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

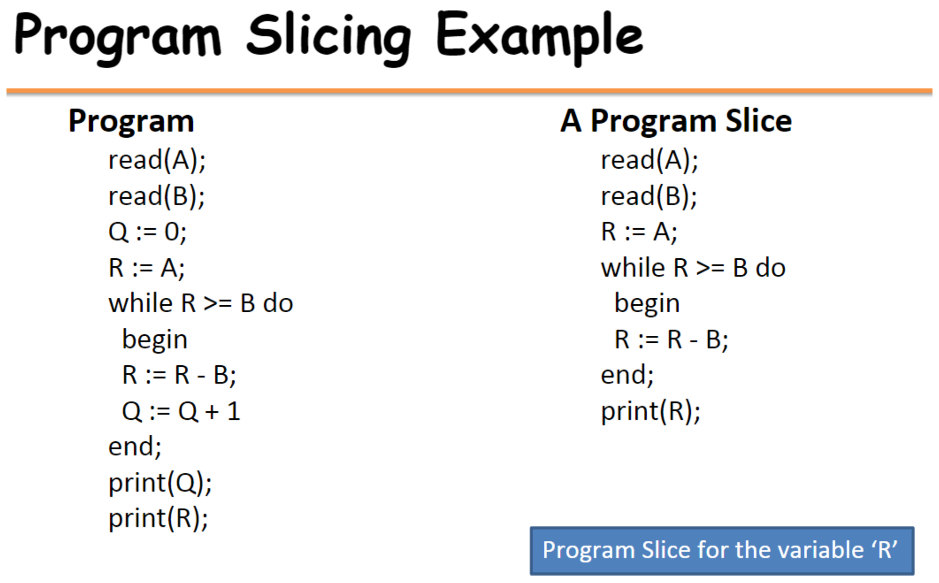
Program Description

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

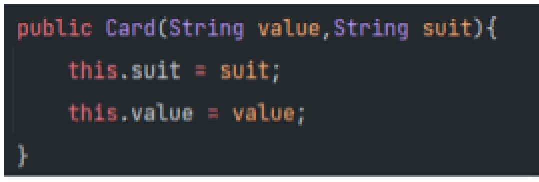
Program
```
read(A);
read(B);
Q := 0;
R := A;
while R >= B do
  begin
  R := R - B;
  Q := Q + 1
end;
print(Q);
print(R);
```

A Program Slice
```
read(A);
read(B);
R := A;
while R >= B do
  begin
  R := R - B;
end;
print(R);
```

Program Slice for the variable 'R'

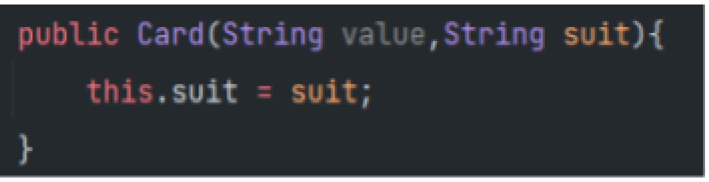
Figure 1: Program Slicing Example [1]

Example 1 - Static Program Slice



```java
public Card(String value,String suit){
    this.suit = suit;
    this.value = value;
}
```

Figure 2: Card method

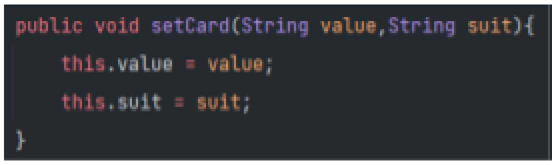


```java
public Card(String value,String suit){
    this.suit = suit;
}
```

Figure 3: Card method after Static slice on suit variable

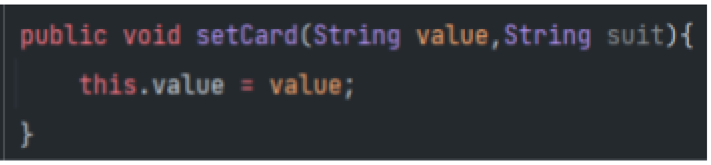
In example 1 there is a static program slice for the suit variable within the Card method. Since the primary focus is on the suit variable anything not related to that variable is removed. After the slice, the Card method will only assign the suit property and allows for independent testing for the only suit.

