Trabajo Práctico
Programación Concurrente
Threads

Ejercicio 1

Listar todas las posibles salidas que podrían obtenerse cuando se ejecuta el siguiente código en un sistema multi-core

```
package ar.edu.itba.pod.concurrency.threads.tp.e1;
// imports
public class ConcurrentThreads {
   public static class T1 implements Runnable {
       @Override
       public void run() {
           System.out.print("A");
           System.out.print("B");
       }
   }

   public static class T2 implements Runnable {
       @Override
       public void run() {
           System.out.print("1");
           System.out.print("2");
       }
   }

   public static void main(final String[] args) {
       final ExecutorService pool = Executors.newFixedThreadPool(2);

       try {
           pool.execute(new T1());
           pool.execute(new T2());
           pool.shutdown();

           if (!pool.awaitTermination(800, TimeUnit.MILLISECONDS)) {
               pool.shutdownNow();
           }
       } catch (InterruptedException e) {
           pool.shutdownNow();
       }

   }
}
```

Ejercicio 2

Correr y analizar la salida del siguiente código para ver los diferentes estados del thread durante su ejecución

```
package ar.edu.itba.pod.concurrency.threads.tp.e2;
// imports
public class ThreadStateViewer {

   public static void main(String[] args) throws InterruptedException {
       String lock = "lock";

       Thread thread = new Thread(() -> {
                           System.out.printf("Hello!,  my  state  is  %s%n",
Thread.currentThread().getState());
           try {
               Thread.sleep(2000);
               synchronized (lock) {
                   lock.wait();
               }
           } catch (InterruptedException e) {
               e.printStackTrace();
           }
       });

       System.out.printf("Thread state: %s%n", thread.getState());

       thread.start();

       Thread.sleep(500);
       System.out.printf("Thread state: %s%n", thread.getState());

       Thread.sleep(2000);
       System.out.printf("Thread state: %s%n", thread.getState());

       synchronized (lock) {
           lock.notifyAll();
       }

       thread.join();

       System.out.printf("Thread state: %s%n", thread.getState());
   }
}
```

Ejercicio 3

Correr y analizar la salida del siguiente código para ver los diferentes comportamientos de los ExecutorServices utilizados.

```
package ar.edu.itba.pod.concurrency.threads.tp.e3;
// imports
public class ExecutorAnalyzer {
   private static final Logger logger =
LoggerFactory.getLogger(ExecutorAnalyzer.class);
   private static final int THREAD_COUNT = 4;

   private static final Function<Integer, Callable<Void>> runnerFactory
= (Integer index) -> () -> {
        logger.info("Starting runner: {}", index);
        Thread.sleep(1500);
        logger.info("Ending runner: {}", index);
        return null;
   };

   public static void execute(ExecutorService pool) {
        try {
            List<Future<Void>> futures = IntStream.range(0,
THREAD_COUNT).mapToObj(index ->
pool.submit(runner.apply(index))).toList();
            for (Future<Void> future : futures) {
                future.get();
            }
            pool.shutdown();
            if (!pool.awaitTermination(800, TimeUnit.MILLISECONDS)) {
                pool.shutdownNow();
            }
        } catch (InterruptedException | ExecutionException e) {
            pool.shutdownNow();
        }
   }

   public static void main(String[] args) {
        logger.info("Cached Thread Pool");
        execute(Executors.newCachedThreadPool());

        logger.info("Fixed Thread Pool");
        execute(Executors.newFixedThreadPool(2));

        logger.info("Single Thread Executor");
        execute(Executors.newSingleThreadExecutor());

        logger.info("Single Thread Executor but rejecting");
        ThreadPoolExecutor executor = new ThreadPoolExecutor(1, 1, 0,
TimeUnit.MILLISECONDS,
        new SynchronousQueue<>(),
        new ThreadPoolExecutor.AbortPolicy());
        execute(executor);
   }
}
```

Ejercicio 4

Para un sistema de atención al público de define al cliente a partir del siguiente modelo

```java
package ar.edu.itba.pod.concurrency.threads.tp.e4;

enum ClientPriority {
   HIGH, PRIORITY, NORMAL
}
```

```java
package ar.edu.itba.pod.concurrency.threads.tp.e4;

public record Client(String name, ClientPriority priority) {
}
```

Y la atención a partir de la siguiente interfaz de servicio

```java
package ar.edu.itba.pod.concurrency.threads.tp.e4;

public interface IBranchClientQueueService {
   void receiveClient(Client client);

   Client clientForPriority(ClientPriority priority);
}
```

Los empleados se definen a partir de dos Roles:

- `Recepcionist`: recibe un cliente y lo ubica en la cola de su prioridad.

```java
package ar.edu.itba.pod.concurrency.threads.tp.e4;
// imports
public class Receptionist implements Callable<Integer > {
   private static final Integer AMOUNT_OF_CLIENTS = 100;
   private final IBranchClientQueueService clientService;

   public Receptionist(final IBranchClientQueueService clientService) {
       this.clientService = clientService;
   }


   @Override
   public Integer call() throws Exception {
       for (int i = 0; i < AMOUNT_OF_CLIENTS; i++) {
           // simulate one client and enqueue
           // sleep for a couple of random seconds.
       }
```

```
        return AMOUNT_OF_CLIENTS;
    }
}
```

➢ `ClientAttendant`: atiende un cliente de la cola de su prioridad

```
package ar.edu.itba.pod.concurrency.threads.tp.e4;
// imports
public class ClientAttendant implements Callable<Integer > {
    private final BranchClientQueueService clientService;
    private final ClientPriority priority;

    public ClientAttendant(BranchClientQueueService clientService,
ClientPriority priority) {
        this.clientService = clientService;
        this.priority = priority;
    }

    @Override
    public Integer call() throws Exception {
    boolean stillWorking = true;
    while (stillWorking) { //if 3 cycles with no client end.
            //get one client and sleep for random amount of seconds to
simulate service time
            // or if no client sleep to simulate waiting time.
        }
        return 0; // how many clients
    }
}
```

El sistema se construye en la siguiente clase:

```
package ar.edu.itba.pod.concurrency.threads.tp.e4;

public class LocalBranch {
    private static Integer AMOUNT_OF_CLIENTS = 200;
    private static Integer AMOUNT_OF_RECEPTIONIST = 2;
    private static Integer AMOUNT_OF_ATTENDANTS_HIGH = 3;
    private static Integer AMOUNT_OF_ATTENDANTS_PRIORITY = 1;
    private static Integer AMOUNT_OF_ATTENDANTS_NORMAL = 2;

    public static void main(String[] args) {


    }
}
```

Se pide modifcar las clases provistas e implementar nuevas para realizar una simulación de manera que el sistema tenga algunos recepcionistas y personas de atención corriendo en diversos threads y cada uno imprima el comienzo y fin de la atención de cada cliente.