# The validInput function

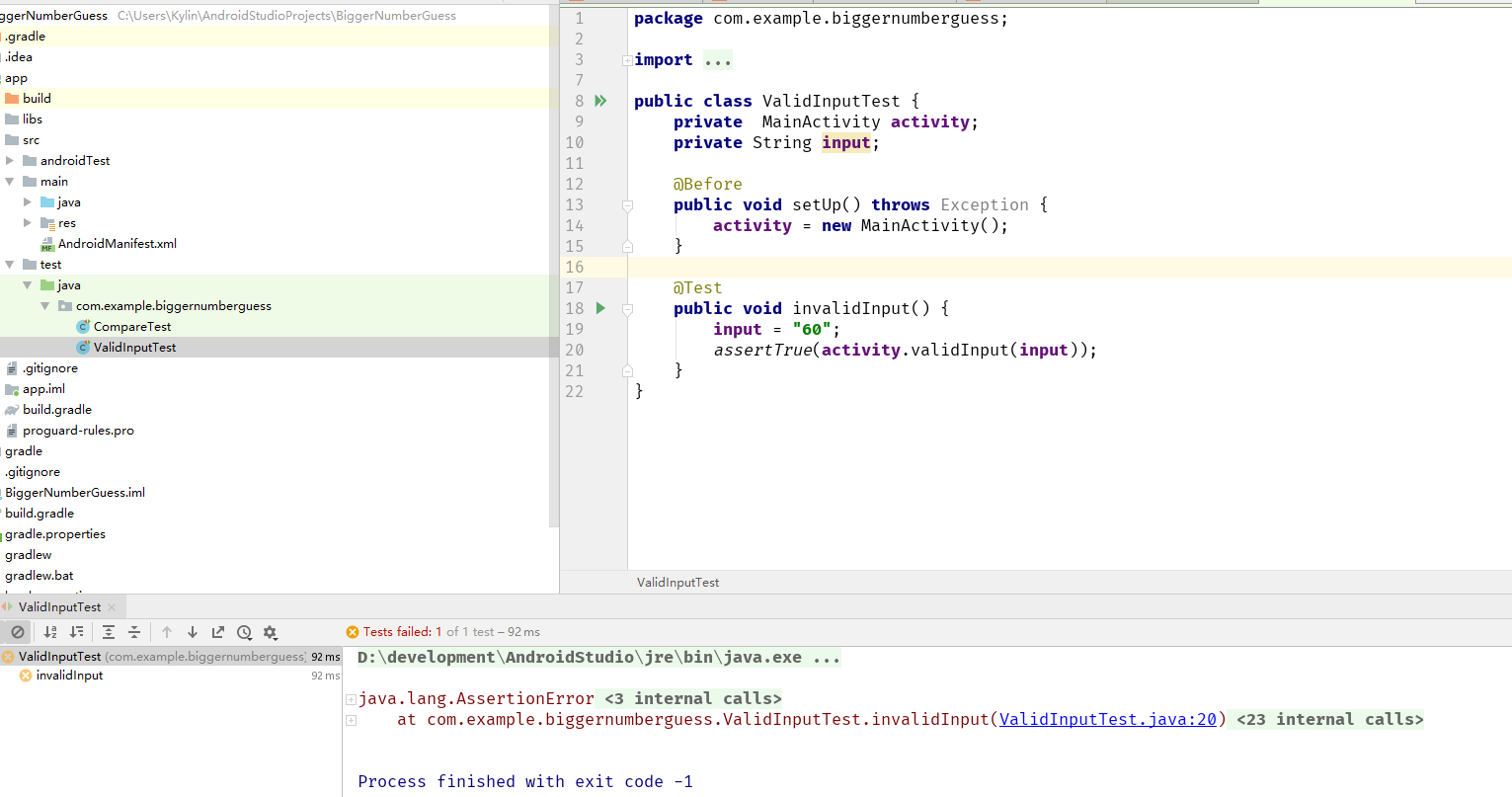
## Description: this function is designed to restrict the input from users. This program should only accept the numbers between 1-100 and “q” or “Q”.

1. procedure

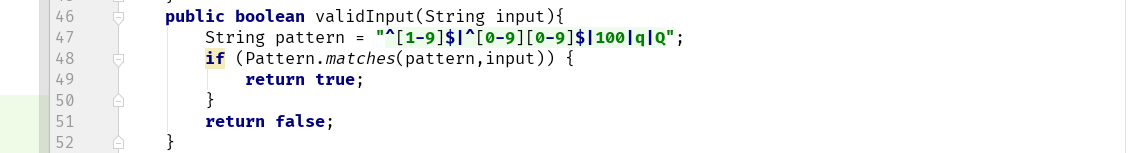
Steps1: Write a framework for that function.