Example 2 - Static Program Slice



```java
public void setCard(String value,String suit){
    this.value = value;
    this.suit = suit;
}
```

Figure 4: setCard method

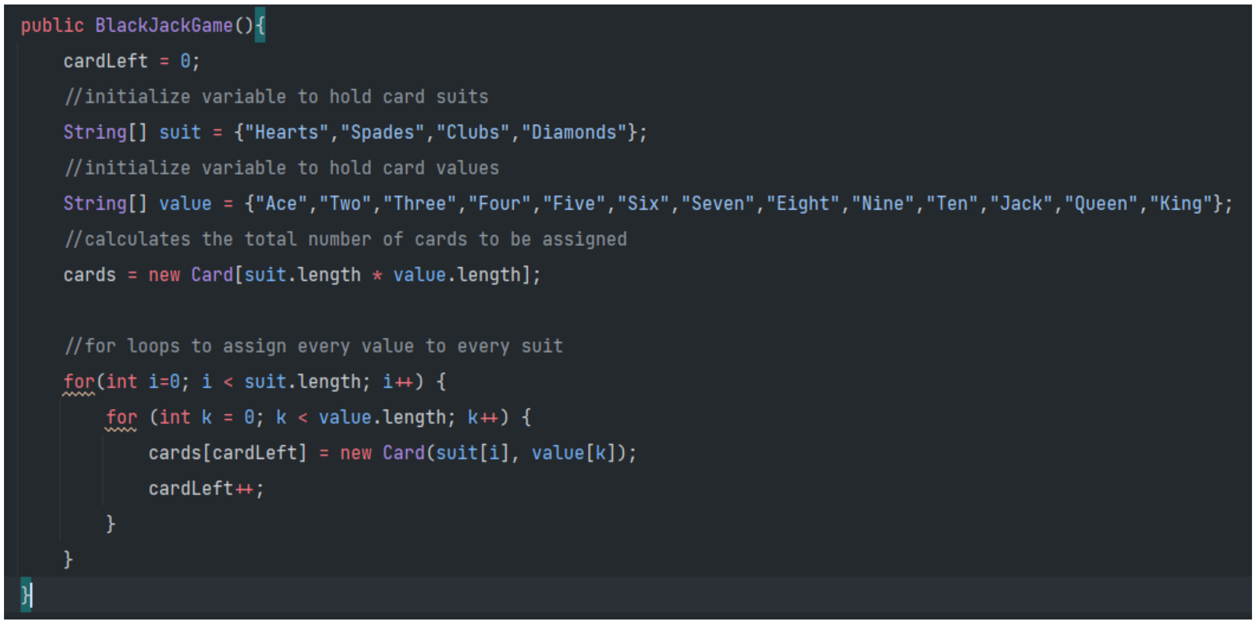


```java
public void setCard(String value,String suit){
    this.value = value;
}
```

