Description: Using Multithreading to Solve a Real-world Problem

In this project, you will be creating a multithreaded application to simulate the daily operations of a community bank. You will use:

- a thread to simulate customers gathering outside of the bank
- a semaphore to simulate a security guard that limits the number of people allowed in the bank at any one time
- a queue to simulate the line a customer gets into while waiting for a teller
- individual threads to simulate tellers

In addition to these constructs you will also use a lock to ensure only one thread can write to the terminal at any time.

Instructions

- Please follow all instructions closely and read this in its entirety before writing ANY code. There is no penalty for going "above and beyond" the assignment requirements. However, failure to meet all required parts can affect your grade. Please consult the rubric for each section.
- For this assignment you will create a single "raw" Python file with the name "Lab6.py".

Setup

- Because we are using several multi-threading components, there will be a few imported modules:

```
from threading import Semaphore, Thread, Lock
from queue import Queue, Empty
from random import randint
from time import sleep
```

- You will also need to create the following variables that define how the simulation will run

```
max_customers_in_bank = 10  # maximum number of customers that can be in the bank at one time
max_customers = 30          # number of customers that will go to the bank today
max_tellers = 3             # number of tellers working today
teller_timeout = 10         # longest time that a teller will wait for new customers
```

Part I – Setting up the utility classes and functions

Customer

The Customer class represents a person that wants to go to the bank to conduct a financial transaction.

This class will have only two methods: __init__() and __str__()

- def __init__(self, name):

- This method accepts a name string and will assign it to self.name
- def __str__(self):
  - This method will output a string in the following format: "{self.name}"

Teller

The Teller class represents a teller that will help customers in the bank.

This class will have only two methods: __init__() and __str__()

- def __init__(self, name):
  - This method accepts a name string and will assign it to self.name
- def __str__(self):
  - This method will output a string in the following format: "{self.name}"

bankprint()

The bankprint() function will be where all print commands are sent. It has the following signature:

```
def bankprint(lock, msg):
```

Where:

- lock is a Lock object that must be acquired before the print() function is called.
  - Hint: Use a with context manager block on the lock object
- msg is the text to print.

Main Block

This block will use the standard check for __name__ and represents the entry point for the program:

```
if __name__ == "__main__":
```

In this block, create the printlock Lock object which will be passed to all of the thread functions and used by bankprint():

```
printlock = Lock()
```

Next, create a Queue called teller_line which represents the line customers get into waiting for a teller (the maxsize parameter need to be set to the max_customers_in_bank value):

```
teller_line = Queue(maxsize=max_customers_in_bank)
```

Now, you need a Semaphore object that represents a security guard. A semaphore only allows a limited number of threads to acquire a resource at the same time, just as a security guard limits the number of customers allowed in the bank at once. Like the queue, this object will also be limited to max_customers_in_bank semaphore resources:

```
guard = Semaphore(max_customers_in_bank)
```

Finally after the guard is created, print several messages to the terminal by using the bankprint() method:

```
bankprint(printlock, "<G> Security guard starting their shift")
bankprint(printlock, "*B* Bank open")
```

Notice that the printlock argument is passed to bankprint(). Also, each message is preceded with a tag of sorts. This will help distinguish output in what will be a very busy terminal/shell. For this code, use the following prefixes:

- `*B*` for bank messages
- `<G>` for security guard messages
- `[T]` for teller messages
- `(C)` for customer messages

You will be adding more code to the main section after that last bankprint() statement in following parts.

Scoring:

Criteria	Points
Teller class	5
Customer class	5
bankprint() function	5
Main block	10
TOTAL	**25**

Part II – Get Customers into the Bank

Now, in the __main__ section, you are going to create the Customer objects that want to go into the bank. This can be done several different ways, such as using a for loop or a comprehension. The key requirements are:

- You must create max_customers Customer objects
- Each Customer object must have a unique name (remember, that is the only argument to Customer's __init__() function)

Now, each customer must be passed to a thread function, wait_outside_bank(), which we have to define now. We will get back to __main__ in a moment…

```
def wait_outside_bank(customer, guard, teller_line, printlock):
```

Where:

- customer is a Customer object
- guard is the security guard semaphore
- teller_line is the Queue
- printlock is the Lock

The thread function must do the following:

- Print a customer message indicating that the customer is waiting outside the bank
- Attempt to acquire a semaphore from the guard object (do NOT use a context manager here since the semaphore is to be released in another method)
- Print a security guard message indicating they have let that customer into the bank
- Print a customer message indicating they are trying to get into line
- Put the customer into the teller_line queue (queue's put() method)

Back in the __main__ section, after the customer objects are created, spawn a Thread object for each one using the following criteria:

- target = wait_outside_bank
- args = (customer, guard, teller_line, printlock)

Now, start each of those threads (you can do it in the same loop as the Thread creation).

Finally, sleep for 5 seconds to give time for the tellers to get to work.

