# Section 1: Setting up the Entry Point

1. Create a new project called Minesweeper. You do not need to select any additional libraries, but you should tick the “Create project from template” box to make it a command line app.
2. If you successfully ticked the “Create project from template” button in the first step you should have a project with a Main.java already created for you. If you do not have this, you should create a new Java file called Main.java. Inside this file you will have a public class Main. You can place a new line between the { and } then type main and press tab. This will auto-generate a main method. The code should look similar to what you see below.

public class Main {

public static void main(String[] args) {

}

}

1. Now create a second Java file called Game.java. This is where the main game logic will be in future sections. The next few steps relate to adding code to this file.
2. Create a private instance variable called scan of type Scanner. This will be used to read in all typed input.
3. Create a default constructor for the Game class. This constructor should initialise the scan variable by assigning it a new Scanner object with System.in as the input source.
4. Create a method called startGame() that returns nothing (void) and takes no parameters.
5. Inside the startGame() method write code to complete the following steps.
   1. Declare two int type variables. You can call these num1 and num2.
   2. Print out the message: "Enter space separated X then Y coordinate: ".
   3. Set num1 to the value returned by scan.nextInt().
   4. Do the same for num2.
   5. Print out a message showing "You entered: num1 num2" where num1 and num2 are the values you read in.
6. Return to Main.java and add the following two lines to the main method.

Game game = new Game();

game.startGame();

1. Run the program from Main.java and test that your code works so far. Try entering 1 2 and pressing enter. You should see the following. (1 2 are bold to show they are input).

Enter space separated X then Y coordinate:

**1 2**

You entered: 1 2

# Section 2: Creating a Board with Cells

1. Create a new class called Cell.java. This will represent the state of a single cell on the board. In later sections the full state information will be set up, but for now cells will simply have a state that is either revealed or not revealed.
2. Declare a private instance variable of type boolean called isRevealed.
3. Create a default constructor that takes no parameters. Inside this method set the value of isRevealed to false.
4. Create a public method called reveal that takes no parameters and returns nothing. This method should set isRevealed to true.
5. Create a public method called getIsRevealed that takes no parameters and returns a boolean value. Use this method to return the value stored in isRevealed.
6. Create a public toString method that takes no parameters and returns a String. Inside this method you should write code to do the following. This method will be used to determine what character to show representing the state of the cell.
   1. Check if isRevealed is true, and if it is then return “0” as a String.
   2. Otherwise return “\*”.
7. Create a new class called Board.java. This will manage a 2D array of cells.
8. Declare a private instance variable of type Cell as a 2D array (by writing [][]) called cells. This will be the grid of all cells we create and manage.
9. Declare a private int called width, and a private int called height. These will be used to remember the dimensions stored inside the array.
10. Create a constructor for Board that takes two parameters. An int width, and an int height. Follow the steps below to initialise this class.
    1. Assign the value from the width and height parameters into their respective instance variables. You can do this by writing for example: this.width = width;
    2. Initialise the cells array to be a new Cell array with width and height by writing:

cells = new Cell[width][height];

* 1. Write a for loop that initialises a variable y to 0, loops while y is less than height, and increments by 1.
  2. Inside this loop write another loop that initialises a variable x to 0, loops while x is less than width, and increments by 1.
  3. Inside the inner loop set the value of cells[x][y] to be a new Cell object.

1. Create a public method called printBoard that takes no parameters and returns nothing. This will print out the contents of the board. The following instructions explain the content of this method.
   1. Copy the nested for loop with y and x that you wrote in steps 19 c to d and paste it into printBoard.
   2. Instead of creating new cells you will need to print out the current value of the cell. We declared a toString method inside the Cell so it can be printed out by referencing the cell. Write a print (instead of println) method call that prints out cells[x][y] followed by a space (in quotes).
   3. Just after the inner for loop ends, but before the outer loop does you should add a System.out.println(); call by itself. This will add the newlines after each row.
2. Create a public method called revealCell that takes two int type variables called x and y and returns nothing. Inside this method just write the line. This will reveal a cell we specify.

cells[x][y].reveal();

1. Open Game.java again and the remaining steps of this section will relate to this file.
2. Declare a private instance variable of type Board called board.
3. Add a statement to the default constructor to initialise the board variable. Initialise board to a new Board type object and pass 10 and 10 as the width and height.
4. In the startGame method complete the following steps.
   1. At the start of the method add a call to board.printBoard();
   2. At the end of the method add a line calling the revealCell method in board passing num1 as the first parameter and num2 as the second parameter.
   3. Then on the line after you should call board.printBoard(); again.
5. Compile and run your program and again test with an input of 1 2. You should see output similar to the following.

