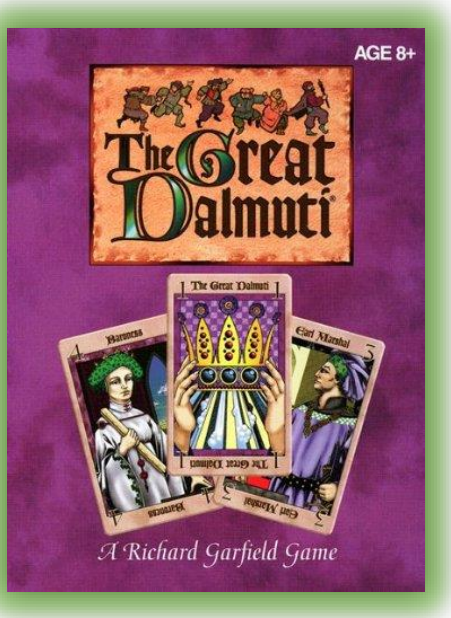
Bluegrass Community and Technical College

Programming Requirements Document

Dalmuti Deal

Narrative Description



There is a card game entitled ***The Great Dalmuti.*** This is a variation of a popular game that dates to the Middle-Ages. ***The Great Dalmuti*** uses a non-standard 80-card deck. The deck consists of 13 types of cards. Each type has a different rank (think class in society). The table below illustrates how many of each type of card exists in a Dalmuti deck.

For example, there is only 1 great Dalmuti card,11 Stonecutter cards, and 2 Jester cards.

For this assignment, you do not need to know how to play the game. However, if you are interested, you can find descriptions online.

The Great Dalmuti – Card Layout		
Rank of Card	Name of Card	Count of this type of card in a deck
1	Dalmuti	1
2	Archbishop	2
3	Earl Marshal	3
4	Baroness	4
5	Abbess	5
6	Knight	6
7	Seamstress	7
8	Mason	8
9	Cook	9
10	Shepherdess	10
11	Stonecutter	11
12	Peasant	12
13	Jester	2

For this assignment, you are not programming the game but rather using the card deck to practice:

- Selection statements
- Repetition statements
- Generating pseudo-random numbers

NOTE: We have not covered arrays and as such you should not use an array for this assignment. We have not covered user-defined methods (static methods) and as such should not be used in this assignment. This project will illustrate the need for these constructs and you may see this assignment again in the future.

Part 1:

This program should:

- Display appropriate titles to the user when the program begins
- Display the standard Dalmuti deck to the user. For example. the output will be:

```
Card 1:   Dalmuti

Card 2:   Archbishop
Card 3:   Archbishop

Card 4:   Earl Marshal
Card 5:   Earl Marshal
Card 6:   Earl Marshal

Card 7:   Baroness
Card 8:   Baroness
Card 9:   Baroness
Card 10:  Baroness

     etc.

Card 67:  Peasant
Card 68:  Peasant
Card 69:  Peasant
Card 70:  Peasant
Card 71:  Peasant
Card 72:  Peasant
Card 73:  Peasant
Card 74:  Peasant
Card 75:  Peasant
Card 76:  Peasant
Card 77:  Peasant
Card 78:  Peasant

Card 79:  Jester
Card 80:  Jester
```

- Notice that the 80 cards are numbered 1 through 80. A for-next loop would be perfect for generating this list. In fact, you are required to a for-next loop for this portion of the program. The control variable for the loop can run from 1 – 80.
- Inside the loop, you can use the card number to determine what should be printed. For example, a 1 will display a Dalmuti card. A value of 2 through 3 would display an Archbishop card. A value of 4 through 6 would display an Earl Marshal card. Use a switch statement to do this. You are being assessed on using a switch statement with break clauses.
- In a switch state, you can specify a range of values by listing each value in the case clause. For example, for a range 2 through 3, you can use:

```
case 2: case 3:
        //some statements here
        break;
```

- Test and get Step 1 working before completing Step 2.

Part 2:

Randomly "deal" two cards from a Dalmuti deck.

- The two cards:

1. The first card is for the user. You can deal a card for the user by generating a random number between 1 and 13. This is the "rank" of the card which is displayed in the table on page 1.

   Once a rank is generated and stored in a variable. Use a selection statement to display the "name" of the card. For example, if a 6 was generated the output would be: Knight

   NOTE: If this were a real game, you would generate a number in the 1-80 range since the odds are not the same for each card rank (for example, there is only 1 Dalmuti but there are 12 Peasants).

   This time, use an if-else structure to display the correct name for the generated card. You will be assessed on using the if-statement.

2. The second card is for the computer. Generate a random number between 1 and 13. Perform error-checking on this second card.

   Since there is only 1 Dalmuti card in the deck, it is possible that the two random numbers generated were both 1 (a Dalmuti card). If that is the case, you need to generate the computer's card again. For all other values, it is fine if the two cards are the same because there are multiples of all other cards.

   When performing error-checking, such as validating user input or validating a variable's value, a while-loop can be used to repetitively receive/generate a new input/value until the data is valid.

   For example, if a user is to key a number between 1 and 12, the following loops validates the data and if an error was made, will ask the user to key a new value:

```
while (number < 1 || number > 12)
{       System.out.println("Enter a number in the range [1-12] ");
        number = scan.nextInt();
}
```

   Use a similar idea to generate a new card for the computer if the user and computer cards are both a 1.

   Once the computer's card is valid, display the card in text format. For example, if a 6 was generated the output would be: Knight

   You can use a if-else structure as above, if you wish. You will notice that you have duplicate code in your program (once for the user's card and once for the computer's card). That is fine for now. We will learn how to use static methods soon to eliminate the duplication.

- Assuming a Dalmuti card is the “best” card in the deck and a Jester is the “worst” card in the deck. Determine and display a winner based upon the card values. If the cards are the same value, declare a “tie”.

Software Requirements

- ☐ The user interface (what the use sees and how they interact with the program) is intuitive, clear, and easy to use.
- ☐ The card deck is displayed as outlined in the narrative.
- ☐ Selection structures are implemented as outlined in the narrative.
- ☐ Repetition structures are implements as outlined in the narrative.
- ☐ Error-checking is performed on the second card.
- ☐ A winner or tie is correctly determined and clearly communicated to the user.
- ☐ Arrays are not used.
- ☐ Static methods and/or user-defined methods are not used.
- ☐ Previous standard requirements are included (documentation, selection of proper data types, constants, etc.).

Security Considerations

The correct data type is selected for numeric values. Error-checking is performed on user input.

Special Notes

None provided for this assignment.