Introduction to Programming I

Lab 11

Munir Makhmutov, Mansur Khazeev, Sami Sellami and Furqan Haider

Collections in JAVA

1. Collections / containers in Java are usually called classes, the main purpose of which is to store a set of other elements.

2. The standard set of Java collections serves to relieve the programmer of the need to independently implement recursive data types (**Lists,Trees)** and provides it with additional features.

3. Collections can store any reference data type.

Collections

1. **List:**

   List is an ordered list. Objects are stored in the order in which they are added to the list. The elements of the list are accessed by index.

2. **Set:**

   Set is a set of non-repeating objects. Only one null reference is allowed in a collection of this type.

3. **Maps:**

   It is used to map each element from one set of objects (keys) to another (values). In this case, each element from the set of keys is assigned a set of values. At the same time, one element from a set of values can correspond to 1, 2 or more elements from a set of keys.

Difference b/w List, Set and Map

List:

1. Index-based methods to insert, update, delete, and search the elements.

2. It can have duplicate elements and we can also store null elements.

3. List preserves the insertion order, it allows positional access and insertion of elements.

Set:

1. Unordered collection of objects.

2. Duplicate values cannot be stored.

3. Duplicate item will be ignored in Set and it will not print in the final output.

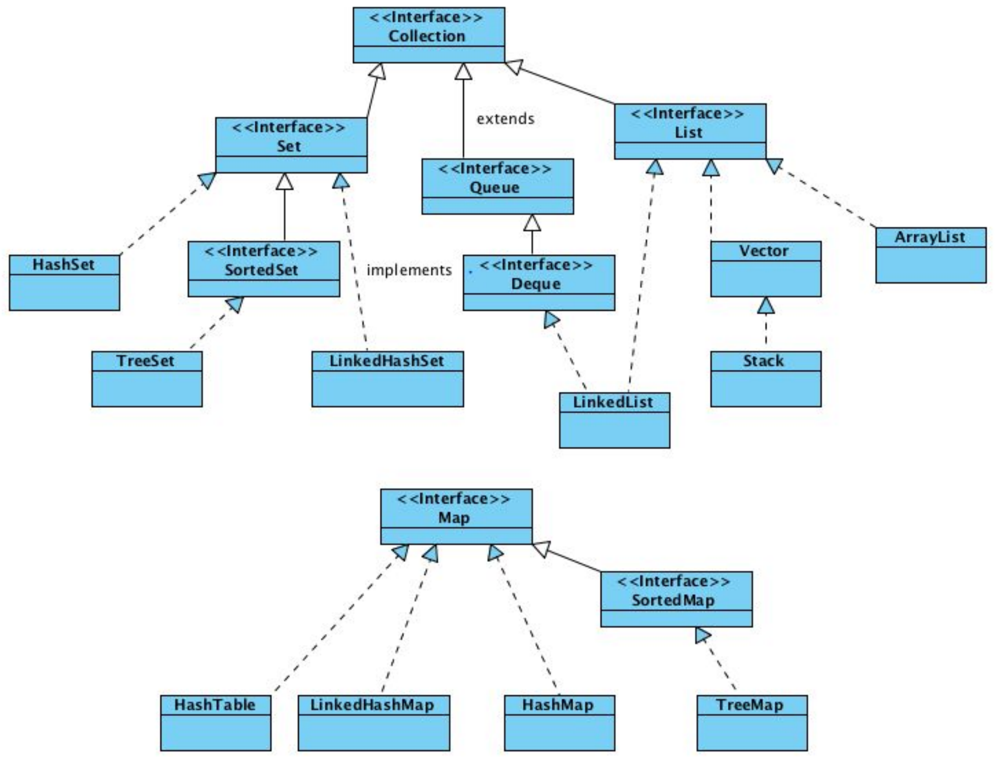
Map:

