Title: ***Understanding While loops and their function***

Author #1: Carmen Petru

Author #2: Edward Madalina

Assignment Definition And General Feedback By Michael Panitz
at Cascadia Community College (http://www.cascadia.edu)

Table of contents:

- Quick Summary *
- When To Use (or Avoid) This *
- Example Of Usage *
- Example Trace *
- Syntax Explanation *
- Help With The Logic*
- Important Details

(Required sections have a * after them in the above list)

Quick Summary:

In Java, ***while*** loops allow the robot to evaluate whether a statement is true; if so it will execute the *true* statement until it is false, thus looping. ***While*** loops are either true or false. Another way to look at this is; it only answers to yes or no questions.

It is important to note that although both ***if*** and ***while*** loop statements are similar in syntax, they serve logically different functions. Remember that ***if*** statements decide whether or not to execute a command and then moves on to the next command; on the other hand, ***while*** loops determine how many times to execute a command based on whether it is true or not.

When To Use This / Avoid This:

There are several situations where you will want to use this technique. It is most helpful when trying to eliminate a repetitive service such as moving forward. For example, if you want to move the robot 4 times till there's a wall in its way, you could write out "*karel.move();* " 4 times, or you could replace it with a ***while*** loop which will tell the robot to move forward if the front is clear, or 'false'. This reduces the clutter and allows you to have clear and concise commands. Another situation where a ***while*** loop is helpful, is if a command or service is true, then execute something else or if a service is false then execute something else. That is, ***while*** something can be picked up, then "*turnLeft();* ". These situations can replace repetitive ***"if"*** statements when

you know there will be more than one instance occurring. Furthermore, not only could you use a ***while*** loop in *main* but you could make a *public void* service that calls out a command to be done at specific times and allows multiple robots to use it.

There are several situations where you will want to avoid this technique. There are times when you don't need the robot to repeat a command; this will lead to a logical error. That is, the ***while*** statement will continue to be true and loop, thus ceasing to move onto the next command. For example, a robot moves to an intersection where there is a *Thing*, the next commands calls a ***while*** loop, which tells the robot to *turnRight* ***while*** it can *pickThing*; the robot will continuously *turnRight* because a *Thing* continues to be underneath it. In this instance an ***if*** command would be more useful as it would be true; the robot would *turnRight* and move on to the next command.

Example Of Usage:

Let's say you have a problem; you want the robot to move from one wall until it reaches another wall, and then returns to its original position having collected any *Things* in its path. We can assume that the distance between the walls is random, as well as the number of things. (For the purposes here an example is given below.) You want to make a new command that would fit any given distance and *Things.* We can make a service that has a ***while*** loop; this command can be called out in main by the respective robot.

The following steps pertain to the code example given below. Notice that the ***while*** *loop statements are highlighted in* ***Grey*** *and are* ***bold*** *as to make it clearer to understand.*

There are 3 simple steps:

First, *(assuming you have already defined a new type of robot, lines 3-8)* create a public void service with the name representing the desired action (Line 16-22). (Note it is not necessary to create a new command in main for the *while* loop to actually work. See both examples below)

Second, within the service brackets **{}** type `while`, this should immediately turn `purple` as it is recognized by the Java language. Next parentheses **()**, it is better to type them out first and then just place your cursor inside; this will reduce the chances of having an error due to forgetting parentheses… Type the statement/command you wish the robot to answer true or false to (Line 18); don't forget parentheses after the statement.

Third, within another set of parentheses (Line 19 & 21), type the action the robot should execute as the statement continues to be true (Line 20).

Line	Code
1.	`import becker.robots.*;`
2.	

```
3.  class Robotcool extends Robot
4.  {
5.    public Robotcool(City theCity, int str, int ave,
      Direction dir, int numThings)
6.      {
7.           super(theCity, str, ave, dir, numThings);
8.      }
9.
10.    public void turnAround()
11.     {
12.        this.turnLeft();
13.        this.turnLeft();
14.     }
15.
16.    public void moveForward()
17.    {
18.        while ( this.frontIsClear() )
19.           {
20.              this.move();
21.           }
22.    }
23.
24.    public void harvest()
25.    {
26.        while ( this.frontIsClear() )
27.           {
28.              this.move();
29.                 if ( this.canPickThing() )
30.                       {
31.                          this.pickThing();
32.                       }
33.           }
34.    }
35.
36. }
37.
38. public class whileExample extends Object
```

```
39.  {
40.      public static void main(String[] args)
41.      {
42.          City Seattle = new City();
43.          Robotcool Eddie = new Robotcool(Seattle, 0, 4,
     Direction.SOUTH, 0);
44.
45.          new Wall(Seattle, 0, 4, Direction.NORTH);
46.          new Wall(Seattle, 4, 4, Direction.SOUTH);
47.
48.          new Thing (Seattle, 1, 4);
49.          new Thing (Seattle, 2, 4);
50.          new Thing (Seattle, 3, 4);
51.
52.          Eddie.moveForward();
53.
54.          while ( ! Eddie.frontIsClear() )
55.             {
56.             Eddie.turnAround();
57.             }
58.
59.          Eddie.harvest();
60.          Eddie.turnAround();
61.       }
62.  }
```

