

Problem Set 3

Due: Friday, September 27

Overview: In this problem set, you will get experience with implementing enumerations, card abstractions, and vectors methods.

Download: To begin this assignment, you should download a copy of the directory `~cs230/download/ps3`. In your local copy of this directory, you will:

- create a file named `MyStringWords.java` or `MyFileWords.java` for Problem 1;
- create a file named `Cards/MyCard.java` for Problem 2;
- edit the file named `VectorSet/VectorSet.java` for Problem 3.

Submission:

- For Problem 1, your hardcopy submission should be your final version of either `MyStringWords.java` or `MyFileWords.java`.
- For Problem 2, your hardcopy submission should be your final version of `Cards/MyCard.java`.
- For Problem 3, your hardcopy submission should be your final version of `VectorSet/VectorSet.java`.

Your softcopy submission for this problem should be the entire `ps3` directory.

Remember to include a signed cover sheet (found at the end of this problem set description) at the beginning of your hardcopy submission.

Problem 1 [30]: Enumerations

Do Problem 3 from Problem Set 2, which is now worth 30 points in the context of PS3. You should put your `MyStringWords.java` or `MyFileWords.java` file in the `ps3` directory.

Problem 2 [20]: Card Abstraction

The Contendo game company is attempting to implement a number of card games in Java on a hand-held computer with a small amount of memory. Because memory is such a limited resource, they are looking for any way they can to reduce the amount of memory their program uses. They hire Abby Stracksen, one of the world's top data abstraction experts, to improve the memory usage of their implementation.

One of the files they show to Abby is their implementation of the `Card` abstraction, which is shown in Appendix A. You should study this appendix now.

Abby notes that every `Card` instance requires space for two variables: one to hold a pointer to the card's value, and another to hold a pointer to the card's suit. Abby observes that the amount of space per `Card` instance can be reduced from two variables to one by using a single integer variable whose contents ranges between 0 and 51 to encode which of the 52 cards in a deck is denoted by the card. In particular, she settles on the following correspondence between integers and cards:

```
0 : 2C
1 : 2D
2 : 2H
3 : 2S
4 : 3C
5 : 3D
6 : 3H
7 : 3S
...
44 : KC
45 : KD
46 : KH
47 : KS
48 : AC
49 : AD
50 : AH
51 : AS
```

In this problem, you will use Abby's idea as the basis for a new implementation of the `Card` class that will replace the one shown above.

To begin this problem, examine the `ps3/Cards` directory, which contains implementations of all of the card abstractions discussed in class. Before you make any modifications, do the following:

- Study the `main()` method and the other class methods within the `Card` class in the file `Card.java`. These methods are used to test if the implementation of the `Card` class is correct. Invoke the `main()` method of the `Card` class by executing `java Card`. Study the output and make sure you understand why it is the correct output. You will be using this `main()` method later in this problem to test your changes to the `Card` class.
- Make a copy of `Card.java` (call it `Card-orig.java` say) so that you will always have the original version to refer to if you should need it.
- Make a copy of `Card.java` called `MyCard.java` defining the `Card` class that you will modify for this problem.

You should begin by removing the instance variables `value` and `suit` from the `Card` class and replacing them by a single integer instance variable named `num`:

```
private int num; // New instance var for Card. Should be in range 0--51.
```

Next, modify the definitions of the constructor method and all the instance methods (but *not* the class methods) of the `Card` class so that they work correctly with the new `num` instance variable in place of the old ones. **You should not change the interfaces to any of these methods; you should just change their implementations!** What you are doing is changing the internals of the `Card` black-box abstraction in such a way that it still appears to work as it did before from the outside world. This is the essence of data abstraction.

For instance, the `Card` constructor should still take two arguments: a `Value v` and a `Suit s`. Your job is to figure out how to generate a number between 0 and 51 that corresponds to the card that has `Value v` and `Suit s`. Similarly the `value()` and `suit()` methods should return the value and suit of the card, just as before.

When you have changed the implementations of the constructor and instance methods in `Card` to be consistent with your new representation, test your changes by executing the `main()` method you studied above. You have succeeded when the test behaves exactly as it did for the original implementation. If the test does not behave correctly, you need to fix your new implementation of `Card`.

