Bug 4 Investigation

Legend: **blue** words are classes, **green** words are instance methods, and **purple** words are instance variables. (If a word is not coloured, that means it is being used in its natural English sense).

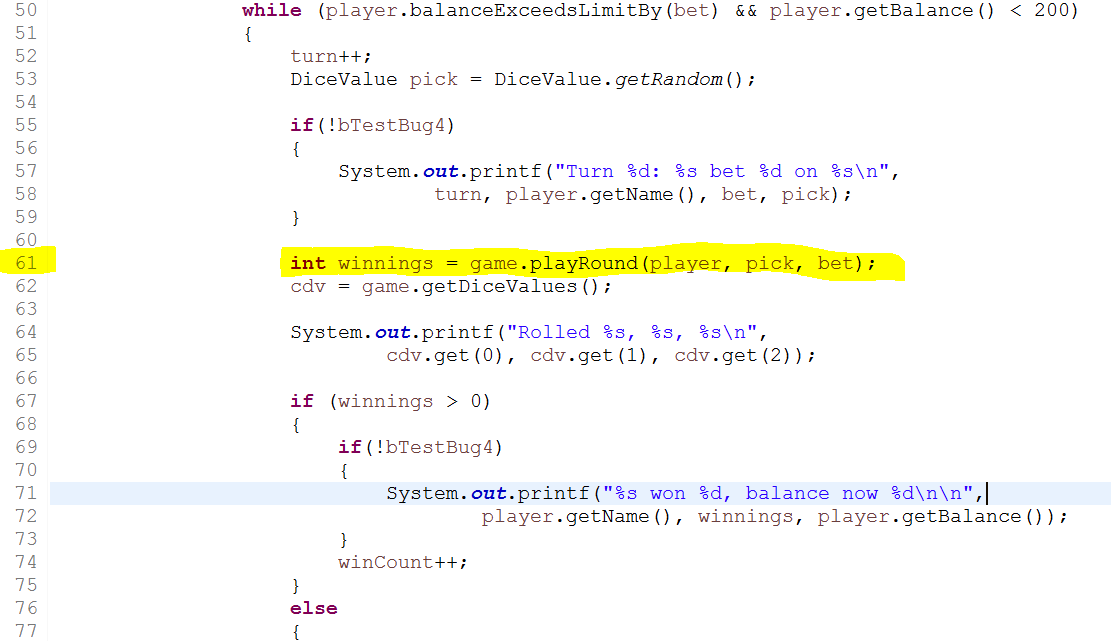
# Description

All rolls are identical in every game of each run through of Main. For example, it could be CROWN, CLUB, HEART repeated for every roll for every game. However, when the run is repeated, this roll can change. For example, in the next run it could be HEART, DIAMOND, DIAMOND repeated on every roll of every game.

