David Javier Boquete Pulleiro 29BPDJ

Práctica 3: Deadlock.

Instrucciones.

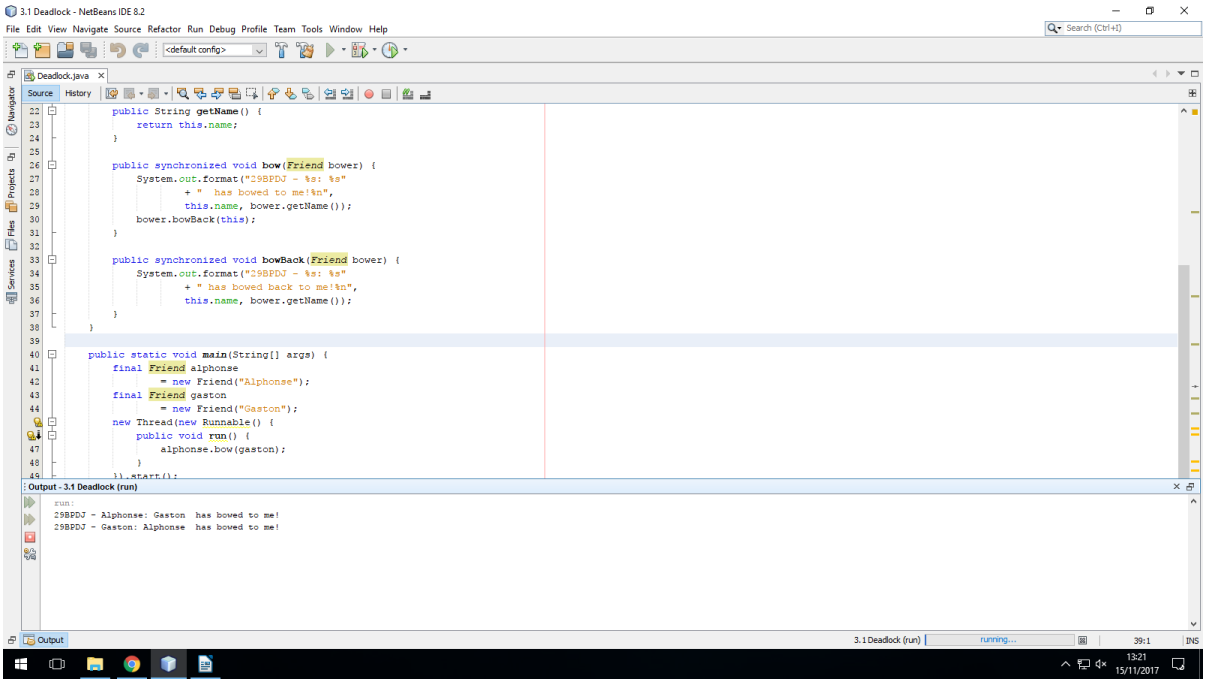
- Implementar en Java las clases que se indican a continuación dentro de un proyecto Netbeans.
- Entregar dos archivos separados:
  - Archivo PDF con las capturas de ejecución que se piden.
  - Archivo ZIP que contenga la carpeta del proyecto Netbeans.