Once you are satisfied that your new `Card` class works, rename it to be `MyCard.java` and rename `Card-orig.java` back to `Card.java`. (You could use your new class in place of the original `Card.java` in future card playing programs – after all, it’s supposed to be equivalent! – but it might contain subtle bugs that could complicate the other programs, so it’s best to reinstall the original implementation just in case.)

Notes/Hints:

- To implement the constructor method `Card(Value v, Suit s)` combine `v.toInt()` and `s.toInt()` in an appropriate way.
- When implementing the `value()` and `suit()` methods, you will find the following methods handy: `Value.intToValue` and `Suit.intToSuit`.
- If `n` and `d` are integers, then `n / d` denotes the integer quotient of `n` divided by `d` while `n % d` finds the integer remainder. That is, `n` is equal to $((d * (n / d)) + (n \% d))$. Study the `main()` method to see how `/` and `%` can be used to convert between numbers and cards.
- Even though `num` is a private instance variable, the instance method of one `Card` instance can examine the `num` variable of another instance.
- All of your method bodies except for `compare` can be one-liners.

Problem 3 [50]: Vector Operations

In this problem, you will write operations that manipulate vectors that represent sets of objects. Later in the semester, we will have a lot to say about representations of set. For the time being, the focus is on vector manipulation, not sets.

For our purposes, a set of objects will be represented by a vector containing all the objects in the set and no others. The objects in the vector need not be in any particular order, and the vector may contain duplicates of some of the objects. For example, here are some valid vector representations of the mathematical set $\{a,b,c\}$:

```
[a,b,c]
[b,a,c] // order doesn't matter
[c,b,a] // order doesn't matter
[a,a,a,b,c,c] // may have duplicates
[b,a,b,b,a,c,a,c] // order doesn't matter, may have duplicates
```

We shall assume that all set elements implement the `Comparable` interface. That is, each set element must have the following method:

```
public boolean compareTo (Object x);
Returns -1 if this object is less than x, 0 if this object is equal to x, and 1 if this object is greater than x.
```

Your goal is implement the following five methods that use this vector representation of sets:

a. [5]

```
public static boolean member (Comparable x, Vector v);
```

Returns `true` if `x` is a member of the set represented by `v`, and `false` otherwise.

b. [5]

```
public static void union (Vector v1, Vector v2);
```

Destructively modifies the set represented by `v1` to be a representation of the set that is the set union of the elements from the sets represented by `v1` and `v2` – i.e., a set that contains all the elements that appear in either the set represented by `v1` or the set represented by `v2`.

c. [10]

```
public static void intersection (Vector v1, Vector v2);
```

Destructively modifies the set represented by `v1` to be a representation of the set that is the set intersection of the elements from the sets represented by `v1` and `v2` – i.e., a set that contains all the elements that appear in both the set represented by `v1` and the the set represented by `v2`.

d. [10]

```
public static void difference (Vector v1, Vector v2);
```

Destructively modifies the set represented by `v1` to be a representation of the set that is the set difference of the sets represented by `v1` and `v2` – i.e., a set that contains all the elements of the set represented by `v1` that do not appear in the set represented by `v2`.

e. [20]

```
public static void canonicalize (Vector v);
```

Destructively modifies `v` so that it represents the same set as the set initially represented by `v` but additionally guarantees that the vector representation (1) contains no duplicates (2) has elements arranged in sorted order from low to high.

You should flesh out the bodies of the above methods in the file `ps3/VectorSet/VectorSet.java`.

You should test your methods using the `main` method of the `VectorSetTest` class which has been provided for you in the file `ps3/VectorSetTest.java` (see Fig. 1). This class provides a simple interpreter that executes vector set commands from a file on an initially empty vector `v`. The format of the testing file is described in the comments at the beginning of the `VectorSetTest` class. An example of such a testing file is the `test.txt` file presented in Fig. 2. The result of executing `java VectorSetTest test.txt` is shown in Fig. 3.

