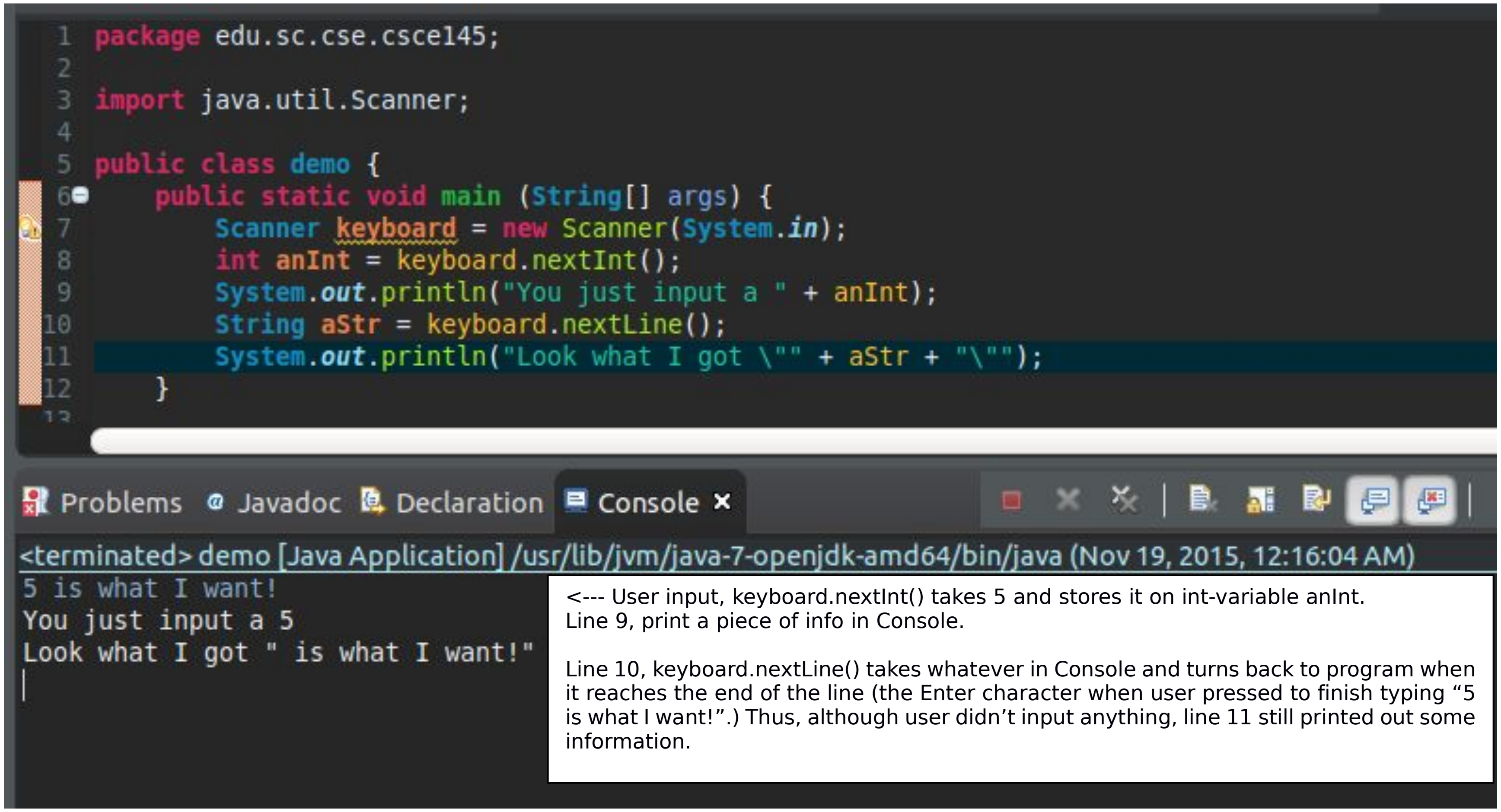
```
import java.io.*;
import java.util.*;

public class DangerZone { <<< Declare a class.
* one public class in one java file.
* one .java file can contain multiple non-public classes
* name of public class MUST be exactly same to the name of .java file

	>>> Declare variables. final means this is a constant and its value cannot be changed in running time.
	static final String INPUT_FILE = "dangerzone.txt";
	static final String OUTPUT_FILE = "NEWdangerzone.txt";
	static final String DELIM = " ";


	//Front end
	public static void main(String[] args)  <<< This is the ENTRY POINT of project lab20
	{
		//Prompts user
		System.out.println("Welcome to Top Gun!"); >>> print a piece of information (in this case, “Welcome to Top Gun!”) in Console
		System.out.println("Enter a word to replace \"danger\" with:"); >>> print a piece of information in Console

		Scanner keyboard = new Scanner(System.in); >>> create a Scanner object, named as keyboard, which will be used to scan
inputs in Console.

		//User input
		String input = keyboard.nextLine(); >>> Scanner keyboard waits for user typing, when user presses Enter (end of a line), it
receives everything (input) in Console (including the Enter, i.e., “\n”) then stores whatever it got on input as a string.
* int anInteger = keyboard.nextInt(); nextInt() will scan the line in Console, receive the first staff (assumed to be an integer) it meets, parses it to
an integer and then stores the integer on anInteger, ignoring everything that follows.
   ↑
This is the reason why we get a “InputMismatchException” thrown in Console when we type “hello world” in Console, because when nextInt() tries to
parse “hello” into an integer and finds out that “hello” is not convertible.
```



```java
 1 package edu.sc.cse.csce145;
 2 
 3 import java.util.Scanner;
 4 
 5 public class demo {
 6     public static void main (String[] args) {
 7         Scanner keyboard = new Scanner(System.in);
 8         int anInt = keyboard.nextInt();
 9         System.out.println("You just input a " + anInt);
10         String aStr = keyboard.nextLine();
11         System.out.println("Look what I got \"" + aStr + "\"");
12     }
13 
```

```
<terminated> demo [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Nov 19, 2015, 12:16:04 AM)
5 is what I want!
You just input a 5
Look what I got " is what I want!"
```