Example Trace:

In a nutshell, this technique allows you to repeat commands in both main; for a specific robot, or while creating a new service. If we look at what happens when the program runs, we notice that the first command *moveForward* (Line 52), calls a ***while*** loop which is define on line 16. So we jump up to line 16 and ask *is the front clear*? If so *move* (Line 20). This is repeated a total of 4 times, because on the 5th time asking if the front is clear? The answer is no/false and so we return back down to main and continue executing the rest of the commands (Line 54).

In order to go over these details more thoroughly, here is a (partial) trace of the above program, with some additional explanation afterwards

Line#	Program Statement	Eddie's Street #	Eddie's Ave #	Eddie's Direction	Eddie's Backpack Contents	True/ False	Wall 1	Wall 2	Thing	Thing	Thing
42	City Seattle = new City();	-	-	-	-	-	-	-	-	-	-
43	Robotcool Eddie = new Robotcool(Seattle, 0, 4, Direction.SOUTH, 0);	0	4	SOUTH	0	-	-	-	-	-	-
45	new Wall(Seattle, 0, 4, Direction.NORTH);	0	4	SOUTH	0	-	0,4,N	-	-	-	-
46	new Wall(Seattle, 4, 4, Direction.SOUTH);	0	4	SOUTH	0	-	0,4,N	4,4,S	-	-	-
48	new Thing (Seattle, 1, 4);	0	4	SOUTH	0	-	0,4,N	4,4,S	1,4	-	-
49	new Thing (Seattle, 2, 4);	0	4	SOUTH	0	-	0,4,N	4,4,S	1,4	2,4	-
50	new Thing (Seattle, 3, 4);	0	4	SOUTH	0	-	0,4,N	4,4,S	1,4	2,4	3,4
52	Eddie.moveForward();	0	4	SOUTH	0	-	0,4,N	4,4,S	1,4	2,4	3,4
16	public void moveForward()	0	4	SOUTH	0	-	0,4,N	4,4,S	1,4	2,4	3,4
18	while (this.frontIsClear())	0	4	SOUTH	0	T	0,4,N	4,4,S	1,4	2,4	3,4
20	this.move();	1	4	SOUTH	0	-	0,4,N	4,4,S	1,4	2,4	3,4
18	while (this.frontIsClear())	1	4	SOUTH	0	T	0,4,N	4,4,S	1,4	2,4	3,4
20	this.move();	2	4	SOUTH	0	-	0,4,N	4,4,S	1,4	2,4	3,4
18	while (this.frontIsClear())	2	4	SOUTH	0	T	0,4,N	4,4,S	1,4	2,4	3,4
20	this.move();	3	4	SOUTH	0	-	0,4,N	4,4,S	1,4	2,4	3,4
18	while (this.frontIsClear())	3	4	SOUTH	0	T	0,4,N	4,4,S	1,4	2,4	3,4
20	this.move();	4	4	SOUTH	0	-	0,4,N	4,4,S	1,4	2,4	3,4
18	while (this.frontIsClear())	4	4	SOUTH	0	F	0,4,N	4,4,S	1,4	2,4	3,4
54	while (! Eddie.frontIsClear())	4	4	SOUTH	0	T	0,4,N	4,4,S	1,4	2,4	3,4
56	Eddie.turnAround();	4	4	SOUTH	0	-	0,4,N	4,4,S	1,4	2,4	3,4

You'll notice that the trace starts after the **main** function, on line 42. It proceeds normally until line 52, which calls a new service. Java then jumps up to define that service (Line 16) which includes a while loop (Line 18), and a command to execute if the statement is true (Line 20).

Syntax Explanation:

Note on syntax of command: The ***while*** loop can execute any service that java already recognizes or a new one you create. It is important to evaluate the given situation and assess whether or not you need a command to repeat multiple times ***"while"*** or once, ***"if"***. If a while statement satisfies the problem then be cautious of the commands you wish to put within the brackets, and the commands that follow right after.

Let's start with the program as it's written here. Take note of the number of times the *move* command must be typed to accomplish the simple task of moving from one wall to another.