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

Enter space separated X then Y coordinate:

**1 2**

You entered: 1 2

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* 1 \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

# Section 3: Validating Input Part 1

1. Create a new Java class called Position.java. This class will be used to pass a coordinate with an x and y property around between different elements that need both pieces of information.
2. Inside the position class declare two public instance variables of type int called x and y.
3. Create a constructor for Position that takes two parameters of type int called x and y.
4. Inside the constructor store the value for x and y from the parameters into their respective instance variables. You can do this with this.x = x; for example.
5. Open Board.java.
6. Create a public method called validPosition that takes a single parameter of type Position called position and returns a boolean result.
   1. A valid position is one that matches with coordinates inside the bounds of the cells array. We already have a width and height variable, and arrays start at 0 so these can be used to check the validity.
   2. The method should return true if all the following conditions are true. If any are false, the result should be false.
      1. X coordinate is greater than or equal to 0.
      2. X coordinate is less than width.
      3. Y coordinate is greater than or equal to 0.
      4. Y coordinate is less than height.
7. Create a public method called isCellRevealed that takes one parameter of type Position called position and returns a boolean result. The method should simply return the result of cells[position.x][position.y].getIsRevealed().
8. Open Game.java. The rest of the work for this section will occur in this file.
9. Create a public method called isPositionInputValid that takes one parameter of type Position called position and returns a boolean result. The following steps should be completed inside this method.
   1. Write an if statement that checks if the position is not valid. You can use the ! character to get the not result and call the validPosition method in Board. The result of this if statement being true should be the text "Coordinate not inside the play space!" and then return false.
   2. Write another if statement after the previous one that checks using your board variable for the result of isCellRevealed. If this is true, you should print out the text "That cell is already revealed!" and then return false.
   3. Finally, if both the previous if statements were false, you should return true. This indicates the position is valid.
10. Create a public method called getPositionInput that returns an object of type Position. The following steps define what you should write for this method.
    1. Declare an object of type Position called input and initialise it with a new Position object set to 0,0. These numbers are just to initialise it and will be overwritten with user input.
    2. Write a do while loop as follows. This code will continue looping until a valid position has been entered.

do {

} while(!isPositionInputValid(input));

* 1. Inside the loop you should start by writing a statement to print out the text "Enter space separated X then Y coordinate: ".
  2. Set the value of input.x to scan.nextInt().
  3. Set the value of input.y to scan.nextInt().
  4. Finally, after the loop at the end of the method return the variable input.

1. As a last step before leaving the Board.java file. Go to your definition of revealCell and change the parameters from int x, int y to Position position. Then use position.x and position.y for the call to reveal.
2. Delete the contents of your startGame method as most of this is now handled inside the getPositionInput method.
3. Inside the startGame method add code to complete the following steps.
   1. Declare a variable of type Position called inputPosition.
   2. Write a do – while loop that has a loop condition set to true; written as while(true);
   3. Inside the loop you should call printBoard using your board object like you did previously.
   4. Then set the value of your inputPosition variable to the result of calling getPositionInput().
   5. Then call the revealCell method in your board object by passing inputPosition to it.
4. Compile and run your program. You should be able to continually enter pairs of x and y coordinates. If you enter numbers less than 0 or 10+ you should see your error message saying, “Coordinate not inside the play space!” and if you specify the same coordinate twice in a row it should output “That cell is already revealed!”. Test for both of these messages and that entering numbers between 0 and 9 for different positions on the grid make the correct cells reveal.

The following page shows some examples of output showing what you should expect to see with some example inputs.

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

Enter space separated X then Y coordinate:

**-1 5**

Coordinate not inside the play space!

Enter space separated X then Y coordinate:

**5 5**

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* 1 \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

Enter space separated X then Y coordinate:

**5 5**

That cell is already revealed!

