Moebius Solutions Application Questions

**================**

> VERSION: 2021-01-02

Name of Candidate: Laurence Alec M. Burce

QUESTION 1. Why does this code not work? How do you fix it?

**----------------**

```

public class Example {

    public List<Integer> removeBigNumbers(List<Integer> data) {

        for (Integer i : data) {

            if (i > 10) {

                data.remove(i);

            }

        }

        return data;

    }

}

```

**Question 1 Answer**

The problem with the code above is, it is not possible to modify the collection used in the for-each loop. For this example, the collection ‘data’ was used for the *for* loop and it was also modified within that loop from the *data.remove(i);* line.

To fix this problem, one method is we can avoid using the collection in the *for* loop by iterating over it manually:

public class Example {

    public List<Integer> removeBigNumbers(List<Integer> data) {

        //  A counter variable 'x' is used to iterate over the indices of 'data'

        for (int x = 0; x < data.size(); x++) {

            // Gets the value for the given index

            if (data.get(x) > 10) {

                // Modify the collection

                data.remove(x);

            }

        }

        return data;

    }

}

Alternatively, we can also use the Iterator class from the *java.util* package.

// Import the Iterator class

import java.util.Iterator;

public class Example2 {

    public List<Integer> removeBigNumbers(List<Integer> data) {

        // Initialize the iterator variable with the collection 'data'

        Iterator<Integer> iterator = data.iterator();

        // While loop to verify if there are more elements to iterate over

        while (iterator.hasNext()) {

            // Obtain the next element

            Integer i = iterator.next();

            // Check if it is greater than 10

            if (i > 10) {

                // Remove if it is

                iterator.remove();

            }

        }

        return data;

    }

}

QUESTION 2: Programming Task

**----------------**

Write a class that implements the following interface, assuming that all methods

are used with approximately the same frequency.

```

interface ItemStore {

    Interface Item {

        String getID();

        String getTag();

    }

    /\*\*

     \* Adds an {@link Item} to the store, replacing any existing item with the

     \* same {@link Item#id} value.

     \*/

    public void put(Item item);

    /\*\*

     \* Retrieves the {@link Item} with the given {@link Item#id} value, or

     \* null if no such {@link Item} exists in the store.

     \*/

    public Item get(String id);

    /\*\*

     \* Delete all {@link Item}s with the specified tag.

     \*/

    public void dropAllByTag(String tag);

    /\*\*

     \* Returns the number of Items in the store

     \*/

     public int size();

}

```

**Question 2 Answer**

For this problem, there was an issue on Line 3 of the given interface ItemStore. I had to correct Interface to interface since this keyword is case sensitive.

ItemStoreTest.java

public class ItemStoreTest implements ItemStore {

    private Map<String, Item> iMap;

    public ItemStoreTest() {

        this.iMap = new HashMap<>();

    }

    @Override

    public void put(Item item) {

        iMap.put(item.getID(), item);

    }

    @Override

    public Item get(String id) {

        return iMap.get(id);

    }

    @Override

    public void dropAllByTag(String tag) {

        iMap.values().removeIf(item -> item.getTag().equals(tag));

    }

    @Override

    public int size() {

        return iMap.size();

    }

    static class ItemTest implements Item {

        private String id;

        private String tag;

        public ItemTest(String id, String tag) {

            this.id = id;

            this.tag = tag;

        }

        @Override

        public String getID() {

            return id;

        }

        @Override

        public String getTag() {

            return tag;

        }

    }

}

Main.java

public class main {

    public static void main(String[] args) {

        ItemStoreTest store = new ItemStoreTest();

        // Create some items

        ItemStore.Item test1 = new ItemStoreTest.ItemTest("101", "Grapes");

        store.put(test1);

        ItemStore.Item test2 = new ItemStoreTest.ItemTest("102", "Apple");

        store.put(test2);

        ItemStore.Item test3 = new ItemStoreTest.ItemTest("103", "Banana");

        store.put(test3);

        ItemStore.Item test4 = new ItemStoreTest.ItemTest("104", "Apple");

        store.put(test4);

        ItemStore.Item test5 = new ItemStoreTest.ItemTest("105", "Pineapple");

        store.put(test5);

        // Retrieve the size of the store

        System.out.println("Size of the store: " + store.size());

        // Retrieve an item by ID

        System.out.println("Item with ID 102: " + store.get("102").getTag());

        // Retrieve an item by ID

        System.out.println("Item with ID 105: " + store.get("105").getTag());

        // Delete all items with tag "Apple"

        store.dropAllByTag("Apple");

        // Check store size

        System.out.println("Size of the store: " + store.size());

    }

}

QUESTION 3: Memory Management

**----------------**

The `SmallMemoryMessageTest` class below passes on our development machines,

but the client reports that it fails on their 64MB VM.

