EXTRA CREDIT # 2: The Nine Card Game

Tutorial: Enum Datatypes:¹

You will be using methods of the library class enum. Read the following theory:

1. Identify the location of the enumeration value within the enumeration: Each value gets an index automatically starting with 0 for the first value within the enumeration. There is a method called ordinal() of the enum class that returns the position as an int (SEE THE API). i.e.

   if m is an enum datatype then
   
   int position = m.ordinal()

   if m has the value FORD then position will receive the value 2 for the enum defined with the values {GM, HONDA, FORD, TOYOTA}

2. Two enum values can be compared by using the equals method of the enum class i.e suppose object a1 and a2 have an enum attribute of type Model and there is an accessor method getModel() then the Model values for the two objects are compared using:

   if(a1.getModel().equals(a2.getModel()))
   {
       //DO SOMETHING
   }

3. Enum also has a useful method called values() that returns an array of strings containing all of the values of the enum in the order they are declared. This method is commonly used in combination with the for-each construct to iterate over the values of an enum type. For the enum datatype “Cars” declared with the values {GM, HONDA, FORD, TOYOTA}, we can iterate and print out the values using the following code:

   for (Cars d : Cars.values()) {
       System.out.println(d);
   }

   We can also print out specific values of the enum using values(). To print out the second value of the enum datatype “Cars”, we can use the following statements:

   String val=Cars.values()[1];
   System.out.println(“The enum value at the second position is ” + val);

¹ Tutorial on Enum datatypes contains material by George Koutsogiannakis
4. One technique for converting a String to an enum data type is to compare every possible enum value to the received String using if/else if and then we set the enum value. This is a time consuming technique, especially if there are quite a few possibilities in the enum data. An easier way is to use the method (see API)

```java
static <T extends Enum<T>> T valueOf(Class<T> enumType, String name)

Returns the enum constant of the specified enum type with the specified name.
```

By using this method we avoid a lot of lines of code since only one line is required then

Example: Suppose the enumeration class is

```java
public enum MyEnum {value1,value2,value3};
```

Suppose we are reading a text file where those values are expected to be read as Strings. Suppose token `tok` is the token that needs to be converted to the proper enum object value.

The technique we previously discussed would require the code:

```java
MyEnum me=null;
if(tok.equals("value1"))
    me=MyEnum.value1;
else if(tok.equals("value2"))
    me=MyEnum.value2;
else if(tok.equals("value3"))
    me=MyEnum.value3;
```

However, an easier way is to substitute the above lines of code with the following two lines of code:

```java
MyEnum me=null;
me=MyEnum.valueOf(tok);
```

That is it. We are saving time and code lines.

The only additional requirement is that we need to add two more catches (it can be with the same try as the IOException

```java
Try{
}
catch (IOException ioe){
    System.out.println(ioe.getMessage());
}
```
`catch(IllegalArgumentException ia){
    System.out.println(ia.getMessage());
}
`catch(NullPointerException np){
    System.out.println(np.getMessage());
}

**Tutorial: Random Number Generator**

The Math class in Java has the random() method to generate a random double value with a positive sign, greater than or equal to 0.0 and less than 1.0. We can use this to generate random int or float or long values within any range. Notice that the range = \( (\text{max} - \text{min} + 1) \). You can use the formula “\((\text{int})(\text{Math.random()} \times (\text{range}) + \text{min})\)” to generate random integers between min(inclusive) and max (inclusive). See the following example to generate a set of random int values with a min value of 7 and max value of 15.

```java
public static void main(String [] args){
    //generate 10 random integers between 7 (min value) and 15 (max value)
    int min=7;
    int max=15;
    int rndVal;
    int range = max-min+1; // this is the range of the values
    for(int i=0;i<10;++i){
        rndVal= (int)(Math.random()*(range)+min); //See how the values at typecast to int only at the end. *Important*
        System.out.println("Random number is " + rndVal);
    }
}
```

**The Nine Card Game:**

- In the nine cards game, the player or user is dealt 9 cards. The house then deals itself 9 cards. The 9 cards of the player are then compared with the cards of the house one by one. In each comparison or round, the card of the higher rank wins. Since the player and the house are dealt cards from different suites, they both can get the same cards. If player wins in more rounds than the dealer, then the player wins. The game can also end up in a tie.

