CREATE PT

Row 1 The program is a digital version of the card game Blackjack. The purpose is to allow a user to play the card game, combining their skill with luck to have fun attempting to win. The video includes a couple rounds of Blackjack, showing examples of wins and losses. You can see the user input as I decide to hit (get dealt another card) or stay (keep the current hand). The video also shows the code for the program, which I wrote using Eclipse and filmed using IntelliJ IDEA.
Row 2 The program is a digital version of the card game Blackjack. I developed a concept for the program that would divide the game into four classes, each assigned to a different function. One class to create the cards, another to create the deck, one pertaining to the hand, and another to play the game. Over time, I revised and refined this idea to include objects for the deck, the playerhand, and the dealer hand. This enables the program to use the same code to create both hands. Further iterative development was shown with the addition of several boolean variables to check the player type and who had won.
Row 3 A difficulty that arose in the development of this program was the unclear value of the card Ace. The issue was that in Blackjack, Ace is worth either 1 or 11 depending on the total sum of your hand (You will always choose for it to be worth as much as possible). I resolved this by creating a method in the Card class called swapAce which would change the value to 1. The value of Ace would be set to 11 by default and swapAce would be called if the total value of the player's hand is over 21. Another issue that appeared later on in the development of the program was that if the player lost during the hit method by having a hand value greater than 21 the program would continue to run. This was solved by creating a boolean called on that the rest of the program checked for using if statements. If false the rest of the program would not run.
Row 4

```
public static void loadHand(Deck d) {

    Hand dlh = new Hand(d.getDeck().remove(0), d.getDeck().remove(0), true);
    System.out.println("Dealer's Hand: ");
    dlh.printHand(bj);

    Hand plh = new Hand(d.getDeck().remove(0), d.getDeck().remove(0), false);
    System.out.println("Your Hand: ");
    plh.printHand(bj);

    if (plh.getSum() == 21 && (dlh.getSum() != plh.getSum())) {
        System.out.println("BLACKJACK! \nYou Win! ");
    } else if (dlh.getSum() == 21 && (dlh.getSum() != plh.getSum())) {
        System.out.println("\nDealer's Hand: ");
        bj = true;
        dlh.printHand(bj);
        System.out.println("The dealer has blackjack. \nYou Lose! ");
    } else {
        hit(plh, dlh, d);
    }
```

Row 5

The code in row four is an algorithm that uses mathematical and logical concepts because it calls several methods. It uses the objects “plh” and “dlh” to create a hand for the player and the dealer. It then prints and determines the value of each hand, comparing them against one another to see if either player has blackjack and wins (exactly 21 points of cards). If not, the algorithm continues and calls the “hit” method. The purpose of this algorithm is to load both hands from the deck, which is done by taking the first value of the arraylist “Deck” and then removing it multiple times.

Row 6

The code shown in row four implements an algorithm that includes multiple nested smaller algorithms. Together, the algorithms allow the program to create and fill two separate hands for the player and dealer, then compare them and allow the player to “hit.” Several algorithms in the method such as “printHand”, ” hit,” and “getSum” call additional methods. In addition, these functions apply mathematical and logical concepts in that “printHand” contains an if/else ladder nested inside a for loop, “getSum” contains a for loop, and “hit” contains user input and an if/else statement nested inside a do/while loop nested within an if/else statement. Independently, the purpose of the hit method in the main class is to allow the player to deal themselves another card until they “bust” (go over 21) or stay, determined by user input from the keyboard. This is done through sequential operations: the boolean playerType determines if the function runs for the player or dealer. If it is the dealer, the algorithm will add a card to the dealer’s hand until the total value is greater than or equal to 17. If it is the player, it will add one card to the player’s hand. After running either function it prints the name of the added card.

Row 7

```
public void printHand (boolean bj){

    for (int i=0; i<hand.size(); i++){
        if (playerType && i==0 && bj==false){
            System.out.print("[?] ");
        }
        else{
            System.out.print("[" + hand.get(i).getName()+ "] ");
        }
    }
    System.out.println("\n");
}
```

Row 8

The selected above code in row seven shows a student-developed abstraction that manages the complexity of the program. This is shown in that it consists of a “for” loop that prints the values of each card in hand. The abstraction is the method itself; Whatever calls it does not need to know how the hand is printed. This allows for less complexity in the program because there are fewer lines of code in the main class. This method is also an example of abstraction because it can be used to print the hand of either the dealer or the player, cutting down further on excess code. Moreover, because each function in the program (such as this one) has a specific purpose, debugging becomes easier.