<--- User input, keyboard.nextInt() takes 5 and stores it on int-variable anInt.
Line 9, print a piece of info in Console.

Line 10, keyboard.nextLine() takes whatever in Console and turns back to program when it reaches the end of the line (the Enter character when user pressed to finish typing "5 is what I want!".) Thus, although user didn't input anything, line 11 still printed out some information.

```
//sends to readDangerZone
readDangerZone(INPUT_FILE, input);
```
>>> invoke a function to do some task, passing whatever information needed by the function. *INPUT_FILE* and *input* contain the information / values that we want to tell *readDangerZone*().

** The pointer will jump to readDangerZone() from here, the following statements in main() won't be executed until readDangerZone finishes its task, then the pointer will jump back and continue following statements.

//Confirms with user
System.out.println("Danger has been replaced with " + input
+ "! The new lyrics are in NEWdangerzone.txt"); >>> print a piece of information in Console
** Since there is no more statement, this is the last statement be performed. The program will terminate here.
} // Exit of program

public static **void** readDangerZone(String fileName, String input) <<< Declare a function
* This function takes in 2 pieces of information, and both of them are strings. -- String fileName, String input
* This function won't give any information back to whom invokes it. -- void
* This function is a Class / static function. -- static
* This function is public, it is accessible by any object from outside (from another .java file) -- public
**** Before you define a function, make sure you know the answers of following questions**
**** what is it for? what is its task? -- this could help you giving your function a more meaningful name
**** it is for others to use, or it only assists other functions (inside same class) to finish certain task? --- public or private
**** any information needs to return / tell whomever invoke this function? If yes, what kinds of information? -- void, or certain type, say String, int, etc.
**** do I need any information to do my task? If yes, how many information? what kinds of information? -- parameter, number of parameters, types of parameters.
### better strategy to design a function: simple and pure. -- one task per function.
### parameters and return value (if it is not **void**) are used to communicate / exchange information between functions.
{

