CS2

Theme:

Object Oriented Programming in Java

Handout : Week8

prepared by N S kumar

Preface

Dear friends,

This time we discuss a few programming problems. We will elaborate on finer points of programming. We will also highlight a few interesting aspects of good programming. We will also discuss the concept of Strings.

FAQ is indefinitely delayed. Hope I get the time and the inclination to compile these question and answers.

I am challenging you to become part of this exercise of making notes. Let me know whether any of you want to contribute towards this effort.

All the best.

N S Kumar
Visiting Professor
Dept. Of Computer Science and Engineering
PES Institutions.
Email : ask.kumaradhara@gmail.com
Date : 8th March 2015

A few guide Lines:

All of you have been programming in the laboratory – hopefully enjoying the assignments and more so the challenging questions. Habit dies hard – getting into a good habit is always a good idea. Follow these coding guidelines.

- Always indent your programs.
- Name the classes and attributes(fields) using nouns.
- Name the methods using verbs.
- Make the names of interfaces meaningful – if necessary use long names.
- Make the names of local variable also meaningful – but here the names can be short.
- Use camelCase for naming.
- Never use **break** unless your control structure is a **switch** statement.
- Never use **continue**.
- Use static fields in a class to represent some shared attribute.
- Do not use static field to store temporary results of computation.
- Make a method return once and only once.
- Prefer a method to return a value.
- Name the method which modifies the object to indicate its behaviour. You may call it a set method.
- Do not have common code across if and else statements.
- Move the code out of the loop if that computation is not affected by the conditions within the loop.
- Make every class represent one and only concept.
- Make every method do one and only one action.
- Choose the boundary conditions of the looping structures correctly.
- Come out of the loop using predicates if looping is no more required.
- Test your methods with variety of test cases.
- Never sacrifice readability.
- Avoid tricks in programming.

It will be nice if you can join me in this activity of making notes. Can you make some examples which violate the above mentioned points and then suggest a remedy for them? I am curious to know how many of you would volunteer to make this special document. Mail me at ask.kumaradhara@gmail.com.

A Few More Examples:

Find the smallest element in an array:

We should decide the method signature.

- What are the arguments required for this method?
- What should the method return?

In this case, a single array shall be passed as argument. Arrays in Java are intelligent. They know their size unlike their C counterpart (which are stupid!).

What shall we return from the method? Shall we return the smallest element itself? We can. What if the client wants to change the smallest element – increase its value? She cannot if she gets back the value. Why not return the position of the element in the array to her?
Thats a good idea.

This shall be a static method as these arrays are not part of any class. So there is no object to invoke the method.

Let us now consider the implementation.
Shall we assume that the smallest element is 0 to start? Compare this with the remaining element and update this if any element in the array is smallest than this element? **Can you think of a case when this would not work?**

How about assuming that the 0th element of the array is the smallest? That is a nice idea. We can now compare the smallest so far with the remaining elements of the array. Whenever an element of the array is smaller than the smallest so far,

we update the smallest so far and also remember its position. So, you might have observed that we are doing as many as n – 1 comparisons where n is the number of elements in the array. **Can we make it better than that?**

// L29/ques.txt
// Check out some of the common errors or pitfalls:
// - off by one errors.
// - difficulty in deciding the boundary condition
// - wrong return from with a selection inside a loop
// - unnecessary looping even after the solution is found
// - unnecessary computation within the loop
// - not using the library routines

```
// Write a method to find the position of the smallest element in an array.
int findMin(int[] a)
{
	int pos = 0;
	int min = a[0];
	for(int i = 1; i < a.length; i++)
	{
		if(a[i] < min)
		{
			min = a[i];
			pos = i;
		}
	}
	return pos;
}
```

Check whether elements of a sorted array are not unique or elements repeat:

- **What should be the argument to this method?**
  - Sorted array
- **What should the method return?**
  - Should we display whether elements repeat or do not repeat within the method? NO. A method should display only if it is called display – it is for displaying. The client cannot do anything if she gets no result back.
  - Should the method return a boolean? May be. If the client wants to remove the repeated element or change it, she cannot as she does not know the position.
  - Should we return the position where the element repeats. YES.
  - What if there is no repetition? We may return any invalid index – in such cases, -1 is the normal choice in Java.
  - What if there are more than one repetition? We will find the leftmost as of now and we will discuss later how to find all repetitions.
- **What should we consider in the implementation?**
  - As the input is sorted, adjacent elements will repeat if the array has repeated elements.
  - In the best case, we shall do at most one comparison.
  - In the worst case, we shall do n – 1 comparisons where n is the number of elements in the array.
  - We require a mechanism to exit the loop as soon as we find the two adjacent elements repeat.
  - Check:
    - Why pos is initialized to -1?
    - Why is the loop condition i < n – 1 and not i < n?
    - Why not say pos = -1; in the else part of the if statement?

