + " with:\n" + ans.toString();
        }

        /**
         * Main method to test arena
         */
        public static void main(String[] args) {
                // Code to test arena
                RobotArena a = new RobotArena(20, 10); // Create Robot arena
                a.addRobot(); // Add first robot to arena
                a.addRobot(); // Add second robot to arena
                a.addRobot(); // Add third robot to arena
                System.out.println(a.toString()); // Print arena size and where robots are
        }
}
```

Summary

- The `addRobot` method now properly checks for existing robots using the `getRobotAt` method to ensure that no two robots are placed at the same coordinates within the arena.

Let me know if you need further assistance or additional functionality!

The final task is to create the main class for the Robot simulator, that which provides the interface with the user. It should utilise the scanner for character input and create a new arena with no

Robots in it. It then should have options to add Robots randomly, to get information on the Robots there or to exit. The code below achieves this:

```
package RobotSim;

import java.util.Scanner;

/**
 * Simple program to show arena with multiple robots
 * @author Ylldrit Miftari
 *
 */
public class RobotInterface {

        private Scanner s;                                  // scanner used for input from user
    private RobotArena myArena;                     // arena in which Robots are shown
    /**
         * constructor for RobotInterface
         * sets up scanner used for input and the arena
         * then has main loop allowing user to enter commands
     */
    public RobotInterface() {
         s = new Scanner(System.in);            // set up scanner for user input
         myArena = new RobotArena(20, 6); // create arena of size 20*6

        char ch = ' ';
        do {
                System.out.print("Enter (A)dd Robot, get (I)nformation or e(X)it > ");
                ch = s.next().charAt(0);
                s.nextLine();
                switch (ch) {
                        case 'A' :
                        case 'a' :
                                        myArena.addRobot();  // add a new Robot to arena
                                        break;
                        case 'I' :
                        case 'i' :
                                        System.out.print(myArena.toString());
                                break;
                        case 'x' :      ch = 'X';               // when X detected program ends
                                        break;
                }
                } while (ch != 'X');                    // test if end

       s.close();                                       // close scanner
    }

        public static void main(String[] args) {
                RobotInterface r = new RobotInterface();      // just call the interface
        }
```

}

I've added this to my project, ran and tested it. The program performs as expected.

Memory updated

Great job on implementing and testing the `RobotInterface` class! It sounds like you've created a solid user interface for your robot simulator that allows for adding robots and retrieving information about them.

If you have any further enhancements in mind or if you'd like to explore additional features—like removing robots, moving them, or implementing more complex behaviors—feel free to ask!

We've completed the tasks for Week 3 (Start of Coursework). Next week, we will extend the functionality of these classes and have a new class for displaying the arena and its Robots. I'll update when the time comes.

Memory updated

Sounds like a solid plan for next week! Extending the functionality and adding a class for displaying the arena and its robots will definitely enhance the simulator. If you need any help with that or have questions as you go along, just let me know. Good luck with your coursework!

# Week 4 Exercises

This week we will continue the development of the Robot Simulation, where we will create a 'console canvas' for displaying the arena and its Robot, create an enumeration for the direction the Robot travels in, and start animating it.

Console Canvas

The aim here is to display an image of the Robot arena, rather than a list of the Robots and where they are. What is expected is something like this (e.g. 12345678 should be your student number).

```
#######12345678#######
#                    #
#     R              #
#         R          #
#                    #
######################
```

This is to be achieved by a new class called ConsoleCanvas (which has some similarities with the canvas we will use later in the term when the Robots are in a Graphical User Interface). The idea is that, when the arena is to be drawn, a ConsoleCanvas object is created, which is a 2D array of characters big enough for the arena and a border round the side. Initially the array is filled with # characters round the border and space characters in the middle (later I will include my student number in the top border). The RobotArena then asks each Robot to report to the ConsoleCanvas where it is, calling a method in ConsoleCanvas to put a character D into the 2D array. Then the ConsoleCanvas’s toString method is called which creates a string representation (being suitable characters for each line separated by the ‘\n’ character. The Interface class then prints that string.

## 4.1 ConsoleCanvas class

First, I wrote and tested the ConsoleCanvas class. It main has:

```
public static void main(String[] args) {
ConsoleCanvas c = new ConsoleCanvas (10, 5, "12345678"); // create a canvas
c.showIt(4,3,'R'); // add a Robot at 4,3
System.out.println(c.toString()); // display result
}
```

I created the class, included a 2D array as data. I wrote the constructor which creates a suitably sized array filled with # (for border) and spaces elsewhere. I wrote the showIt method which puts R at the correct location (remember the border). Then wrote and tested the toString method. Finally, amended the code to put my student number on the middle of the border and checked it.

Here's the implementation of my ConsoleCanvas class:

```
package RobotSim;

public class ConsoleCanvas {

    private int xSize, ySize;
    private char[][] displayChars;   // 2D array of characters

    ConsoleCanvas (int xS, int yS, String userStr) {
        xSize = xS + 2;
        ySize = yS + 2;
        displayChars = new char[xSize][ySize];  // create array big enough for canvas and border
        padChars(' ', '#', userStr);   // now set up array, spaces, and border
    }
    /**
     * function to fill the displayChars array with character pchar, putting bchar in the border
     * @param pchar      character to pad array
     * @param bchar      character used for border
     * @param userStr    8 char string identifying user  (put in centre of top row)
     */
    private void padChars(char pchar, char bchar, String userStr) {
        int topchk = Math.max((xSize - 8) / 2, 0);   // used to ensure stdnum in middle of top row : max ensures not -ve

        for (int xct = 0; xct < xSize; xct++) {
```

```
			for (int yct = 0; yct < ySize; yct++) {
				if (xct > 0 && xct < xSize - 1 && yct > 0 && yct < xSize - 1)
					displayChars[xct][yct] = pchar;
				else if (xct >= topchk && xct < 8 + topchk && yct == 0) displayChars[xct][yct] = userStr.charAt(xct-topchk);
				else displayChars[xct][yct] = bchar;
			}
		}
	}
	/**
	 * specific what is shown at position x,y
	 * @param x
	 * @param y
	 * @param ch
	 */
	public void showIt(int x, int y, char ch) {
		displayChars[x+1][y+1] = ch;	// puts ch into x, y'th element in displayChars
	}
	/**
	 * function to output a row of the building with walls/objects in it if necessary
	 * does so by outputting displayChars array, preceded and followed by pchar character
	 * @param pchar
	 */
	public String toString () {
		String ans = "";
		for (int yct = 0; yct < ySize; yct++) {
			for (int xct = 0; xct < xSize; xct++) ans += displayChars[xct][yct];
			ans += '\n';
		}
		return ans;
	}

	public static void main(String[] args) {
		ConsoleCanvas c = new ConsoleCanvas (20, 6, "32004724"); // create a canvas
		c.showIt(4,3,'R'); // add a Robot at 4,3
		System.out.println(c.toString()); // display result
	}

}
```

When I ran and tested the program, it displayed the arena, but the bottom border is not showing.

```
#######32004724#######
#                    #
#                    #
#                    #
#    R               #
#                    #
#                    #
#                    #
```

Memory updated

It looks like the top and side borders are displayed correctly, but the issue with the bottom border not showing might be due to a logic error in how you are setting the characters in your `padChars` method.

To fix this, let's revisit the logic in the nested loops:

Key adjustment in `padChars` method

Make sure the last row of the array (i.e., the bottom border) is filled with `bchar` as well. In your current logic, it seems like the condition to set `bchar` on the last row or last column isn't being met fully.

Here's a revised version of the `padChars` method:

```java
private void padChars(char pchar, char bchar, String userStr) {
    int topchk = Math.max((xSize - 8) / 2, 0); // ensure student number is centered on top
row

    for (int yct = 0; yct < ySize; yct++) {
        for (int xct = 0; xct < xSize; xct++) {
            if (yct == 0 && xct >= topchk && xct < topchk + 8) { // Center the student num
ber on the top row
                displayChars[xct][yct] = userStr.charAt(xct - topchk);
            } else if (xct == 0 || xct == xSize - 1 || yct == 0 || yct == ySize - 1) {
                displayChars[xct][yct] = bchar; // Create border for the entire boundary
            } else {
                displayChars[xct][yct] = pchar; // Fill inside with pchar
            }
        }
    }
}
```

This modification checks if a position is on any border (top, bottom, left, or right) and fills it with `bchar` correctly.

Reasoning:

- We check if `xct` or `yct` are at the boundaries (`0` or `xSize - 1`, `ySize - 1`). If so, we fill them with the border character `bchar`.
- We maintain the condition to insert the student number on the top row only where applicable.

This should correctly draw the entire border, including the bottom. Try running it with this adjustment, and the bottom border should appear correctly.

## 4.2 Incorporating ConsoleCanvas in Robot simulation

In the Robot class, implement the following method
/**
 * display the Robot in the canvas
 * @param c the canvas used
 */
public void displayRobot(ConsoleCanvas c) {
<< call the showIt method in c to put a R where the Robot is
}

To implement the `displayRobot` method in the `Robot` class, you simply need to call the `showIt` method of the `ConsoleCanvas` object (`c`) using the robot's current coordinates. Here's how you can do it:

Implementation of `displayRobot` method

```java
public void displayRobot(ConsoleCanvas c) {
    c.showIt(this.rx, this.ry, 'R'); // Call showIt to place 'R' at the robot's (rx, ry) position
}
```

Explanation:

- The `showIt` method in `ConsoleCanvas` expects the `x` and `y` coordinates of where to place the character, and the character itself ('R' in this case).
- The method uses `this.rx` and `this.ry` as the robot's current position. It places the character 'R' on the canvas at those coordinates.

Full `Robot` class method implementation:

If the `Robot` class isn't fully shown above, here’s how this specific method fits within the class:

```java
package RobotSim;

public class Robot {
    private int rx, ry; // robot coordinates

    public Robot(int x, int y) {
        rx = x;
        ry = y;
    }

    /**
     * Display the Robot in the canvas
     * @param c the canvas used
     */
    public void displayRobot(ConsoleCanvas c) {
        c.showIt(this.rx, this.ry, 'R'); // Add robot character at the robot's coordinates
    }
```

```
        // Other methods for the Robot class...
    }
```

Next steps:

- Now that the `displayRobot` method is implemented, you can update the code in `RobotArena` to call this method for each robot when displaying the arena. This ensures that each robot's position is drawn onto the canvas.

Would you like to continue with this part of incorporating it in `RobotArena`?

In the RobotArena class implement the following method (you can use an enhanced for loop)
/**
 * show all the Robots in the interface
 * @param c the canvas in which Robots are shown
 */
public void showRobots(ConsoleCanvas c) {
<< loop through all the Robots calling the displayRobot method >>
}
It is also suggested that you write 'getter' methods to return the x and y size of the arena.

To implement the `showRobots` method and the getter methods for the arena size in the `RobotArena` class, follow the steps below:

1. Implementing `showRobots` Method

This method will loop through all the robots in the `manyRobots` ArrayList and call the `displayRobot` method on each robot, passing the provided `ConsoleCanvas` object as a parameter.

2. Adding Getter Methods for `xmax` and `ymax`

These methods will return the size of the arena.

Full Implementation of the Required Code:

```
java


package RobotSim;

import java.util.ArrayList; // import ArrayList class
import java.util.Random; // import Random class

public class RobotArena {

        private int xmax, ymax; // defines size of arena
        private Robot r; // define a single robot
        private int rx, ry; // defines coordinates of first robot
        ArrayList<Robot> manyRobots; // declare an ArrayList called manyRobots

        /**
         * Constructor to create the arena
         */
        public RobotArena(int x, int y) {
                manyRobots = new ArrayList<Robot>(); // create an empty ArrayList
                xmax = x; // set x coordinate for arena
                ymax = y; // set y coordinate for arena
        }

        /**
         * A method that spawns a robot in a random location of the arena
         */
        public void addRobot() {
                Random randomGenerator = new Random(); // create a Random object
                do {
                        rx = randomGenerator.nextInt(xmax);
                        ry = randomGenerator.nextInt(ymax);
                } while (getRobotAt(rx, ry) != null); // ensure no overlap with existing r
obots

                manyRobots.add(new Robot(rx, ry)); // create and add the robot
        }

        /**
         * Search the ArrayList of Robots to see if there is a Robot at (x, y)
         * @param x x-coordinate to check
         * @param y y-coordinate to check
         * @return null if no Robot there, otherwise return Robot
         */
        public Robot getRobotAt(int x, int y) {
                for (Robot r : manyRobots) {
                        if (r.getX() == x && r.getY() == y) {
                                return r;
                        }
                }
                return null;
        }

        /**
         * Show all the Robots in the interface
         * @param c the canvas in which Robots are shown
         */
        public void showRobots(ConsoleCanvas c) {
                for (Robot r : manyRobots) {
                        r.displayRobot(c); // call the displayRobot method for each robot
                }
        }

        /**
         * Get the x-size of the arena
         * @return x-size
         */
        public int getXSize() {
                return xmax;
        }

        /**
         * Get the y-size of the arena
         * @return y-size
         */
        public int getYSize() {
                return ymax;
        }

        /**
         * Return information about the arena and all robot contents
         */
        public String toString() {
                String ans = "";
                for (Robot r : manyRobots) {
                        ans += r.toString() + "\n";
                }
                return "Arena " + xmax + " by " + ymax + " with: \n" + ans;
        }

        // Main method for testing, if needed
        public static void main(String[] args) {
                RobotArena arena = new RobotArena(20, 10);
                arena.addRobot();
                arena.addRobot();
                arena.addRobot();
                System.out.println(arena.toString());
        }
}
```

Explanation:

1. `showRobots` **Method**:
    - This method uses an enhanced for loop (`for (Robot r : manyRobots)`) to iterate through all robots in the `manyRobots` ArrayList.
    - For each robot (`r`), it calls the `displayRobot` method, passing the canvas (`c`) to draw the robot at its current location.
2. **Getter Methods (`getXSize` and `getYSize`)**:
    - `getXSize` returns the x-dimension of the arena.
    - `getYSize` returns the y-dimension of the arena.
3. **Main Method**:
    - Included for testing the `RobotArena` class functionality.

Next Steps:

With this implementation, you can now call the `showRobots` method from the `RobotInterface` class to draw all robots on the canvas.

Would you like to proceed with modifying `RobotInterface` or adding more functionality to the project?

In the RobotInterface class, you need the following
a) Write the following method
/**
/**
* Display the Robot arena on the console
*
*/
void doDisplay() {
// determine the arena size
// hence create a suitable sized ConsoleCanvas object
// call showRobots suitably
// then use the ConsoleCanvas.toString method
}
b) Add an option in RobotInterface such that when the user presses ‘d’ or ‘D’, doDisplay is called.

To incorporate these changes in the `RobotInterface` class, let's complete the following steps:

Part (a): Writing the `doDisplay` Method

1. Determine the size of the arena using the getter methods (`getXSize` and `getYSize`).
2. Create a `ConsoleCanvas` object with the appropriate size and your student number.
3. Call the `showRobots` method on the `myArena` object to draw the robots on the canvas.
4. Print the string representation of the canvas using the `toString` method.

Part (b): Adding an Option for Displaying the Arena

1. Add an option in the `switch` statement to call `doDisplay` when the user presses 'd' or 'D'.

Updated `RobotInterface` Class Code

java

```java
package RobotSim;

import java.util.Scanner;

/**
 * Simple program to show arena with multiple robots
 * @author Ylldrit Miftari
 *
 */
public class RobotInterface {

        private Scanner s; // scanner used for input from user
        private RobotArena myArena; // arena in which Robots are shown

        /**
         * constructor for RobotInterface
         * sets up scanner used for input and the arena
         * then has main loop allowing user to enter commands
         */
        public RobotInterface() {
                s = new Scanner(System.in); // set up scanner for user input
                myArena = new RobotArena(20, 6); // create arena of size 20*6

                char ch = ' ';
                do {
                        System.out.print("Enter (A)dd Robot, (D)isplay arena, get (I)nform
ation or e(X)it > ");
                        ch = s.next().charAt(0);
                        s.nextLine();
                        switch (ch) {
                                case 'A':
                                case 'a':
                                        myArena.addRobot(); // add a new Robot to arena
                                        break;
                                case 'D':
                                case 'd':
                                        doDisplay(); // display the arena with robots
                                        break;
                                case 'I':
                                case 'i':
                                        System.out.print(myArena.toString());
                                        break;
                                case 'x':
                                        ch = 'X'; // when X detected program ends
                                        break;
                        }
                } while (ch != 'X'); // test if end

                s.close(); // close scanner
        }

        /**
         * Display the Robot arena on the console
         */
        void doDisplay() {
                // determine the arena size using getter methods
                int xSize = myArena.getXSize();
                int ySize = myArena.getYSize();

