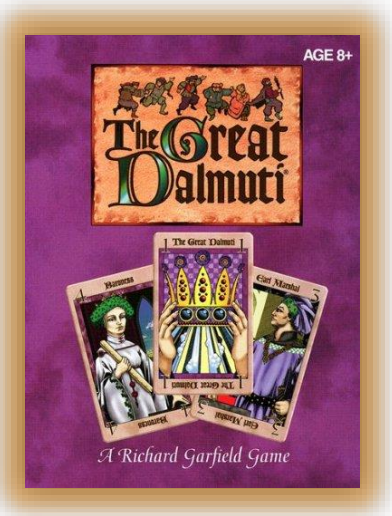
Java Programming – Dalmuti Class

Dalmuti Game



There is a card game entitled ***The Great Dalmuti.*** This is a variation of a popular card game that dates back 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. The table below illustrates how many of each type of card exists in a Dalmuti deck. 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	Number 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

<u>This assignment is a review of creating Java classes. You will do very little coding for this.</u>

Reading this assignment is MUCH worse than completing it. Read this handout thoroughly before beginning. **This is NOT a hard assignment. You will create a class which describes a single card in the Dalmuti deck.**

Scenario: You have been assigned to a team of 3 programmers to develop a computerized version of playing ***The Great Dalmuti***. Your team members are Angela, David, and you. Your team divides the work and here are the assignments:

1. You will create a **DalmutiCard** class which represents a single card from a Dalmuti deck. Details are on the second page. **Good news: This is the smallest part of the game to code!**

2. David will create a **DalmutiDeck** class. It creates an array of 80 **DalmutiCard** objects. This forms the Dalmuti deck. **Good news: David has completed his part and given it to you. You will not have to complete coding for this class.** You will find a copy of **DalmutiDeck.java** in the same ZIP file that contained this PDF programming assignment. You are to use **DalmutiDeck.java** but you are not permitted to change it.

3. Angela will create a **CardGame** class that "plays" the game. **Good news: Angela has completed her part and given it to you to use. You will not have to complete any coding for this class.**

4. At this point, all your team wants to test is creating the card deck, shuffle the cards, and deal the cards to the players (4-8 players). For this assignment, your team will not test the entire game.

5. After your part is ready to test, you should simply

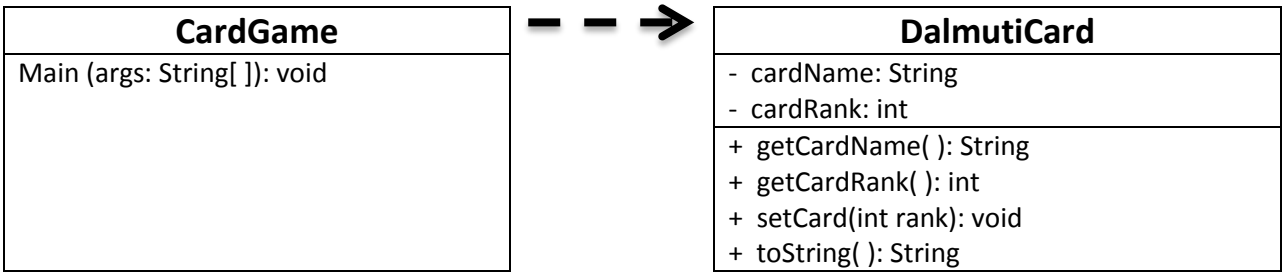
    a. Compile **DalmutiCard** – this what you write
    b. Compile, **DalmutiDeck** - this is what Angela gave you (in the ZIP file)
    c. Compile **CardGame** – this is what David gave you (in the ZIP file)
    d. Run **CardGame**

    Any errors you have will be a result of your work. Do not change the other classes. If you have questions, email your instructor.

Like many real-world examples when programmers work in teams, you need to follow program specifications so that classes work together.

Here are the specifications for your DalmutiCard class:

Below is a UML Class Diagram for the **DalmutiCard** class



(+ represents the public modifier and – represents the private modifier)

1. The name of the Java class must be **DalmutiCard** and saved in a file named **DalmutiCard.java.**

2. Create two instance variables (also called fields or data members). Use the names shown in the UML class diagram (also in bold below):

    a. **cardName** – a string to hold the description of a single card (i.e. Dalmuti, Archbishop, etc.)
    b. **cardRank** – an integer to hold the card's rank (a value from 1-13)

3. Create a constructor with 1 argument (the rank of the card you are creating -- an integer value from 1-13)

    a. Constructors are not typically illustrated in a UML Class Diagram. Constructor names MUST match the name of the class (**DalmutiCard** in this case). Your constructor will have one parameter – the rank of the card to create.

b. Make sure the parameter passed in has a value between 1 and 13. If it does, use the parameter value to update **cardRank**. Also, update **cardName** with an appropriate description (using the table on page 1). If the parameter value passed in was invalid, assign 0 to **cardRank** and "Unknown" to **cardName**.

4. Create an accessor method (also called a getter) for each instance variable. Use the method names in bold below (and as shown in the UML diagram):

   a. **getCardRank**() – returns the current value of cardRank
   b. **getCardDescription**() – returns the current value of cardName

5. Create a mutator method (also called a setter) called **setCard( ).** As indicated in the UML Class diagram on the previous page, this method has one integer argument which represents a new card rank. Like the constructor, validate the parameter passed in. If valid, update **cardRank** and **cardName**. If invalid, do not update anything. *FOR EXTRA CREDIT: After you have completed this method, look for duplicate code in the constructor and this method and eliminate duplication.*

6. Create a **toString**() method which returns a nicely formatted string of data from the **DalmutiCard** class. Sample output might be something like the examples below:

   1 – Dalmuti
   5 – Abbess
   13 – Jester

For grading it is quicker if your instructor as all Java files needed to run and test your solution. Zip together **DalmutiCard.java**, **DalmutiDeck.java**, and **CardGame.java** (no other files). Submit the zipped file in Blackboard for grading. ***DO NOT SUBMIT .class files or any other folders/files created by the IDE you are using.***