Figure 5: setCard method after Static slice on value variable

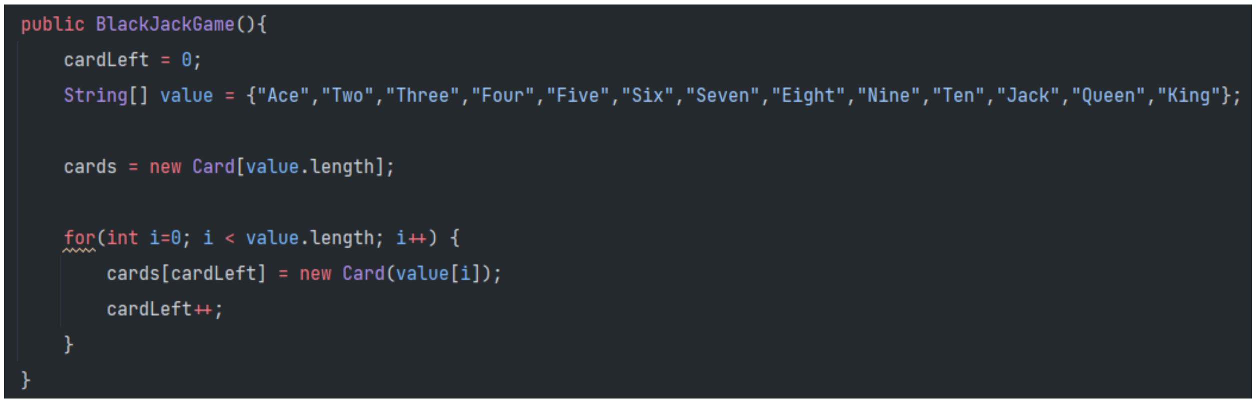
In example 2 there is a static program slice for the value variable within setCard method. Since the primary focus is on the value of the card, the suit variable is removed from the method. This means that the method can only assign a card value.

Example 3 - Static Program Slice



```java
public BlackJackGame(){
    cardLeft = 0;
    //initialize variable to hold card suits
    String[] suit = {"Hearts","Spades","Clubs","Diamonds"};
    //initialize variable to hold card values
    String[] value = {"Ace","Two","Three","Four","Five","Six","Seven","Eight","Nine","Ten","Jack","Queen","King"};
    //calculates the total number of cards to be assigned
    cards = new Card[suit.length * value.length];

    //for loops to assign every value to every suit
    for(int i=0; i < suit.length; i++) {
        for (int k = 0; k < value.length; k++) {
            cards[cardLeft] = new Card(suit[i], value[k]);
            cardLeft++;
        }
    }
}
```

Figure 6: BlackJackGame method

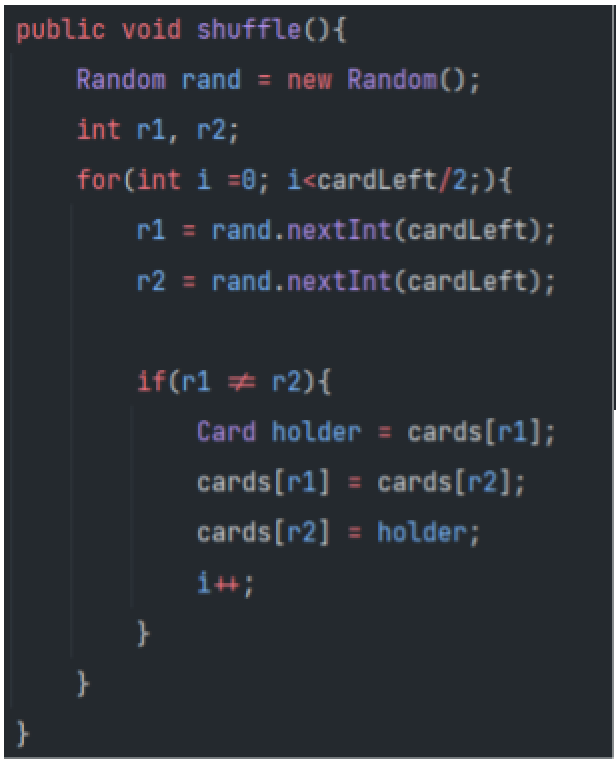


```java
public BlackJackGame(){
    cardLeft = 0;
    String[] value = {"Ace","Two","Three","Four","Five","Six","Seven","Eight","Nine","Ten","Jack","Queen","King"};

    cards = new Card[value.length];

    for(int i=0; i < value.length; i++) {
        cards[cardLeft] = new Card(value[i]);
        cardLeft++;
    }
}
```