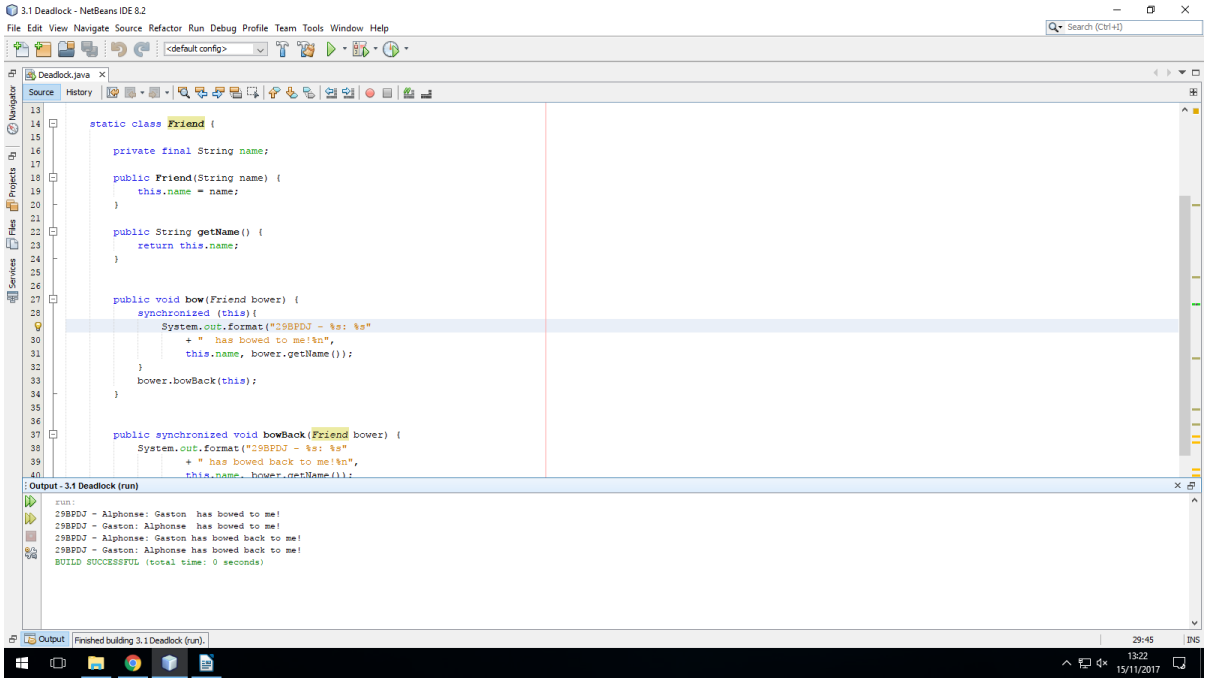
Programa.

Ejemplo: Gaston y Alphonse.

1. Implementar el ejemplo.
2. Concatenar el **identificador de alumno** a cada string que muestra el programa.
3. Capturar los dos escenarios de ejecución que se recogen en la guía.