String newSong = ""; >>> Define an empty string variable *newSong*, which will be used to store the new song generated later.

try >>> when system meets "try", it scans following statements, finding the matching "catch"es (must have at least one "catch"), checks what kinds of Exception(s) you want to catch during execution. Then it monitors (we call this listen in CS) all statements in try block for certain type(s) of exceptions. Whenever any of specific exception(s) thrown, it jumps to corresponding catch block and executes statements in that block. If some exceptions, which won't be caught by catch block, this exception will be handed over to system to handle (it usually prints the execution stack and terminates the program). If no exception thrown, then catch block(s) will be skipped.
{
//Scans file lines
Scanner fileScanner = new Scanner(new File(INPUT_FILE)); >>> create a Scanner object, named as *fileScanner*, which will be used to scan the file that *INPUT_FILE* refers to.

```
// Loop through each line
                Condition
                ↓     ↓     ↓     ↓
while (fileScanner.hasNextLine()) >>> WHILE_LOOP, repeats all statements in this block as long as the condition is TRUE
{
  //scans each line
  String line = fileScanner.nextLine().toLowerCase(); >>> read one line and store the entire line on variable line.

  //splits a string into an array, one word per element, words are separated by the space
  String[] splitLines = line.split(DELIM); >>> store all words in line in an array, one word per element.

  for (int i=0; i<splitLines.length; i++) >>> goes through current line, word by word
  {
        //if the word contains the word danger
        if (splitLines[i].contains("danger")) >>> check whether current word is the word we are looking for
        {>>> YES! We just find a word matches our will.
              //Replaces that particular word with the user input
              splitLines[i] = input; >>> update / change / cover / replace the word (contains "danger") with user's input
        }
        newSong += splitLines[i]+" "; >>> add the ith word to our new song.
```

* if it did not contain “danger”, the original word will be concatenated.
* if it contained “danger”, it has been updated in above IF block, and it is now safe to be added to new song.

Ex. (say, input = “**Right**”)

Original line:	Highway	to	the	Danger	Zone
	↓	↓	↓	↓	↓
Array splitLines:	splieLines[0]	splieLines[1]	splieLines[2]	splieLines[3]	splieLines[4]
Contains?:	NO	NO	NO	**YES**	NO
What to do?	Copy directly	cp directly	cp directly	repl with “Right” then copy	copy directly
	↓	↓	↓	↓	↓
newSong:	Highway	to	the	Right	Zone

```
                  } >>> End of for, Done checking all words in one line
                  newSong += "\n"; >>> Add Enter character at the end of current line
                } >>> End of While, Done scanning the whole file

                //Closes file
                fileScanner.close(); >>> close the file, do not occupy / lock a resource after you have done your work with it.

                //Send to printDangerZone
                printDangerZone(OUTPUT_FILE, newSong); >>> invoke function pringDangerZong,
** The pointer will jump to printDangerZone() from here, the following statements in this function won't be executed until printDangerZone
finishes its task, then the pointer will jump back and continue following statements (although in this case, there is no more statements, the
pointer will jump back here anyway, and jumps back to main() when it finds that this is the End of readDangerZone()).
            }
            catch(Exception e) {>>> catch exception.
                  System.out.println(e.getMessage());
            }
      }>>> end of function readDangerZone, then point jump back to main() and continue from wherever it went out.

      public static void printDangerZone(String fileName, String newSong) >>> Declare a function
* in this function, we want to print newSong into file fileName.
      {
            //Prints to file
            Try >>> a try block
            {
                  PrintWriter fileWriter = new PrintWriter(new FileOutputStream(fileName)); >>> create a PrintWriter object, and
open the file that fileName refers to.
                  fileWriter.println(newSong); >>> output whatever newSong is to file.
                  fileWriter.close(); >>> save changes to the file and close it.
            }
            catch (IOException e)
            {
                  System.out.println("Exception"+e.getMessage());
            }
            catch(Exception e)
            {
                  System.out.println("Exception"+e.getMessage());
                }
        } >>> End of printDangerZone() function, the pointer will go back wherever it came from.
}
```