COP2805C Module 7 Practice Exercise

In this practice exercise we are using the Java Concurrency API by designing a multi-threaded version of our HappyBirthday application.

The Module 6 application accessed the Stream contents to dispatch the greetings but did not modify the queue. Now we want the application to remove the greetings as we process them.

We will do this by creating a thread which **polls** the queue and when pending greetings are detected, processes them and removes them from the queue (processing them here just means printing them out) Polling means we periodically check the queue contents using a predefined time cycle. In real-world designs we frequently have to choose between this approach or using an asynchronous event-driven approach (e.g. the Flow API, which has replaced the deprecated Java Observer/Observable API). The choice is based on the frequency with which an object (the queue in our case) is modified. If the frequency is very high, the Flow is preferred. Since we are not implementing a high-throughput application, we will use simple polling. If we had a million users submitting birthday greetings each day this might change, but for our purposes polling will suffice.

I created a new interface for this purpose, the MessageProcessor:

```
public interface MessageProcessor {
    void processMessages();
}
```

and implemented that in a class called BirthdayCardProcessor:

```
public class BirthdayCardProcessor extends Thread implements
MessageProcessor {
```

Note that this class inherits from the Thread class, we will override the Thread class's run() method to process messages in a new thread.

There are two instance variables in this class:

```
private ConcurrentLinkedQueue<BirthdayCard> safeQueue;  // see
explanation below
private boolean stopped = false;
```

"safeQueue" is our queue shared from the HappyBirthdayApp class (passed to the BirthdayCardProcessor class during instantiation), see below for why it is now called "safeQueue" and why we are now using the ConcurrentLinkedQueue type instead of Queue.

The "stopped" variable is a simple flag that allows the HappyBirthdayApp to signal the polling thread that the application is shutting down in order to allow it to clean up if necessary.

In the BirthdayCardProcessor constructor, after receiving the shared queue reference, we start the polling thread using start():

```
public BirthdayCardProcessor(ConcurrentLinkedQueue<BirthdayCard>
safeQueue) {
    this.safeQueue = safeQueue;

    // start polling (invokes run(), below)
    this.start();
}
```

From run(), we do our polling, sleeping for 1 second each cycle after calling processMessages() to see if anything is in the queue. The sleep() method throws InterruptedException every time the 1 second time period expires, so we have to catch that exception. While we are "awake", we check our flag to see if we need to exit the thread (if we have been awakened by the timer expiring, we would expect the flag to be false, so we would continue on in our loop), this is why the "try" is inside of the loop.

```
// poll queue for cards
public void run() {
    final int SLEEP_TIME = 1000; // ms
    while (true) {
        try {
            processMessages();
            Thread.sleep(SLEEP_TIME);
            System.out.println("polling");
        } catch (InterruptedException ie) {
            System.out.println("BirthdayCardProcessor: sleep
interrupted! " + ie);
            // see if we should exit
            if (this.stopped == true)
                break;
        }
    }
}
```

If the HappyBirthdayApp class wants to stop the polling thread, it calls the endProcessing method in the BirthdayCardProcessor class. It does this just before the main application method ends.

```
public void endProcessing() {
    this.stopped = true;
    interrupt();
}
```

The interrupt method will also cause an InterruptedException in run() so we'll be checking our flag whenever that happens and now would expect it to be true.

Now let's look at how the messages are processed. A naïve approach is to simply loop through the contents of the queue, remove an element, and process it. But this doesn't work, the code raises a **ConcurrentModificationException** exception

```
// remove messages from the queue and process them
public void processMessages() {
    System.out.println("queue size is " + queue.size());
    queue.stream().forEach(e -> {
            // Do something with each element
            e = queue.remove();     <<< exception raised here
when using the original queue
            System.out.print(e);
    });
    System.out.println("queue size is now " + queue.size());
}
```

```
Exception in thread "Thread-0" java.util.ConcurrentModificationException
 at java.base/java.util.LinkedList$LLSpliterator.forEachRemaining(LinkedList.java:1246)
 at java.base/java.util.stream.ReferencePipeline$Head.forEach(ReferencePipeline.java:762)
```

Java does not like a Collection object to be modified (e.g. by using the remove operation) during a forEach loop since it creates an iterator from the object for the loop. If the object were to be modified by another thread it could cause problems for the iterator. Java takes the conservative approach and disallows the modification entirely.

If the object were "normal" and not a Collection object, we could "protect" it while modifying it by using a course-grained lock on the entire method; this is done by declaring the method as synchronized:

```
public synchronized void processMessages() {
```

Here is a finer-grained lock approach, which can be more complicated to code but is more efficient; we are not locking the entire method, but only the object that needs to be protected when it is being modified:

```
public void processMessages() {
    Lock lock = new ReentrantLock();
      try {
        if (lock.tryLock(10, TimeUnit.SECONDS)) { // try for a lock here, timeout after 10s if it
    doesn't happen
            ... modify the shared object
        }
      } catch (InterruptedException e) {
        e.printStackTrace();
      } finally{
        //release lock
```

```
            lock.unlock();
        }
```

But either way we will still get the exception, the problem is in the iteration, not the shared access to the object. Instead, we have to try a different approach: use a **thread-safe** version of the queue. Fortunately Java provides thread-safe version of most Collection objects, the queue is one of these:

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/concurrent/ConcurrentLinkedQueue.html