1. Java Map can store pairs of keys and values.

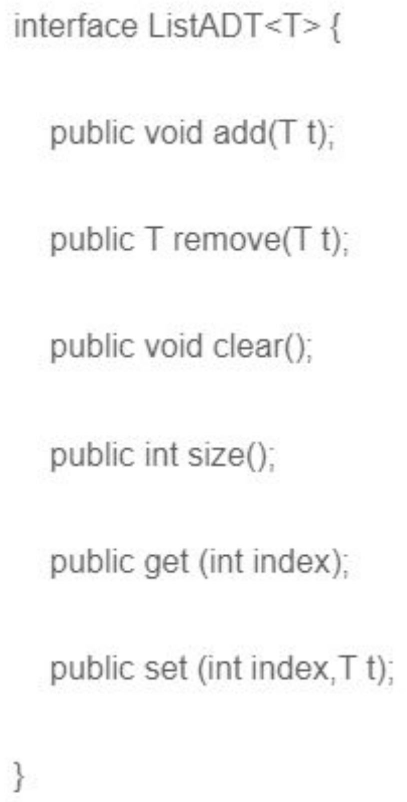
2. Each key is linked to a specific value.

3. Once stored in a Map, you can later look up the value using just the key.

Collections Hierarchy

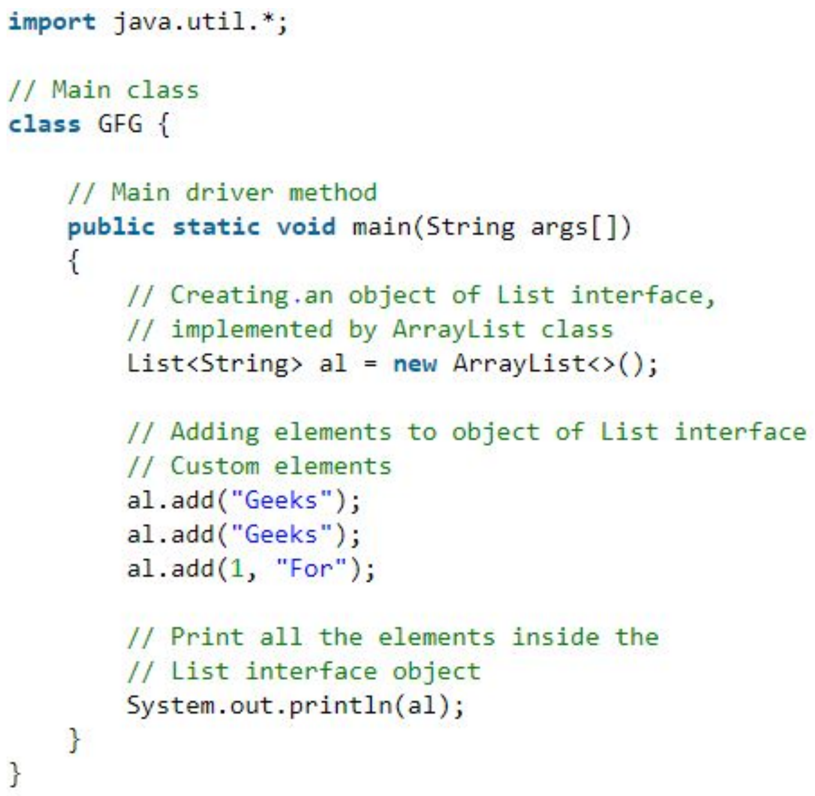


List Template



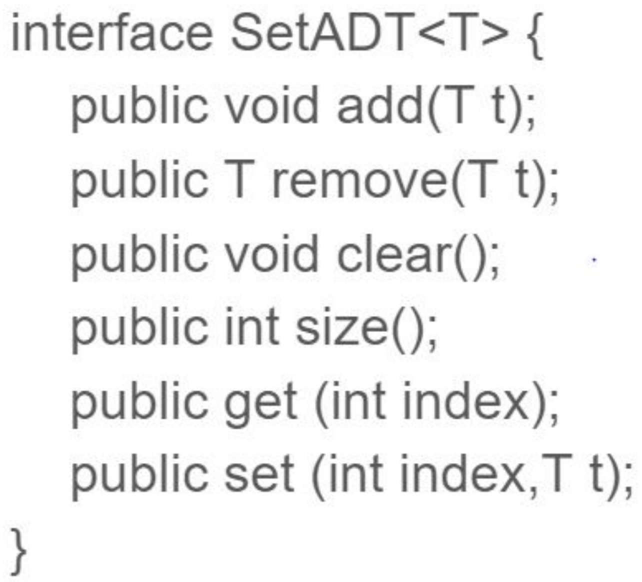
```
interface ListADT<T> {

    public void add(T t);

    public T remove(T t);

    public void clear();

    public int size();

    public get (int index);

    public set (int index,T t);

}
```

