**Bug 1:** Game does not pay out at correct level.

UAT case to replicate issue:

Description:

Confirm that player’s balance does not increase even if bet is won

Pre-conditions:

* Dice class exists
* DiceValue class exists
* Game class exists
* Main class exists
* Player class exists

Post-conditions:

* Game is executed
* Evidence that player’s balance never increases

Data required:

* None

**Debugging log**

Simplification:

To reduce size of output, initial balance set at 10, and limit number of games to 1.

Output when program is run:

Start Game 0:

Fred starts with balance 10, limit 0

Turn 1: Fred bet 5 on CLUB

Rolled CLUB, CROWN, ANCHOR

Fred won 5, balance now 10

Turn 2: Fred bet 5 on CROWN

Rolled CLUB, CROWN, ANCHOR

Fred won 5, balance now 10

Turn 3: Fred bet 5 on DIAMOND

Rolled CLUB, CROWN, ANCHOR

Fred lost, balance now 5

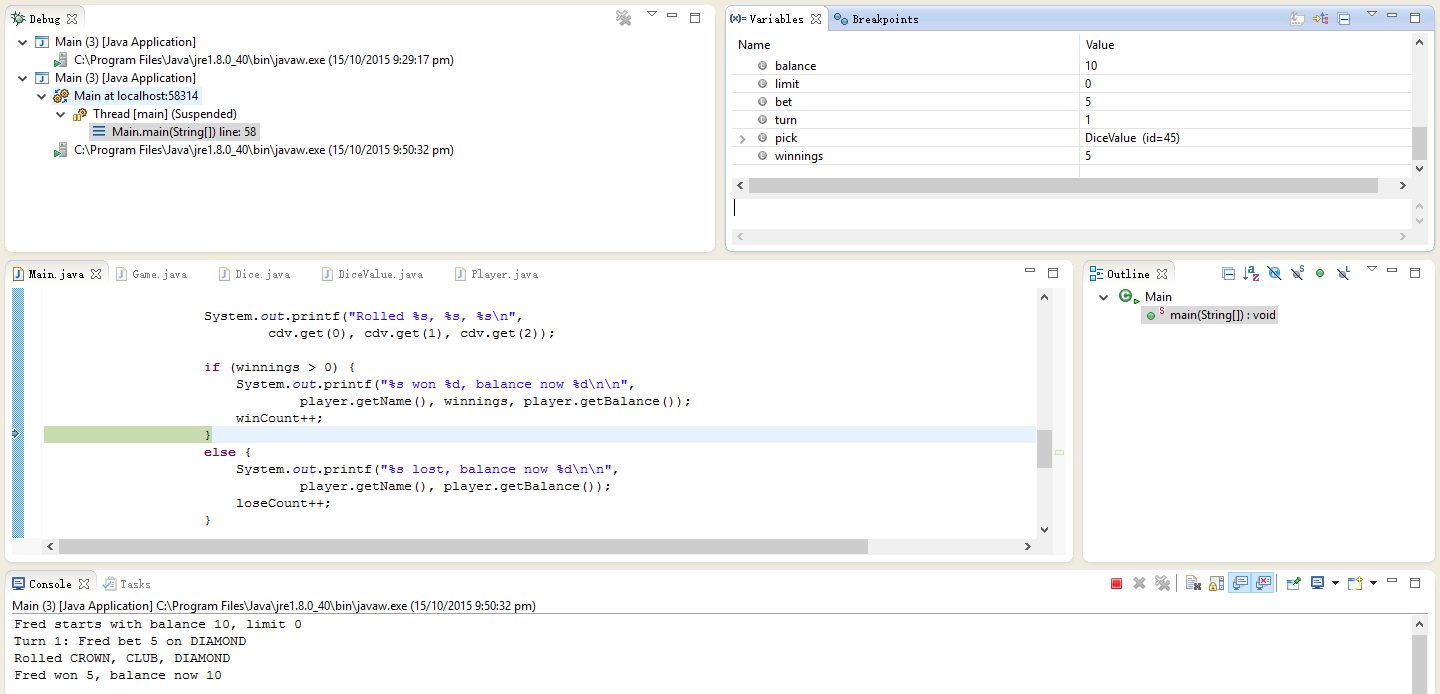
3 turns later.

