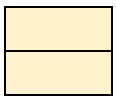
1 EX 12.1 Compare and Contrast data types, abstract data types and data structures.

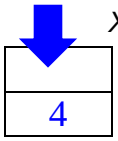
	Data Types	Data Structures	Abstract Data Type
Differences	Categories in which the variables are listed to hold the specific values.	The way in which values and variables are stored in memory. The gathering together of many different data types.	User defined data type model which is designed by programmer to store complex data
Similarities	Data of primitive types	Grouping, or cluster of data of primitive types, including objects.	Special form of data that is derived from data types and data structures.

2 EX 12.4 Hand Trace an initially empty stack X thought the following operations:

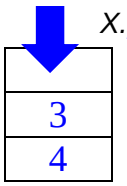
(Assumption: The stack is initially empty)



X.push(new Integer(4));

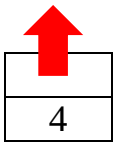


X.push(new Integer(3));

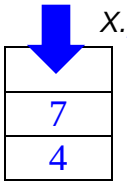


Integer Y = X.pop();

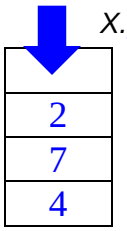
3



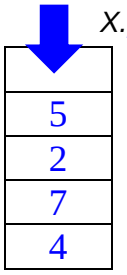
X.push(new Integer(7));



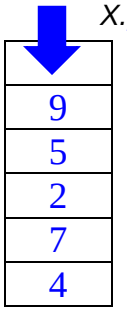
X.push(new Integer(2));



X.push(new Integer(5));

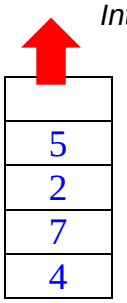


X.push(new Integer(9));

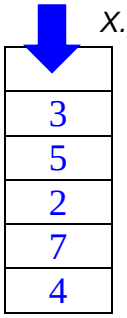


Integer Y = X.pop();

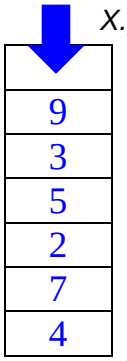
9



X.push(new Integer(3));

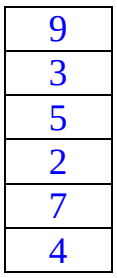


X.push(new Integer(9));



3 EX 12.5 Given the resulting stack X from the previous exercise, what would be the result of each of the following?

a. Y = X.peek(); Y = 9, the stack is unchanged.



b. Y = X.pop(); Y = 9, 9 is removed from the stack.

~~9~~
3
5
2
7
4

Z = X.peek(); Z = 3, the stack is unchanged.

~~9~~
3
5
2
7
4

c. Y = X.pop(); Y = 3, 3 is removed from the stack.

~~9~~
~~3~~
5
2
7
4

Z = X.peek(); Z = 5, the stack is unchanged.

~~9~~
~~3~~
5
2
7
4

Final stack

5
2
7
4

4 EX 12.7 Show how the undo operation in a word processor can be sup- ported by the use of a stack. Give specific examples and draw the contents of the stack after various actions are taken.

a. The UNDO stack is initially empty

b. The user selects some text and changes the font from Arial Courier

stack = |Arial-To-Courier (selection)|

c. Then the user increases the font size from 12 to 14.

stack = |12-to-14(selection)|Arial-To-Courier (selection)|

d. Then the user changes the color from red to green.

stack = |red-to-green (selection)|12-to-14(selection)|Arial-To-Courier (selection)|

e. NOW, the user uses the undo function. The last entered action from the stack is popped and a reverse action is taken...

action popped = red-to-green

reversed action = green-to-red.

f. Selection's color is reverted back to red.

stack = |12-to-14(selection)|Arial-To-Courier (selection)|

g. The user presses undo again,

action popped = 12-to-14(selection)

reversed action = 14-to-12 font size

stack = |Arial-To-Courier (selection)|

5 Complete the implementation of the ArrayStack class presented in this chapter. Specifically, complete the implementations of the peek, isEmpty, size, and toString methods.

See attached file name *ArrayStack.java*

The code below is for reference.

```
package jsjf;

import jsjf.exceptions.*;
import java.util.Arrays;

/**
 * An array implementation of a stack in which the bottom of the
 * stack is fixed at index 0.
 *
 * @author Java Foundations
 * @version 4.0
 */
public class ArrayStack<T> implements StackADT<T>
{
    private final static int DEFAULT_CAPACITY = 100;

    private int top;
    private T[] stack;

    /**
     * Creates an empty stack using the default capacity.
     */
    public ArrayStack()
    {
        this(DEFAULT_CAPACITY);
    }

    /**
     * Creates an empty stack using the specified capacity.
     * @param initialCapacity the initial size of the array
     */
    public ArrayStack(int initialCapacity)
    {
        top = 0;
        stack = (T[])(new Object[initialCapacity]);
    }

    /**
     * Adds the specified element to the top of this stack, expanding
     * the capacity of the array if necessary.
     * @param element generic element to be pushed onto stack
     */
    public void push(T element)
    {
        if (size() == stack.length)
```

```
            expandCapacity();

        stack[top] = element;
        top++;
    }

    /**
     * Creates a new array to store the contents of this stack with
     * twice the capacity of the old one.
     */
    private void expandCapacity()
    {
        stack = Arrays.copyOf(stack, stack.length * 2);
    }

    /**
     * Removes the element at the top of this stack and returns a
     * reference to it.
     * @return element removed from top of stack
     * @throws EmptyCollectionException if stack is empty
     */
    public T pop() throws EmptyCollectionException
    {
        if (isEmpty())
            throw new EmptyCollectionException("stack");

        top--;
        T result = stack[top];
        stack[top] = null;

        return result;
    }

    /**
     * Returns a reference to the element at the top of this stack.
     * The element is not removed from the stack.
     * @return element on top of stack
     * @throws EmptyCollectionException if stack is empty
     */
    public int peek() throws EmptyCollectionException
    {
        if (isEmpty())
            throw new EmptyCollectionException("stack");

      //Force this return value to int, HQP, 02/21/2015
        return((int) stack[top-1]);
    }

    /**
     * Returns true if this stack is empty and false otherwise.
     * @return true if this stack is empty
     */
    public boolean isEmpty()
    {
        // If top is beyond 0, it is empty, Hieu, 02/21/2015
```

```
        if(top==-1)
             return(true);
         else
             return(false);
    }

    /**
     * Returns the number of elements in this stack.
     * @return the number of elements in the stack
     */
    public int size()
    {
        return(top); //Return the size, Hieu, 02/21/2015
    }

    /**
     * Returns a string representation of this stack.
     * @return a string representation of the stack
     */
    public String toString()
    {
        // To be completed as a Programming Project
       return("Size = " + stack.length + "\n"
                    + "top = " + this.top + "\n"
                    + "Capacity = " + ((this.isEmpty()) ? "Empty" : "Still
                    good") + "\n" + "First element is: " + stack[0] + "\n"
                    + "Last element is: " + stack[top-1]);
    }
}
```

6 PP12.4 The array implementation in this chapter keeps the top variable pointing to the next array position above the actual top of the stack. Rewrite The array implementation such that stack[top] is the actual top of the stack.

```
    public int peek() throws EmptyCollectionException
    {
        if (isEmpty())
            throw new EmptyCollectionException("stack");

      //Change to point to top, HQP, 02/21/2015
        return((int) stack[top]);
    }
```

See attached file name *ArrayStack.java*