2. given a sorted array, check whether an element repeats

```
int findPosOfRepeated(int[] a)
{
	int pos = -1; int n = a.length;
	for(int i = 0; i < n - 1 && pos == -1; i++)
	{
		if(a[i] == a[i + 1])
		{
			pos = i;
		}
	}
	return pos;
}
```

The earlier method is not flexible. It tries to find the first occurrence of the repeated elements. What if my requirement is to find all the repetitions?

How about finding repeated elements within a range of positions in the array?

Check the next example.

```
int findPosOfRepeated(int[] a,int start_pos,int stop_pos)
{
	int pos = -1;
	for(int i = start_pos; i < stop_pos && pos == -1; i++)
	{
		if(a[i] == a[i + 1])
		{
			pos = i;
		}
	}
	return pos;
}
```

Should we support both these overloaded methods? Yes. It will be helpful to our client. But something is terribly wrong with this implementation. Both have the same logic. If for some reason, the logic changes, we have to change at both the places. This is called multiple updation. If we change at only one place, our program will be terribly wrong. Integrity goes for a toss.

Can we make one call the other so that the logic of the solution is captured at only one place?

Find position of all repeated elements in a sorted array:

This method calls the method developed earlier until no more repeated elements are found. Observe how the looping structure has been designed, We call the method to find the position of the repeated elements twice – once before the loop and once as the last statement within the loop. This is a characteristic of loops which use a sentinel. In this case, pos becoming -1 acts like a sentinel. In a language like 'C', you may put the assignment itself within the expression of the while – which is not considered a good programming practice in Java.

All said and done this is not a good method. It displays the result even though it is not called display!

Modify the method to return positions of all repetitions.

```
// use find of pos of repeated elements
// find all matches(display the posn to start with)
public static void findAllRepeated(int[] a)
{
	int pos = findPosOfRepeated(a, 0, a.length - 1);
	while(pos != -1)
	{
		System.out.println(pos);
		pos = findPosOfRepeated(a, pos + 1, a.length - 1)
	}
}
```

Remove repeated elements:

- argument : sorted array
- return mechanism
  - Can we display unique elements in the method? NO.

not useful to the client

  - Can we modify the same array itself? We cannot. The attribute length of an array is a readonly attribute. We can modify the elements of the array, but not its length.
  - Can we return a new array as the result? YES.
- Implementation:
  - At any point in time, i shall indicate the position of the last unique element so far – j will start from position 1 and run through the whole array. Whenever the elements a[i] and a[j] are same. We skip the jth element.
  - At the end of the loop, the array a will have unique elements from position 0 till and inclusive of position i. We can use the library function – utility method of the Arrays class – copyOfRange to return the array to the client. Let the client do whatever she wants with it.

```
// input is sorted array; remove repeated elements
public static int[] removeRepeated(int[] a)
{
        int i = 0;
        for(int j = 0; j < a.length; j++)
        {
                if (a[i] != a[j])
                {
                        a[i+1] = a[j];
                        i++;
                }
        }       // i : posn of the last element
        return Arrays.copyOfRange(a, 0, i + 1); //check
```

```
}
```

Recursion and Array:

Let us try a simple example of recursion with arrays. Let us try to find the sum of the elements of an array. The sum of an empty section of an array is 0. That could be the escape hatch or the base condition. Then we can say that the sum of an array section having n elements is given by the sum of the last element and the remaining elements of the section of the array without the last element.

```
// find the sum of the elements of an array
//	using recursion
public static int sum(int[] a, int n)
{
	if(n == 0)
	{
		return 0;
	}
	else
	{
		// last elem + sum of the remaining
		return a[n - 1] + sum(a, n -1);
	}
}
```

Strings:

A sequence of characters is called a string. What I am typing into the computer is a string. My name is a string. Our program is a sequence of strings. Strings are omnipresent.

