2217 Java Software Dev 2
Project 9 – Generic Deck

Purpose

Create a generic class and method for code flexibility and reuse.

Goal

Write a generic class that represents a deck of cards that can be used in a variety of applications.

Project Overview

Many software applications rely on a deck of cards, whether those cards are standard playing cards, Uno cards, flashcards, or role-playing (RPG) cards. This diversity makes it an ideal scenario for using a generic class. By designing the deck to work with an unspecified card type, we preserve flexibility and reusability.

As a review, generic classes allow us to handle a range of data types while preserving type safety. They enable early detection of type-related errors during compilation, rather than at runtime. You've seen generics in action before, most notably when working with the `ArrayList` class.

In this project, your goal is to design and implement a class that defines the fundamental operations of a deck, such as shuffling, drawing a card, and resetting the deck without committing to a specific card type. Though we won't integrate the deck into a playable game, we will write code to test it with different card implementations.

Program Requirements

Download the starter project provided with the assignment. It includes all the necessary files. You are required to complete the implementation tasks marked with `TODO:` comments. **You must use the starter project as-is**—projects built from scratch will not be accepted.

Cards have different behaviors based on the application in which they are used. However, every card object must be able to display a description of itself. They all must have this common behavior. We will use an interface named `ICard` to represent this behavior requirement. Each card type must realize the `ICard` interface.

The `Deck` class represents a collection of card objects, but it does not assume what type of cards are being used or how they behave. To keep the class flexible and reusable, we define it using a generic type parameter **that must realize the `ICard` interface**.

Since the `Deck` class doesn't construct the card objects itself, a fully built list of cards, along with a description of the cards, must be supplied to its constructor. For example, this statement creates a deck of Uno cards:

```
Deck<UnoCard> unoDeck = new Deck<>(unoCardList, "Uno");
```

The argument `unoCardList` is a collection of all the cards needed for a complete Uno deck.

The deck needs the following behaviors:

`shuffle`	Use the `Collections.shuffle` method to put the cards in a random order.
`draw`	Returns a card from the collection. Throws `NoSuchElementException` if the method is called and the collection is empty.
`deal`	Receives a number as a parameter. Returns a list of cards containing the requested number of cards. Throws `IllegalArgumentException` if the method is called and the collection does not contain enough cards.
`reset`	Resets the collection to its original constructed state and then shuffles the cards. This means a clone of the collection of cards sent to the constructor must be kept.
`size`	Returns the number of cards in the deck.
`getDescription`	Returns the description of the cards in the deck that was sent to the constructor.

To test our Deck class, we'll use two different card types. The first is `StandardCard`, which models a traditional playing card, like those found in games such as Rummy, Hearts, Poker, and Solitaire. This class must implement the `ICard` interface.

Within `StandardCard`, we define two enumerated types: one for the suit (e.g., Hearts, Spades) and one for the rank (e.g., Ace, King, Seven). The class requires two instance variables, one for the suit and one for the rank. Both are declared using their respective enumerated types rather than `String`. The constructor receives parameters for the card's suit and rank.

AI Tip: Research how and why we use `enum` types in Java. The `enum` type has several behaviors that will be very useful in this project.

A `StandardCard` object must be able to tell us its suit and rank. It must also implement the `display` method to satisfy the requirement established by the `ICard` interface.

The second is `UnoCard`, which models a card used in the game Uno. This class must implement the `ICard` interface.

Within `UnoCard`, we define two enumerated types: one for the color (e.g., red, green) and one for the value (e.g., one, two, skip, reverse). The class requires two instance variables, one for the color and one for the value. Both are declared using their respective enumerated types rather than `String`. The constructor receives parameters for the card's color and value.

An `UnoCard` object must be able to tell us its color and value. It must also implement the display method to satisfy the requirement established by the `ICard` interface.

To construct a deck of cards, we must provide a list containing all the cards used in the game. We will separate this logic from the rest of the code and place it in a static method in "factory" classes. (This design decision will make more sense to you if/when you study design patterns.)

Creating a list of standard playing cards is a straightforward task: we need one card for each unique combination of suit and rank. Because the Suit and Rank enumerations in the `StandardCard` class are declared public, they are directly accessible from within the factory class.

The Uno card factory is a bit more complex. *AI Tip:* Research the cards needed for a complete Uno deck.

Since we are not writing a working game, we will add testing logic to the `main` method. To avoid repeating code and to practice writing a generic method, we will add a static method named `testDeck`. The generic type for the method must realize the `ICard` interface.

The `testDeck` method should receive a list of cards, a `String` description of the cards, and the size of a starting hand. The method should do the following:

1. Create a deck using the list of cards and the description.
2. Shuffle the deck.
3. Deal a starting hand from the deck.
4. Display the cards in the starting hand.
5. Draw and display a card from the deck.

Once you have completed your code, the output should look something like this:

```
Testing the generic deck implementation.

Dealing 5 standard cards:
        KING of HEARTS
        ACE of DIAMONDS
        JACK of DIAMONDS
        NINE of CLUBS
        EIGHT of CLUBS
Draw 1 standard card: FIVE of CLUBS

Dealing 7 Uno cards:
        YELLOW NINE
        YELLOW ONE
        RED ZERO
        YELLOW TWO
        GREEN FIVE
        GREEN SEVEN
        RED FIVE
Draw 1 Uno card: GREEN REVERSE
```

Expectations

The program is expected to compile successfully, include identification comments at the top, and produce output that exemplifies good grammar and spelling. Failure to meet these standards may result in point deductions.