Note that because the same set can be reprinted by many different vectors, the output shown in Fig. 3 is only one example of a correct output. Any output in which each line beginning `v =` is a vector denoting the same mathematical set (i.e., has the same elements, but perhaps in a different order and with a different number of occurrences of some elements) would also be correct.

Notes:

- The `test.txt` file presented here is only an example. You should develop and use your own testing file(s).

- You are welcome to define any auxiliary methods you find helpful for this problem in `VectorSet.java`.
- You may find it helpful to use enumerations to write some of the methods.
- Beware of the gotchas you were warned about in class. They are waiting to bite you in some of the methods!

A Implementation of the Card class

The original implementation of the `Card` class (before the modifications in Problem 2) is shown in Figs. 4–6.

```

import java.util.*;

public class VectorSetTest {
    // A class for testing vector set operations.
    // Maintain a mutable vector set of strings that is initially empty.
    // Perform memberships, unions, intersections, differences,
    // and canonicalizations as specified in a file in the following format
    // (1) The string "member" on one line followed by a line with
    //     the string to test for membership in the current vector set.
    // (2) The string "union" on one line followed by a line with
    //     a string representation of the vector to union with the
    //     current vector set.
    // (3) The string "intersection" on one line followed by a line with
    //     a string representation of the vector to intersect with the
    //     current vector set.
    // (4) The string "difference" on one line followed by a line with
    //     a string representation of the vector to difference with the
    //     current vector set.
    // (5) The string "canonicalize" on one line.

    private static Vector v = new Vector ();

    public static void status () {
        System.out.println("v = " + v);
    }

    public static void main (String [] args) {
        status();
        String filename = args[0];
        Enumeration lines = new FileLines(filename, false);
        // "false" means don't include terminating newlines
        while (lines.hasMoreElements()) {
            String command = (String) lines.nextElement();
            if (command.equals("member")) {
                String elt = (String) lines.nextElement();
                System.out.println("member (" + elt + ", v) = "
                    + VectorSet.member(elt, v));
            } else if (command.equals("union")) {
                String vecRep = (String) lines.nextElement();
                System.out.println("union(v, " + vecRep + ");");
                VectorSet.union(v, StringVector.fromString(vecRep));
            } else if (command.equals("intersection")) {
                String vecRep = (String) lines.nextElement();
                System.out.println("intersection(v, " + vecRep + ");");
                VectorSet.intersection(v, StringVector.fromString(vecRep));
            } else if (command.equals("difference")) {
                String vecRep = (String) lines.nextElement();
                System.out.println("difference(v, " + vecRep + ");");
                VectorSet.difference(v, StringVector.fromString(vecRep));
            } else if (command.equals("canonicalize")) {
                System.out.println("canonicalize(v);");
                VectorSet.canonicalize(v);
            }
        }
        status();
    }
}

```

```

union
[b,a,c]
member
a
member
b
member
c
member
d
union
[b,f,e,g,c,e,d,h]
intersection
[c,g,a,i,f,h,a,e]
difference
[i,f,j,e,a,f]
canonicalize
union
[c,g,k,c,f,g,b]
canonicalize

```

Figure 2: The contents of `test.txt`, one example of a file to test vector sets.

```

java VectorSetTest test.txt
v = []
union(v, [b,a,c]);
v = [b, a, c]
member (a, v) = true
v = [b, a, c]
member (b, v) = true
v = [b, a, c]
member (c, v) = true
v = [b, a, c]
member (d, v) = false
v = [b, a, c]
union(v, [b,f,e,g,c,e,d,h]);
v = [b, a, c, b, f, e, g, c, e, d, h]
intersection(v, [c,g,a,i,f,h,a,e]);
v = [a, c, f, e, g, c, e, h]
difference(v, [i,f,j,e,a,f]);
v = [c, g, c, h]
canonicalize(v);
v = [c, g, h]
union(v, [c,g,k,c,f,g,b]);
v = [c, g, h, c, g, k, c, f, g, b]
canonicalize(v);
v = [b, c, f, g, h, k]

