Introduction: This program assignment will give you practice with arrays. We suggest that you do following steps:

- Understand the problem and write the specification.
- Design the program by listing the objects, variables and operations.
- Write the program (bottom up)
- Incrementally compile and Test the program and redesign if needed.

Submit your working program as described below. The program should be in a file named RobotGame.java and contain the main program class RobotGame, as well as any other non-standard classes used (including the provided classes Robot and Position). This is an “exercise”, so you may consult freely with any of your classmates. As with all programming projects, each pair should work together and submit only one assignment. The assignment will be graded basically pass/fail (pass = 2 points). The graders may award a single point for good efforts that fall short of complete success.

The problem: Write a program that simulates the popular Unix game “Robots”. “Robots” pits you against evil robots, who are trying to kill you (which is why they are evil). Fortunately for you, even though they are evil, they are not very bright and have a habit of bumping into each other, thus destroying themselves. In order to survive, you must get them to kill each other off, since you have no offensive weaponry.

Getting ready: Create a new directory called hw8 or assignment8 as a subdirectory of your cs12a directory. You may use either tio or Java’s built-in input facilities, although I encourage use to use the built-in facilities. See the sample program JavaIO.java off the class web page for how to use Java’s standard facilities.

The specification: The game is played on a 15 by 15 square board. Your initial position is in the center of the board. At the beginning of the game, you are asked to enter the number of robots and the number of junkheaps to start with. The robots and the junkheaps will then be placed at random locations on the grid. If robots and/or junkheaps get put at the same location you can either try again or just treat the location as a junkheap. It is important that the center square remain vacant for the player. Therefore if robots
and/or junkheaps happen to get put in the center square, then you can either try another random location or just run with fewer robots/junkheaps.

Each turn, the player is allowed to move one square in any direction, stay still, or use his special teleportation device (see the commands below). After the player moves, each of the live robots currently on the grid will try to catch the player by moving one square towards the player’s new location.

Although the player has no offensive weaponry, the player does have a teleportation device that will transport the player to a random unoccupied location on the grid (unfortunately, this new location can be right next to a robot).

There are a few simple rules to follow:

1. If the player moves onto a square already occupied by a junkheap or a robot, the game is over and the player loses.

2. If the player moves off the gameboard, your program can do anything reasonable (player loses game, request new input, teleport, lose turn, etc.).

3. If a robot moves onto the player, the game is over and the player loses (even if two robots collide in the player’s space).

4. If a robot moves onto a junkheap, it dies and joins the junkheap.

5. If two robots run into each other, they die and create a junkheap.

6. If all the robots are destroyed, the game is over and the player wins.

On the board, the robots are represented using ‘R’ characters, junkheaps are represented with ‘#’, the player (the good guy) is represented by a ‘@’ and all other unoccupied squares are represented with ‘-’ characters.

The commands that the player can give in each turn are (look at a numeric keypad):

- ‘4’ – move one square left;
- ‘6’ – move one square right;
- ‘8’ – move one square up;
- ‘2’ – move one square down;
- ‘7’ – move one square up and left;
• ‘9’ – move one square up and right;
• ‘1’ – move one square down and left;
• ‘3’ – move one square down and right;
• ‘5’ – do nothing for one turn;
• ‘t’ – teleport to a random location;

There are a few things your program has to take care of. First, the player is asked for the number of robots and junkheaps initially on the board (make sure there are no more robots and junkheaps than the available locations on the board). Your program has to randomly arrange that many robots and junkheaps on the board, making sure no two of them end up at the same location or in the center of the board (which is the initial position of the player). Consider representing the board using a two dimensional array (see the encodings of the player, robots and junkheaps above). In each turn, the player is asked to specify the action he wants to perform, either move one square in some direction, or teleport (see the commands described above). After the player moves, your program has to move each robot one square towards the player. After one turn (when both the player and all the robots moved) the new state of the board is displayed on the screen. The player wins when there are no robots on the board and loses when he gets caught by a robot.

To start open (create) in your favorite editor (pico, emacs, vi etc.) a new file RobotGame.java. The first lines of your programs should always have the file name and the names and computer accounts of the program authors in comments.

// RobotGame.java
// Author: Firstname Lastname account
// Author: Firstname Lastname account
// Date:
// lab8 for the class cmps12a ..
// Short description what this program does.

Note that you may find it useful to write several classes for this assignment. Each class should be in its own file with similar initial comments.

Each function should also be preceded by a comment describing its interface and what it does.
The Submission: To submit this assignment run the submit command out of your working directory. First, use the move command mv to make sure that none of your backup files have the suffix ".java". Then submit all of your java files as follows (add in any other class files you write):

submit cmps012a-dh.w04 hw8 RobotGame.java Robot.java Position.java

Alternatively, you can use the unix wildcard "*" to submit all of your .java files with

submit cmps012a-dh.w04 hw8 *.java

Submit only your final Java classes for this program. Do not submit Java .class files, or “old” versions of the files. You can submit multiple times, only last version will be saved and graded. To see your files use:

peek cmps012a-dh.w04 hw8

Grading: This assignment is an excercise and will be graded on a pass/fail basis. If you would like more detailed feedback on your program, please consult with lab tutor for a face-to-face analysis.

Example Run: Feel free to modify the dialog so long as the essential usage and financial information is clearly presented.

```
p
> javac Robots.java
> java Robots

Please enter the number of robots on the board (1 - 224):
250
Invalid number of robots.
Please enter the number of robots on the board (1 - 224):
10
Please enter the number of junkheaps on the board (0 - 214):
215
invalid number of junkheaps.
Please enter the number of junkheaps on the board (0 - 214):
5
```
Enter your move: 2

Enter your move: t
Enter your move: 9

Enter your move: 3
Enter your move: 4

Enter your move: 8
Enter your move: 8

GAME OVER!!! You run into a robot!