



## **Assignment 2: Static and Dynamic Analysis**

Project: Black Jack (Java)

Yash Patel 100746810

SOFE 3980U: Software Quality

Dr. Akramul Azim

March 24, 2022

## Software Quality Assignment 2: Black Jack Game Report

### Program Description

The purpose of this assignment is to continue working on the program created in the previous assignment. The program created in the previous assignment was a Black Jack replication using the Agile Process Model. The GitHub repository can be found here at, <https://github.com/yash-patel268/Black-Jack-Game>. The premise of the assignment was to create a program using an engineering process and to validate it using testing. My project was created using Java and was tested using JUnit.

The objective of assignment 2 is to implement program slicing to display static analysis and to implement program instrumentation to display dynamic analysis. Program slicing aims to remove a proportion of the code and reflect on the functionality after the removal. Program instrumentation aims to see what happens during run time and if there's something that can be improved by obtaining the data. The new GitHub can be found here, <https://github.com/yash-patel268/Black-Jack-Game-Analysis>.

### Implementation of Static Analysis technique

For this assignment, the method of static analysis will be done through program slicing. Program slicing is where bits of code are removed from a selected section of code. The program will then be checked to see if any logical discrepancies could potentially result in a buggy program. Another reason for program slicing is to understand code where there is a lack of documentation, the process becomes trial and error to see what kind of results the section of code has.

## Program Slicing Example

Program	A Program Slice
read(A);	read(A);
read(B);	read(B);
Q := 0;	R := A;
R := A;	while R >= B do
while R >= B do	begin
begin	R := R - B;
R := R - B;	end;
Q := Q + 1	print(R);
end;	
print(Q);	
print(R);	

Program Slice for the variable 'R'

Figure 1: Program Slicing Example [1]

```

public class ProgramSlicer {
    public static void main(String[] args) {
        //Setup scanner method
        Scanner scnr = new Scanner(System.in);
        System.out.println("Enter input filename: ");
        //get file for static analysis
        String inputFile = scnr.next();
        System.out.println("Enter input var: ");
        //get var for static analysis
        String input = scnr.next();

        try (BufferedReader br = new BufferedReader(new FileReader(inputFile))) {
            String line;
            while ((line = br.readLine()) != null) {
                //if the var is found in file print it
                if(line.contains(input)) {
                    System.out.println(line);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Figure 2: Program Slicing Class

The class above is a custom class that will take an input of a file to search through and an input of a variable to find. The class will find every instance of the variable and print it to the console to be examined. The working directory for the application is with the company directory which is located within the com directory, which is within the src directory.

*Example 1 - suit variable in Card.java*

```
Enter input filename:
Card.java
Enter input var:
suit
    //initialize variables for card suit and value
    private String suit;
    //getter method for card suit
        return suit;
    public Card(String value,String suit){
        this.suit = suit;
    //initialize setter method to card to have suit and value assigned
    public void setCard(String value,String suit){
        this.suit = suit;
        String name = value + " of " + suit;
```

Figure 3: suit variable program slice output

*Example 2 - value variable in Card.java*

```
Enter input filename:
Card.java
Enter input var:
value
    //initialize variables for card suit and value
    private String value;
    //getter method for card value
        return value;
    public Card(String value,String suit){
        this.value = value;
    //initialize setter method to card to have suit and value assigned
    public void setCard(String value,String suit){
        this.value = value;
    //initialize method to allow card to be display in console, also makes return value string
        String name = value + " of " + suit;
```

Figure 4: value variable program slice output

*Example 3 - cardLeft variable in BlackJackGame.java*

```
Enter input filename:  
BlackJackGame.java  
Enter input var:  
cardLeft  
    private int cardLeft;  
        cardLeft = 0;  
            cards[cardLeft] = new Card(suit[i], value[k]);  
            cardLeft++;  
        for(int i =0; i<cardLeft/2;){  
            r1 = rand.nextInt(cardLeft);  
            r2 = rand.nextInt(cardLeft);  
        cardLeft--;  
        return cards[cardLeft];
```

Figure 5: value variable program slice output

*Example 4 - cards variable in Card.java*

