Lab 3: Memory

Professor: Ronaldo Menezes

TA: Ivan Bogun

Department of Computer Science
Florida Institute of Technology

September 3, 2014

1 Problem statement

In this lab the task will be to emulate memory using pointers. Memory will be represented as an array of bytes. Object stored in the memory will be represented via pointers. Object's pointer will be simply an index in the memory array where the object is being stored. For simplicity in this lab we assume that we will store only arrays of objects (single object can be seen as an array of size 1).

1.1 Memory

Assume we want to create 16 bytes of memory and store there array of integers $a = \{234, 301\}$ and doubles $b = \{13.5\}$.

Figure 1: When initialized memory array will be empty. The arrow shows the index where next object will be allocated.

Fig. 1 shows how the memory array will look like when initialized. Since memory consists of bytes in order to store integers a conversion has to be made. Each byte contains 8 bit while Integer 32, this means that in order to store one Integer we will need $\frac{32}{8} = 4$ bytes [1]. Functions to convert from Integer/Double/Float/Short/Char to byte arrays and back will be provided. When converted to byte arrays $234 = \{0, 0, 0, -22\}$ and $301 = \{0, 0, 1, 45\}$.

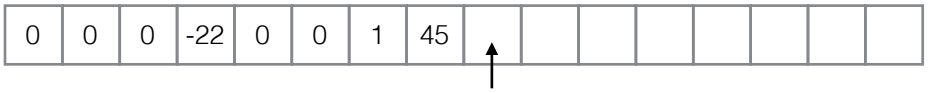

Figure 2: State of the memory after array $\{234, 301\}$ was stored

Similarly we convert double array b into array of bytes and store them into the memory array. In bytes $13.5 = \{64, 43, 0, 0, 0, 0, 0, 0\}$

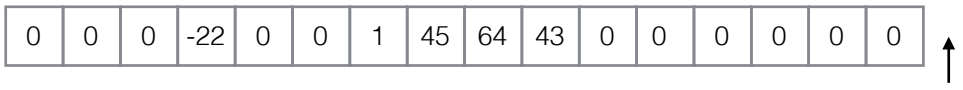

Figure 3: Memory state after insertion of b

[1] Java types

1.2 Pointers

In order to access specific element in the memory we need to know

- index in the memory array (to know where to start)
- type of the object to be accessed (to know how many bytes per element)
- size of the array to be accessed (to know how many can we read)

For example, if we were to access array b we would need to know the index where the first element is stored (8), type (Double) and size (1).

2 Implementation

Implement the class *Memory.java* [2].

```
// Memory.java

public class Memory {

        public byte[] memoryArray;
        public boolean[] occupiedMemory;            // boolean array
            representing which bytes are occupied

        public int indexOfFreeMemory=0;             //  index where next object
            will be stored in the memory


        public Memory(int size) {
        // constructor
        }

        int allocate(int num,Class<?> cls){
        // allocate num elements of type cls in the memory array. Index of the
            first element should be returned.
        // Class<?> cls is simply a way to pass class type in a variable
        }

        // convenience function - nothing to implement here
        int allocate(Object[] a){
                return allocate(a.length, a[0].getClass());
        }


        boolean copy(int pointer,Object[] objs){
```

[2] You can copy code for this file from: Memory.java

```
        // copy objs into memory starting at index pointer. True should be
            returned if it is possible (there is enough memory to store all the
            objects), false otherwise.
        }

        public byte[][] read(int pointer,Class<?> cls, int num){
        // read array of size num of the objects of type cls starting at
            pointer and return it
        }

        public void free(){
        // free everything from the memory
        }

        public void free(int pointer,Class<?> cls,int num){
        // free objects array of size num of type cls starting at pointer
        }

        public String toString() {
        // print out which elements of the memory array are occupied and which
            are not
        }

        String printLocationInMemory(int pointer,Class<?> cls, int num){
        // return a string representing location in the memory of object array
            of size num of type cls starting at pointer
        }


        public int addIntArrays(int pointer1,int pointer2,int num){
        //Given two pointers of Integer array of the same size add them,
            allocate and store result in the memory array and return it's
            pointer
        }


}
```

Class *TypeConverter.java* [3] contains functions necessary to convert different types to and from byte arrays. Nothing to implement here.

```
import java.nio.ByteBuffer;

public class TypeConverter {
```