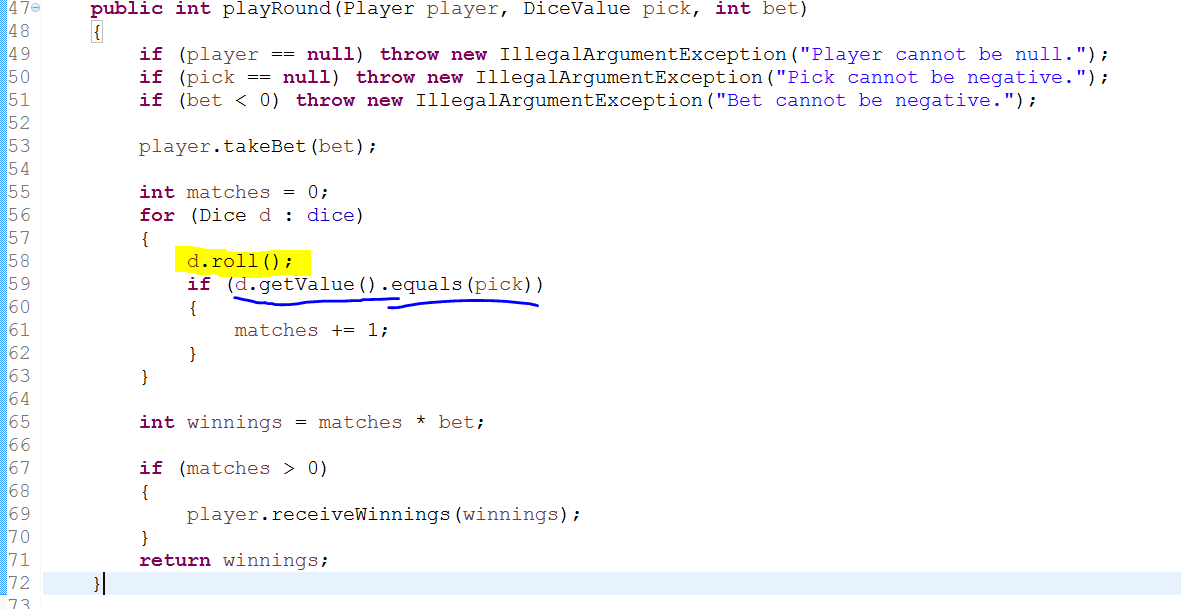
# Static Review

The method that should generate a new **DiceValue** to compare against the pick is **Dice**’s **roll**. This occurs in **Main**’s **main** method, in **Game**’s **playRound** method here:

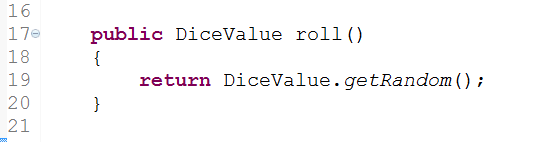
**Main**’s **main**: (highlighted in yellow)



**Game**’s **playRound**: (highlighted in yellow)



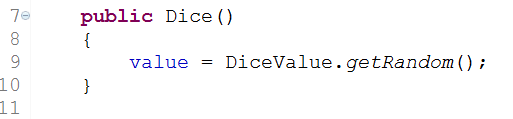
**Dice**’s **roll**:



We notice the following things:

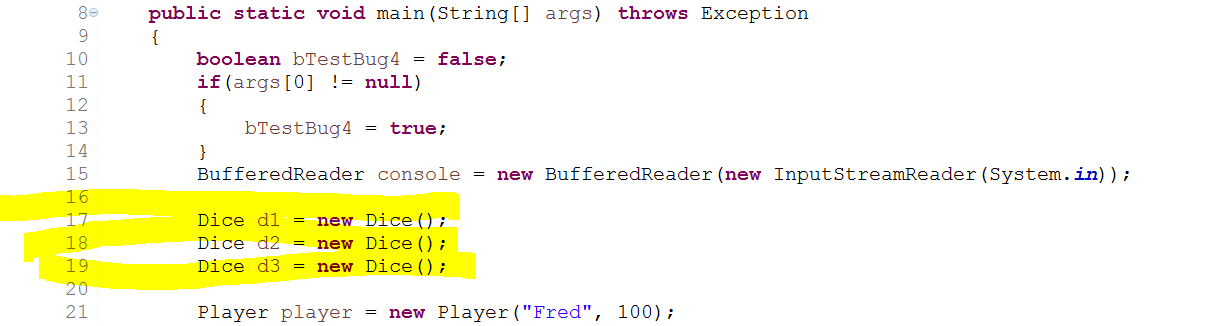
1. The **Dice** class has an instance variable **value** of type **DiceValue.**
2. It is this value that is getting compared against the pick (underlined in blue in **Game**’s **playRound** method, pictured above).
3. The roll method in **Dice** does not change the instance variable **value** (it returns a **DiceValue** object which is not captured – in **playRound** in **Game** it is called like a method with a void return).

