Clase 1: Presentación y Programa ejemplo de programación con procesos en Java

```
      ProcessBuilder pb = new
ProcessBuilder("java","-cp","/home/gilbert/.config/Code/User/workspaceS
torage/78787eb821848071f7cf50e4c766382c/redhat.java/jdt_ws/proyectos_ja
va_6ca5d6b1/bin","ProcesoP01","hola");
           Process proceso = pb.start();
           BufferedReader lectorSalida = new BufferedReader(new
InputStreamReader(proceso.getInputStream()));
           String linea;
           while((linea = lectorSalida.readLine())!=null)
           {
               System.out.println("Extraido del proceso: "+linea);
           }
           System.out.println("Con PID:"+proceso.pid());
```

Clase 2: Terminar programa anterior de gestión de procesos e iniciar otro programa de gestión de procesos de una carrera de caracoles.
Clase 3: Terminamos gestión de procesos con caracoles, vemos un poco de hilos hacemos debug.

```
String[][] parametros = {
               { "10", "B" },
               { "1", "A" }
       };

       try {
           List<Process> listaProcesos = new ArrayList<>();
           for (String[] param : parametros) {
               ProcessBuilder pb = new ProcessBuilder("java", "-cp",

"/home/gilbert/.config/Code/User/workspaceStorage/5f8a6ee0f4555d5ce3e3d
a216edb1520/redhat.java/jdt_ws/proyectos_java_16aea38/bin",
                       "Caracol", param[0], param[1]);
               Process p = pb.start();
               listaProcesos.add(p);

           for (Process pp : listaProcesos) {
               pp.waitFor();
           }
```

Clase 4: Creamos un primer programa con hilos y vemos el ciclo de vida de un hilo en Java

```
class Tarea implements Runnable
{

   @Override
   public void run()
   {
      Thread h1 = new Thread(new Tarea("h1",3000));
       Thread h2 = new Thread(new Tarea("h2",1000));
       Thread h3 = new Thread(new Tarea("h3",3000));

       h1.setPriority(Thread.MIN_PRIORITY);
       h2.setPriority(6);
       h3.setPriority(Thread.MAX_PRIORITY);

       h1.start();
       h2.start();
       h3.start();
       /*
       try {
           h1.join(); // espera que los otros hilos acaben
           System.out.println("FFF h1");
       } catch (InterruptedException e) {
           System.out.println("Exception"+e.getMessage());
       }
           */
       while(h1.isAlive() || h2.isAlive() || h3.isAlive())
       {
           System.out.println("yield");
           Thread.yield(); // cede el control al procesador
       }
       System.out.println("todos los hilos has terminado");
```

Clase 5: Terminamos con el ciclo de vida de un hilo y arrancamos el programa de concurrencia Productor/Consumidor.
Clase 6 terminado problema consumidores productores.
Clase 7 Se ha ejecutado y verificado comportamientos de productores y consumidores con varios tiempos y múltiples consumidores y productores.

```
public class MiBuffer
{
   private final int capacidad;
   private final LinkedList<Integer> cola = new LinkedList<>();
```

```
   public MiBuffer (int in_capacidad)
   {
       this.capacidad=in_capacidad;
   }
   public synchronized void producir(int value) throws
InterruptedException
   {
       while (cola.size()==capacidad)
       {
           System.out.println("BLOQUEADO PRODUCTOR para elemento
"+value);
           wait();
       }
       cola.add(value);
       notifyAll();
   }
   public synchronized int consumir(int nivel) throws
InterruptedException
   {
       while (cola.isEmpty())
       {
           System.out.println("BLOQUEADO CONSUMIDOR NIVEL:"+nivel+"
COLA VACIAAAAAAAAAAAAAAAAAAAAAAAAAA");
           wait();
       }
       int value = cola.removeFirst();
       notifyAll();
       return value;
   }
```