Figure 7: BlackJackGame method are program slice on value variable

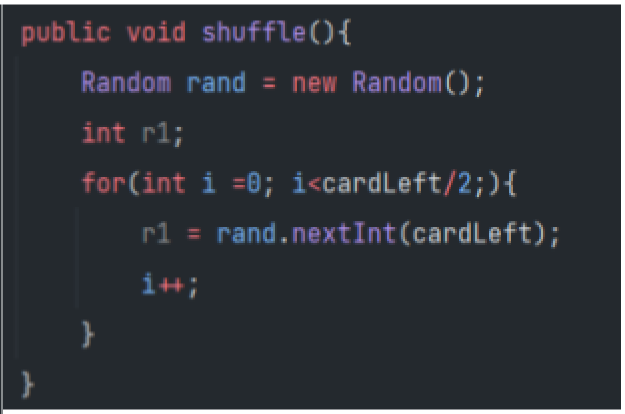
In example 3 there is a static program slice for the value variable within the BlackJackGame method. Since the primary focus is on the value of the cards, the suit variable is removed from the method. This means that the method can only assign a deck of cards that is only made of cards values which means the deck length will only be 13.

Example 4 - Static Program Slice



```java
public void shuffle(){
    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++;
        }
    }
}
```

Figure 8: shuffle method

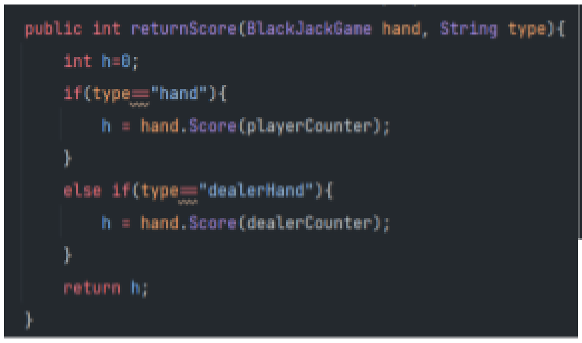


```java
public void shuffle(){
    Random rand = new Random();
    int r1;
    for(int i =0; i<cardLeft/2;){
        r1 = rand.nextInt(cardLeft);
        i++;
    }
}
```

Figure 9: shuffle method after Static slice on r1 variable

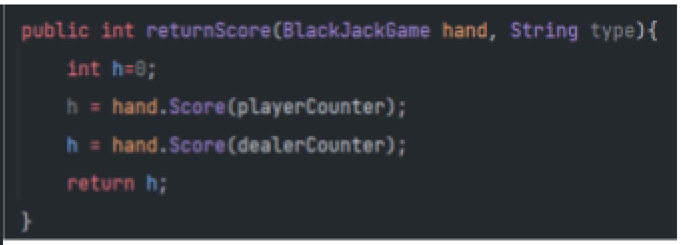
In example 4 there is a static program slice for the r1 variable within the shuffle method. Since the primary focus is on the first half of the deck, the second half is removed from the method. This means that the method can only shuffle the first half of the deck.

Example 5 - Static program slice



```
public int returnScore(BlackJackGame hand, String type){
    int h=0;
    if(type=="hand"){
        h = hand.Score(playerCounter);
    }
    else if(type=="dealerHand"){
        h = hand.Score(dealerCounter);
    }
    return h;
}
```

Figure 10: returnScore method



```
public int returnScore(BlackJackGame hand, String type){
    int h=0;
    h = hand.Score(playerCounter);
    h = hand.Score(dealerCounter);
    return h;
}
```