                // create a ConsoleCanvas object with the appropriate size
                ConsoleCanvas canvas = new ConsoleCanvas(xSize, ySize, "32004724");

                // call showRobots method on the arena
                myArena.showRobots(canvas);

                // print the string representation of the ConsoleCanvas
                System.out.println(canvas.toString());
        }

        public static void main(String[] args) {
                RobotInterface r = new RobotInterface(); // just call the interface
        }
}
```

Explanation:

1. **`doDisplay` Method**:
    - The method retrieves the arena's x and y sizes using `getXSize` and `getYSize`.
    - It creates a new `ConsoleCanvas` object using these dimensions and your student number.
    - It calls `showRobots` to place all the robots onto the canvas.
    - Finally, it prints the string representation of the canvas using `toString`.
2. **Adding an Option for Display**:
    - In the main loop, I've added a new `case 'D'` to call `doDisplay` when the user presses 'd' or 'D'.

Testing the Program

Now, when you run the program and press 'D', the arena should display with the border, student number, and all added robots.

Let me know if you need further modifications or assistance!

## 4.3 DirectionEnum

The next main task is to get the Robots to move – and at this stage we will constrain them to moving in the directions North, East, South, and West.
Create a new Enum class called Direction for these four directions and add a method to return a random direction and one which returns the next direction (eg if East, next is South)

Memory updated

Here's an outline for implementing the `Direction` enum class:

1. **Create the Enum**: Define the directions (`NORTH`, `EAST`, `SOUTH`, `WEST`) and implement methods for generating a random direction and obtaining the next direction in sequence.

2. **Implement Random and Next Direction Methods**.

Here is the complete code:

```java
package RobotSim;

import java.util.Random;

public enum Direction {
    NORTH, EAST, SOUTH, WEST;

    // Method to get a random direction
    public static Direction getRandomDirection() {
        Random random = new Random();
        return values()[random.nextInt(values().length)];
    }

    // Method to get the next direction in sequence
    public Direction nextDirection() {
        return values()[(this.ordinal() + 1) % values().length];
    }
}
```

Explanation:

- **Random Direction**: Uses `values()` to get an array of all enum values, and selects one randomly.
- **Next Direction**: Uses the ordinal value of the current direction, incrementing it and using modulo to cycle through the directions.

This setup will allow you to easily retrieve a random direction and get the next direction in sequence for future robot movements.

Then add to Robot an extra variable to specify the direction the Robot will travel (of type Direction).

To add an extra variable to specify the direction of travel for a robot, follow these steps:

1. **Declare a variable of type `Direction` in the `Robot` class**.

2. **Initialize this variable** within the `Robot` constructor or a method.

Here's the updated `Robot` class:

```java
package RobotSim;

public class Robot {
    private int x, y;  // Coordinates of the Robot
    private Direction direction; // Variable to specify direction of the Robot

    /**
     * Constructor to create a Robot at a specified position
     * @param x Position on the x-axis
     * @param y Position on the y-axis
     */
    public Robot(int x, int y) {
        this.x = x;
        this.y = y;
        this.direction = Direction.getRandomDirection(); // Initialize with a random direction
    }

    /**
     * Getter method for x-coordinate
     */
    public int getX() {
        return x;
    }

    /**
     * Getter method for y-coordinate
     */
    public int getY() {
        return y;
    }

    /**
     * Display the Robot on the ConsoleCanvas at its current coordinates
     * @param c ConsoleCanvas to show the Robot
     */
    public void displayRobot(ConsoleCanvas c) {
        c.showIt(x, y, 'R');
    }

    /**
     * Getter for the Robot's current direction
     */
    public Direction getDirection() {
        return direction;
    }

    /**
     * Setter to change the Robot's direction
     */
    public void setDirection(Direction direction) {
        this.direction = direction;
    }
}
```

Changes Made:

1. **Declared a variable** `direction` of type `Direction`.
2. **Initialized the** `direction` variable with a random direction in the constructor using `Direction.getRandomDirection()`.

3. **Added getter and setter methods** for the `direction` variable to allow easy access and modification if needed.

This setup ensures that each robot has a specific direction associated with it, which can be utilized in future movement logic.

> Then, add an extra argument to the constructor, passing the initial. [You should change the main of Robot to have a call something like.
> Robot d = new Robot(5, 2,Direction.SOUTH);
> Also, change Robot's toString so that the direction is also added to the string – note that if dir is the robot direction, then dir.toString() returns that direction as a string.
> In RobotArena, where you create a Robot object and add it to the list ensure that the call to the constructor has the extra parameter (note the RobotArena has a random generator which you should use to generate a random direction).

Memory updated

To pass an initial direction to the `Robot` class, you'll need to modify the constructor to include a `Direction` parameter. Additionally, you should update the way the robot is created in `RobotArena`. Here's how to achieve this:

Changes to `Robot` class

1. **Add an extra argument to the constructor**: Modify the `Robot` constructor to accept a `Direction` argument. Set this new direction to the robot's instance variable.

2. **Update the main method in `Robot`**: When creating a robot in the main method or elsewhere, pass a direction as well.

3. **Update `toString` to include direction**: Ensure the `toString` method outputs the robot's direction.

Here is the updated code for the `Robot` class:

```java
package RobotSim;

public class Robot {
    private int x, y;
    private Direction dir; // Add variable for direction

    public Robot(int x, int y, Direction d) {
        this.x = x;
        this.y = y;
        this.dir = d; // Set the robot's direction
    }

    public String toString() {
        return "Robot at (" + x + "," + y + ") facing " + dir.toString();
    }

    public void displayRobot(ConsoleCanvas c) {
        c.showIt(x, y, 'R');
    }

    // Getter methods for x, y, and direction
    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public Direction getDirection() {
        return dir;
    }
}
```

Changes to `RobotArena` class

1. **Modify the `addRobot` method** to use the new constructor with a random direction.

Here’s how the updated `addRobot` method should look:

```
java


package RobotSim;

import java.util.ArrayList;
import java.util.Random;

public class RobotArena {
    private int xmax, ymax;
    private ArrayList<Robot> manyRobots;
    private Random randomGenerator;

    public RobotArena(int x, int y) {
        manyRobots = new ArrayList<>();
        xmax = x;
        ymax = y;
        randomGenerator = new Random();
    }

    public void addRobot() {
        int rx = randomGenerator.nextInt(xmax);
        int ry = randomGenerator.nextInt(ymax);
        Direction randomDirection = Direction.getRandomDirection(); // Get a random direct
ion
        manyRobots.add(new Robot(rx, ry, randomDirection)); // Create and add a robot
    }

    public void showRobots(ConsoleCanvas c) {
        for (Robot r : manyRobots) {
            r.displayRobot(c);
        }
    }

    // Getter methods for xmax and ymax
    public int getXMax() {
        return xmax;
    }

    public int getYMax() {
        return ymax;
        }
    }
```

Changes to the `main` method of `Robot` class for testing

You can test creating a robot by directly calling:

```java
    public static void main(String[] args) {
        Robot d = new Robot(5, 2, Direction.SOUTH);
        System.out.println(d.toString());
    }
```

Summary

- The `Robot` class constructor now includes an extra parameter for direction.
- When adding a robot in `RobotArena`, a random direction is generated and passed to the robot.
- The `toString` method in `Robot` includes the direction information.

Test that the RobotInterface program suitably shows the direction of each Robot when you select the I option.

To test that the `RobotInterface` program shows the direction of each robot when you select the `I` option, you can run the program after making the following changes. The goal is to verify that the output includes the direction each robot is facing when the user selects 'I' for information.

Steps to Test:

1. **Update the `toString` method of the `Robot` class**: Ensure that it returns the direction along with the coordinates.
2. **Add robots with random directions** using the updated `addRobot` method in the `RobotArena` class.
3. **Run the `RobotInterface` class** and add a few robots.
4. **Select the 'I' option** to print information about all robots.

Code Walkthrough:

Make sure that the `toString` method of the `Robot` class includes direction information:

```java
    @Override
    public String toString() {
        return "Robot at (" + x + "," + y + ") facing " + dir.toString();
    }
```

This line will ensure that when you retrieve information, it shows the position and the direction of each robot.

Running the `RobotInterface` program:

- Open your main program in `RobotInterface`.
- Select the 'A' or 'a' option to add robots. This will add robots with random positions and directions.
- Select the 'I' or 'i' option to print information about all robots.

Expected Output Example:

When you select the 'I' option after adding some robots, you should see output like:

```
java

  Robot at (5, 2) facing SOUTH
  Robot at (10, 3) facing EAST
  Robot at (7, 4) facing NORTH
  Robot at (3, 1) facing WEST
```

Running the Program

1. Run the program using your IDE or the command line.
2. In the console, press 'A' multiple times to add robots with random directions.
3. Press 'I' to display information.
4. Confirm that the output includes the direction for each robot, as demonstrated in the example.

If everything is working correctly, you'll see that the direction is included in the information printed when you select the 'I' option!

## 4.4 Moving the Robots – 30 mins
The next stage is to add code so that the Robots move in the appropriate direction. The idea is as follows, which is tried for each Robot in turn.

    Calculate the next x,y position depending on its direction
    If the Robot can move there, update its x,y position to this new position
    Otherwise, change its direction to the next one in the NESW sequence.

A Robot can move to the next position if
    The new position is within the arena
    There is not a Robot already there

This requires a method, say called tryToMove, which is in Robot (as the Robot knows its current position and direction). This first calculates a potential new position, it then tests if this is valid and decides whether to go to that new position or change direction.

Testing of the new position is valid is achieved by calling a method canMoveHere which returns true if the Robot can go to the specified position. This needs to be in RobotArena as the arena knows its size and hence whether the new position is within the arena, and it knows about all the other Robots, and so can tell if there is one at that positon.

So that tryToMove in class Robot can call canMoveHere in class RobotArea, tryToMove should be passed an argument which is the arena it is in.

Hence, in Robot, write method tryToMove which calculates the new position, calls canMoveHere and puts the Robot there if it can, or changes direction.

In RobotArena, write canMoveHere. This returns false if the x,y position passed is outside the arena otherwise it returns whether there is a Robot there (use your existing getRobotAt method).

In RobotArena, write a method moveAllRobots, which calls tryToMove on all Robots, passing itself as the arena (use this as in moveBall in BallPark).

In RobotInterface, add an option to command the Robot arena to move all its Robots once, to then
redraw the arena and list information about all the Robots.

Test the program by first adding to RobotInterface a command called if the user types m or M which calls RobotArena.moveAllRobots and then redraws the area, and then running the program.

Here's how to implement and test this task step-by-step:

Step 1: Implement `tryToMove` Method in `Robot` Class

The `tryToMove` method should:

1. Calculate the new position based on the current direction.
2. Call the `canMoveHere` method in `RobotArena` to check if the new position is valid.
3. If the new position is valid, update the robot’s position.
4. If the new position is not valid, change the direction to the next one in the NESW sequence.

`Robot` Class - Add `tryToMove` Method:

```java
public void tryToMove(RobotArena arena) {
    int newX = x, newY = y;
    switch (dir) {
        case NORTH: newY--; break;
        case EAST: newX++; break;
        case SOUTH: newY++; break;
        case WEST: newX--; break;
    }
```

```
        if (arena.canMoveHere(newX, newY)) {
            x = newX;
            y = newY;
        } else {
            dir = dir.next(); // Assuming you've implemented a `next` method in Direction enum
        }
    }
```

Step 2: Implement `canMoveHere` Method in `RobotArena` Class

The `canMoveHere` method should:

1. Return `false` if the new position is outside the arena.
2. Check if there is already a robot at the specified position using `getRobotAt`.
3. Return `true` if the new position is within the arena and there is no robot at that position.

`RobotArena` Class - Add `canMoveHere` Method:

```java
    public boolean canMoveHere(int newX, int newY) {
        // Check if the new position is within the arena boundaries
        if (newX < 0 || newX >= xSize || newY < 0 || newY >= ySize) {
            return false;
        }

        // Check if there is already a robot at the new position
        return getRobotAt(newX, newY) == null;
    }
```

Step 3: Implement `moveAllRobots` Method in `RobotArena`

The `moveAllRobots` method should:

1. Loop through all robots in the list.
2. Call `tryToMove` for each robot, passing itself (`this`) as the arena.

`RobotArena` Class - Add `moveAllRobots` Method:

```java
    public void moveAllRobots() {
        for (Robot r : robots) {
            r.tryToMove(this);
        }
    }
```

Step 4: Add Option to Move All Robots in `RobotInterface`

In the `RobotInterface` class:

1. Create a method `doMove` to command the arena to move all robots, redraw the arena, and list information about all the robots.
2. Add an option in the main menu to call `doMove` when the user presses 'm' or 'M'.

`RobotInterface` Class - Add `doMove` Method:

```
java


  void doMove() {
      myArena.moveAllRobots(); // Move all robots once
      doDisplay(); // Redraw the arena
      System.out.print(myArena.toString()); // Print information about all robots
  }
```

Step 5: Modify Main Loop in `RobotInterface` to Include 'm' or 'M' Option

In the main loop of `RobotInterface`, modify the `switch` statement to include the 'm' or 'M' option:

```
java


  case 'M':
  case 'm':
      doMove();
      break;
```

Step 6: Test the Program

1. **Run the program** using your IDE or command line.
2. **Add some robots** using the 'A' or 'a' option.
3. **Display the arena** using the 'D' or 'd' option to check the initial positions.
4. **Move the robots** using the 'M' or 'm' option.
5. **Display the arena** again to see the updated positions and verify the movement.

Expected Output

When you press 'M' or 'm', you should see the robots move in their respective directions. If a robot hits a boundary or another robot, it should change direction according to the NESW sequence.

This solution successfully enables the robots to move within the arena while avoiding obstacles, which will be crucial when we move towards implementing animation in the next stage!

I've tested the RobotInterface. The program works, but I'm unsure if the direction is changing or not, and when the robots hit a wall (they do not move away from the wall) or if the robots touch each other (one robot disappears/remains in the same place).

Here's my Robot class to review:

package RobotSim;

public class Robot {

private int x, y, dx, dy, robotId;   // robot position, id and how moves in x, y direction

```
    private static int robotCount = 0;             // used to give each robot a unique number
private Direction direction; // specify direction of the Robot

    /**
     * construct robot at position rx, ry
     * @param rx    x-position of robot
     * @param ry    y-position of robot
     * @param dir   direction of robot
     */
    public Robot (int rx, int ry, Direction dir) {
        x = rx;
        y = ry;
        dx = 1;    // default direction of movement
        dy = 1;
        dir = direction;
        direction = Direction.getRandomDirection(); // Initialize with a random direction
        robotId = robotCount++;
    }

    /**
     * Is the robot at this x,y position
     * @param sx       x position
     * @param sy       y position
     * @return         true if robot is at sx,sy, false otherwise
     */
    public boolean isHere (int sx, int sy) {
        return (x == sx && y == sy);
    }

    /**
     * display the Robot in the canvas
     * @param c the canvas used
     */
    public void displayRobot(ConsoleCanvas c) {
        c.showIt(this.x, this.y, 'R');
    }

/**
 * Getter for the Robot's current direction
 */
public Direction getDirection() {
    return direction;
}

/**
 * Setter to change the Robot's direction
 */
public void setDirection(Direction direction) {
    this.direction = direction;
}
```

```
    public void tryToMove(RobotArena a) {
        int newX = x + dx;
        int newY = y + dx;
        switch (a.canMoveHere(newX, newY)) {
                case 0 :
                        x = newX;
                        y = newY;
                        break;
                case 1:
                        dx = -dx;
                        break;
                case 2:
                        dy = -dy;
                        break;
                case 3:
                        dx = -dx;
                        dy = -dy;
                        break;
        }
    }

        /**
         * return info about robot in string
         */
        public String toString() {
                return "Robot " + robotId + " at (" + x + ", " + y + ")" + " facing " + direction.toString();
        }


        public static void main(String[] args) {
                Robot d = new Robot(5, 3, Direction.SOUTH);                // create Robot
                System.out.println(d.toString());       // print where is
        }

