Lecture 22

Static Variables, Nested Lists

In this lecture we will discuss following topics:

- static variables
- nested lists

Before starting discussion on static variables it is useful to revise local variables. Local variables are defined within a function. Local variables have following attributes:

- invisible outside function (can access within defined function)
- local variables life start with function call and ends when function terminates/ returns

First point means local variables of one function are not accessible in other function. We may send values of multiple local variables as arguments to other functions. However, function can return single variable only. Second points is local variable is created when function is called and destroyed when function terminates/ return. The longest life function for every program is main function which starts with the execution of program and terminates with the termination of program. Hence local variables of main method have longest life. Local variables in other functions born and die as many times function is called.

Static variables are declared anywhere within class boundary and outside the functions. Keyword "static" is written with static variables. Static variables are visible to every function of the class. Writing private may stop their access outside the class. Otherwise outside class we may access them static variables of other class with class name followed by period, followed by variable name:

```
class One{
  static int n=10;
  ...
}
```

```
class Two{
  public static void main(String a[]){
    ?ln(One.n);
  }
}
```

Any function of the class can access and change value of static variable. Therefore at one side this is convenience to share variables among functions of class at other side this is problem for debugging. A situation where you have to trace why output is not accurate? Anyhow in this lecture we will discuss use of static variables. For this we will discuss a scenario. Suppose we want to make address book. Address book has contacts. A contact has name, address, email, mobile no. We have to store many contacts. Therefore we have to use array and this is clearly example of parallel arrays. Here we need four arrays because there are four attributes. Sample data is shown below, I will share file of 35000 contacts, free available from internet:

FirstName	Address	Phone	Email
Cara Hingst	1717 Elm Hill Pike	615-883-8408	cara@hingst.com
Diane Ort	3 Embarcadero Ctr	415-393-6166	diane@ort.com
Zane Opunui	416 Park St	434-296-3666	zane@opunui.com
Lea Knighton	3202 W Charleston Blvd	702-878-2130	lea@knighton.com
Patsy Rezac	4 W Main St	828-524-5325	patsy@rezac.com
Lorene Taglauer	2 Bridge St	401-861-6066	lorene@taglauer.com
Dessie Diekrager	403 Charles St	301-870-8720	dessie@diekrager.com
Samuel Monn	811 S Bellview Ave	856-829-9200	samuel@monn.com
Darwin Howington	201 E 5th St	620-231-1773	darwin@howington.com

Data available on: http://www.briandunning.com/sample-data/

In address book we have following operations mainly:

1. add new contact
2. search existing contact by any field
3. remove existing contact
4. update existing contact
5. show all contacts of address book

Here we are assuming that maximum capacity of contacts is 200 as our mobiles have something similar. However, current count of contacts may vary with operations of add and remove. Therefore, we will take a variable currentSize. Now we have 5 static variables in our class:

```
class AddressBook{
  public static String name[]=new String[200];
  public static String add[]=new String[200];
  public static String mob[]=new String[200];
  public static String email[]=new String[200];
  public static int currentSize=0;
}
```

Let start discussion with operations required. First operation is add. Add operation requires a contact and it will add it into the static variable so that it become available for all functions. Add operation will simply place new contact at current size and will increment current size by 1:

```
public static void add(String n, String a, String m, String em){
  name[currentSize]=n;
  add[currentSize]=a;
  mob[currentSize]=m;
  email[currentSize]=em;
  currentSize++;
}
```

Student may add if condition to check that current size remain below actual array size, and your function may return boolean value true and false. If current size is below actual size record will be added and true value is sent; whereas; otherwise record is not added and false value is sent back. Next we will discuss searchByName function. If name is found complete record will be shown:

```
public static void searchByName(String n){
  int i;
  for (i=0;i<currentSize;i++)
    if (n.equals(name[i])
      ?ln(name[i]+"\t"+add[i] +"\t"+mob[i] +"\t"+email[i]);
}
```

Student may add similar function to search by other fields like address or mobile or email. Also search function can return index instead of printing record. Modify function can be made similarly by replacing print statement with new value assignment. A little bit tricky operation is delete or remove. This function will again proceed in similar way as search but in order to delete it will do following tasks. Copy last record to current than reduce current size by 1:

```
public static void remove(String n){
  int i;
  for (i=0;i<currentSize;i++)
    if (n.equals(name[i]){
```

```
     name[i]=name[currentSize];
     add[i]=add[currentSize];
     mob[i]=mail[currentSize];
     email[i]=email[currentSize];
     currentSize--;
   }
}
```

Student may think some other way as well but this is the most convenient way to remove as well as manage size as well. Lastly show all contacts is pretty simpler function. Just use loop and print all contacts in single line. Simply if we remove if condition from search operation we will get show all function. Create a menu driven main and enjoy the address book.

Now we will discuss the trickiest but interesting thing of this semester that is nested list. Each of them can store other object of ArrayList or Vector as well. Like list of players where record of each player is stored in list as well. List of players has record of players where record of each player is also a list. Before that we use list as linear list where we can store simple values and for that purpose we declare list like:

```
ArrayList<int> marks=new ArrayList<int>();

ArrayList<double> heights=new ArrayList<double>();

ArrayList<String> months=new ArrayList<String>();
```

If we have to add array list as object in array list, we will declare it as:

```
ArrayList<ArrayList> lists=new ArrayList<ArrayList>();
```

Before discussing a sample code it is important to understand why we need nested list. Nested list are convenient to pass and return data to other functions. As we may have to send single argument instead of multiple arguments and in returning we have no option to return multiple things. A sample code is given in last lab, I am pasting same code here:

```
class NestedList{
 PSVM (String args[]){
  ArrayList<String> names;
  names=new ArrayList<String>();
  ArrayList<Integer> marks;
  marks=new ArrayList<Integer>();
  names.add("Zahid");
  names.add("Saeed");
  names.add("Furrukh");
  marks.add(34);marks.add(29);
  marks.add(37);
  ArrayList lists=new ArrayList();
  lists.add(names);
  lists.add(marks);
  showAll(lists);
 }
PS void showAll(ArrayList<ArrayList> ls)
{
  ArrayList nm=lists.get(0);
  ArrayList mk=lists.get(1);
  int i;
  for (i=0;i<nm.size();i++)
   ?ln(nm.get(i)+"\t"+mk.get(i));
 }
}
```