Step2: Write the test class for that function and test it with arbitrary input. The test should be failed because there is no concrete content inside the validInput function.



Step3: Fulfill the validInput function



## Test case

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Purpose | Test Data | Expected Result | Actual  Result | Status |
| 1 | Test with valid number | 60 | true |  | Pass |
| 2 | Test with space | “ ” | false |  | Pass |
| 3 | Test with latter | “ss” | false |  | Pass |
| 4 | Test with “q” | “.” | true |  | Pass |
| 5 | Test with invalid number | 101 | false |  | Pass |
| 6 | Test with 0 | 0 | false |  | Pass |

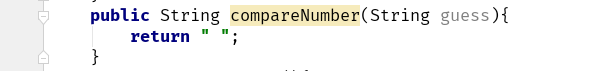
## Conclusion

As test case above, this function can correctly assess the invalid input.

# The compareNumber function

1. Description: this function compares the user’s input with the random number and react different ways with different situations.
2. procedure

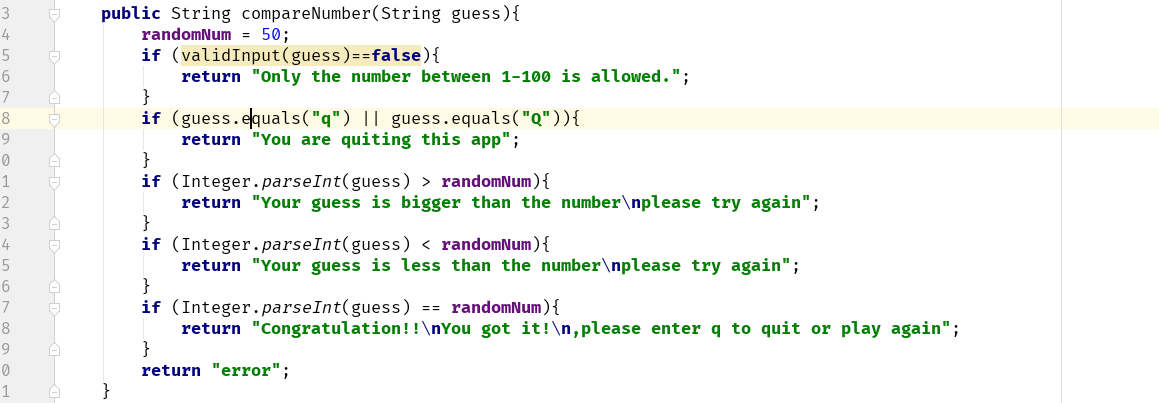
Steps1: Write a framework for that function.



Step2: create test for target method. Then run it with expected input. The result should be failed because the target method has not been finished.



Step3: Finish the compareNumber function. For the testing purpose, I set a fake random number 50 in here.

