Generics Chapter

ArrayList & Stack Lab

Contents

[Intro 1](#_Toc100561791)

[Notes 1](#_Toc100561792)

[ArrayList 1](#_Toc100561793)

[Stack 4](#_Toc100561794)

[Learning Goals 9](#_Toc100561795)

[Task 9](#_Toc100561796)

[Submit 9](#_Toc100561797)

Intro:

Learning about different data structures will help you with implementing efficient and optimized solutions for the real-world problems that you will be solving. Data Structures are one of the most fundamental subjects in Computer Science. It is the basis to develop your thinking in terms of code. Learning about different data structures can give you a massive edge over the application or website you'll be building.

A majority of self-taught programmers skip the fundamentals of Computer Science. They jump directly to the development process and the most in-demand skill in the industry. The most popular technologies can change overnight, but the fundamentals like Data Structures and Algorithms stay.

These subjects are the core of the industry, and learning about them would give you an edge over other developers who skip them mindlessly.

Data Structures are the building blocks of any large scale software.

There are instances when you should use an array, and there are places where a hash table would shine. I have seen a lot of beginner programmers never use stacks and queues. But in reality, they are massively used in the industry.

Secondly, apart from data structures, you must learn the basics of algorithms. If you know how to write recursive solutions, you make your code significantly shorter and elegant.

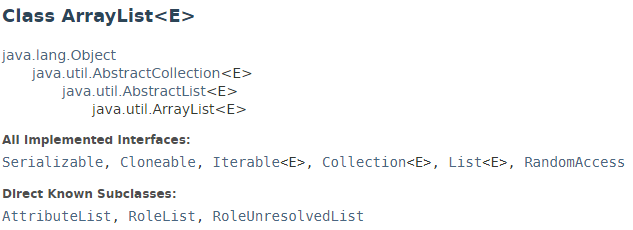
Notes:

ArrayList:

An ArrayList is a data structure; it is a resizable “array” with useful predefined methods.

It holds a group of like elements – reference data types; it does this by utilizing Generics.

ArrayList class implements List interface and it is based on the Array data structure. ArrayList can dynamically grow and shrink after addition and removal of elements; whereas, arrays are fixed. In addition to these benefits ArrayList class enables us to use predefined methods of it which makes our task of programming easier.



* ArrayList inherits AbstractList class (which inherits Collection) and implements List interface.
* ArrayList is initialized by a size, however the size can increase if collection grows or shrinks as objects are removed from the collection.
* capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically.
* ArrayList **cannot** be used for primitive types, like int, char, etc.
* ArrayList in Java can be seen as similar to vector in C++.

Advantages of ArrayLists:

* Resizable – dynamically grows and shrinks as elements are added or removed
* Can store an unlimited number of object elements
* Permits all reference data types, allows null.
* Can be accessed in order
* Can store an object in a specified spot
* Can remove an object, remove an object at a specified spot, and remove all objects

ArrayList is known as a generic class with a generic type E. You specify the concrete class type to replace the E when instantiating a new ArrayList.

Constructors in Java ArrayList:

1. ArrayList(): Constructs an empty list with an initial capacity of ten, size of zero
2. ArrayList(Collection c): Constructs a list containing the elements (from Collection c) of the specified collection, in the order they are returned by the Collection's iterator.
3. ArrayList(int capacity): This constructor is used to build an array list with initial capacity being specified

Syntax for instantiation of an ArrayList:

ArrayList<ClassDataType> indentifier =

new ArrayList<ClassDataType>();

or

ArrayList<ClassDataType> indentifier = new ArrayList<>();

// type inference allows this statement to compile

Example:

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

ArrayList <String> listOfWords = new ArrayList< >();

Methods in ArrayList: <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

Methods inherited from class java.util.AbstractList

* equals
* hashCode

Methods inherited from class java.util.AbstractCollection

* containsAll
* toString

Methods inherited from class java.lang.Object

* finalize
* getClass
* notify
* notifyAll
* wait
* wait
* wait

Methods inherited from interface java.util.List

* containsAll
* equals
* hashCode

Methods inherited from interface java.util.Collection

* parallelStream
* stream

Textbook Program:

public class TestArrayList {

public static void main(String[] args) {

// Create a list to store cities

ArrayList<String> cityList = new ArrayList<String>();

// cityList is empty, has a size of 0, and a capacity of 10

// Add some cities in the list

cityList.add("London");

// cityList now contains [London]

cityList.add("Denver");

// cityList now contains [London, Denver]

cityList.add("Paris");

// cityList now contains [London, Denver, Paris]

cityList.add("Miami");

// cityList now contains [London, Denver, Paris, Miami]

cityList.add("Seoul");

// contains [London, Denver, Paris, Miami, Seoul]

cityList.add("Tokyo");

// contains [London, Denver, Paris, Miami, Seoul, Tokyo]

System.out.println("List size? " + cityList.size());

System.out.println("Is Miami in the list? " +

cityList.contains("Miami"));

System.out.println("The location of Denver in the list? "

+ cityList.indexOf("Denver"));

System.out.println("Is the list empty? " +

cityList.isEmpty()); // Print false

// Insert a new city at index 2

cityList.add(2, "Xian");