}
```

Here's my RobotArena class to review:

```
package RobotSim;

import java.util.ArrayList; // import ArrayList class
import java.util.Random; // import Random class

public class RobotArena {

        /**
         * Class which creates the robot arena for many robots to spawn in randomly
         */

        private int xmax, ymax;      // defines size of arena
        private Robot r;            // define a single robot
```

```
    private int rx, ry;             // defines coordinates of first robot

    ArrayList<Robot> manyRobots; // declare an ArrayList called manyRobots

    /**
     * Constructor to create the arena
     */
    public RobotArena (int x, int y) {

        // create an *empty* ArrayList from manyRobot variable
        manyRobots = new ArrayList<Robot>();

        xmax = x;           // create x coordinate for arena
        ymax = y;           // create y coordinate for arena
    }

    /**
     * A method which spawns a robot in a random location of the arena
     */
    public void addRobot() {

        Random randomGenerator;     // declare a Random object
        randomGenerator = new Random();     // creating a Random object

        do {
            rx = randomGenerator.nextInt(xmax); // Create x-coordinate for robot within arena
            ry = randomGenerator.nextInt(ymax); // Create y-coordinate for robot within arena
        } while (getRobotAt(rx, ry) != null); // Ensure no robot exists at the chosen position

        Direction randomDirection = Direction.getRandomDirection();

        manyRobots.add(r = new Robot(rx, ry, randomDirection)); // Creates a robot and adds to manyRobots ArrayList
    }

    /**
     * Search ArrayList (manyRobots) to check if there is a robot at x,y
     * @param x       Check x-coordinate
     * @param y       Check y-coordinate
     * @return    null if no Robot is found, otherwise return Robot
     */
    public Robot getRobotAt(int x, int y) {
        for (Robot r : manyRobots) {
            if (r.isHere(x, y)) {
                return r;
            }
        }
        return null;
    }

    /**
```

```
 * Show all the Robots in the interface
 * @param c the canvas in which Robots are shown
 */
public void showRobots(ConsoleCanvas c) {
	for (Robot r : manyRobots) {
		r.displayRobot(c); // call the displayRobot method for each robot
	}
}

/**
 * Get x size of the arena
 * @return x
 */
public int getX() {
	return xmax;
}

/**
 * Get y size of the arena
 * @return y
 */
public int getY() {
	return ymax;
}

public int canMoveHere(int x, int y) {
	int status = 0;	// default - ok to go here
	if (x<0 || x>=xmax)
		status += 1;	// indicate hit vertical wall
	if (y<0 || y>= ymax)
		status += 2;	// indicate hit horizontal wall
	return status;
}

public void moveAllRobots() {
    for (Robot r : manyRobots) {
        r.tryToMove(this);
    }
}

/**
 * return information about arena and all robot contents
 */
public String toString() {
	String ans = "";

	// Enhanced for loop method which concatenates many robots' coordinates to ArrayList
	for (Robot r : manyRobots) ans += r.toString() + "\n";

	return "Arena " + xmax + " by " + ymax + " with: " + "\n" + ans;
}
```

```
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// Code to test arena
		RobotArena a = new RobotArena(20, 10); // create Robot arena
		a.addRobot();	// add first robot to arena
		a.addRobot();	// add second robot to arena
		a.addRobot();	// add third robot to arena
		System.out.println(a.toString());	// print arena size and where robots are
	}

}
```

Here's my RobotInterface to review:

```
package RobotSim;

import java.util.Scanner;

/**
 * Simple program to show arena with multiple robots
 * @author Ylldrit Miftari
 *
 */
public class RobotInterface {

	private Scanner s;				// scanner used for input from user
    private RobotArena myArena;			// arena in which Robots are shown

    /**
	 * constructor for RobotInterface
	 * sets up scanner used for input and the arena
	 * then has main loop allowing user to enter commands
     */
    public RobotInterface() {
    	 s = new Scanner(System.in);			// set up scanner for user input
    	 myArena = new RobotArena(20, 6);	// create arena of size 20*6

    	char ch = ' ';
    	do {
        	System.out.print("Enter (A)dd Robot, (D)isplay arena, get (I)nformation, (M)ove Robots or
e(X)it > ");
        	ch = s.next().charAt(0);
        	s.nextLine();
        	switch (ch) {
    			case 'A' :
    			case 'a' :
        					myArena.addRobot();  // add a new Robot to arena
        					break;
```

```
                case 'D':
                case 'd':
                            doDisplay(); // display the arena with robots
                            break;
            case 'I' :
            case 'i' :
                        System.out.print(myArena.toString());
                    break;
            case 'M':
            case 'm':
                  doMove();
                  break;
            case 'x' :  ch = 'X';               // when X detected program ends
                        break;
        }
        } while (ch != 'X');                 // test if end

    s.close();                               // close scanner
  }

    /**
     * Display the Robot arena on the console
     */
    void doDisplay() {
        // determine the arena size using getter methods
        int xSize = myArena.getX();
        int ySize = myArena.getY();

        // create a ConsoleCanvas object with the appropriate size
        ConsoleCanvas canvas = new ConsoleCanvas(xSize, ySize, "32004724");

        // call showRobots method on the arena
        myArena.showRobots(canvas);

        // print the string representation of the ConsoleCanvas
        System.out.println(canvas.toString());
    }

    void doMove() {
      myArena.moveAllRobots(); // Move all robots once
      doDisplay(); // Redraw the arena
      System.out.print(myArena.toString()); // Print information about all robots
    }

    public static void main(String[] args) {
        RobotInterface r = new RobotInterface();    // just call the interface
    }

}
```

Here's my enum class Direction, for further reference, if helpful:

```
package RobotSim;

import java.util.Random;

public enum Direction {
        NORTH, EAST, SOUTH, WEST;

        /**
         * Method to get a random direction
         * Uses values() to get an array of all enum values, and selects one randomly
         */
    public static Direction getRandomDirection() {
        Random random = new Random();
        return values()[random.nextInt(values().length)];
    }

    /**
     * Method to get next direction in sequence
     * Uses the ordinal value of the current direction, incrementing it
     * and using modulo to cycle through the directions
     */
    public Direction nextDirection() {
        return values()[(this.ordinal() + 1) % values().length];
    }
}
```

Your code shows good effort towards implementing robot movement, but there are a few areas that could lead to unexpected behavior in how the robots handle collisions, walls, and direction changes. Here's a breakdown of the potential issues and improvements:

Problems & Suggestions:

1. **Updating Robot Positions with Correct Direction**: In the `` `tryToMove` `` method, you need to update the robot's position based on its current direction (`` `NORTH` ``, `` `EAST` ``, etc.). However, the code currently uses `` `dx` `` and `` `dy` ``, which are not linked to the robot's direction. Instead, we should compute `` `newX` `` and `` `newY` `` based on the robot's direction.

   Solution: Implement a switch statement or conditional structure that updates `` `newX` `` and `` `newY` `` based on the robot's current direction (`` `NORTH` ``, `` `EAST` ``, `` `SOUTH` ``, `` `WEST` ``).

2. **Handling Direction Changes**: If a robot hits a wall or another robot, the current implementation uses `` `dx` `` and `` `dy` `` to switch directions. Instead, you should be leveraging the `` `Direction` `` enumeration and using the `` `nextDirection()` `` method to set a new direction.

Solution: In `tryToMove`, when a robot can’t move, call `direction.nextDirection()` to update the robot’s direction.

3. **Fixing the Disappearance Issue**: One robot may be disappearing or remaining in the same spot because of how you're checking for a collision. The method `canMoveHere` only checks for the position and boundary conditions. However, if two robots are on the same spot, it needs to ensure that one robot doesn't overwrite the other’s position or stay there.

   Solution: Update `canMoveHere` to check not only for the boundaries but also for existing robots at the target location using `getRobotAt`.

Recommended Code Updates:

Robot Class:

```java
public void tryToMove(RobotArena a) {
    // Calculate the new coordinates based on the current direction
    int newX = x, newY = y;
    switch (direction) {
        case NORTH:
            newY--; // Move up
            break;
        case EAST:
            newX++; // Move right
            break;
        case SOUTH:
            newY++; // Move down
            break;
        case WEST:
            newX--; // Move left
            break;
    }