List Example



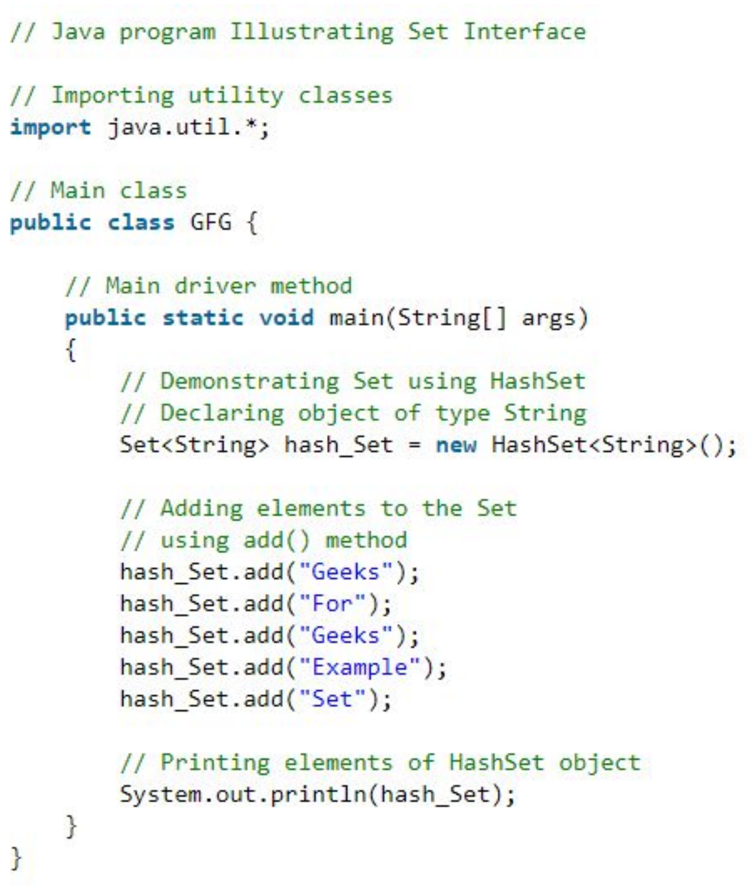
```
import java.util.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Creating an object of List interface,
        // implemented by ArrayList class
        List<String> al = new ArrayList<>();

        // Adding elements to object of List interface
        // Custom elements
        al.add("Geeks");
        al.add("Geeks");
        al.add(1, "For");

        // Print all the elements inside the
        // List interface object
        System.out.println(al);
    }
}
```