// contains [London, Denver, Xian, Paris, Miami, Seoul, Tokyo]

// Remove a city from the list

cityList.remove("Miami");

// contains [London, Denver, Xian, Paris, Seoul, Tokyo]

// Remove a city at index 1

cityList.remove(1);

// contains [London, Xian, Paris, Seoul, Tokyo]

// Display the contents in the list

System.out.println(cityList.toString());

// Display the contents in the list in reverse order

for (int i = cityList.size() - 1; i >= 0; i--)

System.out.print(cityList.get(i) + " ");

System.out.println();

// Create a list to store two circles

java.util.ArrayList<CircleFromSimpleGeometricObject> list

= new java.util.ArrayList<CircleFromSimpleGeometricObject>();

// Add two circles

list.add(new CircleFromSimpleGeometricObject(2));

list.add(new CircleFromSimpleGeometricObject(3));

// Display the area of the first circle in the list

System.out.println("The area of the circle? " +

((CircleFromSimpleGeometricObject)list.get(0)).getArea());

}

}

Stack:

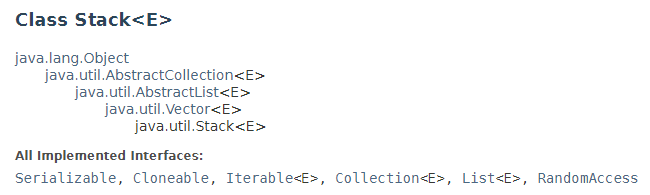
The Stack class represents a last-in-first-out (LIFO) stack of objects. It extends class Vector with five operations that allow a vector to be treated as a stack. The usual push and pop operations are provided, as well as a method to peek at the top item on the stack, a method to test for whether the stack is empty, and a method to search the stack for an item and discover how far it is from the top.

When a stack is first created, it contains no items.

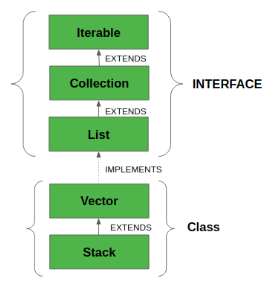
Note: Please note that the Stack class in Java is a legacy class and inherits from Vector in Java. It is a thread-safe class and hence involves overhead when we do not need thread safety. It is recommended to use ArrayDeque for stack implementation as it is more efficient in a single-threaded environment.

A more complete and consistent set of LIFO stack operations is provided by the Deque interface and its implementations, which should be used in preference to this class. For example:

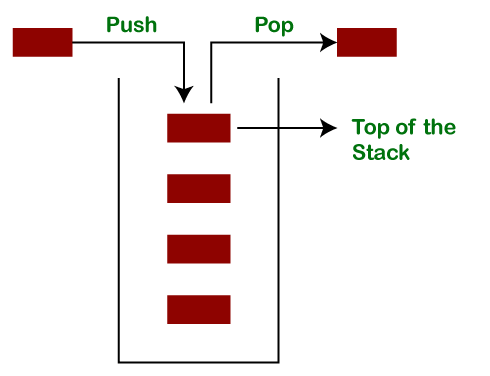
Deque<Integer> stack = new ArrayDeque<Integer>();



The below diagram shows the hierarchy of the Stack class:



To instantiate/create a stack: Stack <E> myStack = new Stack<>();



Methods in Stack Class

empty() It returns true if nothing is on the top of the stack. Else, returns false.

peek() Returns the element on the top of the stack, but does not remove it.

pop() Removes and returns the top element of the stack. An ‘EmptyStackException’

An exception is thrown if we call pop() when the invoking stack is empty.

push(Object element) Pushes an element on the top of the stack.

search(Object element) It determines whether an object exists in the stack.

If the element is found,

It returns the position of the element from the top of the stack.

Else, it returns -1.

Methods inherited from class java.util.Vector

add(Object obj) Appends the specified element to the *end* of this Vector.

add(int index, Object obj) Inserts the specified element at the *specified* *position* in this Vector.

addAll(Collection c) Appends all of the elements in the specified Collection to the *end* of this Vector, in the order that they are returned by the specified Collection’s Iterator.

addAll(int index, Collection c) Inserts all the elements in the specified Collection into this Vector at the specified position.

addElement(Object o) Adds the specified component to the end of this vector, increasing its size by one.

capacity() Returns the current capacity of this Vector.

clear() Removes all the elements from this Vector.

clone() Returns a clone of this Vector.

contains(Object o) Returns true if this vector contains the specified element.

containsAll(Collection c) Returns true if this Vector contains all the elements in the specified Collection.

copyInto(Object []array) Copies the components of this Vector into the specified array.

elementAt(int index) Returns the component at the specified index.

elements() Returns an enumeration of the components of this Vector.

ensureCapacity(int minCapacity) Increases the capacity of this Vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

equals() Compares the specified Object with this Vector for equality.

firstElement() Returns the first component (the item at index 0) of this Vector.

get(int index) Returns the element at the specified position in this Vector.

hashCode() Returns the hash code value for this Vector.