End Game 0: Fred now has balance 5

Win count = 2, Lose Count = 1, 0.67

Clearly bug is still present after simplification.

Hypothesis: Whenever Fred wins a bet, he does not receive his original bet back.



Here, Fred won the bet. The winnings variable correctly shows 5 but the console incorrectly shows that balance is still 10. All variable values were correct until this point. It should be noted that the variable “balance” in the variable inspector represents the “balance” variable in Main.java and not Player.java. As such, its state is unimportant beyond setting the initial balance of the player.

Resolution:

In the playRound() method of Game.java, the last if statement was changed.

Output:

Start Game 0:

Fred starts with balance 10, limit 0

Turn 1: Fred bet 5 on ANCHOR

Rolled ANCHOR, DIAMOND, CROWN

Fred won 5, balance now 15

Initial balance and number of games was returned to 100. Results checked out to be sane.

**Bug 2:** Player cannot reach betting limit

UAT case to replicate issue:

Description:

Confirm that player’s balance can never reach lower limit

Pre-conditions:

* Dice class exists
* DiceValue class exists
* Game class exists
* Main class exists
* Player class exists

Post-conditions:

* Game is executed
* Evidence that player’s balance never reaches lower limit

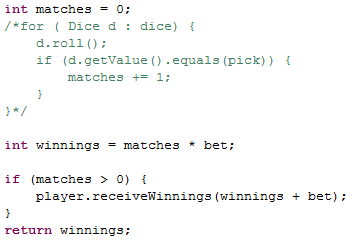
Data required:

* None

**Debugging log**

Simplification:

To reduce size of output, initial balance set at 10, and limit number of games to 1. Furthermore, to ensure the Fred always loses, the following code was commented out:



Output when program is run:

Start Game 0:

Fred starts with balance 10, limit 0

Turn 1: Fred bet 5 on CROWN

Rolled DIAMOND, CROWN, DIAMOND

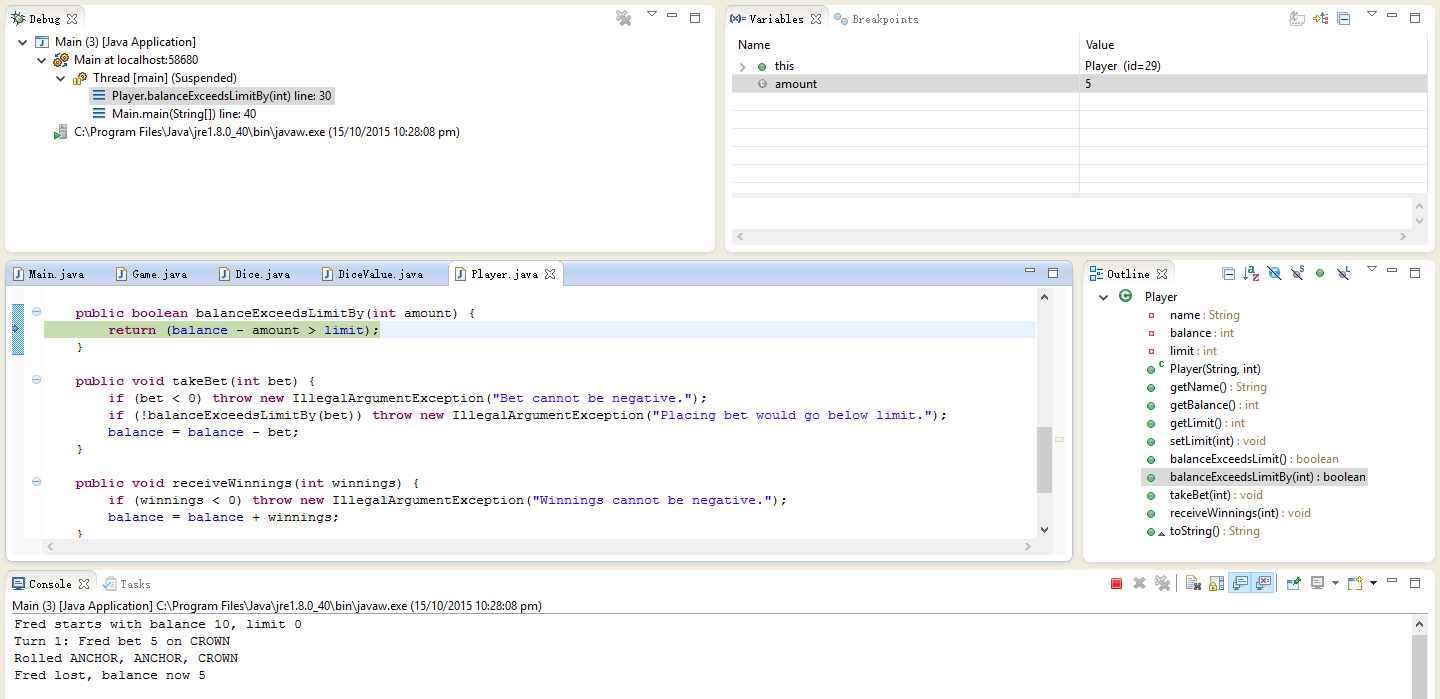
Fred lost, balance now 5

1 turns later.

End Game 0: Fred now has balance 5

Win count = 0, Lose Count = 1, 0.00

Hypothesis: balanceExceedsLimitBy() method in Player.java is not behaving properly.



The bug is clearly shown here. Current balance is 5, amount is also 5. According to the code, balance – amount must > 0 which will resolve to false in this case. As a result, the while loop in Main.java ends.

Resolution:

In the balanceExceedsLimitBy() method of Player.java, the > symbol in the return statement was changed to >=.

Output:

Start Game 0:

Fred starts with balance 10, limit 0

Turn 1: Fred bet 5 on CROWN

Rolled CROWN, HEART, HEART

Fred lost, balance now 5

Turn 2: Fred bet 5 on HEART

Rolled CROWN, HEART, HEART

Fred lost, balance now 0

2 turns later.

End Game 0: Fred now has balance 0

Win count = 0, Lose Count = 2, 0.00

Initial balance and number of games was returned to 100. Commented out section restored. Results checked out to be sane.

**Bug 3:** Odds in the game do not appear to be correct

UAT case to replicate issue:

Description:

Confirm win:loss ratio after playing is incorrect

Pre-conditions:

* Dice class exists
* DiceValue class exists
* Game class exists
* Main class exists
* Player class exists

Post-conditions:

* Game is executed
* Evidence that final win:loss ratio does not fit probability

Data required:

* None

**Debugging log**

Simplification:

Make sample size sufficiently large such that the end win:loss ratio could be reasonably expected to correlate to expected result according probability. To achieve this, the code was modified to play one game, but to take 10,000 turns. To avoid the game ending prematurely, conditionals were disabled for the loop and the balanceExceedsLimitBy() method in Player.java was modified to always return true.

Output when program is run:

Turn 9997: Fred bet 5 on DIAMOND

Rolled HEART, ANCHOR, CLUB

Fred lost, balance now 10865

Turn 9998: Fred bet 5 on CLUB

Rolled HEART, ANCHOR, CLUB

Fred won 5, balance now 10870

Turn 9999: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, CLUB

Fred lost, balance now 10865

Turn 10000: Fred bet 5 on CROWN

Rolled HEART, ANCHOR, CLUB

Fred lost, balance now 10860

10000 turns later.