```

Figure 3: One possible output of `java VectorSetTest test.txt`. Any output in which each line beginning `v =` contains the same elements, with possible reorderings and different number of occurrences of the same elements, would also be correct.

```

public class Card {

    // Instance Variables:
    private Value value; // The value of this card.
    private Suit suit; // The suit of this card.

    // Constructor Method:
    public Card (Value v, Suit s) {
        // Returns a new card with the given value and suit.
        value = v;
        suit = s;
    }

    // Instance Methods:

    public Value value() {
        // Returns the value of this card.
        return value;
    }

    public Suit suit() {
        // Returns the suit of this card.
        return suit;
    }

    public boolean equals (Card c) {
        // Returns true if c has the same value and suit as this card; false otherwise.
        return ((value.equals(c.value)) && (suit.equals(c.suit)));
    }

    public int compare (Card c) {
        // Returns -1 if this card precedes c in the card ordering;
        // 0 if this card is equal to c in the card ordering;
        // and 1 if this card follows c in the card ordering.
        if (value.equals(c.value)) {
            return suit.compare(c.suit);
        } else {
            return value.compare(c.value);
        }
    }

    public String toString() {
        // Returns a string representation of this card.
        return value.toString() + suit.toString();
    }

    public static Card fromString (String s) {
        if (s.length() != 2) {
            throw new RuntimeException ("Card.fromString: unrecognized string -- " + s);
        } else {
            return new Card (Value.fromString((new Character(s.charAt(0))).toString()),
                Suit.fromString((new Character(s.charAt(1))).toString()));
        }
    }

    public static Card intToCard (int n) {
        return new Card(Value.intToValue((n / 4) + 2), Suit.intToSuit(n % 4));
    }
}

```

Figure 4: The Card class (Part 1).

```

// -----
// Testing code

public static void main(String [] args) {
    for (int i = 0; i < 52; i++) {
        // Perform simple tests on each of the 52 cards:
        testCard(i, intToCard(i));
    }
    System.out.println();
    compareTable();
    System.out.println();
    equalsTable();
}

public static void testCard (int n, Card c) {
    System.out.println(n + ": c = " + c
        + "; c.value() = " + c.value()
        + "; c.suit() = " + c.suit()
        + "; Card.fromString(c.toString()) = "
        + Card.fromString(c.toString())
    );
}

public static void compareTable() {
    Deck d = new Deck();
    System.out.println("          " + d);
    for (int n = 0; n < 52; n++) {
        Card c = intToCard(n);
        System.out.print(c + ".compare(...): ");
        for (int i = 0; i < d.size(); i++) {
            System.out.print(formatInt(c.compare(d.get(i)), d.get(i).value().toInt()));
        }
        System.out.println();
    }
}

public static void equalsTable() {
    Deck d = new Deck();
    System.out.println("          " + d);
    for (int n = 0; n < 52; n++) {
        Card c = intToCard(n);
        System.out.print(c + ".equals(...): ");
        for (int i = 0; i < d.size(); i++) {
            System.out.print(formatBool(c.equals(d.get(i)), d.get(i).value().toInt()));
        }
        System.out.println();
    }
}
}

```

Figure 5: The Card class (Part 2).

```
public static String formatInt (int n, int val) {
    if (n == 0) {
        return " = ";
    } else if (n < 0) {
        return " < ";
    } else {
        return " > ";
    }
}

public static String formatBool (boolean b, int val) {
    if (b) {
        return " + ";
    } else {
        return " - ";
    }
}
}
```

Figure 6: The Card class (Part 3).

*Problem Set Header Page
Please make this the first page of your hardcopy submission.*

CS230 Problem Set 3

Due Friday September 27

Name:

Date & Time Submitted:

Collaborators (*anyone you worked with on the problem set*):

By signing below, I attest that I have followed the collaboration policy as specified in the Course Information handout.

Signature:

*In the **Time** column, please estimate the time you spend on the parts of this problem set. Please try to be as accurate as possible; this information will help me design future problem sets. I will fill out the **Score** column when grading your problem set.*

Part	Time	Score
General Reading		
Problem 1 [30]		
Problem 2 [20]		
Problem 3 [50]		
Total		