TITLE: JAVA NIO CHANNELS AND BUFFERS

OBJECTIVES:

→ To identify and list the supported socket options for various java NIO ~~the~~ network channels.

→ To implement and demonstrate core NIO Buffer manipulations, specifically: filling, draining, duplicating slicing and compacting.

THEORY:

Non Blocking I/O (NIO)

→ Java NIO offers an alternative to the standard Java I/O API. Unlike standard I/O, which is stream-oriented and blocking.

→ NIO is buffer oriented and non-blocking.

Channel

→ It represents an open connection to an I/O entity, such as a hardware device, a file, a network socket etc.

→ It is bidirectional.

→ Data is always read from a channel ~~into~~ into buffer, or written from a buffer into channel.

→ Types: SocketChannel, DatagramChannel
ServerSocketChannel

Buffer

→ A buffer is a container for a fixed amount of data of a specific primitive type.

→ It acts as an endpoint for sending and receiving data in NIO.

→ Filling : writing data into the buffer (put())

→ Draining : Reading data out of buffer (get())

Properties * Capacity : total size

Position : index of next element to be read or written

Limit : index of first element that should not be read or written.

Selector

→ It is multiplexor of SelectableChannel obj objects. It allows a single thread to monitor multiple channels for events.

SOURCE CODE:

```
package lab7:
import java.io.*;
import java.net.*;
import java.nio.*;
import java.util.Set;
```

```
public class Solution{
    public static void main (string[] args){
        listAllSocket();
        fillingAndDraining();
    }

    private static void listAllSocket(){
      try{
        SocketChannel sc = SocketChannel.open();
        printOptions ("SocketChannel", sc.supportedOptions());
        sc.close();

        ServerSocketChannel ssc = ServerSocketChannel.open();
        printOptions("ServerSocketChannel", ssc.supportedOptions());
        ssc.close();

        DatagramChannel dc = DatagramChannel.open()
        printoptions(" DatagramChannel",
                    dc.supportedOptions());

        dc.close();
    }
    close catch (IOException e){
        System.out.println (e.getMessage());
    }
}
```

```
private static void printOptions(String channelType,
        Set<SocketOption<?>> options){
    System.out.println("Supported options for: " + channelType);
    for (SocketOption<?> o : options){
        System.out.println("-" + o.name());
    }
    System.out.println();
}
```

```
public static void fillingAndDraining(){
    CharBuffer buffer = CharBuffer.allocate(10);
    System.out.println("Initial state...");
    printStats(buffer);

    System.out.println("In filling Buffer (H,e,l,l,o)");
    buffer.put("Hello");
    printStats(buffer);

    System.out.println("Flipping (Prepare to Drain)");
    buffer.flip();
    printStats(buffer);

    System.out.println("Draining first 2 chars: "+ buffer.get()+
                                                    buffer.get());
    printStats(buffer);

    System.out.println("Duplicating...");
    CharBuffer dup = buffer.duplicate();
    System.out.println("Duplicate Buffer stats: "+dup);

    System.out.println("slicing");
    CharBuffer slice = buffer.slice();
    System.out.println("slice content:" + slice.toString());

    System.out.println("Compacting");
    buffer.compact();
```

```
        printStats(buffer);
        buffer.put("world");
        buffer.flip();
        System.out.println("Content after compact + write: " +
                                    buffer.toString());
    }

    private static void printStats(CharBuffer cb){
        System.out.println("Pos:" + cb.position() + ", Limit:"
                        + cb.limit(); ", Cap:" + cb.capacity());
    }
}
```

CONCLUSION:

→ In this lab, we explored the fundamentals of Java NIO.

→ we verified that different channels support different sets of options.

→ we successfully demonstrated filling, flipping, duplicating and slicing and compacting.