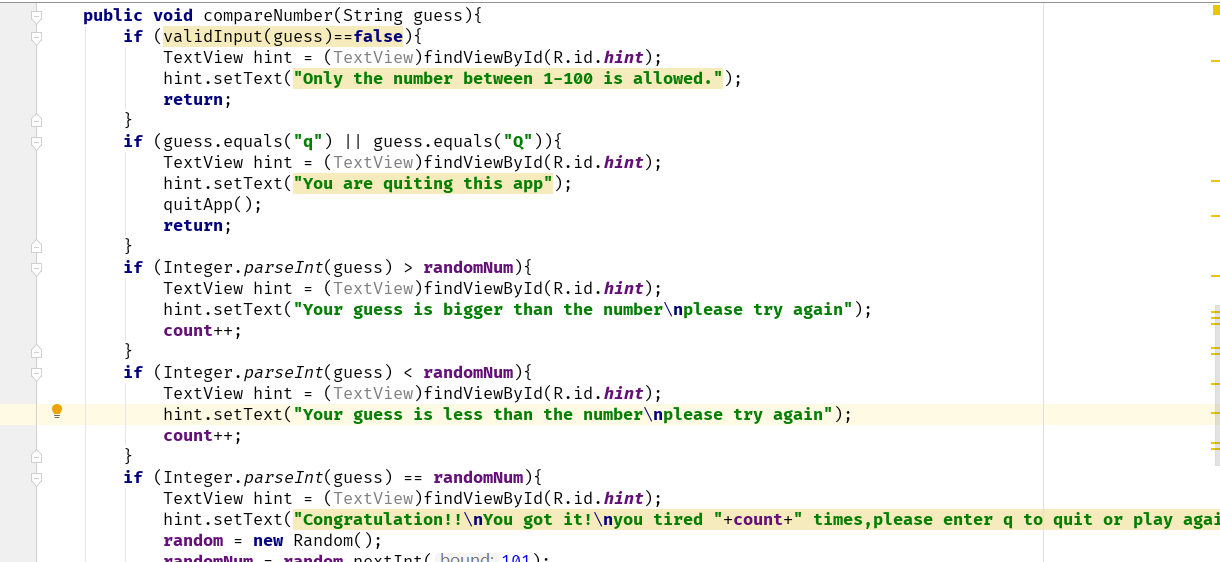


## Test case

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Purpose | Test Data | Expected Result | Actual  Result | Status |
| 1 | Test with invalid data | “ek” | "Only the number between 1-100 is allowed." |  | Pass |
| 2 | Test with “q” | “q” | "You are quitting this app" |  | Pass |
| 3 | Test with the number is larger than 50 | 51 | "Your guess is bigger than the number\nplease try again" |  | Pass |
| 4 | Test with the number is less than 50 | 49 | "Your guess is less than the number\nplease try again" |  | Pass |
| 5 | Test with same number 50 | 50 | "Congratulation!!\nYou got it!\n,please enter q to quit or play again" |  | Pass |

## Conclusion

This method can correctly compare the user’s input with the fake random number to return suitable feedback. Thus, I can change the method return type to void and add more function to pass these feedbacks to users’ screen as below.



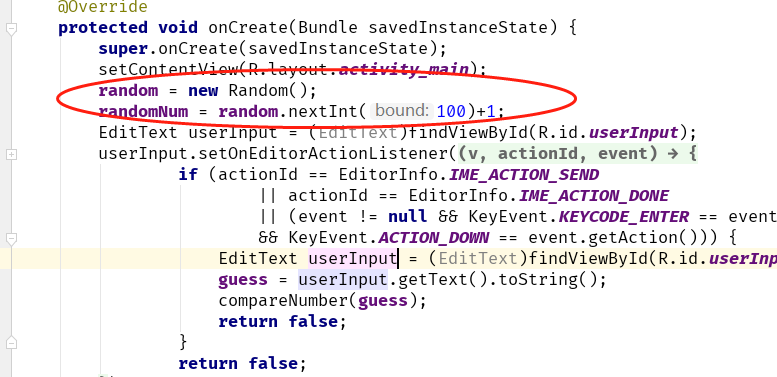
# Code refactoring

By smelling the code for this program, several problems that can be changed to improve the quality of the code have been found and listed below:

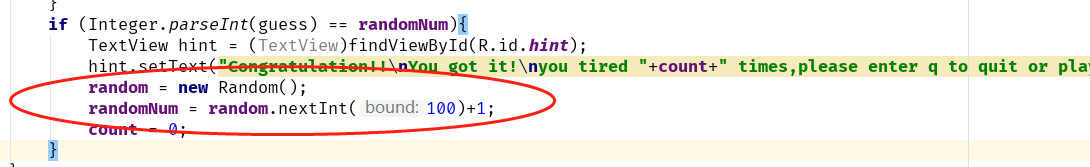
## 1) Duplicate code

The code for create random number has been repeatedly used in both at the start of activity and where the user successfully guesses the random number.

Original one:



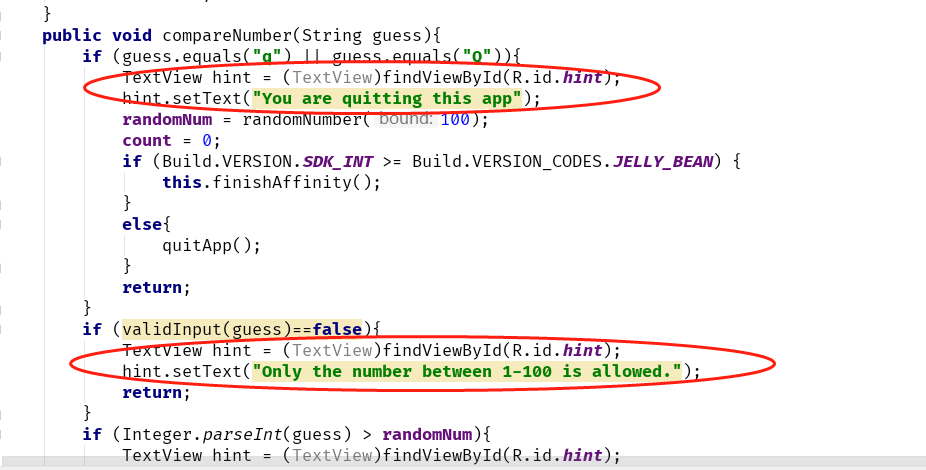
Improved one:



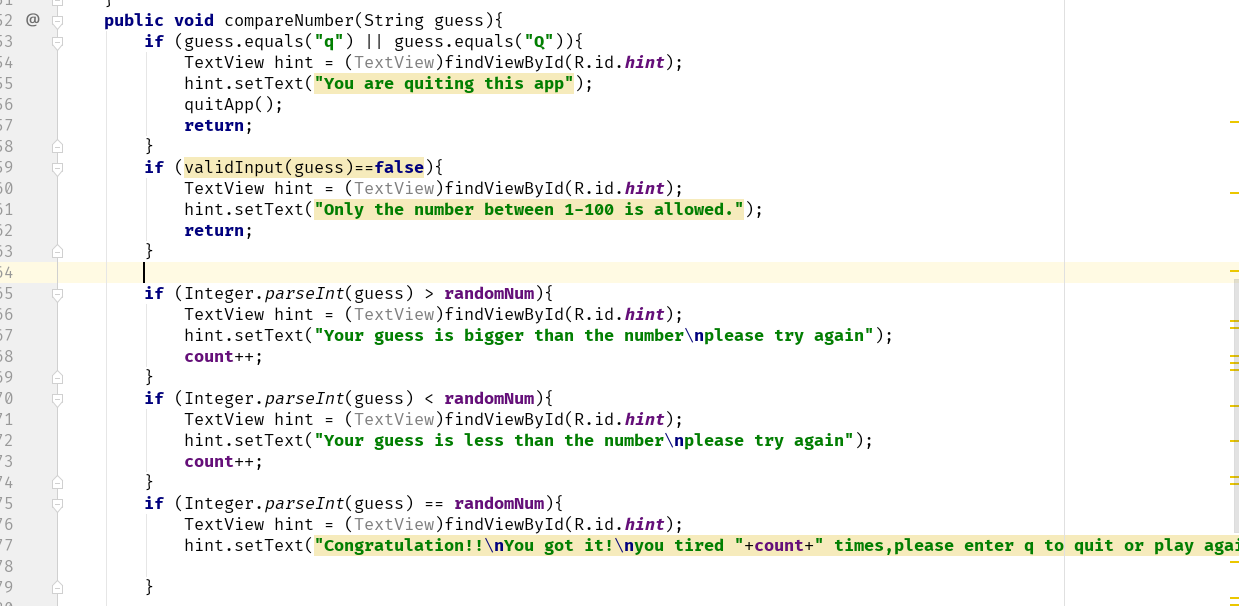
These two-parts present same function and they are in same class. Therefore, they can be extracted to one method and called in both places.

## Another duplicate code

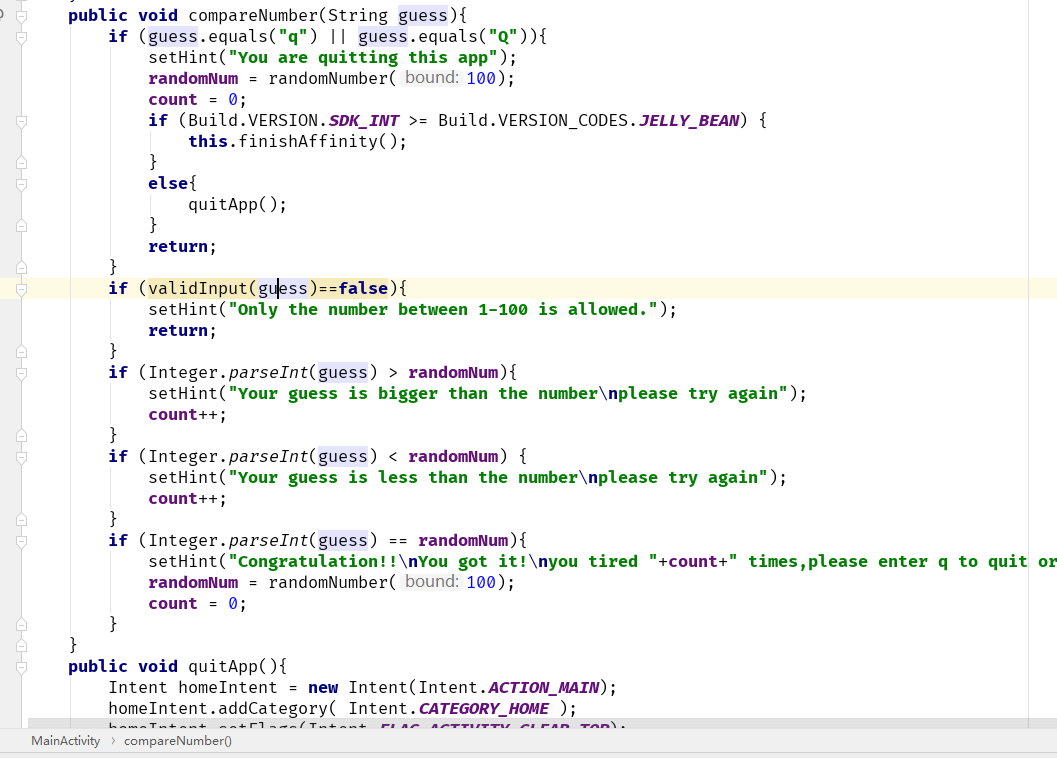
Same as random function, the function for setting the hint text also can be separated.