[3] TypeConverter.java

```
public static byte[] toByteArray(Object value){

        Class<?> cls=value.getClass();
        byte[] bytes = new byte[numberOfBytesPerClass(cls)];

        if(cls==Double.class){
                ByteBuffer.wrap(bytes).putDouble((Double)value);
        }

        else if(cls==Short.class){
                ByteBuffer.wrap(bytes).putShort((Short)value);
        }

        else if(cls==Integer.class){
                ByteBuffer.wrap(bytes).putInt((Integer)value);
        }

        else if(cls==Long.class){
                ByteBuffer.wrap(bytes).putLong((Long)value);
        }

        else if(cls==Character.class){
                ByteBuffer.wrap(bytes).putChar((Character)value);
        }

        else if(cls==Float.class){
                ByteBuffer.wrap(bytes).putFloat((Float)value);
        }

        return bytes;
}

public static byte[] toByteArray(double value) {
        byte[] bytes = new byte[8];
        ByteBuffer.wrap(bytes).putDouble(value);
        return bytes;
}


public static byte[] toByteArray(short value) {
        byte[] bytes = new byte[2];
        ByteBuffer.wrap(bytes).putShort(value);
        return bytes;
}


public static byte[] toByteArray(char value) {
        byte[] bytes = new byte[2];
        ByteBuffer.wrap(bytes).putChar(value);
```

```
        return bytes;
}


public static byte[] toByteArray(int value) {
        byte[] bytes = new byte[4];
        ByteBuffer.wrap(bytes).putInt(value);
        return bytes;
}

public static byte[] toByteArray(float value) {
        byte[] bytes = new byte[4];
        ByteBuffer.wrap(bytes).putFloat(value);
        return bytes;
}


public static byte[] toByteArray(long value) {
        byte[] bytes = new byte[8];
        ByteBuffer.wrap(bytes).putLong(value);
        return bytes;
}

public static Double byteArrayToDouble(byte[] bytes){
        return ByteBuffer.wrap(bytes).getDouble();
}

public static Short byteArrayToShort(byte[] bytes){
        return ByteBuffer.wrap(bytes).getShort();
}

public static Integer byteArrayToInt(byte[] bytes){
        return ByteBuffer.wrap(bytes).getInt();
}

public static Long byteArrayToLong(byte[] bytes){
        return ByteBuffer.wrap(bytes).getLong();
}

public static Character byteArrayToChar(byte[] bytes){
        return ByteBuffer.wrap(bytes).getChar();
}

public static Float byteArrayToFloat(byte[] bytes){
        return ByteBuffer.wrap(bytes).getFloat();
}

public static int numberOfBytesPerClass(Class<?> cls){
```

```
                int numberOfBytes=0;

                if (cls==Short.class  ) {
                        numberOfBytes=2;
                }
                else if (cls==Integer.class) {
                        numberOfBytes=4;
                }
                else if (cls==Long.class) {
                        numberOfBytes=8;
                }
                else if (cls==Double.class) {
                        numberOfBytes=8;
                }
                else if (cls==Character.class) {
                        numberOfBytes=2;
                }
                else if (cls==Float.class) {
                        numberOfBytes=4;
                }
                else {
                        System.out.println("Unknown type");
                }

                return numberOfBytes;
        }

}
```

3 Sample input-output

Create the file *Driver.java* [4] whose modified version will be used for testing.

3.1 Input

```
public class Driver {

        public static void main(String[] args) {

                int n=32;
                Memory ram = new Memory(n);

                // things we will store in the memory
                Integer[] a={234,301};
```

[4] Driver.java

```
Double[] b={13.5};

System.out.println("Initially memory is free - all bytes are
    avaliable");
System.out.println(ram);

// allocate segment in the memory for a.
int a_p=ram.allocate(a);

// copy array a into the memory starting at pointer
boolean status=ram.copy(a_p, a);

// allocate segment for b and copy it inot
int b_p=ram.allocate(b);
ram.copy(b_p, b);


Integer[] a_back=new Integer[a.length];
Double[] b_back=new Double[b.length];

if (status) {

        // read from the memory
        byte[][] bytes=ram.read(a_p, Integer.class, a.length);

        for (int i = 0; i < a.length; i++) {
                a_back[i]=TypeConverter.byteArrayToInt(bytes[i]);
                System.out.println(a_back[i]);
        }

        bytes=ram.read(b_p, b[0].getClass(),b.length);
        for (int i = 0; i < b_back.length; i++) {
                b_back[i]=TypeConverter.byteArrayToDouble(bytes[i]);
                System.out.println(b_back[i]);
        }

        // print state of the memory
        System.out.println(ram);
}

// print location of bytes occupied by b
System.out.println("Location of the array b in the memory");
System.out.println(ram.printLocationInMemory(b_p, Double.class,
    b.length));


Integer[] c={-100,-200};

int c_p=ram.allocate(c);
```

```
                ram.copy(c_p, c);

                int sum_p=ram.addIntArrays(a_p, c_p, c.length);


                System.out.println(ram);
                byte[][] bytes=ram.read(sum_p, Integer.class, a.length);

                Integer[] sum=new Integer[2];

                System.out.println("Array sum:");
                for (int i = 0; i < sum.length; i++) {
                        sum[i]=TypeConverter.byteArrayToInt(bytes[i]);
                        System.out.println(sum[i]);
                }

                // print location of bytes occupied by the sum array
                System.out.println("Location of the array sum in the memory");
                System.out.println(ram.printLocationInMemory(sum_p,
                    Integer.class, sum.length));

        }

}
```

3.2 Output

```
Initially memory is free - all bytes are avaliable
oooooooooooooooooooooooooooooo
234
301
13.5
xxxxxxxxxxxxxxxxoooooooooooooo
Location of the array b in the memory
oooooooxxxxxxxxoooooooooooooooo
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Array sum:
134
101
Location of the array sum in the memory
ooooooooooooooooooooooooxxxxxxxx
```

4 Grade breakdown

basis	grade
Implementation	(60)
read & allocate	20
addIntArrays()	25
other	15
Comments	(20)
General	10
Javadocs	10
Overall	(20)
Compiled	5
Style	5
Runtime	10
Total	100