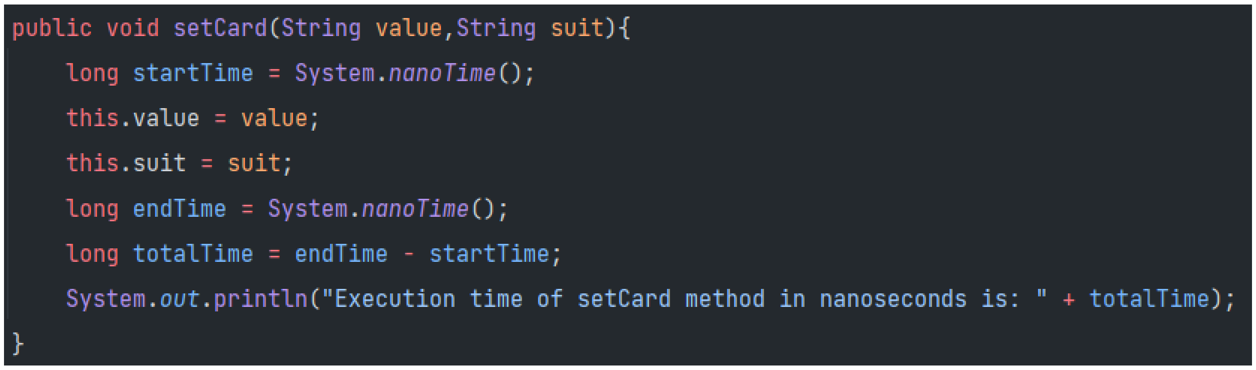
Figure 11: returnScore method after Static slice on hand variable

In example 5 there is a static program slice for the hand variable within the returnScore method. Since the primary focus is on the hand, the type of the hand is removed. This means that the method will return scores regardless of the type.

<u>Implementation of Dynamic Analysis</u>
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 12: setCard method with internal clock

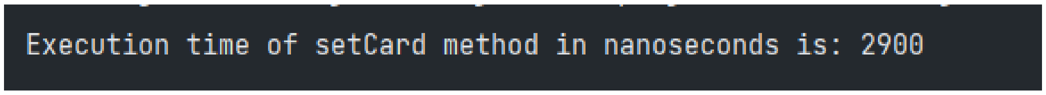

```
Execution time of setCard method in nanoseconds is: 2900
```

Figure 13: execution of setCard method

Example 2 - toString method execution time

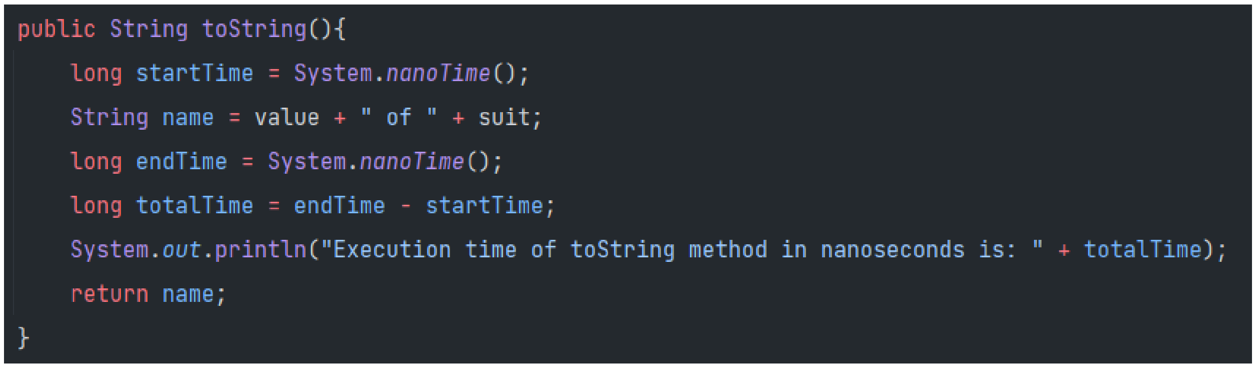

```
public String toString(){
    long startTime = System.nanoTime();
    String name = value + " of " + suit;
    long endTime = System.nanoTime();
    long totalTime = endTime - startTime;
    System.out.println("Execution time of toString method in nanoseconds is: " + totalTime);
    return name;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

Figure 14: toString method with internal clock

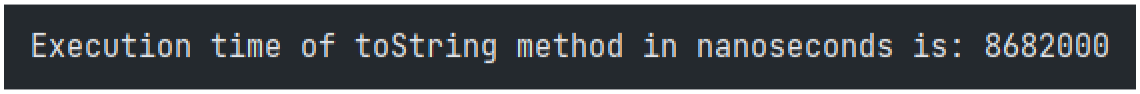