Set Template



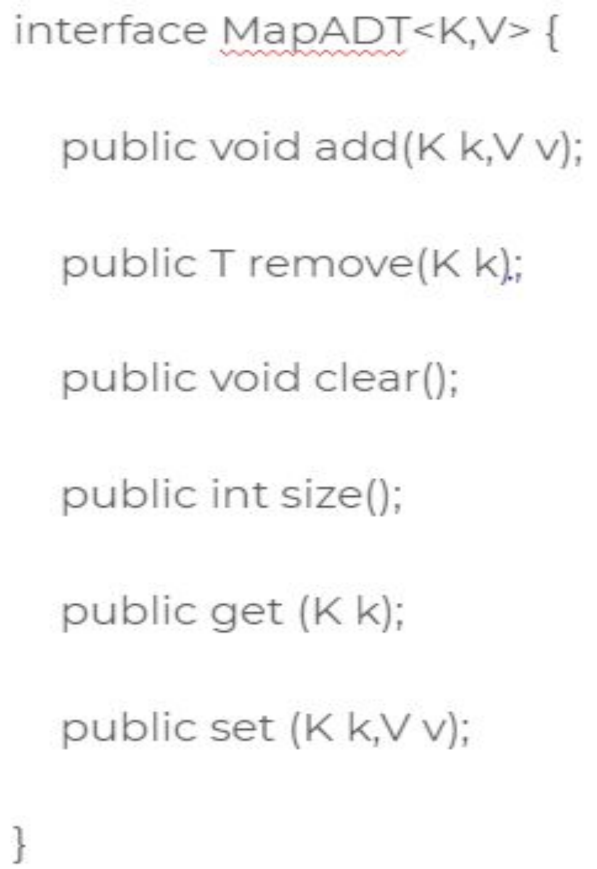
```
interface SetADT<T> {
    public void add(T t);
    public T remove(T t);
    public void clear();
    public int size();
    public get (int index);
    public set (int index,T t);
}
```

Set Example



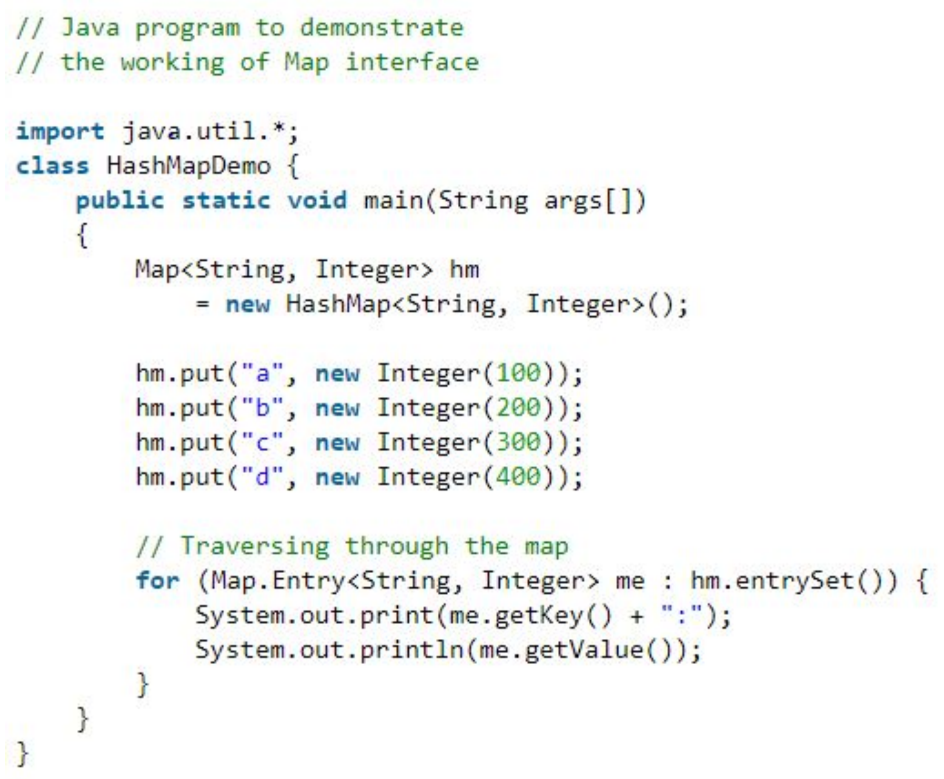
```
// Java program Illustrating Set Interface

// Importing utility classes
import java.util.*;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Demonstrating Set using HashSet
        // Declaring object of type String
        Set<String> hash_Set = new HashSet<String>();

        // Adding elements to the Set
        // using add() method
        hash_Set.add("Geeks");
        hash_Set.add("For");
        hash_Set.add("Geeks");
        hash_Set.add("Example");
        hash_Set.add("Set");

        // Printing elements of HashSet object
        System.out.println(hash_Set);
    }
}
```

