Programming in JAVA

lecture 9

Dynamic data structures
(ArrayList and HashMap)

Dynamic data structures

1. To store sets of similar data you can use conventional tables.
2. The other option is to use **dynamic data structures**.
3. These data structures are not as fast as conventional tables, but more convenient in business applications.
4. These data structure allow to add/remove elements to/from tables after creation (coventional tables have fixed size after creation).

Types of dynamic data structures

Some of types od dynamic data structures in JAVA

ArrayList

HashMap

ArrayList declaration in JAVA

```
ArrayList<Float> myList1 = new ArrayList<Float>();

ArrayList<Integer> myList2 = new ArrayList<Integer>();

ArrayList<String> myLista3 = new ArrayList<String>();
```

You have to specify type of data that will be stored inside ArrayList

by typing it inside `< … >` brackets.

Adding elements to ArrayList

```
myList1.add( new Float("7637.3") );

myList2.add( 3 , new Float("7637.3") );
```

The first type od add function adds new element at the end of the list.

The second type of add function is used to add data after an element.

(in the example in the position of index 3).

Adding entire list to ArrayList

```
myList.addAll( other_ArrayList );

myList.addAll( 3 , other_ArrayList );
```

Retrieve an element from ArrayList

```
int i = 5;

Float f = myList1.get( i );
```

Size of ArrayList

```
myList1.size();
```

Removing from ArrayList

```
int i = 5;

myList1.remove( i );

myList1.remove( fromIndex , toIndex );

myList1.clear();
```

2 dimensional ArrayList

```
ArrayList<ArrayList<Float>> list2D
                    = new ArrayList< ArrayList<Float>>();

for(int i =0 ; i<5 ; i++)

list2D.add( new ArrayList<Float>() );

list2D.get(2).add(new Float("356.34"));

list2D.get(2).get(0);
```

HashMap declaration in JAVA

```
HashMap<String,Float> map
                          = new HashMap<String,Float>();
```

1. The HashMap structure stores pairs key-data.

   Thus we specify inside < … > brackets the type of key and type

   of data.

2. By using HashMap we do not care about order like in ArrayList.

3. If we iterate over entire HashMap we do not have guarantee that the

   iteration will be in the same order as we put data to HashMap.

Adding new data to HashMap

```
map.put("sales for dep. A" , new Float("7637.3") );

map.putAll( other_HashMap );
```

Obtaining data from HashMap

```
map.get("sales for dep. A");
```

1. To obtain data from HashMap you need the key

   (not index like in ArrayList).

2. However it is possible to iterate over entire HashMap if you do not

   have the key set (next slides).

Iterating over entire HashMap
(2 possibilities)

```
//option 1
Set<String> keys = myMap.keySet();


//option 2
Collection<Float> dataSet = myMap.values();
```

Iterating over entire HashMap (option 1 - iterator)

```
Float f;
Iterator it = myMap.keySet().iterator();
while (it.hasNext())
{

   f = myMap.get( it.next());
   System.out.println( f.toString()
}
```

Iterating over entire HashMap (option 2 – the for..each loop)

```
for( Float f : myMap.values() )
{
     System.out.println( f.toString() );
}
```

EXAMPLES

Example 1 - ArrayList

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

myList.add("abc1");
myList.add("abc2");
myList.add("abc3");

System.out.println("Printing list data : ");
for(int i=0; i<myList.size() ; i++)
{
   System.out.println(myList.get(i));
}
```

Example 2 - stack created with ArrayList
(class declaration not included)

```
private ArrayList<Integer> core =
                                    new ArrayList<Integer>();
```

Example 2 - stack created with ArrayList

```
public void push(Integer i)
{
    core.add(i);
}

public Integer pop()
{
    return core.remove( core.size()-1 );
}

public Integer peek()
{
    return core.get( core.size()-1 );
}

public boolean isEmpty()
{
   return core.isEmpty();
}
```

Example 3 – removing from ArrayList

```
ArrayList<Integer> tab = new ArrayList<Integer>();

for(int i=0; i<95 ; i++)
          tab.add(i%5);
```

Example 3 – removing from ArrayList – part 2

```
/*  this code is wrong  : */

for(int i=0 ; i<tab.size() ; i+=5)
            tab.remove(i);
```

Example 3 – removing from ArrayList – part 3

```
/*  this code is correct : */

for(int i=tab.size()-1 ; i>=0 ; i-=5)
        tab.remove(i);
```