```
Execution time of toString method in nanoseconds is: 8682000
```

Figure 15: execution of toString method

Example 3 - shuffle method execution time

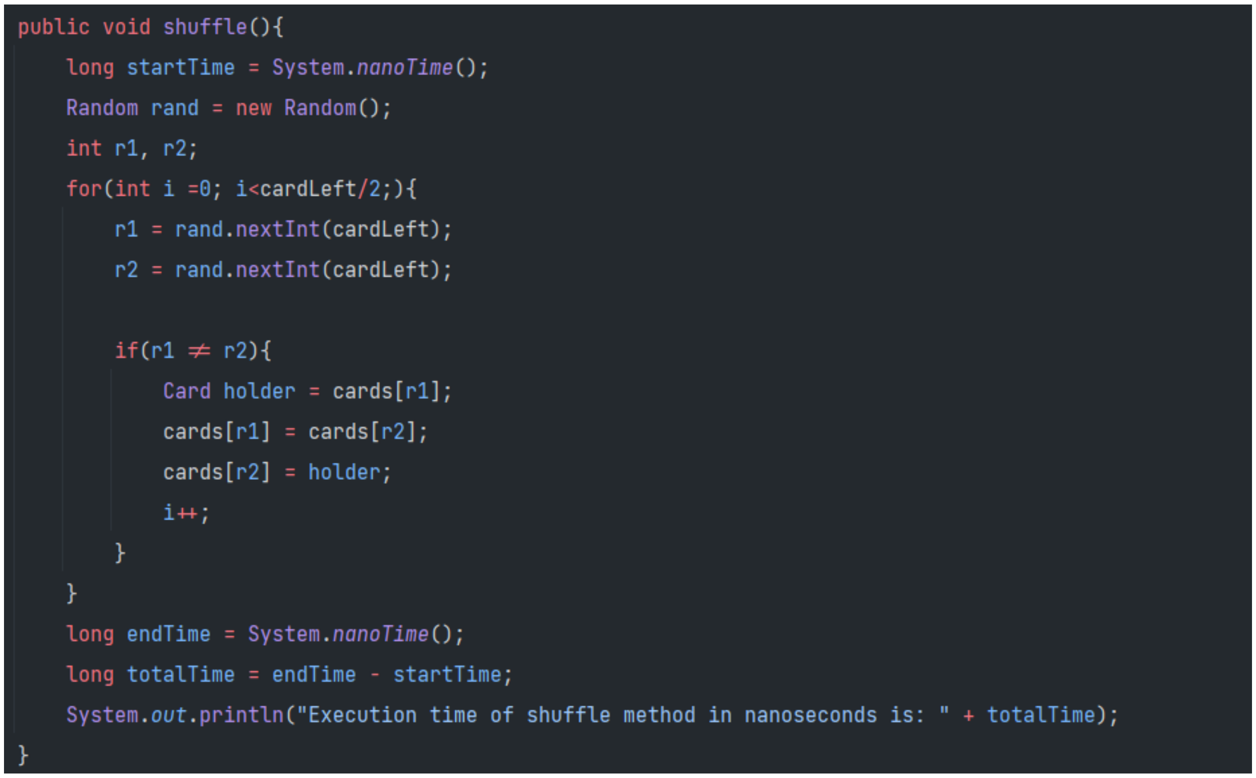
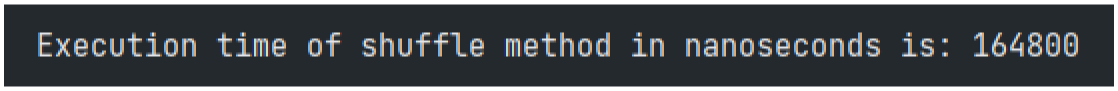