Map Template



```
interface MapADT<K,V> {

    public void add(K k,V v);

    public T remove(K k);

    public void clear();

    public int size();

    public get (K k);

    public set (K k,V v);

}
```

Map Example



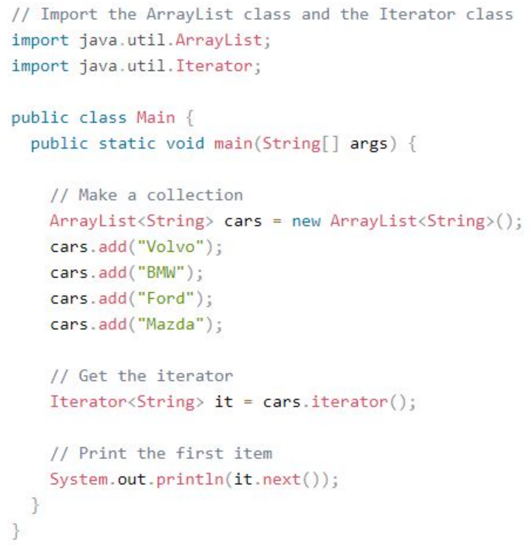
```java
// Java program to demonstrate
// the working of Map interface

import java.util.*;
class HashMapDemo {
    public static void main(String args[])
    {
        Map<String, Integer> hm
            = new HashMap<String, Integer>();

        hm.put("a", new Integer(100));
        hm.put("b", new Integer(200));
        hm.put("c", new Integer(300));
        hm.put("d", new Integer(400));

        // Traversing through the map
        for (Map.Entry<String, Integer> me : hm.entrySet()) {
            System.out.print(me.getKey() + ":");
            System.out.println(me.getValue());
        }
    }
}
```

Iterator in JAVA

An Iterator is an object **that can be used to loop through collections**, like ArrayList and HashSet. It is called an "iterator" because "iterating" is the technical term for looping. To use an Iterator, you must import it from the java. util package.

The iterator method can be used to get an Iterator for any collection.

Implementation of Iterator --------------------->



```java
// Import the ArrayList class and the Iterator class
import java.util.ArrayList;
import java.util.Iterator;

public class Main {
  public static void main(String[] args) {

    // Make a collection
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");

    // Get the iterator
    Iterator<String> it = cars.iterator();

    // Print the first item
    System.out.println(it.next());
  }
}
```

List methods

```
List listA = new ArrayList();
List listB = new LinkedList();
List listC = new Vector();
List listD = new Stack();

list.add(element);

list.add(0, "element 4"); // Insert at a specific index

// Copy from another list
List<String> listSource = new ArrayList<>();
listSource.add("123");
listSource.add("456");
List<String> listDest   = new ArrayList<>();
listDest.addAll(listSource);

String element1 = listDest.get(1); // Get an element

// Search for an element
int index1 = list.indexOf("element 1"); // first occurence
int index2 = list.lastIndexOf("element 1"); // last occurence

// Check if element exists
boolean containsElement = list.contains("element 1");
```

```
list.remove(element); // Remove an element by object
list.remove(0); // Remove an element by index
list.clear(); // Remove all elements from a list

int size = list.size(); // Get list size

List<String> sublist = list.subList(1, 3); // Sublist of a List

import java.util.Collections;
Collections.sort(list); // Sort a list

// Iterate a list using Iterator
import java.util.*;
Iterator<String> iterator = list.iterator();
while(iterator.hasNext()) {
   String next = iterator.next();
}

// Iterate a list using a for-each loop
for(String element : list) {
   System.out.println(element);
}

// Iterate a list using a for loop
for(int i=0; i < list.size(); i++) {
   String element = list.get(i);
}
```