indexOf(Object o) Returns the index of the *first* occurrence of the specified element in this vector, or -1 if this Vector does not contain the element.

indexOf(Object o, int index) Returns the index of the *first* occurrence of the specified element in this Vector, *searching forwards* from the index, or returns -1 if the element is not found.

insertElementAt(Object o, int index) Inserts the specified object as a component in this vector at the specified index.

isEmpty() Tests if this Vector has no components.

iterator() Returns an iterator over the elements in this list in proper sequence.

lastElement() Returns the last component of the Vector.

lastIndexOf(Object o) Returns the index of the *last* occurrence of the specified element in this vector, or -1 If this Vector does not contain the element.

lastIndexOf(Object o, int index) Returns the index of the last occurrence of the specified element in this Vector, *searching backward* from the index, or returns -1 if the element is not found.

listIterator() Returns a list iterator over the elements in this list (in proper sequence).

listIterator(int index) Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.

remove(int index) Removes the element at the specified position in this Vector.

remove(Object o) Removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.

removeAll(Collection c) Removes from this Vector all of its elements that are contained in the specified Collection.

removeAllElements() Removes *all* components from this vector and sets its size to zero.

removeElement(Object o) Removes the *first* (lowest-indexed) occurrence of the argument from this vector.

removeElementAt(int index) Deletes the component at the specified index.

removeRange(int fromIndex, int toIndex) Removes from this list all the elements whose index is between fromIndex, *inclusive*, and toIndex, *exclusive*.

retainAll(Collection c) Retains only the elements in this Vector that are contained in the specified Collection.

set(int index, Object o) Replaces the element at the specified position in this Vector with the specified element.

setElementAt(Object o, int index) Sets the component at the specified index of this Vector to be the specified object.

setSize(int newSize) Sets the size of this Vector.

size() Returns the number of components in this Vector.

subList(int fromIndex, int toIndex) Returns a view of the portion of this List between fromIndex, *inclusive*, and toIndex, *exclusive*.

toArray() Returns an array containing all of the elements in this Vector in the correct order.

toArray(Object []array) Returns an array containing all of the elements in this Vector in the correct order; the runtime type of the returned array is that of the specified array.

toString() Returns a string representation of this Vector, containing the String representation of each element.

trimToSize() Trims the capacity of this vector to be the vector’s current size.

Example:

public class StackExample {

public static void main(String[] args) {

//creating an instance of Stack class

//Stack<Integer> stk= new Stack<>();

Deque<Integer> stk = new ArrayDeque<Integer>();

// checking stack is empty or not

boolean isStackEmpty= stk.empty();

System.out.println("Is the stack empty? " + isStackEmpty);

// pushing elements into stack

stk.push(20);

stk.push(13);

stk.push(89);

stk.push(90);

stk.push(11);

stk.push(45);

stk.push(18);

//print elements of the stack

System.out.println("Elements in Stack: " + stk);

//repeating - check to see if the Stack is empty

boolean isStackEmpty = stk.empty();

System.out.println("Is the stack empty? " + isStackEmpty);

// pop elements of the stack

stk.pop(); // pops off 18

stk.pop(); // pops off 45

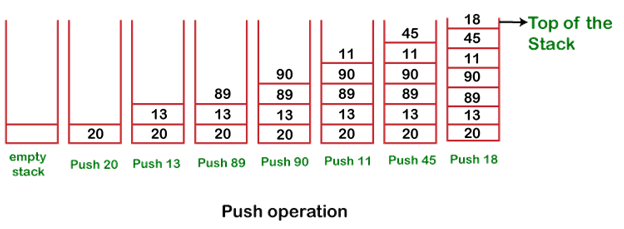
Integer num11 = stk.pop();// pops off 11

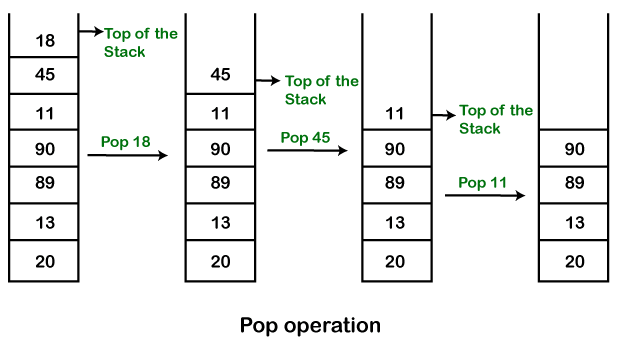
System.out.println("element popped off: " + num11);

} // end main

} // end StackExample class

What is happening in memory?





Learning Goals:

To understand and utilize two basic data structures: Arraylist and Stack.

Task:

1. Download A\_S\_Lab.java and rename it with mySFAusername\_A\_S\_Lab.java.
2. Instantiate an ArrayList of CoffeeCup objects
   1. add at least 3 CoffeeCup objects
   2. Print the logo of each CoffeeCup object.
3. Instantiate a DequeStack of CoffeeCup objects
   1. Push at least 3 CoffeeCup
   2. Print the logo of each CoffeeCup object.

Submit: Submit mySFAusername\_A\_S\_Lab.java in the Dropbox in Brightspace by D2L.