```
public void shuffle(){
    long startTime = System.nanoTime();
    Random rand = new Random();
    int r1, r2;
    for(int i =0; i<cardLeft/2;){
        r1 = rand.nextInt(cardLeft);
        r2 = rand.nextInt(cardLeft);

        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);
}
```

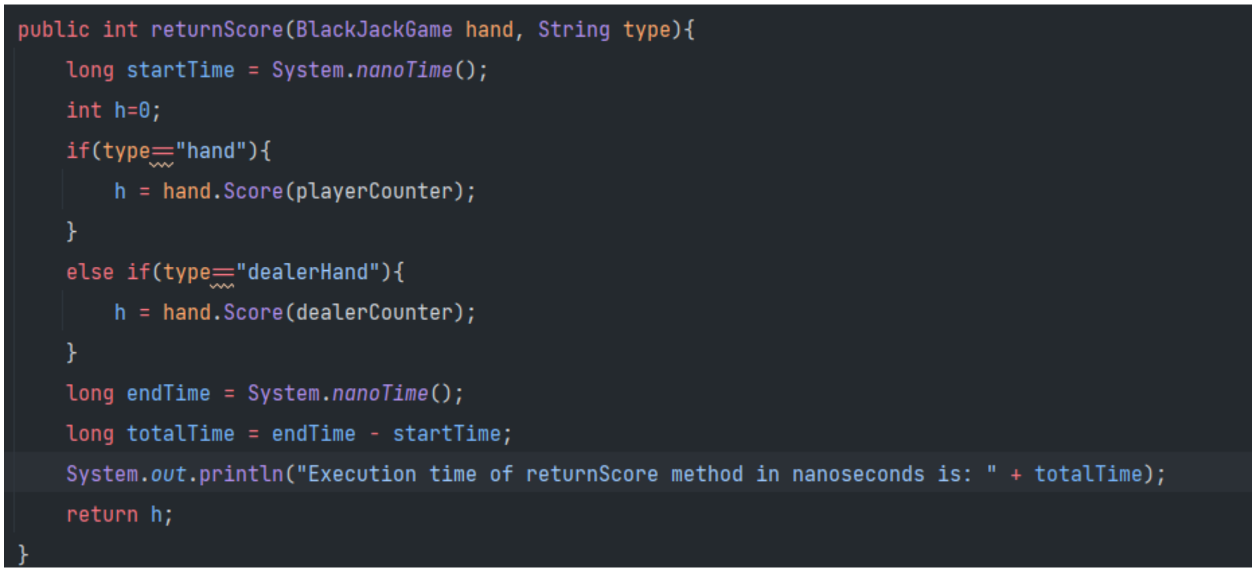
Figure 16: shuffle method with internal clock



```
Execution time of shuffle method in nanoseconds is: 164800
```

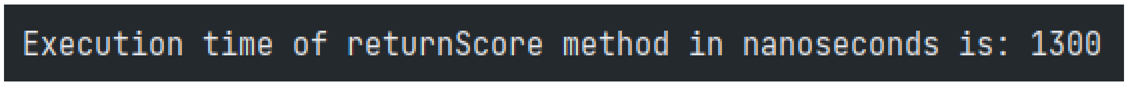
Figure 17: execution of shuffle method

Example 4 - returnScore method execution time