    // Check if the new position is valid
    if (a.canMoveHere(newX, newY)) {
        x = newX;
        y = newY;
    } else {
        // Change the direction if the move is not possible
        direction = direction.nextDirection();
    }
}
```

RobotArena Class:

1. **Updating `canMoveHere` Method**:

```java
public boolean canMoveHere(int x, int y) {
    // Check if the new coordinates are within the arena
    if (x < 0 || x >= xmax || y < 0 || y >= ymax) {
        return false; // Position is outside the arena
    }

    // Check if there is already a robot at the new coordinates
```

```
        return getRobotAt(x, y) == null;
    }
```

2. **Update `moveAllRobots` to call `tryToMove`:**

```
java

    public void moveAllRobots() {
        for (Robot r : manyRobots) {
            r.tryToMove(this); // Pass the current instance of RobotArena
        }
    }
```

Key Changes:

- **Movement Calculation**: Instead of relying on `dx` and `dy`, the robot's movement is now determined based on its `Direction` enum.
- **Boundary and Collision Checking**: The `canMoveHere` method is more comprehensive, ensuring that the robot doesn't move outside the arena or onto an occupied position.
- **Direction Change**: When a move is not possible, the direction changes to the next one in the NESW sequence using the `nextDirection` method in the `Direction` enum.

Testing Suggestions:

- **Test Moving into Walls**: Start the robot next to the arena boundaries to confirm it changes direction when hitting a wall.
- **Test Collisions with Other Robots**: Add multiple robots and confirm that they don't overlap or disappear when they reach each other.
- **Test the Direction Changing**: Print the direction of each robot after each move attempt to verify it changes correctly when required.

These changes should solve the current issues with robot movement, collisions, and direction changes. Let me know if you need further clarification or additional help!

## 4.5 Animation

In RobotInterface, add an option so that
the program moves all the Robots 10 times. Each time the Robots move the program should redraw the arena, list the Robots and their positions and then delaying by 200ms.

Test the program.

Memory updated

To implement the animation, you'll add an option in the `RobotInterface` class to move all the robots 10 times. During each movement, the arena should be redrawn, and the positions of the robots should be listed. Additionally, you'll include a 200ms delay after each redraw. Here's the step-by-step breakdown:

Step 1: Modify `RobotInterface` to Include Animation Option

Add a new case to handle the animation command:

1. Add a method called `doAnimate` to handle the animation.
2. In this method, move all robots 10 times, redraw the arena each time, list the robot positions, and include a 200ms delay.

Step 2: Implement the `doAnimate` Method

Here's the updated `RobotInterface` code with the `doAnimate` method:

```java
// Add this new method to the RobotInterface class
void doAnimate() {
    for (int i = 0; i < 10; i++) {
        myArena.moveAllRobots(); // Move all robots once
        doDisplay(); // Redraw the arena
        System.out.print(myArena.toString()); // Print information about all robots

        // Delay for 200 milliseconds
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
            System.err.println("Animation interrupted");
        }
    }
}
```

Step 3: Add Option for Animation in the Main Loop

In the main loop, add a case to handle the user's choice for animation:

```java
do {
    System.out.print("Enter (A)dd Robot, (D)isplay arena, get (I)nformation, (M)ove Robot
s, (R)un Animation or e(X)it > ");
    ch = s.next().charAt(0);
    s.nextLine();
    switch (ch) {
        case 'A':
        case 'a':
            myArena.addRobot(); // add a new Robot to arena
            break;
        case 'D':
        case 'd':
            doDisplay(); // display the arena with robots
            break;
        case 'I':
        case 'i':
            System.out.print(myArena.toString());
            break;
        case 'M':
        case 'm':
            doMove();
            break;
        case 'R': // New case for running animation
        case 'r':
            doAnimate();
            break;
        case 'x':
            ch = 'X'; // when X detected program ends
            break;
    }
} while (ch != 'X'); // test if end
```

Testing

Run the program and choose the new option (e.g., `R`) to run the animation. Observe if the robots move smoothly, and if their positions update correctly.

This should allow you to see the robots animate across the arena, changing direction if they hit a wall or another robot.

We have successfully completed the tasks for week 4. Next week, we will extend the simulation so as to create new arenas and to save the arena and its Robots to a file, or load a new arena from a file. I'll inform you when we'll work on these tasks.

Memory updated

Great job completing Week 4 of your coursework! Let me know whenever you're ready to work on the new tasks for creating and saving arenas.

# Week 5 Tasks New Simulation and File options

This week we will continue the development of the Robot simulation, where you will add extra functionality – to allow for a new Simulation, and to Save a simulation of Load one from files.

## 5.1 New Arena
Add an option to RobotInterface whereby the user can specify a new arena.
The user should be prompted for the x and y sizes of the arena, thereby producing an empty arena. After that, the user can add new Robots.

To implement this, we'll add a new option in the `RobotInterface` class to create a new arena. This involves prompting the user to input the new x and y sizes for the arena.

Here's the updated plan for `RobotInterface`:

1. **Prompt for New Arena Sizes:** Add a new option to ask the user for the new arena size.
2. **Create the New Arena:** Use the sizes provided by the user to create a new `RobotArena` object.
3. **Clear the Old Arena:** When a new arena is created, discard any existing robots in the old arena.

Code Changes for `RobotInterface` Class

Step 1: Add the New Option

We will add a new case for creating a new arena with a user-specified size.

Updated `RobotInterface` Class

```java
package RobotSim;

import java.util.Scanner;

/**
 * Simple program to show arena with multiple robots
 */
public class RobotInterface {

    private Scanner s; // scanner used for input from user
    private RobotArena myArena; // arena in which Robots are shown

    /**
     * constructor for RobotInterface
     * sets up scanner used for input and the arena
     * then has main loop allowing user to enter commands
     */
    public RobotInterface() {
        s = new Scanner(System.in); // set up scanner for user input
        myArena = new RobotArena(20, 6); // create arena of size 20*6

        char ch = ' ';
        do {
            System.out.print("Enter (A)dd Robot, (D)isplay arena, get (I)nformation, (M)ove Robots, (N)ew Arena, or e(X)it > ");
            ch = s.next().charAt(0);
            s.nextLine();
            switch (ch) {
                case 'A':
                case 'a':
                    myArena.addRobot(); // add a new Robot to arena
                    break;
                case 'D':
                case 'd':
                    doDisplay(); // display the arena with robots
                    break;
                case 'I':
                case 'i':
                    System.out.print(myArena.toString());
                    break;
                case 'M':
                case 'm':
                    doMove();
                    break;
                case 'N':
                case 'n':
                    createNewArena(); // call the method to create a new arena
                    break;
                case 'x':
                    ch = 'X'; // when X detected, program ends
                    break;
            }
        } while (ch != 'X'); // test if end
        s.close(); // close scanner
    }

    /**
     * Create a new arena based on user input
     */
    private void createNewArena() {
        System.out.print("Enter new x size of arena: ");
        int xSize = s.nextInt();
        System.out.print("Enter new y size of arena: ");
        int ySize = s.nextInt();
        s.nextLine(); // clear the buffer
        myArena = new RobotArena(xSize, ySize); // create a new arena with specified size
        System.out.println("New arena created with size " + xSize + " x " + ySize);
    }

    /**
     * Display the Robot arena on the console
     */
    void doDisplay() {
        // determine the arena size using getter methods
        int xSize = myArena.getX();
        int ySize = myArena.getY();

        // create a ConsoleCanvas object with the appropriate size
        ConsoleCanvas canvas = new ConsoleCanvas(xSize, ySize, "32004724");

        // call showRobots method on the arena
        myArena.showRobots(canvas);

        // print the string representation of the ConsoleCanvas
        System.out.println(canvas.toString());
    }