Many languages provide String as a builtin type. There have been languages for processing strings alone – SNOBOL. Is one such example.
String is a builtin class in Java.

Do look at the String class in Java documentation.

Let us summarize the special characteristics of String in Java.

- String is a reference type.
- String object can be created using a String constant(we call it a literal) or by using new. There is very little difference between the two ways of creating a String object.
- We can find the length of the string using the method length. This is not an attribute (not as in arrays).
- We can not index on a String the way we do with arrays. Instead to get individual characters, we use a method called charAt(index) on a String object.
- We can not modify a String once created. We cannot modify the individual characters. We cannot modify the length. We say that the Strings are immutable in Java. However we can create a new String with the same name by assigning to the same variable.
- Let us discuss a few operators and/or methods which are not self explanatory.
    - Difference between == and equals
        - == is based on the location. If both the objects refer to the same location, then the result of the comparison will be true. This concept is called reference semantics.

        - equals is based on the values. If both the objects have the same content, the result of the comparison will be true. The creator of the class (in this case, String class) should provide the meaningful implementation for this method. This concept is called value semantics.
        - Observe the reference semantics definitely implies value semantics.
    - compareTo()
        - This is a three way comparison method. This is similar to strcmp of 'C'. If the two Strings are same, the result is 0. Otherwise this method finds leftmost mismatch in the two strings and returns the difference of their internal coding. This method is required for sorting an array.
    - IndexOf, substring, replace
        - indexOf searches a given string for the occurrence of another string
        - substring returns part of a string
        - replace allows us check whether a given string exists, modify that by another string and return the changed string.
            - **Check the difference between replace and replaceAll**

```
//L30/Example1.java
// Example on strings
// - is a type
// - has a sequence of characters
// - can be treated as a single unit
// - in 'C' :
//              is an array null terminated
// - in Java :
//              is a reference type
//              is like a structure of 'C'
//              has a sequence of characters
//              knows its length (it is a behaviour and not an attribute)
```

```
//              cannot be changed( immutable)
//              has # of methods to access
//                      to return the modified string
//      check the documentation of String class
public class Example1
{
        public static void main(String[] args)
        {

                // creation of strings; no difference
                String s1 = "java";
                String s2 = new String("java");
                System.out.println("string s1 : " + s1);
                System.out.println("string s2 : " + s2);
                System.out.println("length : " + s1.length()); // is a behaviour
                // no support for indexing using [] operator
                // access each element
                for(int i = 0; i < s1.length(); i++)
                {
                        System.out.println(s1.charAt(i));
                }
                // convert to uppercase
                // toUpperCase()
                s1.toUpperCase();
                System.out.println("string s1 : " + s1); // no change!
                // immutability
                s1 = s1.toUpperCase(); //  a new string is created!
                System.out.println("string s1 : " + s1); // has changed!

                // comparison
                String s3 = "java";
                System.out.println("equal : " + (s2 == s3)); // false
```

```
        System.out.println("equal : " + s2.equals(s3)); // true



        // some more comparisons
        System.out.println("in order : " + "amar".compareTo("anthony")); //
negative
        System.out.println("same: " + "amar".compareTo("amar")); // 0
        System.out.println("not in order : " + "amar".compareTo("akbar")); //
positive



        // substring
        String s4 = "together";
        System.out.println(s4.substring(0, 2));
        System.out.println(s4.substring(2, 5));
        System.out.println(s4.substring(5));

        // replace
        String s5 = "good programming good java good teacher";
        String s6 = s5.replaceAll("good", "best");
        System.out.println("old string : " + s5);
        System.out.println("new string : " + s6);

        // search
        String s7 = "bengaluru";
        System.out.println("pos : " + s7.indexOf("gal"));
    }
}
```

Combining Concepts:

Command Line Arguments:

Arrays are not Strings. Strings are not arrays. There is no relationship between them. One does not throw a shadow on the other. We say these are orthogonal. In such cases we can combine them. We can make arrays of Strings. (But they are not exactly orthogonal – as we cannot make Strings of Strings or Strings of Arrays!).