Enter space separated X then Y coordinate:

**5 6**

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* 1 \* \* \* \*

\* \* \* \* \* 1 \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

Enter space separated X then Y coordinate:

# Section 4: Validating Input Part 2

1. Open Game.java. You may have tried to enter something that was not numbers in the previous testing and found it crashed your program. The changes in this section will make those crashes not possible, prepare for later steps where the user will be able to flag cells, and provide a way to quit at any time.
2. Create a public method called getStringOrQuit that takes no parameters and returns a String. The following steps should be completed inside this method.
   1. Declare a variable of type String called input and initialise it to scan.nextLine().trim(). The scan.nextLine() will retrieve the entire line of current input and trim() will remove any white space at the start/end of the line.
   2. Write an if statement that checks if input is the word “quit” you can do this by calling equalsIgnoreCase and supplying “quit” as the argument. This means it will be true regardless of how quit is capitalised.
   3. If the if statement is true, you should output a line saying “Thanks for playing! Goodbye!”. Then call System.exit(0) to terminate the application.
   4. After the if statement return input.
3. Modify the getPositionInput method with the following additional changes.
   1. After the line telling the user what to enter, but before the line setting input.x write an if statement that checks if scan.hasNextInt() is false. You can do this with a ! if you wish. If this if statement is true call getStringOrQuit(), output “Invalid X coordinate.”, and write continue; by itself on a line.

This will make sure that the line setting input.x will only be reached if the input has a valid int type value. And if it is not valid the input line is cleared with the call to getStringOrQuit() while at the same time checking if the input was just “quit”. Then the question is reasked to the user so they can enter a new coordinate because continue will return to the start of the loop.

* 1. Repeat the previous step between the lines that set input.x and input.y so that you can verify the input for the Y coordinate is a valid number. Of course, you will need to change the message to say “Invalid Y coordinate” for this, but it will otherwise be the same if statement block.
  2. After both input.x and input.y are set inside the loop add lines to subtract 1 from each of them. This means that users can input numbers from 1 to 10 instead of 0 to 9.
  3. In the next steps we will be making the numbers to input more obvious, so modify the previous message “Enter space separated X then Y coordinate: “ to say “Enter space separated X (bottom) then Y (right) coordinate: “.

1. Open Board.java.
2. Modify your printBoard method with the following changes.
   1. In the loop that prints out your board add an additional space into the loop to keep spacing working when numbering shows a 10 later on.
   2. Inside your outer loop you had a println() method call that had nothing in it. Modify this line so that it prints out “ |” followed by the value (y+1). Note that there is a space before the | character to give a space after the end of the row. This will show numbers down the right side so users can see the row number for the y coordinate.
   3. At the end of the method outside the loop write a new for loop. This for loop should loop for a value of x starting at 0, while x is less than width, and increment by 1. Inside the loop simply use print (not println) to print “\_ “ a \_ followed by two spaces. This will create a line of spaced-out dashes.
   4. After this for loop use a println call by itself to move to the next line.
   5. Write another for loop like the last one to start x at 0, loop while x is less than width, and increment by 1. Inside the loop you should print (x+1) followed by a space. Then an if statement that checks if x+1 is less than 10. If this is true, you should print out an additional space. This will keep the spacing the same for single and double digits.
   6. After the final loop write another println statement to move to the next line.
3. Compile and test your program. You should test entering letters for each of the X and Y coordinates one at a time. Test normal input still works. And finally test that quit works for one or more of the scenarios where it will. The output shown below is an example of what you should expect to see at this point.

\* \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

**a 1**

Invalid X coordinate.

1 \* \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

**1 a**

Invalid Y coordinate.

1 1 \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* \* \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

**7 7**

1 1 \* \* \* \* \* \* \* \* |1

\* \* \* \* \* \* \* \* \* \* |2

\* \* \* \* \* \* \* \* \* \* |3

\* \* \* \* \* \* \* \* \* \* |4

\* \* \* \* \* \* \* \* \* \* |5

\* \* \* \* \* \* \* \* \* \* |6

\* \* \* \* \* \* 1 \* \* \* |7

\* \* \* \* \* \* \* \* \* \* |8

\* \* \* \* \* \* \* \* \* \* |9

\* \* \* \* \* \* \* \* \* \* |10

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_

1 2 3 4 5 6 7 8 9 10

Enter space separated X (bottom) then Y (right) coordinate:

**quit**

Invalid X coordinate.

That cell is already revealed!

Enter space separated X (bottom) then Y (right) coordinate:

Thanks for playing!

Section 5