Title: Multi-file Programs

Author: Charlie Schuy

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
- Important Details

Quick Summary:

In Java, you can create programs using multiple files. Creating multiple files helps organize your program, and as a result, it becomes easier to modify.

When To Use This / Avoid This:

There are several situations where you will want to use this technique. The most important instance is when creating a program that uses many, many different classes. By creating different files for each class, you will be able to navigate your source code much easier and more efficiently than if you had put all of the code in one source. Another situation where you will want to use this technique is for reusing code. If you create a new type of robot in a different file, then when you want to use it in another program all you need to do is copy the file rather than copy a portion of a file.

The only times you would want to avoid creating multiple files is if the length of the program is going to be very short and if the program is going to very specialized that the code can't be reused elsewhere.

Example Of Usage:

Let's say you want a newly defined robot to move to a wall and then return to its original position, but instead of putting the class of the robot in the same file as the main function, you want to put it in a different file.

There are four steps:
First, create the main file that will be run with the setup for the city. For this example, the city

needs to have a robot facing towards a wall any number of spaces away from the robot. The robot should be the new type that you will be defining in a different file.

Second, create a new file for the new type of robot. The name of the file must be the name of the type of robot. For example, if you named the type of robot "SuperBot", then the file must be named "SuperBot.java". Make sure that you import the becker.robots file in this file as well. The class must be defined as public so that it can be accessed from other files.

Third, create the robot as you would normally do, but in a new separate file. When creating functions, make sure that every function that you want to be called outside of the file is set to public. If you wanted a function called "turnRight" that you wanted to be used outside of the file, you would define it like "public void turnRight()". In this example, we want the robot to move until it hits a wall, then turn around and return to its original position. We need to make a public function to do that.

Fourth, call the function that you defined in the previous step in the main function of the main file. The function can be called exactly like any function that you made that was defined in the same file. Afterwards, you want to compile all of the files that you created and run the main file.

Main file: multi_file_demo.java

Line	Code
1.	`import becker.robots.*;`
2.	
3.	`public class multi_file_demo extends Object`
4.	`{`
5.	`   public static void main(String[] args)`
6.	`   {`
7.	`      City woodinville = new City(8, 8);`
8.	`      OutsideRobot rob = new OutsideRobot(woodinville, 2, 2, Direction.EAST, 0);`
9.	
10.	`      new Wall(woodinville, 2, 6, Direction.EAST);`
11.	
12.	`      rob.pace();`
13.	`      rob.move();`
14.	`   }`
15.	`}`

Robot's file: OutsideRobot.java

Line	Code

```
1.  import becker.robots.*;
2.
3.  public class OutsideRobot extends Robot
4.  {
5.     public OutsideRobot(City ct, int st, int ave, Direction d,
       int n)
6.     {
7.        super(ct, st, ave, d, n);
8.     }
9.
10.    private void turnAround()
11.    {
12.       this.turnLeft();
13.       this.turnLeft();
14.    }
15.
16.    public void pace()
17.    {
18.       int spacesMoved = 0;
19.
20.       while (this.frontIsClear())
21.       {
22.          this.move();
23.          spacesMoved++;
24.       }
25.
26.       this.turnAround();
27.
28.       for (int i = 0; i < spacesMoved; i++)
29.       {
30.          this.move();
31.       }
32.    }
33. }
```

Example Trace:

In a nutshell, this technique allows you to split the program into different parts. The trace works much like a program that was all in one file would, but instead of all being contained in the single file, the trace jumps between the different files as needed.

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	rob Street	rob Avenue	rob Direction	Bob Backpack	Wall	True / False
12	rob.pace();	2	2	East	0	(2, 6, E)	-
-	<jumps to OutsideRobot.java>	2	2	East	0	(2, 6, E)	-
16	public void pace()	2	2	East	0	(2, 6, E)	-
18	int spacesMoved = 0;	2	2	East	0	(2, 6, E)	-
20	while (this.frontIsClear())	2	2	East	0	(2, 6, E)	True
22	this.move();	2	3	East	0	(2, 6, E)	-
23	spacesMoved++;	2	3	East	0	(2, 6, E)	-
24+	…	..	..	..	0	(2, 6, E)	-
-	< returns to main file >	2	2	West	0	(2, 6, E)	-
13	rob.move();	2	1	West	0	(2, 6, E)	-

You'll notice that the trace starts in the **main** function as normal, but when rob's pace command gets called, the trace jumps to the OutsideRobot.java file, finds the pace function and executes the function. After the function is done executing, the program jumps back to the **main** function and continues onward. In this case it executes a single move command.