- The cards have the following signs (SPADE, CLUB, DIAMOND, HEART) and values (ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING).
- When comparing two cards, we first compare their signs. The signs are arranged in the following order {SPADE, CLUB, DIAMOND, HEART}, with HEART greater than the other three signs, DIAMOND greater than CLUB and SPADE, and so on. If both the cards have the same sign, then we compare their values. The values are arranged in the following order {ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING} with KING having the highest value, followed by QUEEN, then JACK and so on. For example,

<table>
<thead>
<tr>
<th>Player’s Card</th>
<th>House’s card</th>
<th>Who has the higher rank card?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPADE/THREE</td>
<td>HEART/TWO</td>
<td>House</td>
</tr>
<tr>
<td>CLUB/NINE</td>
<td>CLUB/TWO</td>
<td>Player</td>
</tr>
<tr>
<td>DIAMOND/ACE</td>
<td>CLUB/KING</td>
<td>Player</td>
</tr>
</tbody>
</table>

- The first card of the player is compared with the first card of the dealer, the second card of the player with that of the dealer, and so on. The person who win in the most number of comparisons or rounds wins. If the player and dealer wins in equal number of rounds, it is a tie.

**Programming Task:**

- Your task is to write a Java program that will implement the nine card game. Some of the java classes that you will be needing has been provided with this assignment. Please go through them. The places where you need to add your code has been marked with a “Task” comment.

**You will download and modify the given source files from the course website:**

- Card.java: implements a service class called “Card” that stores the sign and value of a card using enum types CardSign and CardValue, respectively. You will need to modify this file.
- CardSign.java: has the declaration of the CardSign enum datatype. No modification needed.
- NineCards.java: has the client class. Please go through the comments. It should be self-explanatory. You will need to modify this file.

- Here are the specific tasks (also listed as comments /*Task......*/ in the source code). Tasks can be completed in any order.
  - **Task 1**: declare an enum type called CardValue with the following values {ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING}. This should be done in a separate java source file and placed in the package carddeck.service.classes;
  - **Task 2**: Import necessary user defined classes in the Card.java and Ninecards.java
o **Task 3**: Implement the class method `public int compareCards (Card aCard)` in `Card` Java. This method will compare the ranks of two cards and return a value as described by the comments.

o **Task 4**: Implement the code in the main method of the `Ninecards` class, that will take the values read from the text file, create `Card` objects and populate the array `yourHand`. `yourHand` is an attribute of the `Ninecards` class.

o **Task 5**: Implement the class method `generateHouseHand()` in `Ninecards` class. This method will use a random generator to generate signs, and values of the cards that are dealt to the house.

o **Task 6**: Implement the class method `getScore()` in `Ninecards` class. This method will compare the cards of the house and player and provide a score. If the score is zero, it means the game is a tie. If the score is positive, then the player has won the game.

---

**Make sure your code compiles and generates the required output. See the sample output below for reference.**

**You should demo your code to the TA in the class and upload it on the BB.**

**SAMPLE OUTPUT**

```console
>java library.client.classes.Ninecards hand.txt
Select an option:
Type "P" to play a round of Nine Cards
Type "Q" to Quit
P
Comparing my DIAMOND:ACE with the house's DIAMOND:TEN
House's card wins
Comparing my SPADE:FIVE with the house's HEART:SIX
House's card wins
Comparing my CLUB:TWO with the house's CLUB:JACK
House's card wins
Comparing my DIAMOND:KING with the house's CLUB:SEVEN
My card wins
Comparing my HEART:QUEEN with the house's DIAMOND:FOUR
My card wins
Comparing my SPADE:JACK with the house's HEART:QUEEN
House's card wins
Comparing my CLUB:NINE with the house's DIAMOND:TEN
House's card wins
```
Comparing my DIAMOND:FOUR with the house's CLUB:TEN
My card wins
Comparing my CLUB:TEN with the house's DIAMOND:ACE
House's card wins
House Wins :-(
Select an option:
Type "P" to play a round of Nine Cards
Type "Q" to Quit