    /**
     * Move all Robots and redraw the arena
     */
    void doMove() {
        myArena.moveAllRobots(); // Move all robots once
        doDisplay(); // Redraw the arena
        System.out.print(myArena.toString()); // Print information about all robots
    }

    public static void main(String[] args) {
        RobotInterface r = new RobotInterface(); // just call the interface
    }
}
```

Summary of Changes:

- Added a new case for creating a new arena when the user inputs 'N' or 'n'.
- Added a method `createNewArena()` to prompt for new arena dimensions and create the arena.
- When a new arena is created, all existing robots are removed, and the new arena is set up based on the given dimensions.

Testing

To test this:

1. Run the program and choose the new `(N)ew Arena` option.
2. Specify dimensions for the new arena.
3. Add new robots and display the arena to confirm it is working correctly with the new dimensions.

Let me know if you encounter any issues or need further refinements!

### File Options

Next we are to add options to allow the arena and its contents to be saved to file or loaded from file.
The user should be able to select the name of the files – it is suggested you use JFileChooser, Exception handling should be provided in case there are errors.
You can choose whether to use binary or text files. As the arena can be created from a text string, text files may be easier, especially as you can adapt the file handling for the ball example provided.

## 5.2 Save File
First add an option to save information about the arena to a file and test it.

Memory updated

To implement the "Save File" feature, you'll need to follow these steps:

1. **Choose the File Format**: Since you decided on text files, we'll focus on saving the arena's dimensions and robots' positions, IDs, and directions to a text file.
2. **Use JFileChooser**: To make file selection user-friendly, you can use `JFileChooser` for choosing where to save the file.
3. **Handle Exceptions**: You'll want to catch potential I/O exceptions to handle cases where file operations fail.

Step-by-Step Code

Here’s a suggestion for adding the save functionality to your `RobotInterface` class:

1. Modify RobotArena Class for Saving Data

You'll need a method in `RobotArena` to generate a string representation that can be saved to a file.

```java
// RobotArena Class

public String toSaveString() {
    StringBuilder sb = new StringBuilder();
    sb.append(xmax).append(" ").append(ymax).append("\n"); // save arena size
    for (Robot r : manyRobots) {
        sb.append(r.getX()).append(" ").append(r.getY()).append(" ").append(r.getDirection
()).append("\n");
    }
    return sb.toString();
}
```

2. Update RobotInterface for Saving to a File

Here’s an updated version of `RobotInterface` with a `saveArena` method:

java

```java
// RobotInterface Class

import javax.swing.JFileChooser;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

void saveArena() {
    JFileChooser fileChooser = new JFileChooser();
    int userSelection = fileChooser.showSaveDialog(null);

    if (userSelection == JFileChooser.APPROVE_OPTION) {
        File fileToSave = fileChooser.getSelectedFile();
        try (FileWriter writer = new FileWriter(fileToSave)) {
            writer.write(myArena.toSaveString()); // Save arena data
            System.out.println("Arena saved successfully to " + fileToSave.getAbsolutePath());
        } catch (IOException e) {
            System.out.println("Error saving arena: " + e.getMessage());
        }
    } else {
        System.out.println("Save command cancelled by user.");
    }
}
```

3. Add Save Option in Main Loop

Add an option in the main loop to trigger the `saveArena()` method.

java

```java
// Inside RobotInterface class

public RobotInterface() {
    // existing code...

    char ch = ' ';
    do {
        System.out.print("Enter (A)dd Robot, (D)isplay arena, get (I)nformation, (M)ove Robots, (N)ew arena, (S)ave arena, or e(X)it > ");
        ch = s.next().charAt(0);
        s.nextLine();
        switch (ch) {
            // existing cases...
            case 'S':
            case 's':
                saveArena();
                break;
        }
    } while (ch != 'X');
}
```

Testing

1. Run the program and add some robots to your arena.
2. Use the newly added `Save Arena` option.
3. Verify that the file is created successfully with the expected contents.

Would you like guidance on loading the arena next?

## 5.3 Load File

An update with my classes. I imported a class called TextFile (created by my lecturer, adapted by myself) within my RobotSim package so that I can save data on the arena and its robots' positions.

Here's the TextFile class:

```
package RobotSim;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;

    /**
     * A class for reading from and writing to text files
     * User can define the extension for such files
     * Now exist methods to read/write whole file as single string lines separated by \n
     * @author 32004724
     *
     */

public class TextFile {

    private String extension;               // extension of files that are opened/create   eg txt
    private String extDescription;          // description used for these eg Text Files
    private String nameOfFile = "";         // name of file used in last operation
    private BufferedReader inBuffer;        // buffer used for reading files
    private BufferedWriter outBuffer;       // buffer used for writing files
    private FileReader reader;              // used for reading files
    private String fileLine;                // string containing latest line from file
    private JFileChooser chooser;           // object used to allow user to select files

    /**
     *  constructor used to set up object ... pass it the description of files and extension
     *  @param ftypeD is string defining type of files - eg Text files
     *  @param fExtension is string defining extension eg "txt" for text files
     */
    public TextFile(String ftypeD, String fExtension) {
        extDescription = ftypeD;                        // remember these arguments
        extension = fExtension;
```

```
                // now set up the dialog whereby user sets file name
        String curDir = System.getProperty("user.dir");                     // find current
folder
        chooser = new JFileChooser(curDir);                                     // and set
chooser to start there
        chooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);   // set chooser up
to find
        FileNameExtensionFilter filter = new FileNameExtensionFilter(extDescription, extension);
                                            // define filter so only look for files with
extension
        chooser.setFileFilter(filter);              // set up chooser
    }

    /**
     * report name of file used for reading/writing - empty string if open/create file failed
     * @return  name of file
     */
    public String usedFileName() {
        return    nameOfFile;
    }

    /**
     * method to open a text file for reading
     * @return true if file has been opened
     */
    public boolean openFile() {
        nameOfFile = "";                                        // by default no file specified
        int oDVal = chooser.showOpenDialog(null);               // run dialog for asking user to
select file

        if (oDVal == JFileChooser.APPROVE_OPTION) {                 // if successful

            File selFile = chooser.getSelectedFile();           // get the file chosen
            if(selFile.isFile()){                               // if it is a file
                try {
                    reader = new FileReader(selFile);           // set up reader for the file
                    inBuffer = new BufferedReader(reader);      // set up buffer used for
reading lines from file
                    nameOfFile = selFile.getAbsolutePath();     // remember name of file
(shows operation a success)
                } catch (FileNotFoundException e) {
                    e.printStackTrace();                        // if there is a problem, reporr it
                }
            }
        }
        return nameOfFile.length()>0;                           // return true if a file name
specified and file opened ok
    }

    /**
     * Close the file which has been read
```

```
         */
        void closeFile() {
                try {
                        inBuffer.close();                                       // close the file
                } catch (IOException e) {
                        e.printStackTrace();                                    // report error if this did not
work
                }
        }

        /**
         * function to get the next line from the text file
         * @return true if a line is there
         */
        boolean getNextline() {
                boolean ok = false;                                                     // assume false
                try {
                        fileLine = inBuffer.readLine();                         // read line from inBuffer
                        if (fileLine!=null) ok = true;                  // if not null, then line is there
                } catch (IOException e) {
                        e.printStackTrace();                                    // report any error
                }
                return ok;                                                              // report if successful
        }

        /**
         * return the last line that was read successfully by getNextline
         * @return the line is returned as a string
         */
        String nextLine() {
                return fileLine;                                                // just return fileLine
        }

        /**
         * read all of file, returning one string; each line separated by \n
         * @return
         */
        public String readAllFile() {
                String ans = "";
                while (getNextline())                   // while there is a line to read
                        ans = ans + nextLine() + "\n";          // get it and add to answer
                closeFile();                            // close file
                return ans;
        }


        /**
         * create a new file for writing
         * @return true if file is created
         */
        public boolean createFile() {
```

```
        nameOfFile = "arenafile.txt";                            // no name specified yet
        int oDVal = chooser.showSaveDialog(null);
        if (oDVal == JFileChooser.APPROVE_OPTION) {              // call select file to save
dialog
            File wFile = chooser.getSelectedFile();          // get name of file
                                                                 // check has the right
extension
            if (!chooser.getFileFilter().accept(wFile)) {
                wFile = new File(wFile.getAbsolutePath() + "."+extension);
            }                                                    // add extension if not
            try {
                nameOfFile = wFile.getAbsolutePath();        // remember file name (also
indicates file created)
                outBuffer = new BufferedWriter(new FileWriter(wFile));
                                                                 // set up the output buffer
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return nameOfFile.length()>0;                        // return true if a file name
specified and file created ok
    }

    /**
     * writes a string then newline to the file
     * @param s : string to be written
     */
    void putNextLine(String s) {
        try {
            outBuffer.write(s);                                  // write string
            outBuffer.newLine();                             // and newline
        } catch (IOException e) {
            e.printStackTrace();                             // handle any error
        }
    }