Original one:

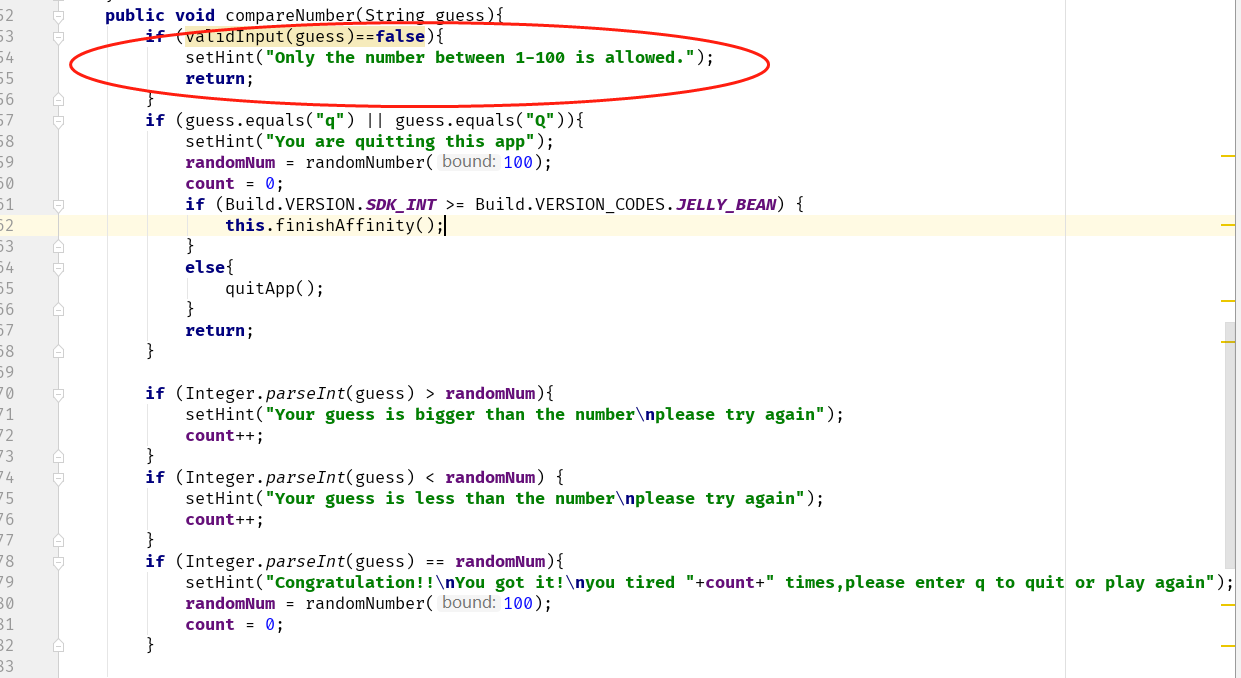


Improved one:



## Optimization for if statement

The order of if statement is important. The if statement could be expensive if the statement of it is unpredictable. Because Modern CPUs will handle fetch, decode, execute and write-back at same time, they will try to predict the result of if statement. In the compareNumber function, this function should go through as less if statements as it can. And the possibility of receiving an arbitrary invalid from user is much more than receiving “q” or “Q”. Thus, the validation for invalid input should be put at the first position to return the function as soon as possible.



## Comment

Comments also are important for long term maintenance and change of developer. Suitable comments make the code maintainable and understandable.