Clase 8 Concurrencia con semáforos

```
Semaphore sem = new Semaphore(1);
```

```
       for (int i = 1; i < 9; i++)
       {
           int timer = new Random().nextInt(i*1000-500+1)+500;
           new Thread(new Person(sem,"persona"+i,timer)).start();
       }
public class Person implements Runnable
{
   private final Semaphore sem;
   private final String nombre;
```

```
    private final int tardon;
    public Person (Semaphore sem, String nombre, int tardon)
    {
        this.sem=sem;
        this.nombre=nombre;
        this.tardon = tardon;
    }
    @Override
    public void run()
    {
        try
        {
            System.out.println("Pide permiso "+nombre);
            sem.acquire(); // pide permiso para entrar en un hueco
            System.out.println("Trabajando "+nombre+" con tardon
"+tardon);
            Thread.sleep(tardon);
         } catch (InterruptedException e)
        {
            System.out.println("Hilo interrumpido "+e.getMessage());
        }
        finally
        {
            System.out.println("Termina "+nombre);
            sem.release(); // libera un hueco
        }
    }
}
```

Clase 9 Aparcamiento con monitores

Clase 10 Pruebas y profundización sobre el ejercicio del aparcamiento

```
public class Parking
{
    private final int capacidad;
    private int n_ocupadas = 0;

    public Parking(int in_capa)
    {
        this.capacidad=in_capa;
    }

    public synchronized void entrar(String nombre_coche) throws
InterruptedException
```

```
    {
        while (n_ocupadas==this.capacidad)
        {
            System.out.println("BLOQUEADOOOOOOOO El coche
"+nombre_coche+" no puede pasar");
            wait();
        }
        n_ocupadas++;
        System.out.println("El coche "+nombre_coche+" YA HA ENTRADO Y
APARCADO");
    }

    public synchronized void salir(String nombre_coche)
    {
        n_ocupadas--;
        System.out.println("SALEEEE el coche "+nombre_coche+" YA HA
ENTRADO Y APARCADO");
        notifyAll();
    }


}
```

Clase 11 Explicación de Tarea 1 y ejemplo con ArrayBlockingQueue

```
  BlockingQueue <Integer> buffer = new ArrayBlockingQueue<>(5);
```

```
        Thread productor = new Thread(() -> {

            try {
                for (int i = 0; i < 10; i++)
                {
                    buffer.put(i);
                    System.out.println("Introducido elemento "+i);
                }
            }
            catch (InterruptedException ex)
            {
                System.out.println("error: "+ex.getMessage());
            }
        });
        Thread consumidor = new Thread(() -> {
```

```
            try {
                for (int i = 0; i < 10; i++)
                {
                    buffer.take();
                    Thread.sleep((long)(Math.random()*10000));
                    System.out.println("Introducido consumido "+i);
                }
            }
            catch (InterruptedException ex)
            {
                System.out.println("error: "+ex.getMessage());
            }
        });


        consumidor.start();
        productor.start();
```

Clase 12 Ejemplo Callcenter con SynchronousQueue

```
        SynchronousQueue<String> callCenter = new SynchronousQueue<>();

        long inicio = System.currentTimeMillis();

        for (int i = 0; i < 3; i++)
        {
            int operadorId = i;
            new Thread( () ->
            {
                while(true)
                    {
                System.out.println("Operador "+operadorId+" esperando
llamada");
                String llamada;
                try {

                        llamada = callCenter.take();
                        System.out.println("Operador "+operadorId+"
recibe la llamada del cliente "+llamada);
                        Thread.sleep(5000);
                        System.out.println("Operador "+operadorId+"
finaliza llamada");
```

```
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            }
            ).start();
        }
        for (int i = 0; i < 15; i++)
        {
            int clienteId = i;
            try
            {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            new Thread( () ->
            {
                try {

                    System.out.println("Cliente "+clienteId+"
llamando...");
                    callCenter.put(""+clienteId);
                    long tiempoTrans = (System.currentTimeMillis() -
inicio);
                    System.out.println("Cliente "+clienteId+" está
siendo atendido en tiempo "+tiempoTrans);

                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            ).start();
```