Set methods

```
Set setA = new HashSet();
Set setB = new LinkedHashSet();
Set setC = new TreeSet();

// Unmodifiable (immutable) Set
Set<String> set = Set.<String>of("val1", "val2", "val3");

setA.add("element 1"); // Add an element

// Copy from another set
Set<String> set1 = new HashSet<>();
set1.add("one");
set1.add("two");
set1.add("three");
Set<String> set2 = new HashSet<>();
set2.add("four");
set2.addAll(set);

// Check if element exists
boolean contains123 = set1.contains("123");
set1.remove("object-to-remove"); // Remove an element
set1.clear(); // Remove all elements

// Remove all elements present in another set
set1.removeAll(set2);
```

```
// Retain (keep) all elements present in another set
set1.retainAll(set2);

// Get set size
int size = set1.size();

// Check if set is empty
boolean isEmpty = set1.isEmpty();

// Iterate a list using Iterator
import java.util.*;
Iterator<String> iterator = set1.iterator();
while(iterator.hasNext()) {
   String next = iterator.next();
}

// Iterate a set using a for-each loop
for(String element : set1) {
   System.out.println(element);
}

// Iterate a set using a for loop
for(int i=0; i < set1.size(); i++) {
    String element = set1.get(i);
}
```

Map methods

```java
// Map creation
Map mapA = new HashMap();
Map mapB = new TreeMap();

// Add an element
Map<String, String> map = new HashMap<>();
map.put("key1", "element 1");
map.put(null, "value for null key");
map.put("D", null);

// Get an element (by key)
String element1 = (String) map.get("key1");
String value = map.get(null);
String value = map.get("D");

// Get or Default Value
String value = map.getOrDefault("E", "default value");

// Check if key exists
boolean hasKey = map.containsKey("123");

// Check if value exists
boolean hasValue = map.containsValue("value 1");
```

```java
// Remove an element
map.remove("key1");

// Remove all elements
map.clear();

// Replace an entry in a map
map.replace("key", "val2"); // value of "key" replaced

// Get map size
int entryCount = map.size();

// Check if map is empty
boolean isMapEmpty = map.isEmpty();

// Inserting elements from another Map
Map<String, String> map1 = new HashMap<>();
map1.put("key1", "value1");
map1.put("key2", "value2");
Map<String, String> map2 = new HashMap<>();
map2.putAll(map1);
```

Map methods : Iteration

```java
// Iterating the Keys of a Java Map
// Iterating the Values of a Java Map
// Iterating the Entries of a Java Map

// Iterate the keys of a map using Iterator
Iterator iterator = map.keySet().iterator();
while(iterator.hasNext() {
   String key   = iterator.next();
   String value = map.get(key);
}

// Iterate the values of a map using Iterator
Iterator<String> iterator = map.values().iterator();
while(iterator.hasNext()) {
   String nextValue = iterator.next();
}

// Iterate the entries of a map using Iterator
Set<Map.Entry<String, String>> entries = map.entrySet();
Iterator<Map.Entry<String, String>> iterator = entries.iterator();
while(iterator.hasNext()) {
   Map.Entry<String, String> entry = iterator.next();
   String key   = entry.getKey();
   String value = entry.getValue();
}
```

```java
// Iterate a map using a for-each loop with keys
for(String key : map.keySet()) {
   String value = map.get(key);
}
// Iterate a map using a for-each loop with values
for(String value : map.values()) {
   System.out.println(value);
}
// Iterate a map using a for-each loop with entries
for(Map.Entry<String, String> entry : map.entrySet()){
   String key = entry.getKey();
   String value = entry.getValue();
}
```

Warm-up

Write a program to create a List of animals. Add the animals, remove the animals and update the animals.