When we run a command like ls, we can specify the directory name(s) following the command ls in unix flavours. These are called command line arguments. Can we do the same when we run our Java programs?

We can.

For example, we can say

$ java Example1 I love java very much

These five Strings, I love java very much will be stored in the array called args which is the name of the parameter of the main method of our class. The following program shows how we can display all the command line arguments. Please note that the name of the parameter could be anything.

```
//L31/Example1.java
// playing with array of strings
// display the command line arguments
public class Example1
{
	public static void main(String[] args)
	{
		int l = args.length; // observe that args is an array and not a String
		for(int i = 0; i < l; i++)
		{
			System.out.println(args[i]);
		}
```

```
            for(String s : args)

            {
                  System.out.println(s);
            }
      }
}
```

Sort an array of Strings:

Let us understand a few points in this program.

- The statement below creates a reference to an array of strings.

String[] names;

- names is not initialized. Using names at this point shall give a compile time error. Ex: names.length
- The statement below creates an array of String references and not Strings!

names = new String[n];

- names.length and not names.length() is valid at this point.
- names[0] is perfectly alright as any object created using new (on the heap) will be initialized to a default value based on the type. All reference types will be null.
- Arrrays.sort() can sort array of any type! Is it not wonderful? How does it know how to compare the elements of this array – which are Strings. It internally uses compareTo method of the String class. We will discuss more of these in the next lectures.

```
//L31/Example2.java
// sort an array of strings
// observe step by step :
// step 1:
// create an array of strings and display them (all null)
```

```
// step 2:
// read from the scanner and populate each string and display them
// step 3:
// sort the strings and disp
//       check how the strings are getting sorted


import java.util.*;
public class Example2
{
        public static void main(String[] args)
        {
                int n;
                String[] names;
                Scanner scanner = new Scanner(System.in);
                System.out.print("enter # of strings : ");
                n = scanner.nextInt();
                names = new String[n];
                // display and check that each string will be null
                // disp(names);
                read(names, scanner);
                // disp(names);

                Arrays.sort(names);
                disp(names);
        }
```

```
    public static void disp(String[] names)
    {
        for(String s : names)
        {
            System.out.println(s);
        }
    }

    public static void read(String[] names, Scanner scanner)
    {
        int l = names.length;
        for(int i = 0; i < l; i++)
        {
            names[i] = scanner.next();
        }
    }
}
```

Mutable String:

There are many cases where we may want to modify a string – append or insert characters or substrings, modify individual characters. String class is not geared for it. Instead we use a class called StringBuffer class.

- An object of StringBuffer class should be created using the operator new.
- We can convert an object of StringBuffer to a String by using toString method of the StringBuffer class.
- We can append and insert into an object of a StringBuffer class.
- We can reverse an object of StringBuffer class.

This example checks whether a given string is a palindrome.

```
//L31/Example3.java
// difference between String and StringBuilder
// understand immutability
// check other functions like :
//              append
//              setCharAt
//              setLength



public class Example3
{

        public static void main(String[] args)
        {
                String s = "mahopaadhyaaya";
//              StringBuilder sb = "mahopaadhyaya"; // Error
                StringBuilder sb = new StringBuilder("mahopaadhyaaya");
                System.out.println(s);
                System.out.println(sb);
                sb.insert(3, "aamah");
                System.out.println(sb);
                // check for palindrome
                String s1 = "malayalam";
                String s2 = "kannada";
                System.out.println(new StringBuilder(s2).reverse());
                System.out.println(new StringBuilder(s1).reverse().toString() == s1);
                System.out.println(new StringBuilder(s2).reverse().toString() == s2);
                System.out.println(new
```

```
StringBuilder(s1).reverse().toString().equals(s1));
                System.out.println(new
StringBuilder(s2).reverse().toString().equals(s2));
        }
}
```

Some exercises for you to try:

- Replicate a given String n times

replicate(“aha”, 3) should return ahaahaaha

- Encode a string.

encode(“abcdz”) should return bcdea

- BinarySearch a sorted Array of Strings
- Given two arrays of Strings, find Strings which are in both the arrays.
- Given a String representing an expression, find the number of left and the number of right parentheses.

We will converse again next week. Have a nice time.