If we look at the constructor of **Dice**:



We notice some additional things:

1. **DiceValue**’s **getRandom** is being called rather than the more natural **roll** method (given it is the same class). Although this shouldn’t matter because all **roll** does is call **getRandom**, and return exactly what **getRandom** returns.
2. The variable **value** is set from the return value of **getRandom**.
3. Nowhere else in the class **Dice** is **value** set in any way.
4. Conclusion from 5 and 6: **value** is invariant over the life of the instance.
5. If we look at the whole game loop in **Main**’s main we notice that here is the only place that new **Dice** are created:



From this we can guess that all **Dice** used in each run of the program are the same three objects. This will be tested to be sure (see *Testing Hypotheses* later in this document).

1. From 2, 7 and 8 (8 subject to testing as described later in this document) we can see why the rolls are always the same:
   * The **value** is invariant
   * The same **Dice** are used in every **Game** (therefore using the same invariant **value**)
   * It is the **value** that is displayed by the program as a roll
2. From 9: Obviously the roll will be identical every **Game**!

Therefore, our hypotheses are the following:

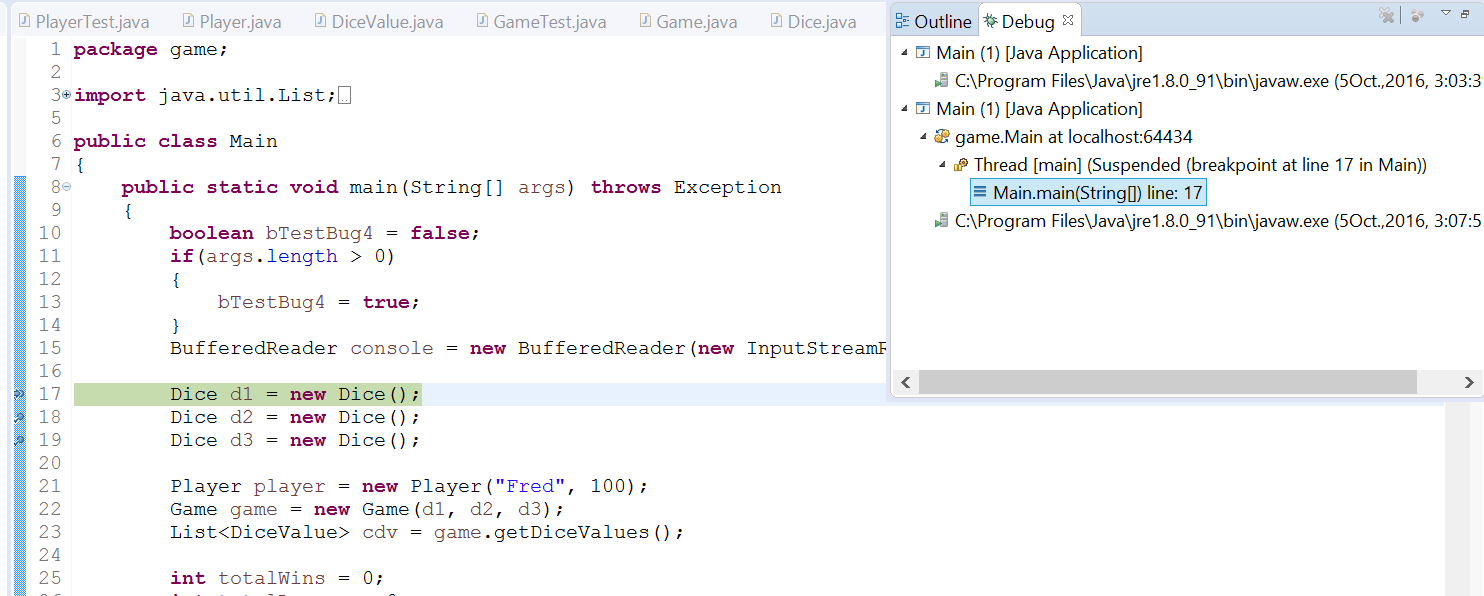
1. **Dice** are created only once per run of the program, and then reused for each **Game**.
2. The **value** is invariant over the life of any particular instance of **Dice**.
3. The **value** of the **Dice** is what is used as each roll & compared to the pick to determine if the player wins or not.