    /**
     * close the file that has been written to
     */
    void closeWriteFile() {
        try {
            outBuffer.close();                               // close it
        } catch (IOException e) {
            e.printStackTrace();                             // print error
        }
    }

    /**
     * write data to created file; data is series of strings separated by \n
     * @param data
     */
    public void writeAllFile(String data) {
        String[] manyStrings = data.split("\n");         // split data into lines
        for (int ct=0; ct<manyStrings.length; ct++)           // for each line
            putNextLine(manyStrings[ct]);                     // put into file
            closeWriteFile();                                 // close file
        }


    public static void main(String[] args) {
        // Code to test text file class
        TextFile tf = new TextFile("Text files", "txt");          // create object looking for *.txt Text files

        if (tf.openFile()) {                                          // open file
            System.out.println("Reading from " + tf.usedFileName());
            System.out.println(tf.readAllFile());                     // read whole file into str which is printed to console

        }
        else System.out.println("No read file selected");

        if (tf.createFile()) {                                        // create file to be written to
            System.out.println("Writing to " + tf.usedFileName());
            tf.writeAllFile("30 10" + "\n" + "0 5 6 EAST"+ "\n" + "1 2 7 SOUTH");
        }
        else System.out.println("No write file selected");
    }

}
```

Here's my RobotInterface class, where I've managed to test run a simulation by saving an arena and position of robots to a text file. Currently, I'm working on an option which loads a saved text file and reads the information shown on the console.

```
package RobotSim;

import java.util.Scanner;

/**
 * Simple program to show arena with multiple robots
 * @author Ylldrit Miftari
 *
 */
public class RobotInterface {

    private Scanner s;                            // scanner used for input from user
  private RobotArena myArena;             // arena in which Robots are shown
  private TextFile tf = new TextFile("Text files", "txt");  // object used to write and read files

  /**
     * constructor for RobotInterface
```

```
         * sets up scanner used for input and the arena
         * then has main loop allowing user to enter commands
     */
    public RobotInterface() {
         s = new Scanner(System.in);                     // set up scanner for user input
         myArena = new RobotArena(20, 6); // create arena of size 20*6

        char ch = ' ';
        do {
                System.out.print("Enter (A)dd Robot, (D)isplay arena, get (I)nformation, "
                                + "(M)ove Robots, (N)ew Arena, (S)ave Arena , (R)un Animation or e(X)it > ");
                ch = s.next().charAt(0);
                s.nextLine();
                switch (ch) {
                        case 'A' :
                        case 'a' :
                                                myArena.addRobot();  // add a new Robot to arena
                                                break;
                                case 'D':
                                case 'd':
                                                        doDisplay(); // display the arena with robots
                                                        break;
                        case 'I' :
                        case 'i' :
                                                System.out.print(myArena.toString()); // display information on arena & robots
                                        break;
                        case 'M':
                        case 'm':
                                   doMove(); // moves all robots at once, then redraws the arena
                                   break;
                case 'N':
                case 'n':
                    createNewArena(); // call the method to create a new arena
                    break;
                case 'S':
                case 's':
                        if (tf.createFile()) // if can create file
                                tf.writeAllFile(myArena.fileString()); // write arena to file
                case 'R':
                case 'r':
                    doAnimate(); // runs animation on robots, moving 10 times
                    break;
                        case 'x' :        ch = 'X';                       // when X detected program ends
                                                break;
                }
                } while (ch != 'X');                             // test if end

       s.close();                                                 // close scanner
    }
```

```
/**
 * Create a new arena based on user input
 */
private void createNewArena() {
    System.out.print("Enter new x size of arena: ");
    int xSize = s.nextInt();
    System.out.print("Enter new y size of arena: ");
    int ySize = s.nextInt();
    s.nextLine(); // clear the buffer
    myArena = new RobotArena(xSize, ySize); // create a new arena with specified size
    System.out.println("New arena created with size " + xSize + " by " + ySize);
}

    /**
     * Display the Robot arena on the console
     */
    void doDisplay() {
            // determine the arena size using getter methods
            int xSize = myArena.getX();
            int ySize = myArena.getY();

            // create a ConsoleCanvas object with the appropriate size
            ConsoleCanvas canvas = new ConsoleCanvas(xSize, ySize, "32004724");

            // call showRobots method on the arena
            myArena.showRobots(canvas);

            // print the string representation of the ConsoleCanvas
            System.out.println(canvas.toString());
    }

    /**
     * Move all robots simultaneously, then update + redraw arena
     */
    void doMove() {
        myArena.moveAllRobots(); // Move all robots once
        doDisplay(); // Redraw the arena
        System.out.print(myArena.toString()); // Print information about all robots
    }

    /**
     * Method called doAnimate which moves all the robots 10 times,
     * redraws the arena each time, lists the robots' positions,
     * and includes a 200ms delay
     */
    void doAnimate() {
        for (int i = 0; i < 10; i++) {
            myArena.moveAllRobots(); // Move all robots once
            doDisplay(); // Redraw the arena
            System.out.print(myArena.toString()); // Print information about all robots
        // Delay for 200 milliseconds
        try {
          Thread.sleep(200);
        } catch (InterruptedException e) {
          System.err.println("Animation interrupted");
        }
      }
    }

    public static void main(String[] args) {
        RobotInterface r = new RobotInterface();      // just call the interface
    }

}
```

I'm currently struggling on the loading option.

In RobotArena, I added a second constructor to create the arena from the loaded text file (providing I had one saved - I did when I did a test run, but it didn't work). Not sure if this constructor is needed if I'm honest.

```
    /**
     * Second constructor to create an arena from a loaded file
     */
    public RobotArena (String fs) {

    }
```

Memory updated

For implementing the load functionality in the `RobotInterface` class, you'll need to adjust `RobotArena` to interpret the text file data, initializing the arena's size and adding robots based on the saved information. Here's a breakdown of the process:

1. Implement the Constructor in `RobotArena`

The second constructor `RobotArena(String fs)` should parse `fs` (the loaded file string) to set up the arena size and robot positions. Here's how it could look:

```java
  /**
   * Second constructor to create an arena from a loaded file
   * @param fs String containing arena and robot details
   */
  public RobotArena(String fs) {
      String[] lines = fs.split("\n"); // split lines from loaded data
      if (lines.length > 0) {
          String[] arenaSize = lines[0].split(" "); // first line for arena size
          if (arenaSize.length == 2) {
              int x = Integer.parseInt(arenaSize[0]);
              int y = Integer.parseInt(arenaSize[1]);
              this.xSize = x; // set arena dimensions
              this.ySize = y;
              this.robots = new ArrayList<>(); // initialize robot list
          }
      }

      // Parse each robot line and add robots to the arena
      for (int i = 1; i < lines.length; i++) {
          String[] robotData = lines[i].split(" ");
          if (robotData.length == 4) {
              int id = Integer.parseInt(robotData[0]);
              int xPos = Integer.parseInt(robotData[1]);
              int yPos = Integer.parseInt(robotData[2]);
              Direction dir = Direction.valueOf(robotData[3]);
              robots.add(new Robot(id, xPos, yPos, dir)); // add robot to list
          }
      }
  }
```

2. Implement Load File Option in `RobotInterface`

Add a new case in the `RobotInterface` menu for loading an arena:

```java
  case 'L':
  case 'l':
      if (tf.openFile()) { // open file chooser dialog
          String loadedData = tf.readAllFile();
          myArena = new RobotArena(loadedData); // pass data to arena constructor
          doDisplay(); // display loaded arena
          System.out.println("Arena loaded successfully from " + tf.usedFileName());
      } else {
          System.out.println("Failed to load arena");
      }
      break;
```

Explanation

- **Constructor in `RobotArena`**: This takes the loaded file's content as a single string, splits it into lines, and extracts the arena dimensions and robot data.
- **File Reading**: The `openFile()` and `readAllFile()` methods from `TextFile` handle the file loading, while `RobotInterface` assigns the loaded data to `myArena`.

Testing

Make sure that the saved file is formatted correctly and matches the expected input format in `RobotArena` to ensure accurate parsing. If it's successful, this setup should load and display the arena as intended. Let me know how this works or if you'd like help tweaking it further!

ChatGPT can make mistakes. Check important info.