1. Run on a 64M VM and copy the failing stack trace.

2. Modify `main()` to work on a 64MB VM.

3. Ensure there are no more than one performance warnings.

To set 64Mb VM, run using `java -Xmx64M SmallMemoryMessageTest`.

```

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import java.util.function.Predicate;

/\*\*

\* The following mock program passes on our development machines,

\* but the client reports that it fails on their 64MB VM.

\*

\* 1. Run on a 64M VM and copy the failing stack trace

\* 2. Modify main() to work on a 64MB VM.

\* 3. Ensure at most one performance warning.

\*

\* (To set 64Mb VM) run using java -Xmx64M SmallMemoryMessageTest

\*/

/\*\* A given interface to process messages (DO NOT CHANGE) \*\*/

interface MessageProcessor {

void processMessage(Message msg);

}

/\*\* A given interface to archive select messages (DO NOT CHANGE) \*\*/

interface MessageArchiver {

void archiveMessages(List<Message> messages, Predicate<Message> filter);

}

/\*\* A given class to represent a message (DO NOT CHANGE) \*\*/

class Message {

private String subject;

private String body;

public Message(String subject, String body) {

this.subject = subject;

this.body = body;

}

public String getSubject() {

return subject;

}

public String getBody() {

return body;

}

@Override

public String toString() {

return "Message{" +

"subject: " + Util.abbreviate(subject, 20) +

", body: " + Util.abbreviate(body, 40) + "}";

}

}

/\*\*

\* Test class that works on developer boxes but fails on small VMs.

\*

\* Fix this class to work on 64Mb VM

\*/

public class SmallMemoryMessageTest {

public static void main(String []args) {

MessageProcessor processor = Util.createMessageProcessor();

MessageArchiver archiver = Util.createMessageArchiver();

List<Message> messages = new ArrayList<>();

for (int i = 0; i < Util.EXPECTED\_TOTAL; i++) {

Message msg = Util.random();

processor.processMessage(msg);

messages.add(msg);

}

archiver.archiveMessages(messages, m -> m.getSubject().startsWith("A"));

/\*

\* DO NOT CHANGE ANYTHING BELOW THIS LINE.

\* PROGRAM MUST EXIT SUCCESSFULLY

\*/

Util.validate();

}

}

/\*\* A given utility class (DO NOT CHANGE) \*\*/

class Util {

static final int EXPECTED\_TOTAL = 98765;

static final int EXPECTED\_ARCHIVED = 3799;

static Message random() {

String subject = randomSubject();

String body = randomBody();

Message m = new Message(subject, body);

return m;

}

static int count = 0;

static String randomSubject() {

StringBuilder sb = new StringBuilder(128);

sb.append((char) ((int) 'A' + (count++ % 26)));

Random r = new Random();

while (sb.length() < 128) {

char c = (char) r.nextInt(127);

if (Character.isLetterOrDigit(c)) {

sb.append(c);

}

}

return sb.toString();

}

static String randomBody() {

StringBuilder sb = new StringBuilder(4096);

Random r = new Random();

while (sb.length() < 4096) {

char c = (char) r.nextInt(127);

if (!Character.isISOControl(c)) {

sb.append(c);

}

}

return sb.toString();

}

static String abbreviate(String s, int n) {

return s.length() > n + 3 ? s.substring(0, n - 3) + "..." : s;

}

static MessageProcessor createMessageProcessor() {

return new TestProcessor();

}

static MessageArchiver createMessageArchiver() {

return new TestArchiver();

}

static void validate() {

if (TestArchiver.count != EXPECTED\_TOTAL ||

TestArchiver.archived != EXPECTED\_ARCHIVED) {

throw new IllegalStateException("Failed to archive all messages!");

}

System.out.println("SUCCESS. PROCESSED: " + TestArchiver.count + " ARCHIVED: " + TestArchiver.archived);

}

}

/\*\* Noop implementation of a message processor (DO NOT CHANGE) \*\*/

class TestProcessor implements MessageProcessor {

long count = 0;

@Override

public void processMessage(Message msg) {

if(++count%1000 == 0) {

System.out.println("Processed: " + (count) + " Latest: " + msg);

System.out.flush();

}

}

}

/\*\* Noop implementation of a message archiver (DO NOT CHANGE) \*\*/

class TestArchiver implements MessageArchiver {

static int count = 0;

long bytes = 0;

static long archived = 0;

@Override

public void archiveMessages(List<Message> messages, Predicate<Message> filter) {

if(messages.size() < 1000) {

System.err.println("WARNING: message list too short, this will drastically reduce performance!");

}

messages.stream().filter(filter).forEach(this::archiveOne);

System.out.println("Archived: " + bytes + " bytes.");

count += messages.size();

bytes = 0;

}

private void archiveOne(Message msg) {

bytes += msg.getSubject().getBytes().length + 1;

bytes += msg.getBody().getBytes().length;

archived++;

}

}