```

Enter input filename:
BlackJackGame.java
Enter input var:
cards
    private Card[] cards;
    //initialize variable to hold total cards
    //initialize variable to hold player cards
    //initialize variable to hold dealer cards
    //calculates the total number of cards to be assigned
    cards = new Card[suit.length * value.length];
        cards[cardLeft] = new Card(suit[i], value[k]);
    if(index>=0 && index < cards.length) {
        cards[index].setCard(s, f);
        Card holder = cards[r1];
        cards[r1] = cards[r2];
        cards[r2] = holder;

        if(cards[i].getValue().equals("Ace"))
        else if(cards[i].getValue().equals("Two"))
        else if(cards[i].getValue().equals("Three"))
        else if(cards[i].getValue().equals("Four"))
        else if(cards[i].getValue().equals("Five"))
        else if(cards[i].getValue().equals("Six"))
        else if(cards[i].getValue().equals("Seven"))
        else if(cards[i].getValue().equals("Eight"))
        else if(cards[i].getValue().equals("Nine"))

```

```

//initialize method to deal cards which will start at the last position of card array and work down
    return cards[cardLeft];
//initialize method to check for blackjack or if higher than 21 when cards are assigned for either player or dealer

```

Figure 6: cards variable program slice output

*Example 5 - value variable in Card.java*

```

Enter input filename:
BlackJackGame.java
Enter input var:
r1

    int r1, r2;
        r1 = rand.nextInt(cardLeft);
        if(r1 != r2){
            Card holder = cards[r1];
            cards[r1] = cards[r2];

```

Figure 7: r1 variable program slice output

Implementation of Dynamic Analysis

For the assignment, the method of dynamic analysis will be done through program instrumentation. Program instrumentation is done externally or internally measuring the code for specific criteria. Items that can be measured include run time, function calls, percentage of code usage, etc. The goal of dynamic analysis is to observe potential uses, collect data, analyze data, or find potential bugs.

The requirement for this section is to measure the execution time of 5 portions of the code. This will be done through initializing an internal clock that will be mapping during method calls from start to end and will display the total time taken. Since Java has a built-in library to measure time all that is required to initialize a start time variable at beginning of the method and an end time variable at the end of the method. The variables are subtracted and the value is printed to the console. This will be for all methods.

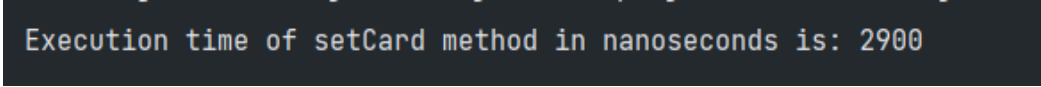
*Example 1 - setCard method execution time*

```

public void setCard(String value,String suit){
    long startTime = System.nanoTime();
    this.value = value;
    this.suit = suit;
    long endTime = System.nanoTime();
    long totalTime = endTime - startTime;
    System.out.println("Execution time of setCard method in nanoseconds is: " + totalTime);
}

```

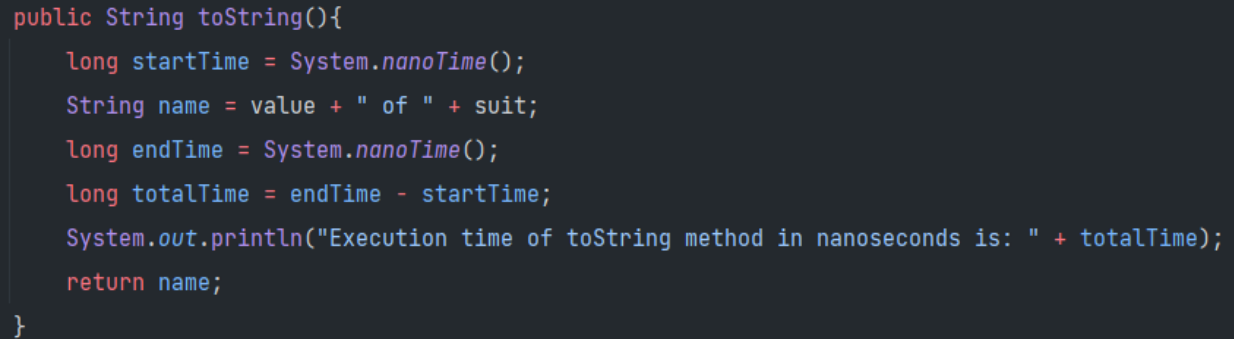
Figure 8: setCard method with internal clock



Execution time of setCard method in nanoseconds is: 2900

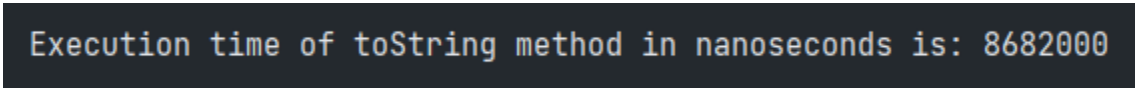
Figure 9: execution of setCard method

*Example 2 - toString method execution time*



