Lab 5 Walkthrough

Compilation Errors:

In Model.java

Error #1

```
private ArrayList<Card> faceUpUnmatchedCards = new ArrayList<>()

public Model() {
    reset();


void selectCard(int n) {
    card selectedCard = cards.get(n);
    if (selectedCard.isFaceUp) {
        return;
```

The first error in this screenshot is on the first line. We need a semicolon to indicate the end of statements in Java.

The second error is that our constructor public Model() does not have a closing curly-brace } to indicate where the constructor ends. We would place this closing curly brace on the line after "reset();".

Error #2

```
void selectCard(int n) {
    card selectedCard = cards.get(n);
    if (selectedCard.isFaceUp) {
        return;
    }

    if (faceUpUnmatchedCards.size() == 2) {
        Card card1_== faceUpUnmatchedCards.get(0);
        Card card2_== faceUpUnmatchedCards.get(1);
        if (!card1.isMatched) {
            card1.flip();
            card2.flip();
        }

        faceUpUnmatchedCards.clear();
        selectedCard.flip();
        faceUpUnmatchedCards.add(selectedCard);
    } else if (faceUpUnmatchedCards.size() == 1) {
        selectedCard.flip();
        faceUpUnmatchedCards.add(selectedCard);
        checkMatch();
    } else if (faceUpUnmatchedCards.isEmpty()) {
        selectedCard.flip();
        faceUpUnmatchedCards.add(selectedCard);
    }
}

private checkMatch() {
    Card card1_== faceUpUnmatchedCards.get(0);
    Card card2_== faceUpUnmatchedCards.get(1);
    if (card1.equals(card2)) {
        card1.setMatched();
        card2.setMatched();
    }
}
```

In both the selectCard and checkMath methods we have the statements "**Card card1 = = faceUpUnmatchedCards.get(0)**" and "**Card card2 = = faceUpUnmatchedCards.get(1)**". The problem here is that we're using the boolean operation = = which returns a boolean value if two values are equal. For example, 5 = = 6 returns false while 5 = = 5 returns true. Instead, we only want to use one equal sign to assign values to variables card1 and card2:

Card card1 = faceUpUnmatchedCards.get(0);
Card card2 = faceUpUnmatchedCards.get(1);

Error #3

```
private checkMatch() {
    Card card1 = faceUpUnmatchedCards.get(0);
    Card card2 = faceUpUnmatchedCards.get(1);
    if (card1.equals(card2)) {
        card1.setMatched();
        card2.setMatched();
    }
}
```

We need to specify a return type for each method we create. Since checkMatch does not return any values, we would write: private **void** checkMatch().

Error #4

```
void reset() {
    Color colors = {Color.GREEN, Color.BLUE, Color.ORANGE, Color.YELLOW, Color.CYAN, Color.RED, Color.GRAY, Color.MAGENTA};
    cards.clear();
    faceUpUnmatchedCards.clear();
    for (int n = 0; n <= NUM_PAIRS; ++n) {
        cards.add(new Card(colors[n]));
        cards.add(new Card(colors[n]));
    }
    Collections.shuffle(cards);
}
```

Over here we're trying to set the variable colors, which is of type Color (representing a single color), to a group of colors. Remember that when we want to store a group of objects or primitive data types, we tend to use arrays.

Single integer vs. group of integers:

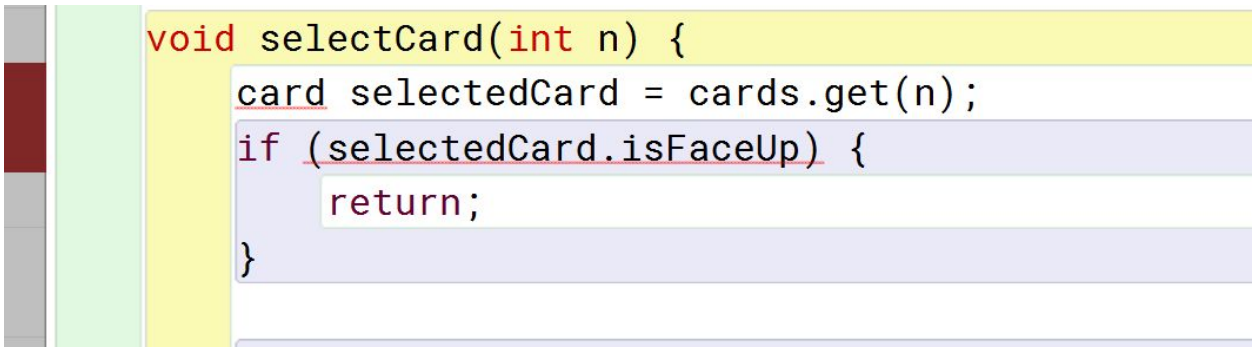
int x = 5;

int**[]** x = {1, 2, 3, 4, 5}

As you can see, the square brackets after int indicate that we are defining an array of integers. Similarly, if we want to declare an array of colors, we use the same square brackets:

Color[] colors = {Color.GREEN, Color.BLUE, Color.ORANGE, Color.YELLOW, Color.CYAN, Color.RED, Color.GRAY, Color.MAGENTA};

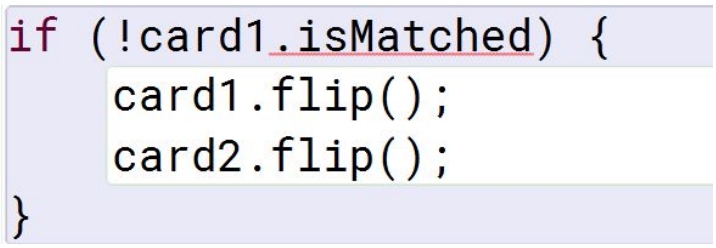
Error #5



```
void selectCard(int n) {
    card selectedCard = cards.get(n);
    if (selectedCard.isFaceUp) {
        return;
    }
```

