Quickdraw Bug Report

Yang Li and Michael E. Locasto

Dec 1, 2011

Program Name: quickdraw.jar
Version: 1.314
Software Source: http://pages.cpsc.ucalgary.ca/QuickDraw/

Problem Description:

Students can use quickdraw.jar program to generate some geometry shapes by typing the command that is defined in quickdraw.jar . They also can write a script or a program in C++/C, Java, Python, etc, to output some quickdraw commands then pipes to quickdraw.jar for the desired graph. For example,

java -jar quickdraw.jar < script.txt

where the script.txt is a file written by student and contains commands for quickdraw.jar

There is a command in quickdraw called “begingroup” which allows student to define a group of shapes. Then, the defined group can be drew later by calling “drawgroup”. The problem we found is if we use “begingroup” to define a group of shapes in terminal, and “drawgroup” is called in the terminal, then the group of desired shapes can be drew properly. However, if we write exactly same codes into a script, then use ioredirection technique (shown above) to run the quickdraw, the group we defined in the script can not be drew out. The interesting thing is the codes in the script where before we start to define the group can be drew properly. Also, if we put core codes in the above group into a script, then run the program, and these core codes can be drew properly as well. For same input, but different manners to input to the program, the program is stuck by one of them. Thus, we think the problem is caused by the BufferedReader class which is used to read text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.

Analysis:

By using the Java Debugger (jdb), we found the program stops at BeginGroup.java line 35. At there, program just reads an input through InputStreamReader which is a bridge from byte streams to character streams and reads bytes and decodes them into characters using a specified charset. Then the input is stored in the memory for BufferedReader. The input is read in BeginGroup.java comes from user or other sources and then is passed to another function to analyze. However, the input is null at this point. In addition, we found in the constructor of BeginGroup.java a piece of memory is allocated to the BufferedReader. However, before the BeginGroup is involved, another piece of memory is allocated earlier in AppController.java to the BufferedReader to handle the input. Thus, the problem is clear now. The input file, script.txt, is read as a string by the program, and it is passed to the BufferedReader memory where the first time it is called. Later, when the BufferedReader in BeginGroup.java try to get input from the script.txt, actually there is nothing left in the file. The input has all been stored into the memory in AppController.java even some are not used. That is, the shell makes all the input of the file available, but the first BufferedReader in AppController is actually greedy enough to consume all this available input even before the BufferedReader in BeginGroup gets a chance to read from stdin (System.in).

To make the problem clear, we write another program to simulate the situation.
Codes:

```
class prob
{
   public static void main(String args[])
   {
     BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
     while(true){
        try{
          System.out.println("\nIn main, reading a line: ");
          String line = in.readLine();
          System.out.println(line);
          AnotherReadLine();
        }
        catch(Exception e){}
     }
   }
   public static void AnotherReadLine()
   {
     BufferedReader in_new = new BufferedReader(new InputStreamReader(System.in));
     try{
        System.out.println("In another function, reading a line: ");
        String line_new = in_new.readLine();
        System.out.println(line_new);
     }
     catch(Exception e){}
   }
}
```

In script.txt:
1
2
3
4

Run the program with ioredirection: java prob < script.txt
Output:

In main, reading a line:
1
In another function, reading a line:
null

In main, reading a line:
2
In another function, reading a line:
null

In main, reading a line:

3
In another function, reading a line:
null

In main, reading a line:
4
In another function, reading a line:
null

Run the program without ioredirection (input by hand): java prob
Output:

In main, reading a line:
1 <--user types input
1
In another function, reading a line:
2 <--user types input
2

In main, reading a line:
3 <--user types input
3
In another function, reading a line:
4 <--user types input
4

Now we see the script.txt is stored into the memory of BufferedReader in the main() function. It can be read later only in main(), but another function can not touch the same memory. And this explains why "begingroup" works by typing but not by ioredirection.

In addition, the program itself erroneously ignored any Exception (such as the NullPointerException originating in StringParser) generated by the code in BeginGroup -- which makes a problem like this one a bit harder to diagnose.

Solution:

Michael edits the program and gives a solution below.

* AppController passes its reference to "in" to Parser
* Needed to modify Parser's constructor to take this field
* When Parser dynamically loads classes add a test if class is an instance of "BeginGroup"
* if so, pass the reference to "in" that Parser received from AppController
* add a test of line==null so that we don't keep having a nullpointerexception (when there is no input -which may happen for long stretches of time)

For more details, please email locasto@ucalgary.ca