# Hypothesis testing

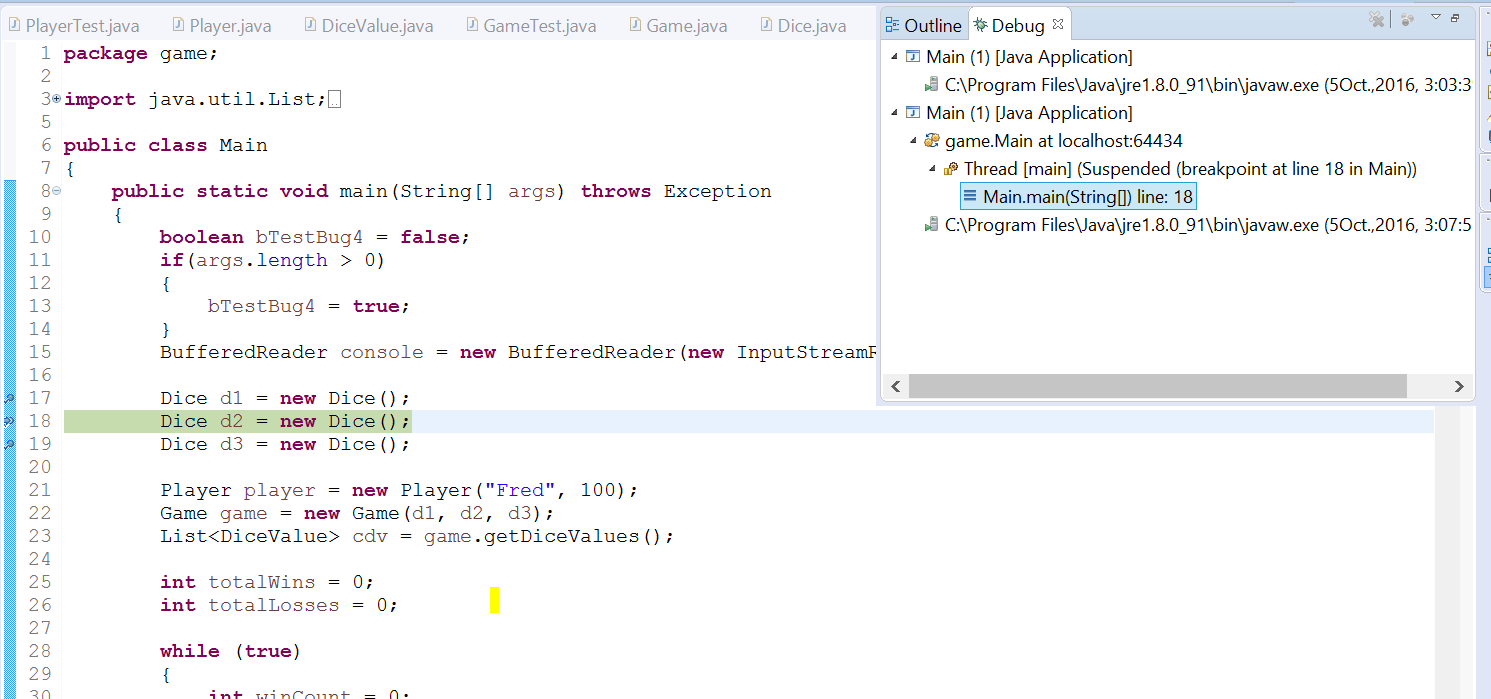
## Hypothesis 1

To test this one, we place breakpoints in **main** where the three **Dice** are created (as highlighted above) to prove that it isn’t orphan code that doesn’t get executed. We also place a breakpoint in **Dice**’s constructor, and run the program in debug mode and count how many times the point of execution enters the constructor. From the static review, we see that there are three **Dice** objects required in a **Game**. Therefore, we expect that the point of execution will enter **Dice**’s constructor exactly three times. If it does this, this will verify hypothesis 1 as true.