End Game 0: Fred now has balance 10860

Win count = 6076, Lose Count = 3924, 0.61

This is not close to the expected ratio of 0.42. Furthermore, every single time the dice rolls resulted in the exact same result. Another observation is that “Spade” is never shown.

Hypothesis: New dice rolls are not being accounted for in the game.

It is observed that the pick variable in Main.java seems to vary randomly as expected, whereas cdv does not. Therefore, it would make sense to investigate where the implementations differ as they both eventually make use of the DiceValue.getRandom() method. The main difference is, whereas the pick variable directly calls the DiceValue.getRandom() method, cdv is mapped to game.getDiceValues(). It is further found that game.getDiceValues() is dependent on the method playRound() which is supposed to modify the class variables each time the roll() method is activated for any given dice.

To check whether roll() is in fact changing the value of the dice, a couple of print statements were added into the code. The output provided the answer:

Start Game 0:

Fred starts with balance 100, limit 0

Turn 1: Fred bet 5 on CROWN

What d.roll() returns: CLUB

What d.getValue() returns: HEART

What d.roll() returns: SPADE

What d.getValue() returns: CLUB

What d.roll() returns: ANCHOR

What d.getValue() returns: HEART

Rolled HEART, CLUB, HEART

Fred lost, balance now 95

It is apparent that the roll() is effectively not doing anything at all. That explains why each round always had the same dice values, which would have a severe impact on odds and therefore skew the resulting win:loss ratio.

The fact that SPADE never shows up could mostly likely be attributed to the way the DiceValue.getRandom() method works. Sure enough, upon observation, the RANDOM.nextInt call is used incorrectly as the int parameter is excluded from the random number range.

Resolution:

The roll() method inside Dice.java was fixed to ensure each roll actually changed the value of the die. To make SPADE available, + 1 was added onto the RANDOM.nextInt() parameter.

Output:

Turn 160: Fred bet 5 on CROWN

Rolled ANCHOR, SPADE, HEART

Fred lost, balance now 10

Turn 161: Fred bet 5 on ANCHOR

Rolled CROWN, HEART, CLUB

Fred lost, balance now 5

Turn 162: Fred bet 5 on CROWN

Rolled DIAMOND, HEART, HEART

Fred lost, balance now 0

162 turns later.

End Game 99: Fred now has balance 0

Win count = 9260, Lose Count = 12768, 0.42

Program was reverted to its usual running state to test results. The expected ratio of 0.42 was observed on multiple runs.

**Bug 4:** Program has an infinite while loop

UAT case to replicate issue:

Description:

Confirm presence of infinite loop

Pre-conditions:

* Program can run

Post-conditions:

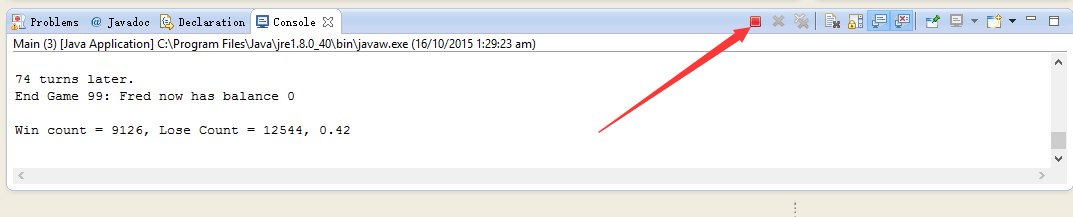
* Program does not terminate

Data required:

* None

Simplification:

The infinite loop is plain to see, no simplification needed.



Hypothesis: The loop is redundant and all the content inside can simply be taken out of it.

Resolution:

The while loop was deleted with all content moved outside of it. A couple of random lines of code were also deleted as they are not used and cause a compile error. Program now terminates successfully and also shows the final print statement.

