Title: Having one new command call another new command

Author #1: Henry M Le

Author #2: Gino Pineda

Author #2: Maxat Kenzhebayev

Assignment Definition and General Feedback by Michael Panitz at CascadiaCommunity College (http://www.cascadia.edu)

Table of contents:

Quick Summary:

In Java, you can have one new command call another new command by extending a class.

When to Use This/Avoid This:

At times you will find it tedious to write a command repetitively to accomplish a task. By extending a class with new services, it will allow programmers to simplify the code and use time more efficiently.

This technique should only be used as stated above. Avoid using this technique if the intent is not to simplify the code.

Example of Usage:

Let's say you want the robot bob to walk up a flight of stairs and use a new command that will run the entire program. Within the method that defines the new command, another new command is created which is called upon for simplicity.

There are three steps:

First, create the whole city with a robot and the stairs.

Second, create a new command that will run the entire program.

Third, create an extending class that will house the methods used to define the new commands.

Line	Code
1.	`import becker.robots.*;`
2.	
3.	`class RobotSmart extends Robot`
4.	`{`
5.	`   RobotSmart (City c, int st, int ave, Direction dir, int num)`
6.	`   {`
7.	`      super(c, st, ave, dir, num);`
8.	`   }`
9.	
10.	`      public void walkUpStairs()`
11.	`      {`
12.	`         this.turnLeft();`
13.	`         this.move();`
14.	`         this.turnRight();`
15.	`         this.move();`
16.	`         this.turnLeft();`
17.	`         this.move();`

```
18.             this.turnRight();
19.             this.move();
20.             this.turnLeft();
21.             this.move();
22.             this.turnRight();
23.             this.move();
24.         }
25.
26.         public void turnRight()
27.         {
28.             this.turnLeft();
29.             this.turnLeft();
30.             this.turnLeft();
31.         }
32. }
33.
34. public class Example_1 extends Object
35. {
36.     public static void main(String[] args)
37.     {
38.         City seattle = new City();
39.         RobotSmart bob = new RobotSmart (seattle, 4, 0,
```

```
        Direction.EAST, 0);
40.
41.             new Wall (seattle, 4, 1, Direction.WEST);
42.             new Wall (seattle, 4, 1, Direction.NORTH);
43.             new Wall (seattle, 3, 2, Direction.WEST);
44.             new Wall (seattle, 3, 2, Direction.NORTH);
45.             new Wall (seattle, 2, 3, Direction.WEST);
46.             new Wall (seattle, 2, 3, Direction.NORTH);
47.
48.             bob.walkUpStairs();
49.         }
50.     }
```

Example Trace:

In a nutshell, this technique in adding a new service allows you to reduce the number of "turnLeft" command.

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	Bob's St #	Bob's Ave #	Bob's direction	Bob's Backpack
38.	City seattle = new City();	-	-	-	-
39	RobotSmart bob = new RobotSmart (seattle, 4, 0, Direction.EAST, 0);	4	0	East	0
48	bob.walkUpStairs();	4	0	East	0
10	public void walkUpStairs()	4	0	East	0
12	this.turnLeft();	4	0	North	0
13	this.move();	3	0	North	0

14	this.turnRight();	3	0	North	0
26	public void turnRight()	3	0	North	0
28	this.turnLeft();	3	0	West	0
29	this.turnLeft();	3	0	South	0
30	this.turnLeft();	3	0	East	0
15	this.move();	3	1	East	0
16	this.turnLeft();	3	1	North	0
17	this.move();	2	1	North	0
18	this.turnRight();	2	1	North	0
26	public void turnRight()	2	1	North	0
28	this.turnLeft();	2	1	West	0
29	this.turnLeft();	2	1	South	0
30	this.turnLeft();	2	1	East	0
19	this.move();	2	2	East	0
20	this.turnLeft();	2	2	North	0
21	this.move();	1	2	North	0
22	this.turnRight();	1	2	North	0
26	public void turnRight()	1	2	North	0
28	this.turnLeft();	1	2	West	0
29	this.turnLeft();	1	2	South	0
30	this.turnLeft();	1	2	East	0
23	this.move();	1	3	East	0

You'll notice that the trace starts at the beginning of the **main** function, on line 38, like normal. Creating the stairs was omitted from the trace for brevity. It proceeds normally until line 48 where the new command 'walkUpStairs' is called and goes back to line 10 where the method is created.