Scoring:

Criteria	**Points**
Customer object creation	5
wait_outside_bank()	5
Proper output and use of bankprint()	5
Acquire semaphore from security guard	5
Put customer into the teller_line queue	5
Thread spawning	10
TOTAL	**35**

Part III – Get the Tellers to Service the Customers

Now, you need to create a thread method, teller_job(), that will allow customers to be served by tellers:

```
def teller_job(teller, guard, teller_line, printlock):
```

Where:

- teller is a Teller object
- guard is the security guard semaphore
- teller_line is the Queue
- printlock is the Lock

As you can see, the parameters are almost the same as wait_outside_bank(). The major difference is the first one which is now a teller since, unlike the other thread which was customer-centric, this one will be teller-centric.

- Print a teller message indicating that this teller has started work
- Create an infinite loop and put the rest of the code in it
    - Use a try block, and place the following code within it (see the except code later):
        - Get a customer from the teller_line (use queue's get method() – provide teller_timeout for the timeout argument)
        - Print a teller message indicating this teller is now helping this customer
        - Sleep a random amount of time (1 – 4 seconds) to simulate the teller working with the customer
        - Print a teller message indicating this teller is done helping this customer
        - Print a security guard message indicating they are letting this customer out of the bank
        - Release the semaphore from the guard object
    - The except block should handle a Queue.Empty error (just Empty is OK due to the import)
        - Print a teller message indicating that the teller is going on break and exit the loop

Back in __main__ it is time to get the tellers to work. So, immediately after the 5 second sleep completed in Part II, do the following:

- Print a bank message indicating the tellers are going to start working now
- Create a list of Teller objects (the number should be max_tellers)
- Create a list of teller threads that target the teller_job method using the following arguments to the Thread initializer method:
    - target = teller_job
    - args = (one of the teller objects, guard, teller_line, printlock)
- Launch all the teller threads
- Wait for all threads to complete
- Print a bank message indicating the bank is closed and let the program end

Scoring:

Criteria	**Points**
teller_job()	5
Proper output and use of bankprint()	5
Proper retrieval of customer from teller line	5
Release of semaphore	5

Teller object creation	10
teller_job thread spawning	10
TOTAL	**40**

Sample Output (max_customers_in_bank = 3, max_customers = 7, max_tellers = 2):

```
$ python lab6.py
<G> Security guard starting their shift
*B* Bank open
(C) 'Customer 1' waiting outside bank
<G> Security guard letting 'Customer 1' into the bank
(C) 'Customer 1' getting into line
(C) 'Customer 2' waiting outside bank
<G> Security guard letting 'Customer 2' into the bank
(C) 'Customer 2' getting into line
(C) 'Customer 3' waiting outside bank
<G> Security guard letting 'Customer 3' into the bank
(C) 'Customer 3' getting into line
(C) 'Customer 6' waiting outside bank
(C) 'Customer 7' waiting outside bank
(C) 'Customer 5' waiting outside bank
(C) 'Customer 4' waiting outside bank
*B* Tellers starting work
[T] 'Teller 1' starting work
[T] 'Teller 1' is now helping 'Customer 1'
[T] 'Teller 2' starting work
[T] 'Teller 2' is now helping 'Customer 2'
[T] 'Teller 1' is done helping 'Customer 1'
<G> Security guard is letting 'Customer 1' out of the bank
[T] 'Teller 1' is now helping 'Customer 3'
<G> Security guard letting 'Customer 6' into the bank
(C) 'Customer 6' getting into line
[T] 'Teller 2' is done helping 'Customer 2'
<G> Security guard is letting 'Customer 2' out of the bank
[T] 'Teller 2' is now helping 'Customer 6'
<G> Security guard letting 'Customer 7' into the bank
(C) 'Customer 7' getting into line
[T] 'Teller 1' is done helping 'Customer 3'
<G> Security guard is letting 'Customer 3' out of the bank
[T] 'Teller 1' is now helping 'Customer 7'
<G> Security guard letting 'Customer 5' into the bank
```

```
(C) 'Customer 5' getting into line
[T] 'Teller 2' is done helping 'Customer 6'
[T] 'Teller 1' is done helping 'Customer 7'
<G> Security guard is letting 'Customer 7' out of the bank
<G> Security guard is letting 'Customer 6' out of the bank
[T] 'Teller 1' is now helping 'Customer 5'
<G> Security guard letting 'Customer 4' into the bank
(C) 'Customer 4' getting into line
[T] 'Teller 2' is now helping 'Customer 4'
[T] 'Teller 2' is done helping 'Customer 4'
[T] 'Teller 1' is done helping 'Customer 5'
<G> Security guard is letting 'Customer 4' out of the bank
<G> Security guard is letting 'Customer 5' out of the bank
[T] Nobody is in line, 'Teller 2' is going on break
[T] Nobody is in line, 'Teller 1' is going on break
*B* Bank closed
```