A ConcurrentLinkedQueue is "an unbounded thread-safe queue based on linked nodes."; in other words, a thread-safe version of the linked-list based queue we are using. Here's our processMessages method now:

```
// remove messages from the queue and process them
public void processMessages() {
    System.out.println("queue size is " + safeQueue.size());
    safeQueue.stream().forEach(e -> {
            // Do something with each element
            e = safeQueue.remove();
            System.out.print(e);
    });
    System.out.println("queue size is now " + safeQueue.size());
}
```

I have changed our queue variable to "safeQueue" and the type to ConcurrentLinkedQueue throughout the application. Here is how it is declared now in the HappyBirthdayApp class:

```
// Use a thread-safe Queue<LinkedList> to act as message queue
for the dispatcher
ConcurrentLinkedQueue safeQueue = new ConcurrentLinkedQueue(
       new LinkedList<BirthdayCard>()
);
```

queue -> safeQueue is just a name change and not required, but I wanted to emphasize that this is now a thread-safe queue, e.g.

```
private Stream<BirthdayCard> stream = safeQueue.stream();
```

Here is the class declaration and constructor for the BirthdayCardProcessor:

```
public class BirthdayCardProcessor extends Thread implements
MessageProcessor {

    private ConcurrentLinkedQueue<BirthdayCard> safeQueue;
    private boolean stopped = false;
```

```
    public
BirthdayCardProcessor(ConcurrentLinkedQueue<BirthdayCard>
safeQueue) {
        this.safeQueue = safeQueue;

        // start polling (invokes run(), below)
        this.start();
    }
```

In the generateCards() method in the HappyBirthdayApp class, I now also clear the ArrayList containing the birthdays after generating the cards so they don't pile up:

```
birthdays.clear(); // clear the list
```

This is safe to do since we are only accessing the list from the main application thread, the ArrayList object is not shared by anyone else. Otherwise we would have to protect the ArrayList (e.g. by using Collections.synchronizedList).

I've made some other changes to clean things up -- moved buildCard into the BirthdayCard class, which now implements a new **BirthdayCardBuilder** interface:

```
public class BirthdayCard implements BirthdayCardBuilder {
```

I removed some print statements from some places and added them in others, in particular to the BirthdayCardProcessor class to watch what the polling thread is doing (e.g. when we get the shutdown notification and to check the queue size).

I wanted to add a little load to the queue, but I didn't feel like adding a bunch of new users, so in the main application method I just resend the same users their cards after sleeping for a couple of seconds (they're like fruitflies now - multiple birthdays in a matter of seconds

Here's how we do a sleep, this can be used in any thread, including the main application thread; the sleep method is static. We could wrap this in a method to save some space since I call it multiple times, but we may deal with the sleep interrupt in different ways so I just use the explicit code.

```
// wait for a bit
try {
    Thread.sleep(2000);
} catch (InterruptedException ie) {
    System.out.println("sleep interrupted! " + ie);
}
```

Here is the output for the application:

```
before processing, queue size is 0
after processing, queue size is now 0
Dianne Romero
```

```
Sorry, today is not their birthday.
Sally Ride
René Descartes
Johannes Brahms
Charles Kao
polling
before processing, queue size is 4
Birthday card for Sally Ride
|---------------------------------------------|
|                  Mar 7, 2023                |
|           Happy Birthday Sally Ride         |
|  Hope all of your birthday wishes come true! |
|---------------------------------------------|
Birthday card for René Descartes
|---------------------------------------------|
|                  7 mars 2023                |
|      Joyeux Anniversaire René Descartes     |
|  Hope all of your birthday wishes come true! |
|---------------------------------------------|
Birthday card for Johannes Brahms
|---------------------------------------------|
|                  07.03.2023                 |
|   Alles Gute zum Geburtstag Johannes Brahms  |
|  Hope all of your birthday wishes come true! |
|---------------------------------------------|
Birthday card for Charles Kao
|---------------------------------------------|
|                   2023年3月7日                  |
|               生日快乐 Charles Kao               |
|  Hope all of your birthday wishes come true! |
|---------------------------------------------|
after processing, queue size is now 0
polling
before processing, queue size is 0
after processing, queue size is now 0
Dianne Romero
Sorry, today is not their birthday.
Sally Ride
René Descartes
Johannes Brahms
Charles Kao
polling
before processing, queue size is 4
Birthday card for Sally Ride
|---------------------------------------------|
|                  Mar 7, 2023                |
```

```
|           Happy Birthday Sally Ride          |
|  Hope all of your birthday wishes come true! |
|----------------------------------------------|
Birthday card for René Descartes
|----------------------------------------------|
|                  7 mars 2023                 |
|      Joyeux Anniversaire René Descartes      |
|  Hope all of your birthday wishes come true! |
|----------------------------------------------|
Birthday card for Johannes Brahms
|----------------------------------------------|
|                  07.03.2023                  |
|   Alles Gute zum Geburtstag Johannes Brahms  |
|  Hope all of your birthday wishes come true! |
|----------------------------------------------|
Birthday card for Charles Kao
|----------------------------------------------|
|                   2023年3月7日                  |
|               生日快乐 Charles Kao               |
|  Hope all of your birthday wishes come true! |
|----------------------------------------------|
after processing, queue size is now 0
polling
before processing, queue size is 0
after processing, queue size is now 0
poll thread received exit signal

Process finished with exit code 0
```