Syntax Explanation:

Note on syntax of command: The name of the new command can be anything you want with few restrictions. The new command cannot have a space between the words and must be all together. First word should be in lowercase and then the following word(s) in uppercase.

Let's start with the program as it's written here without creating the new commands.

```
Line   Program Source Code
#
```

```
1.  import becker.robots.*;
2.
3.  public class Example_1 extends Object
4.  {
5.     public static void main(String[] args)
6.     {
7.           City seattle = new City();
8.           RobotSmart bob = new RobotSmart (seattle, 4,
    0, Direction.EAST, 0);
9.
10.          new Wall (seattle, 4, 1, Direction.WEST);
11.          new Wall (seattle, 4, 1, Direction.NORTH);
12.          new Wall (seattle, 3, 2, Direction.WEST);
13.          new Wall (seattle, 3, 2, Direction.NORTH);
14.          new Wall (seattle, 2, 3, Direction.WEST);
15.          new Wall (seattle, 2, 3, Direction.NORTH);
16.
17.          bob.turnLeft();
18.          bob.move();
19.          bob.turnLeft();
20.          bob.turnLeft();
21.          bob.turnLeft();
22.          bob.move();
23.          bob.turnLeft();
24.          bob.move();
25.          bob.turnLeft();
26.          bob.turnLeft();
```

```
27.          bob.turnLeft();
28.          bob.move();
29.          bob.turnLeft();
30.          bob.move();
31.          bob.turnLeft();
32.          bob.turnLeft();
33.          bob.turnLeft();
34.          bob.move();
35.      }
36.  }
```

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 RobotSmart extends Robot`
4.	`{`
5.	`   RobotSmart (City c, int st, int ave, Direction dir, int num)`
6.	`   {`
7.	`      super(c, st, ave, dir, num);`
8.	`   }`
9.	
10.	`      public void walkUpStairs()`
11.	`      {`
12.	`         this.turnLeft();`
13.	`         this.move();`
14.	`         this.turnRight();`
15.	`         this.move();`
16.	`         this.turnLeft();`

```
17.             this.move();
18.             this.turnRight();
19.             this.move();
20.             this.turnLeft();
21.             this.move();
22.             this.turnRight();
23.             this.move();
24.         }
25.
26.         public void turnRight()
27.         {
28.             this.turnLeft();
29.             this.turnLeft();
30.             this.turnLeft();
31.         }
32. }
33.
34. public class Example_1 extends Object
35. {
36.    public static void main(String[] args)
37.    {
38.          City seattle = new City();
39.          RobotSmart bob = new RobotSmart (seattle, 4,
     0,   Direction.EAST, 0);
40.
41.          new Wall (seattle, 4, 1, Direction.WEST);
42.          new Wall (seattle, 4, 1, Direction.NORTH);
43.          new Wall (seattle, 3, 2, Direction.WEST);
44.          new Wall (seattle, 3, 2, Direction.NORTH);
45.          new Wall (seattle, 2, 3, Direction.WEST);
46.          new Wall (seattle, 2, 3, Direction.NORTH);
47.
48.          bob.walkUpStairs();
49.    }
50. }
```

Help With The Logic:

This is good to use whenever a command is repeated. If you run into a situation where you have to turn right more than once and in order to turn right you have to turn left three times, creating a new command would help simplify that task.

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 CascadiaCommunity College (perhaps using the Faculty And Staff Directory at http://www.cascadia.edu/pages/searchtemplate.aspx)