Line #	Program Source Code
1.	import becker.robots.*;
2.	
3.	class Robotcool extends Robot
4.	{
5.	public Robotcool(City theCity, int str, int ave, Direction dir, int numThings)
6.	{ {
7.	super(theCity, str, ave, dir, numThings);
8.	}
9.	
10.	public void turnAround()
11.	{ {
12.	this.turnLeft();
13.	this.turnLeft();
14.	}
15.	}
16.	
17.	public class PrewhileExample extends Object
18.	{
19.	public static void main(String[] args)
20.	{ {
21.	City Seattle = new City();
22.	Robotcool Eddie = new Robotcool(Seattle, 0, 4, Direction.SOUTH, 0);
23.	
24.	new Wall(Seattle, 0, 4, Direction.NORTH);
25.	new Wall(Seattle, 4, 4, Direction.SOUTH);
26.	
27.	new Thing (Seattle, 1, 4);
28.	new Thing (Seattle, 2, 4);
29.	new Thing (Seattle, 3, 4);
30.	
31.	Eddie.move();
32.	Eddie.move();
33.	Eddie.move();

```
34.          Eddie.move();
35.
36.          Eddie.turnAround();
37.
38.          Eddie.move();
39.          Eddie.pickThing();
40.          Eddie.move();
41.          Eddie.pickThing();
42.          Eddie.move();
43.          Eddie.pickThing();
44.          Eddie.move();
45.          Eddie.turnAround();
46.    }
47. }
```

Below, you can see the finished program with the differences highlighted in yellow, so it's easy to see what's been added/changed).

Line #	Program Source Code
1.	import becker.robots.*;
2.	
3.	class Robotcool extends Robot
4.	{
5.	public Robotcool(City theCity, int str, int ave, Direction dir, int numThings)
6.	{
7.	super(theCity, str, ave, dir, numThings);
8.	}
9.	
10.	public void turnAround()
11.	{
12.	this.turnLeft();
13.	this.turnLeft();
14.	}
15.	
16.	**public void moveForward()**
17.	**{**
18.	**while (this.frontIsClear())**
19.	**{**
20.	**this.move();**
21.	**}**
22.	**}**
23.	
24.	**public void harvest()**
25.	**{**

```
26.          while ( this.frontIsClear() )
27.             {
28.              this.move();
29.                 if ( this.canPickThing() )
30.                    {
31.                     this.pickThing();
32.                    }
33.             }
34.       }
35.
36.  }
37.
38.  public class whileExample extends Object
39.  {
40.   public static void main(String[] args)
41.      {
42.             City Seattle = new City();
43.              Robotcool Eddie = new Robotcool(Seattle, 0, 4, Direction.SOUTH, 0);
44.
45.             new Wall(Seattle, 0, 4, Direction.NORTH);
46.             new Wall(Seattle, 4, 4, Direction.SOUTH);
47.
48.             new Thing (Seattle, 1, 4);
49.             new Thing (Seattle, 2, 4);
50.             new Thing (Seattle, 3, 4);
51.
52.             Eddie.moveForward();
53.
54.             while ( ! this.frontIsClear() )
55.                {
56.                Eddie.turnAround();
57.                }
58.
59.             Eddie.harvest();
60.             Eddie.turnAround();
61.       }
62.  }
63.
```

The definition of the new command starts on line 16. Well what does it mean to "*moveForward"*? Move to line 18, which contains a ***while*** statement: if the *"frontIsClear",* Java understands this is true, then executes the following command: *"move" on line 20.* Another example is on line 54 when the ***while*** statement replaces a simple "*turnAround"* command.

In this instance, if we did not know where the wall would be, we could not determine when to turn around. However, if we use a ***while*** loop to allow the robot to turn around once it does reach

a wall we remove the unknown variable and the program runs without crashing.

How do we correctly write a while loop statement? We begin with the correct syntax, parenthesis, who will perform the command, what the command is and parenthesis :

```
while ( <robot name or <this>>.service to be true or false () )
```

Then Java needs to know where the command, that is true, is. We let Java know this by using an open bracket, **({)** to represent the start and a closed **(})** bracket, to represent the finish.

Lastly in between those brackets write the action the robot should execute. Take note that if the command is in main, use the name of the robot that will be doing the action, vs. if the command is called out in a `public void` scenario where, you would want to be consistent and use `this.`

Command call out

```
{
   this.<<action>>();
}
```

Main statement

```
{
   Eddie.<<action>>();
}
```

Help With The Logic:

This is good to use whenever you want to repeat a command for every instance its true, it is also good to use when you have unknown variables such as undefined lengths in between x command and y command, undefined number or objects or "Things" and when you have a set of commands you would like to repeat.

Important Details:

- Always evaluate whether you need an *"if"* statement, which is executed only once if true, or a *"**while**"* loop, which is repeatedly executed until it is false.

- Remember that if the ***while*** loop is true, it will execute all commands within the brackets in order. It is important to note that the program can still crash while executing the

commands even if the ***while*** loop is true. Any command wished to be completed after the statement is no longer true, should be put outside the closing bracket.

Licensing

Plagiarism

If you believe that some or all of this document infringes on your intellectual property (i.e., part or all of this document is copied from something you've written) please immediately contact Mike Panitz at Cascadia Community College (perhaps using the Faculty And Staff Directory at http://www.cascadia.edu/pages/searchtemplate.aspx)