Syntax Explanation:

Note on syntax of command: the main thing to note about creating a new type of robot in a different file is that everything that you want to access from outside of the file must be set to public.

Let's start with the program as it's written here.

Line #	Program Source Code
1.	`import becker.robots.*;`
2.	
3.	`class OutsideRobot extends Robot`
4.	`{`
5.	`   public OutsideRobot(City ct, int st, int ave,` `   Direction d, int n)`
6.	`   {`

```
7.          super(ct, st, ave, d, n);
8.       }
9.
10.      private void turnAround()
11.      {
12.         this.turnLeft();
13.         this.turnLeft();
14.      }
15.
16.      public void pace()
17.      {
18.         int spacesMoved = 0;
19.
20.         while (this.frontIsClear())
21.         {
22.         this.move();
23.         spacesMoved++;
24.         }
25.
26.         this.turnAround();
27.
28.         for (int i = 0; i < spacesMoved; i++)
29.         {
30.            this.move();
31.         }
32.      }
33.   }
34.
35.   public class combined_file extends Object
36.   {
37.      public static void main(String[] args)
38.      {
39.         City woodinville = new City(8, 8);
40.         OutsideRobot rob = new OutsideRobot(woodinville,
           2, 2, Direction.EAST, 0);
41.
42.         new Wall(woodinville, 2, 6, Direction.EAST);
43.
44.         rob.pace();
45.         rob.move();
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).

multi_file_demo.java

Line #	Program Source Code
1.	`import becker.robots.*;`
2.	
3.	`public class multi_file_demo extends Object`
4.	`{`
5.	`   public static void main(String[] args)`
6.	`   {`
7.	`      City woodinville = new City(8, 8);`
8.	`      OutsideRobot rob = new OutsideRobot(woodinville, 2, 2, Direction.EAST, 0);`
9.	
10.	`      new Wall(woodinville, 2, 6, Direction.EAST);`
11.	
12.	`      rob.pace();`
13.	`      rob.move();`
14.	`   }`
15.	`}`

OutsideRobot.java

Line #	Program Source Code
1.	`import becker.robots.*;`
2.	
3.	`public class OutsideRobot extends Robot`
4.	`{`
5.	`   public OutsideRobot(City ct, int st, int ave, Direction d, int n)`
6.	`   {`
7.	`      super(ct, st, ave, d, n);`
8.	`   }`
9.	
10.	`   private void turnAround()`
11.	`   {`
12.	`      this.turnLeft();`
13.	`      this.turnLeft();`
14.	`   }`
15.	
16.	`   public void pace()`
17.	`   {`
18.	`      int spacesMoved = 0;`
19.	

```
20.         while (this.frontIsClear())
21.         {
22.            this.move();
23.            spacesMoved++;
24.         }
25.
26.         this.turnAround();
27.
28.         for (int i = 0; i < spacesMoved; i++)
29.         {
30.            this.move();
31.         }
32.      }
33.   }
```

The original file has been split up into two different files. One contains the main function and the other contains the new type of robot. It is important to know that since these are two completely separate files, you must import becker.robots.* in both of the files because they are treated as code that could be completely unrelated to each other.

On line 3 of the OutsideRobot.java file, the line defining the robot has the word public added to it. Making OutsideRobot public allows it to be used outside of this file. Also notice that "pace()" is public as well, and it too can be accessed from outside of the file because of the public keyword. If a function is private, then it can only be used inside the file. Other than these two important details, everything else works exactly the same as if you had combined the two files.

Important Details:

- Remember to import what you need for each specific file. What you import isn't necessarily the same for both files. For example, if you want to get input from the keyboard in the main function, you will need to import the appropriate library in the main function, but you do not need to import the same library in the files that are not receiving any input from the keyboard.

- Be conscious of whether a function should be public or private. If a function is meant to be accessed outside of the file, set it to public. If a function is meant to only be used in the files, set it to private.

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)