Warm-up solution

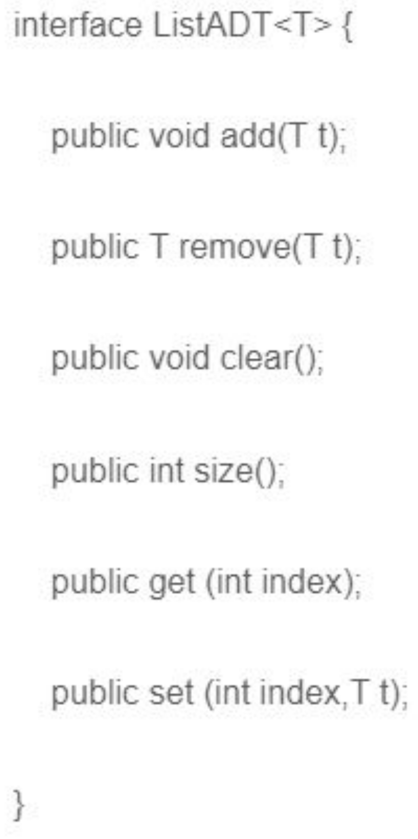
```java
public class Animal {
   String name;
   String species;
   int age;

   public Animal () {}

   public Animal (String name, String species, int age){
       this.name = name;
       this.species = species;
       this.age = age;
   }

   @Override
   public String toString() {
       return String.format("Hi, I am "+name+", I am a "+species
                +" and I am "+age+" years old.\n");
   }
}
```

```java
import java.util.List;
import java.util.ArrayList;

public class Main {

   public static void main(String[] args) {

       List<Animal> animals = new ArrayList<>();

       Animal spot = new Animal("Spot", "Panda", 2);
       Animal elle = new Animal("Elle", "Elephant", 3);

       animals.add(spot);
       animals.add(elle);
       System.out.println("Printing List after add:\n" + animals);

       animals.remove(elle);
       System.out.println("Printing List after remove:\n" + animals);

       animals.get(0).age = 4;
       System.out.println("Printing List after update:\n" + animals);
   }
}
```

Exercise 1(List)

Implement your own List class regarding to the interface shown at right side.

Hint: You can extend the warm-up exercise and apply polymorphism.



```
interface ListADT<T> {

   public void add(T t);

   public T remove(T t);

   public void clear();

   public int size();

   public get (int index);

   public set (int index,T t);

}
```

Exercise 1 solution:

```java
interface ListADT<T> {

      public void add(T t);

      public T remove(int index);

      public void clear();

      public int size();

      public T get(int index);

      public T set(int index, T t);
}
```

Exercise 1 solution:

```java
import java.util.Arrays;
import java.lang.IndexOutOfBoundsException;

@SuppressWarnings("unchecked")
public class ListADTImpl<T> implements ListADT<T> {

   private int size = 0;
   private T[] array;
   private int DEFAULT_CAPACITY = 10;

   public ListADTImpl() {
     size = 0;
     array = (T[]) new Object[DEFAULT_CAPACITY];
   }

   public void add(T t) {
       if(this.size==array.length) { growDouble(); }
       array[size] = t;
       size++;
   }

   public T remove(int index) throws IndexOutOfBoundsException {
       if(index<0 || index>=this.size) throw
               new IndexOutOfBoundsException();
       T returnElement = array[index];
       for(int i = index + 1; i < this.size; i++) {
           array[i-1]=array[i];
       }
       array[this.size]=null;
       this.size--;
       return returnElement;
   }

   public void clear() {
       Arrays.fill(array, null);
       size = 0;
   }

   public int size(){
       return this.size;
   }
```