```

**Question 3 Answer**

There are several ways to optimize memory usage for any application. One method that we can use for this case is batch processing. This means processing groups of messages rather than storing all of them in memory. With the solution below, it is able to process only 160,550 bytes per batch while the original solution processed 16,050,775 bytes in one process

public class SmallMemoryMessageTest {

    public static void main(String []args) {

        MessageProcessor processor = Util.createMessageProcessor();

        MessageArchiver archiver = Util.createMessageArchiver();

        //Define the batch size

        final int BATCH\_SIZE = 1000;

        // Outer loop continues as long as the total number of messages

        // processed (x) is less than the expected total number of messages

        for (int x = 0; x < Util.EXPECTED\_TOTAL; x += BATCH\_SIZE) {

            List<Message> messages = new ArrayList<>();

            // Inner loop continues as long as we haven't processed the entire

            // batch (x < BATCH\_SIZE) and the total number of messages processed (x + y)

            // is less than the expected total number of messages

            for (int y = 0; y < BATCH\_SIZE && x + y < Util.EXPECTED\_TOTAL; y++) {

                Message msg = Util.random();

                processor.processMessage(msg);

                messages.add(msg);

            }

            archiver.archiveMessages(messages, m -> m.getSubject().startsWith("A"));

        }

        /\*

         \*  DO NOT CHANGE ANYTHING BELOW THIS LINE.

         \*  PROGRAM MUST EXIT SUCCESSFULLY

         \*/

        Util.validate();

    }

}

QUESTION 4: Debugging

**----------------**

```

package com.moesol.hr.bugs;

public class Bug1 {

    private Integer rating;

    public int rating() {

        return rating;

    }

    public static void main(String[] args) {

        System.out.println("rating:"

            + new Bug1().rating());

    }

}

```

The program above throws a `NullPointerException` with this stack trace:

```

Exception in thread "main" java.lang.NullPointerException

    at com.moesol.hr.bugs.Bug1.rating(Bug1.java:7)

    at com.moesol.hr.bugs.Bug1.main(Bug1.java:12)

```

What is happening? How can it be fixed?

**Question 4 Answer**

The rating **variable** was declared as an Integer (which is a wrapper class) while the rating() method was declared as an int (which is a primitive). Wrapper classes can apply object-oriented capabilities which means it can hold a null value while the *rating()* method is declared to only return a primitive int value which leads to the `NullPointerException`.