```java
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;
}
```

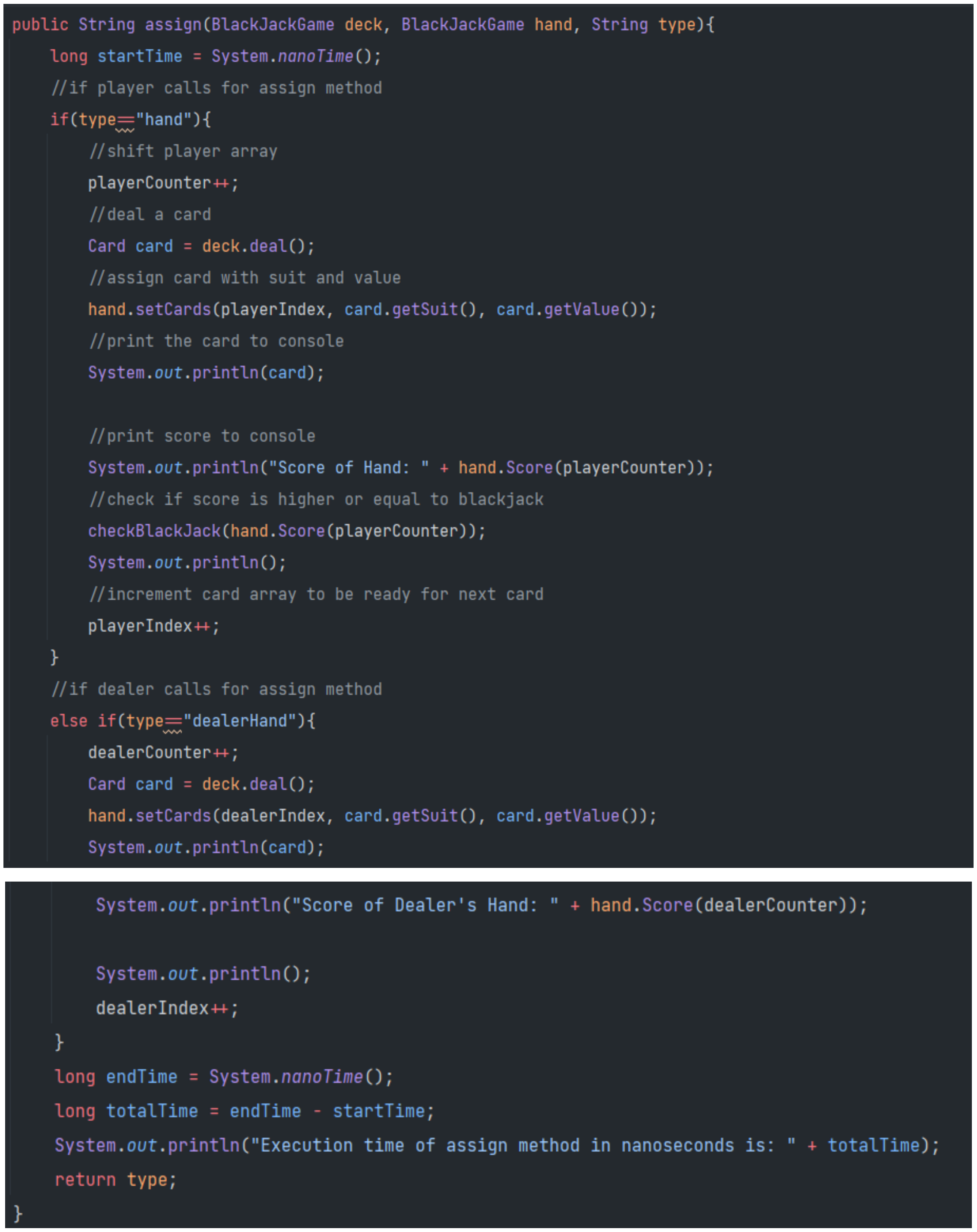
Figure 18: returnScore method with internal clock



```
Execution time of returnScore method in nanoseconds is: 1300
```

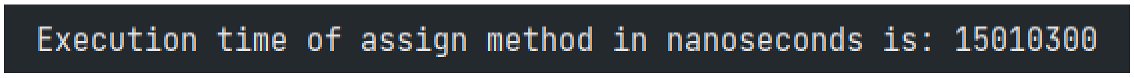
Figure 19: execution of returnScore method

Example 5 - returnScore method execution time



```java
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 20: assign method with internal clock



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

Figure 21: 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.