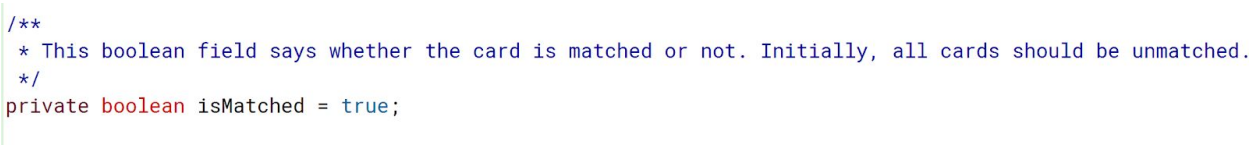
Remember that when we distinguished between primitive data types and objects, we said that objects had their first letter capitalized. For example, we use lower-case for type int but upper-case for type String. Therefore, we need to make the following change:
Card selectedCard = cards.get(n);

Error #6



```
if (!card1.isMatched) {
    card1.flip();
    card2.flip();
}
```

The red line indicates that we are not able to access the variable isMatched for card1. If we look at the Card.java file, we can see an explanation for why this is happening.



```
/**
 * This boolean field says whether the card is matched or not. Initially, all cards should be unmatched.
 */
private boolean isMatched = true;
```

Since isMatched is declared as **private**, we are unable to access the variable outside of the Card class. Therefore, to fix the issue, you would either have to change the keyword private to **public** or just remove the word "private."

In Card.java

Error #7

```
void flip() {
    return !isFaceUp;
}
```

The keyword **void** in front of our method name indicates that this method is not supposed to return any values. Therefore, returning !isFaceUp would not make sense. Recalling what the method is supposed to do, we want the card's current value for isFaceUp to be the opposite of what it was previously. For example, if the card was face up (isFaceUp = true) previously, we'd want it to now be face down (isFaceUp = false). We can do this using the symbol !, which basically means "not". So !true = false and !false = true. Therefore we would replace "return !isFaceUp" with:
isFaceUp = !isFaceUp;

Error #8

```
boolean equals(Card other) {
    if (color.equals(other.color)) {
        return true;
    }
}
```

So here, our code kind of makes sense, since we want to return the value **true** if the color of both cards match. However, we are not taking into account the case where the colors don't match. To fix this, we can either use an else statement or just return false after the if statement since returning a value takes you out off the method:

```
boolean equals(Card other) {
        if (color.equals(other.color)) {
                return true;
        }
        return false;
}
```

Some of you might have noticed that there's actually a better way of writing this:

```
boolean equals(Card other) {
        return color.equals(other.color);
}
```

Since the .equals method already returns a boolean value based on whether the values are equal are not, we can just return the result of this method directly.

Runtime Errors:

Error #9

In Model.java

```
void reset() {
    Color[] colors = {Color.GREEN, Color.BLUE, Color.ORANGE
    cards.clear();
    faceUpUnmatchedCards.clear();
    for (int n = 0; n <= NUM_PAIRS; ++n) {
        cards.add(new Card(colors[n]));
        cards.add(new Card(colors[n]));
    }
    Collections.shuffle(cards);
}
```

When we try to run the program after fixing all the compilation errors, we get an **ArrayIndexOutOfBoundsException** pointing to this line. If we look at the array of colors named **colors**, you see that it has a size of 8 (meaning it holds 8 colors). Given that arrays are zero-indexed, this means the last element of colors that we can access is at the index of 7. The value NUM_PAIRS in this case is 8, since we have 16 cards on the board and 16/2 = 8. Since we're looping as long as n <= 8 as we keep incrementing the value of n, we eventually get an index out of bounds exception since colors[8] does not exist. Therefore, we can fix this by changing the <= to an < sign, so the loop only goes up to index 7 which would be the last element of the array colors.

Logical Errors:

In Card.java

At this point, the game runs seemingly error-free, but clicking on each card doesn't do anything. We will now go through why.

Error #10

```
final Color color = Color.WHITE;

/**
 * This special method is the card constructor. It can be called like:
 *   new Card(Color.BLUE)
 * and it returns a new "instance" of the Card class.
 */
public Card(Color color) {
    // Fields can either be initialized where they're declared, like isFaceUp and isMatched
    // initialized in class constructors.
    color = this.color;
}
```

Here we have several problems. First of all, we don't want to set color equal to Color.WHITE. Instead, we want to leave setting the card's color to the constructor of Card. The constructor allows us to create an object of type Card with a certain color. For example:
Color c1 = new Color(Color.GREEN);
would create a new Card c1 with the color green. Inside the method, we want to assign this color Green to the color of the card. We do this by saying:
this.color = color;
The keyword **this** indicates that we're referring to a field within the class rather than the parameter we pass through the constructor. If this is confusing to you, just thing of the color in public Card(Color color) as our requested color.

Error #11

```
/**
 * This method is responsible for noting when a card has been matched.
 */
void setMatched() {
    isMatched = false;
}
```

The error here is that we are setting isMatched equal to false within the setMatched() method. If we read the comment, we see that the method is supposed to note that a card has been matched two cards of the same color have been flipped over. Therefore, instead of setting isMatched to false, we want to state:
isMatched = **true**;

Error #12

```
boolean isMatched = true;
```

Our FINAL error is that we are setting isMatched to true at the beginning of the game. As you might notice intuitively, this doesn't make sense, since all the cards have not been matched yet when the game starts. Therefore, to indicate that a card is unmatched at the beginning of the game, we instead say:
boolean isMatched = **false**;

Now your game should finally be working!