There are two ways to fix this

1. Change the wrapper variable to a primitive  
 private int rating;

2. Change the method type to a Wrapper  
 public Integer rating() {  
        return rating;  
    }

QUESTION 5: Wrong Result

**----------------**

The following program produces inconsistent results. It should always output

this:

```

counter is 20000

```

Please correct the program.

```

public class WrongAnswer {

    private int counter = 0;

    public static void main(String[] args) {

        new WrongAnswer().run();

    }

    private void run() {

        try {

            Thread t1 = new Thread(this::incrementToOnHundred);

            Thread t2 = new Thread(this::incrementToOnHundred);

            t1.start();

            t2.start();

            t1.join();

            t2.join();

            System.out.println("counter is " + counter);

        } catch (InterruptedException e) {

            System.err.println("fatal error, unexpected interrupt exception");

            System.exit(2);

        }

    }

    private void incrementToOnHundred() {

        for (int i = 0; i < 10\_000; i++) {

            doSomeFakeWork();

        }

    }

    private void doSomeFakeWork() {

        counter++;

    }

}

```

**Question 5 Answer**

The program makes use of two threads that use and increments the counter variable at the same time - without synchronization this will lead to an uncertain output. The solution for this is to set only one thread to access the counter variable at any given time. We can do this by simply adding the **synchronized** keyword to the incrementToOnHundred declaration.

private **synchronized** void incrementToOnHundred()

BONUS: Programming Puzzle Bonus!

**----------------**

A string of ASCII text has been encrypted using the following algorithm:

\* Use the bytes "MOEBIUS" as the key.

\* Working one byte at a time, xor the next byte of data with the next byte in

the key.

\* When the end of the key is reached, wrap around to the first byte again.

\* Base-64 encode the data.

For example, the text "Simple is better than complex." encrypts to

"HiYoMiUwcyQ8ZSAsIScoPWU2ITQ9bSwqLzk5NjVh". The key is repeated as follows

while XOR-ing bytes:

```

Simple is better than complex.

MOEBIUSMOEBIUSMOEBIUSMOEBIUSMO

```

Write a program to reverse this process and decrypt the following message. Use

any programming language you like. Provide both your program and the decrypted

message.

```

DiArJTs0JzgjJDYgOj0+Y2U7JiBzPiApNCwxczknIGIrOj04PGUzPDAgOSYqLGhfWW1vZW

JpdXNtb2ViaXVzbW9lYml1c21vZWJpdXNtb2ViaXVzY2hibEN1c21vZWJpe3RqYWViaXVz

bWFlYml1c21vZWhucnltb2VicwoPYhB/Yml1c21hT2JpdXNtb38dFXoMd29lYhYJe2IQZW

Jnb31nEBltFn9zbW9/YmYJc3dvZWxue2ljaGtIaXV9amhreGl6D211ZWJpe3xkE2ViaXJp

amVlbRV1eW11ZWJue31qYWViZGhpInV4b0N1aRITah1zcn13dX9saXVzbWhlaG5yeW1vZW

JjdXRjaGpsbnUMEWdqHW57dHdoa2VDdWltYBlic3Vpd3V/eGl1c21vbx0Vegxnb2ViaXV+

cG8qYnR4c21gbB5pdXNtaE9iaXJ9Y2hlYm5vaXdoZWJpdXNnb2oeaX9zbW9lYmdyfGNoa2

JpdXRHb2ViaXVzbW9lYml1c21vZWJpdXljYW9iaXVzbW9lYmlvWQ==

```

> NOTE: This block uses the "MIME" flavor of Base64 encoding.

> Depending upon your Base64 decoder, you may need to remove the "=" padding.

>

**Bonus Question Answer**

import java.util.Base64;

public class test {

    public static void main(String[] args) {

        String key = "MOEBIUS";

        String encryptedMessage = "";

        // Decode the Base64 encoded message

        byte[] decodedBytes = Base64.getDecoder().decode(encryptedMessage);

        // Decrypt the message

        byte[] decryptedBytes = new byte[decodedBytes.length];

        int keyLength = key.getBytes().length;

        // Iterate through each byte of the encrypted message

        for (int i = 0; i < decodedBytes.length; i++) {

            // Apply XOR operation between encrypted byte and corresponding key byte

            // Then store the result in the decrypted byte array

            decryptedBytes[i] = (byte)(decodedBytes[i] ^ key.getBytes()[i % keyLength]);

        }

        // Display the decrypted message

        String decryptedMessage = new String(decryptedBytes);

        System.out.println("Decrypted Message: " + decryptedMessage);

    }

}

A screenshot of a computer program

Description automatically generated