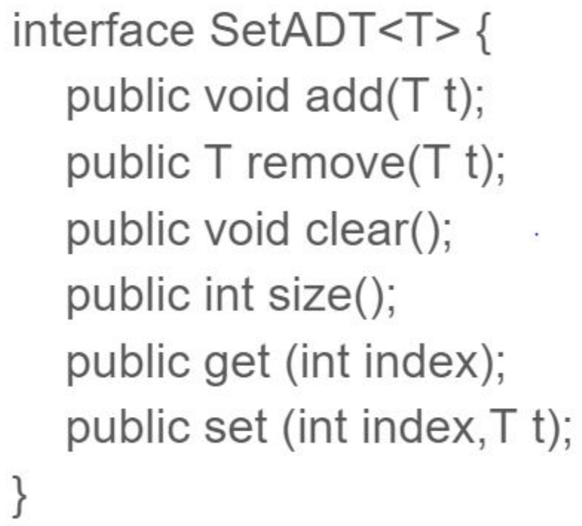
```java
   public T get(int index) throws IndexOutOfBoundsException{
       if(index<0 || index>=this.size) throw
                new IndexOutOfBoundsException();
       return array[index];
   }

   public T set(int index, T element) throws IndexOutOfBoundsException{
       if(index<0 || index >= this.size) throw new
                IndexOutOfBoundsException();
       T returnElement = array[index];
       array[index] = element;
       return returnElement;
   }


   /*
   * Extends the array capacity by double by initializing
   * a new array of size 2 times original array.
   */
   private void growDouble() {
       int arrayLength = array.length;
       T[] tempArray = (T[])new Object[2 * arrayLength];
       for(int i=0;i<this.size;i++){
           tempArray[i] = array[i];
       }
       array = tempArray;
   }
}
```

Exercise 1 solution:

```
public class ListADTImplTest{
    public static void main(String[] args){
       ListADTImpl<Integer> newList = new ListADTImpl<Integer>();
       System.out.println(newList.size());
       newList.add(1);
       newList.add(2);
       newList.add(3);
       newList.add(4);
       System.out.println("Number of elements present in list is "+newList.size());
       newList.add(6);
       System.out.println("Number of elements present in list is "+newList.size());
       System.out.println(newList.get(4));
       System.out.println(newList.set(4,9));
   }
}
```

Exercise 2 (Set)

Implement your own Set class regarding to the interface shown at right side.

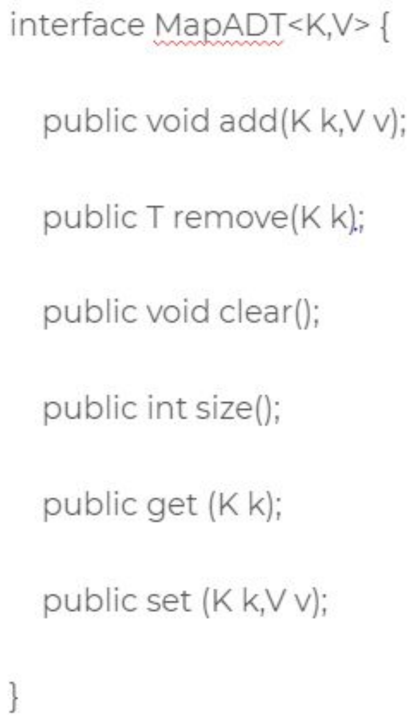


```
interface SetADT<T> {
    public void add(T t);
    public T remove(T t);
    public void clear();
    public int size();
    public get (int index);
    public set (int index,T t);
}
```

Exercise 3 (Map)

Implement your own Map class regarding to

the interface shown at right side.



```
interface MapADT<K,V> {

   public void add(K k,V v);

   public T remove(K k);

   public void clear();

   public int size();

   public get (K k);

   public set (K k,V v);

}
```

Exercise 4

Write a program which contains a main function and demonstrates the correct implementation of the aforementioned classes.

Create an instance of every implemented class and evaluate the functionality of the data structure.

References

- geeksforgeeks.org/list-interface-java-examples/
- https://www.geeksforgeeks.org/set-in-java/
- https://www.geeksforgeeks.org/map-interface-java-examples/
- https://docs.oracle.com/javase/7/docs/api/java/util/Set.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Map.html