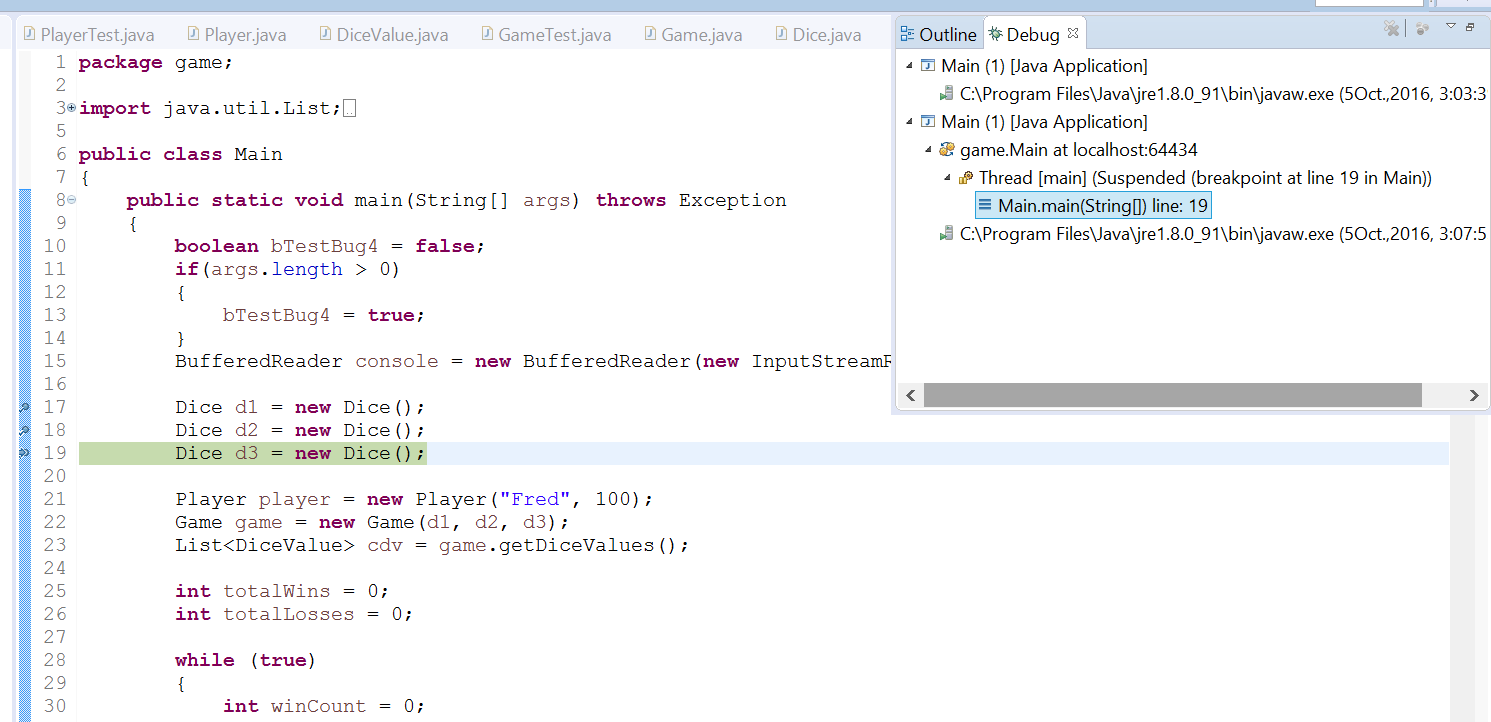
Hit creation of first Dice:



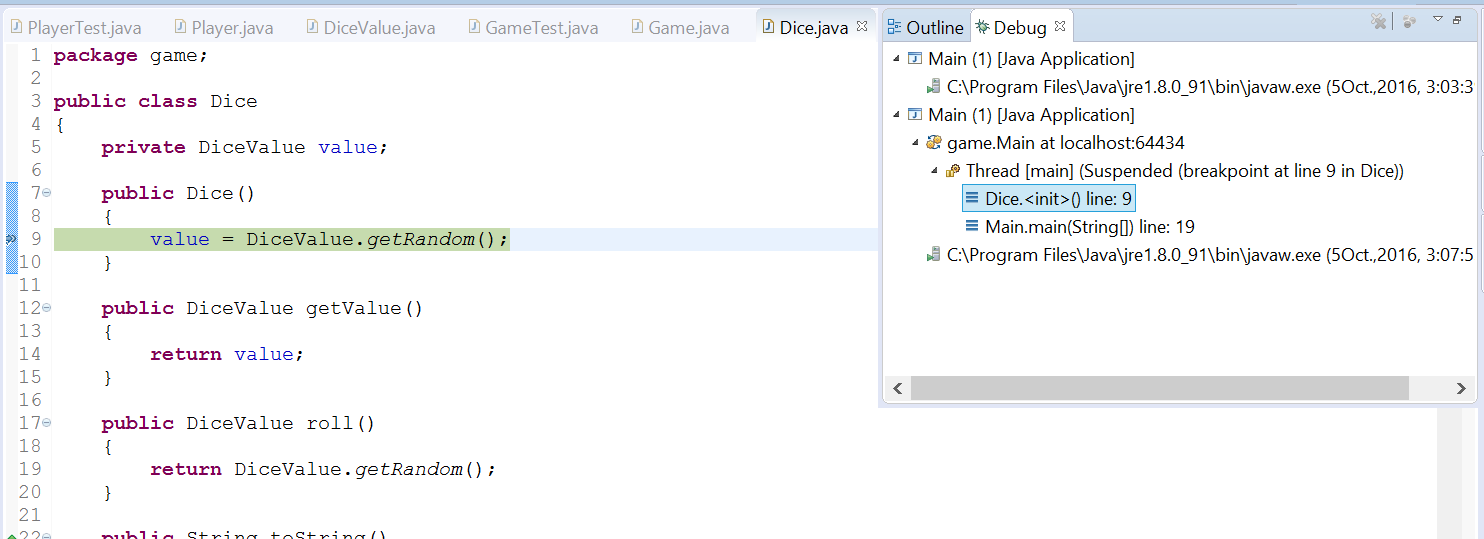
Hit creation of second Dice:



Hit creation of third Dice:



In between, it also hits the constructor (as expected, given it says “new Dice()”):



And most importantly, it only goes into the constructor three times (once after each “new Dice()” statement, as expected), and never again. This proves hypothesis 1: The three Dice created at the beginning of main are reused in each Game.

## Hypothesis 2

The instance variable **value** has a **private** access modifier meaning that the only way it can be changed is through methods in the class itself (there cannot be anything like: Dice dice = new Dice(); dice.value = DiceValue.SPADES; for example). In our static review we found that the only place in the **Dice** class that **value** is set is in the constructor (there is no setValue method for example). So, we would expect that **value** would only be set at creation and never changed again. However, maybe it is possible for a program to be sneaky and call the constructor after already being created? We don’t need to worry about this, as the previous test (hypothesis 1) has already tested for this, by putting breakpoints in the constructor, and noting that it only gets called three times (on creation of the three **Dice** objects used in **main**).

So, we don’t need to do any tests. Hypothesis 2 is proven during testing of hypothesis 1.

## Hypothesis 3

This one is quite clear from the static review, but we can still test it. We will put a breakpoint just after the rolls are displayed in the console window, and have a look at what the value of the corresponding **Dice** object’s **value** is at that moment. To assure our audience that it is not a coincidence, we will do this three times.