

Collection. Generics. Enumeration types

1. Design a class that acts as a library for the following kinds of media: book, video, and newspaper. Provide one version of the class that uses generics and one that does not. Feel free to use any additional APIs for storing and retrieving the media.

2. A predicate is a boolean-valued function with one parameter. Some languages use predicates in generic programming. Java doesn't, but this exercise looks at how predicates might work in Java.

In Java, we could use "predicate objects" by defining an interface:

```
public interface Predicate {
    public boolean test(Object obj);
}
```

The idea is that an object that implements this interface knows how to "test" objects in some way. Define a class `Predicates` that contains the following generic methods for working with predicate objects:

```
public static void remove(Collection coll, Predicate pred)
    // Remove every object, obj, from coll for which
    // pred.test(obj) is true.

public static void retain(Collection coll, Predicate pred)
    // Remove every object, obj, from coll for which
    // pred.test(obj) is false. (That is, retain the
    // objects for which the predicate is true.)

public static List collect(Collection coll, Predicate pred)
    // Return a List that contains all the objects, obj,
    // from the collection, coll, such that pred.test(obj)
    // is true.

public static int find(ArrayList list, Predicate pred)
    // Return the index of the first item in list
    // for which the predicate is true, if any.
    // If there is no such item, return -1.
```

3. A `LinkedList` can be used as a queue by using the `addLast()` and `removeFirst()` methods to enqueue and dequeue items. But, if we are going to work with queues, it's better to have a `Queue` class. The data for the queue could still be represented as a `LinkedList`, but the `LinkedList` object would be hidden as a private instance variable in the `Queue` object. Use this idea to write a generic `Queue` class for representing queues of `Objects`. Also write a generic `Stack` class that uses either a `LinkedList` or an `ArrayList` to store its data.

4. Using generic collection types, write a program which shuffles a deck of 52 cards, deals all the cards to 4 players (each gets 13 cards), and prints the cards on each player's hand to standard output. The deck of cards should be of type `<Type<Card>>`, where `Type` is an appropriate container type.

Hints. An enumeration is a fixed set of constants. In Java an `enum` is a class that represents an enumeration. You use enumerations whenever you have a set of items whose values are known at compile time. Here are two enumeration types:

```
public enum Rank {
    DEUCE, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE
}

public enum Suit {
    DIAMONDS, CLUBS, HEARTS, SPADES
}
```

The Rank enum is saved in a source file named Rank.java and the compiled bytecode is in a file named Rank.class. The compiler generates a special method named values when it generates the class for your enum declaration. The values method returns an array of the enum values.

You may define a Card class for defining all cards in a deck. With this class you can write:

```
for (Rank rank : Rank.values()) {
    cards[i++] = new Card(rank, suit);
}
```

The static valueOf method, inherited from java.lang.Enum, is used to convert a String value to its corresponding enum value:

```
Rank rank = Rank.valueOf("FIVE");
```

You can define a deck with the following class:

```
import java.util.*;
public class Deck {
    private static Card[] cards = new Card[52];
    public Deck() {
        int i = 0;
        for (Suit suit : Suit.values()) {
            for (Rank rank : Rank.values()) {
                cards[i++] = new Card(rank, suit);
            }
        }
    }
}
```

If you ever need the integer value of an enum element, you can use the static method ordinal inherited from java.lang.Enum.

```
for(Suit s : Suit.values()) {
    System.out.print(s.ordinal() + " ");
}
```

The output is

```
0 1 2 3
```

The name method is inherited from Enum and returns the corresponding element name.

An enum can declare methods and constructors, as well as other fields that are not a part of the enumerated list of elements. The enumeration list must be declared first in the enum, followed by a semicolon.

```
public enum Suit {
    DIAMONDS, CLUBS, HEARTS, SPADES;
    public String toString() {
        return this.name().toLowerCase();
    }
}
```