```
public String toString(){
    long startTime = System.nanoTime();
    String name = value + " of " + suit;
    long endTime = System.nanoTime();
    long totalTime = endTime - startTime;
    System.out.println("Execution time of toString method in nanoseconds is: " + totalTime);
    return name;
}
```

Figure 10: toString method with internal clock



Execution time of toString method in nanoseconds is: 8682000

Figure 11: execution of toString method



*Example 3 - shuffle method execution time*

```

public void shuffle(){
    long startTime = System.nanoTime();
    Random rand = new Random();
    int r1, r2;
    for(int i =0; i<cardLeft/2;){
        r1 = rand.nextInt(cardLeft);
        r2 = rand.nextInt(cardLeft);

        if(r1 != r2){
            Card holder = cards[r1];
            cards[r1] = cards[r2];
            cards[r2] = holder;
            i++;
        }
    }
    long endTime = System.nanoTime();
    long totalTime = endTime - startTime;
    System.out.println("Execution time of shuffle method in nanoseconds is: " + totalTime);
}

```

Figure 12: shuffle method with internal clock

```

Execution time of shuffle method in nanoseconds is: 164800

```

Figure 13: execution of shuffle method

*Example 4 - returnScore method execution time*

```

public int returnScore(BlackJackGame hand, String type){
    long startTime = System.nanoTime();
    int h=0;
    if(type=="hand"){
        h = hand.Score(playerCounter);
    }
    else if(type=="dealerHand"){
        h = hand.Score(dealerCounter);
    }
    long endTime = System.nanoTime();
    long totalTime = endTime - startTime;
    System.out.println("Execution time of returnScore method in nanoseconds is: " + totalTime);
    return h;
}

```

Figure 14: returnScore method with internal clock

Execution time of returnScore method in nanoseconds is: 1300

Figure 15: execution of returnScore method

*Example 5 - returnScore method execution time*

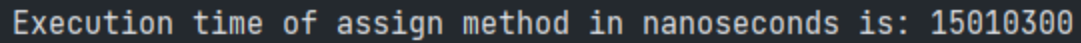
```
public String assign(BlackJackGame deck, BlackJackGame hand, String type){
    long startTime = System.nanoTime();
    //if player calls for assign method
    if(type=="hand"){
        //shift player array
        playerCounter++;
        //deal a card
        Card card = deck.deal();
        //assign card with suit and value
        hand.setCards(playerIndex, card.getSuit(), card.getValue());
        //print the card to console
        System.out.println(card);

        //print score to console
        System.out.println("Score of Hand: " + hand.Score(playerCounter));
        //check if score is higher or equal to blackjack
        checkBlackJack(hand.Score(playerCounter));
        System.out.println();
        //increment card array to be ready for next card
        playerIndex++;
    }
    //if dealer calls for assign method
    else if(type=="dealerHand"){
        dealerCounter++;
        Card card = deck.deal();
        hand.setCards(dealerIndex, card.getSuit(), card.getValue());
        System.out.println(card);

        System.out.println("Score of Dealer's Hand: " + hand.Score(dealerCounter));

        System.out.println();
        dealerIndex++;
    }
    long endTime = System.nanoTime();
    long totalTime = endTime - startTime;
    System.out.println("Execution time of assign method in nanoseconds is: " + totalTime);
    return type;
}
```

Figure 16: assign method with internal clock



```
Execution time of assign method in nanoseconds is: 15010300
```

Figure 17: execution of assign method

### Challenges and Lessons Learned

The challenge experienced within this assignment was the lack of experience with implementing static and dynamic analysis. Although through reading the lecture notes and online forums a suitable understanding was reached and allowed for the completion of this assignment.

Implementing the analysis was fairly easy as the static analysis was code-dependent, and dynamic analysis was done through the codes' libraries. Overall, the assignment has allowed me to understand analysis through a hands-on approach and resulted in the successful completion of the assignment.

## References

- [1] A. Azim, Class Lecture, Topic: “Test-Driven Development (TDD), Static Analysis and Dynamic Analysis.” SOFE3980, Ontario Tech University, Mar, 19, 2022.