David Javier Boquete Pulleiro 29BPDJ

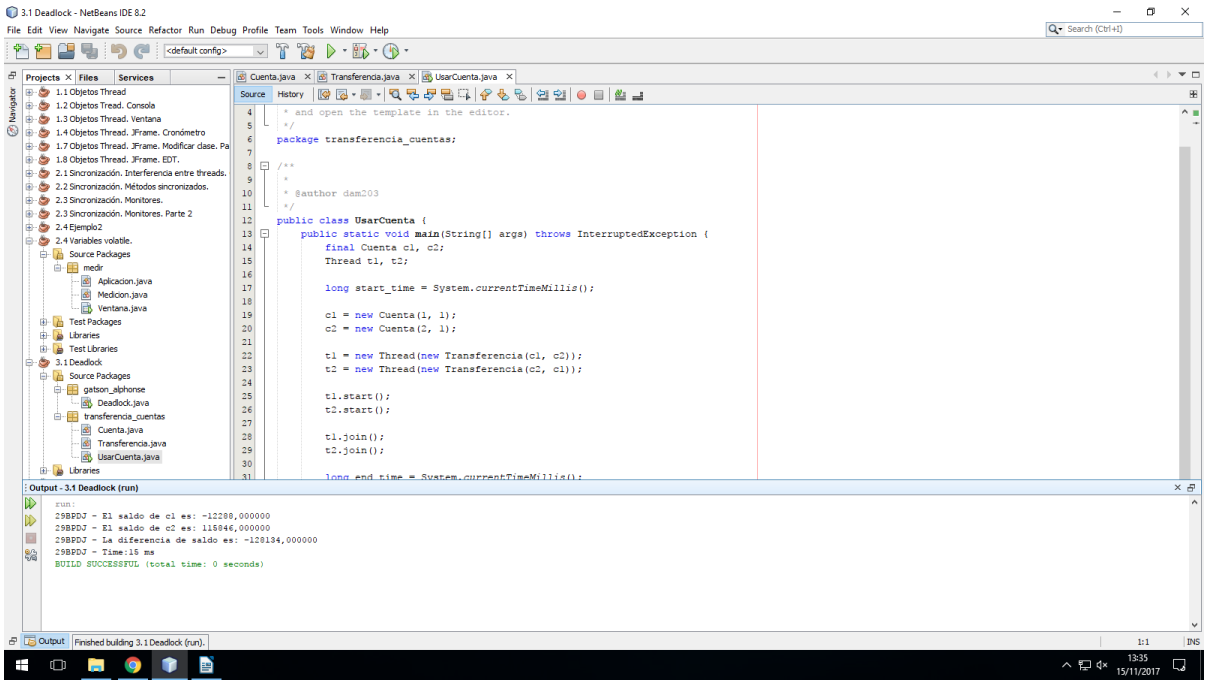




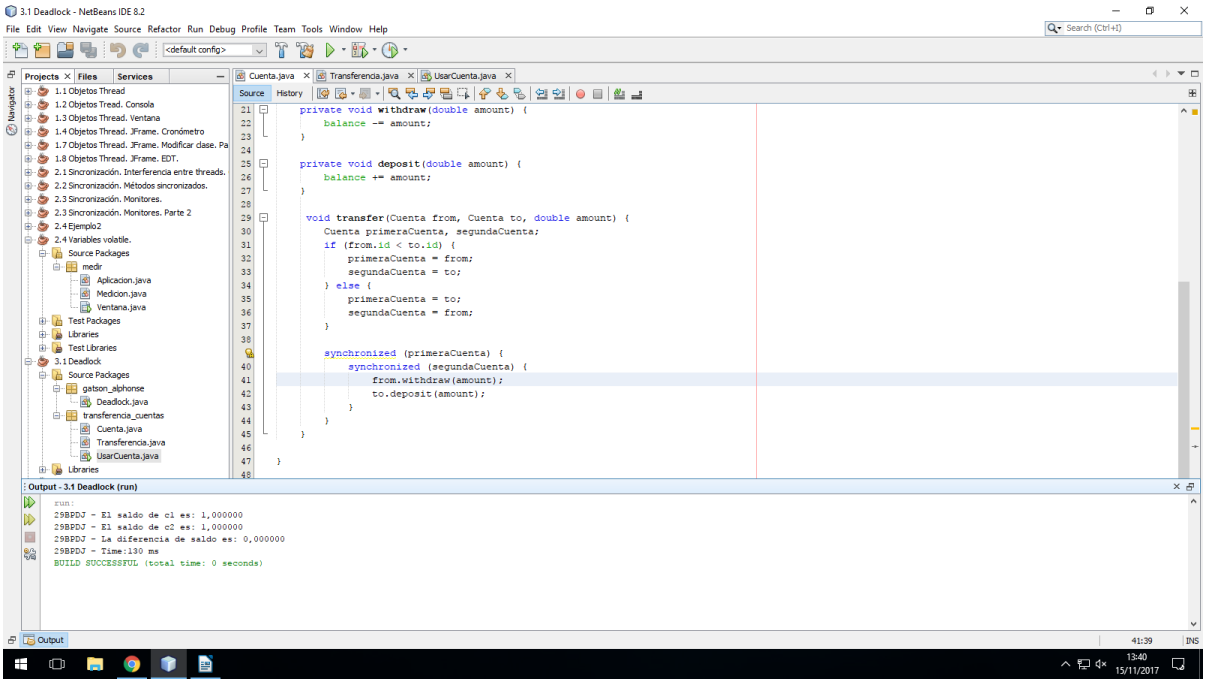
David Javier Boquete Pulleiro 29BPDJ

Ejemplo: Transferencias entre cuentas bancarias.

4. Implementar el ejemplo.
5. Concatenar el **identificador de alumno** a cada string que muestra el programa.
6. Indicar el tiempo de duración del programa para cada ejemplo.
7. Capturar los dos escenarios de ejecución relativos a deadlock que se recogen en la guía.



David Javier Boquete Pulleiro 29BPDJ



Resolver deadlock genérico.

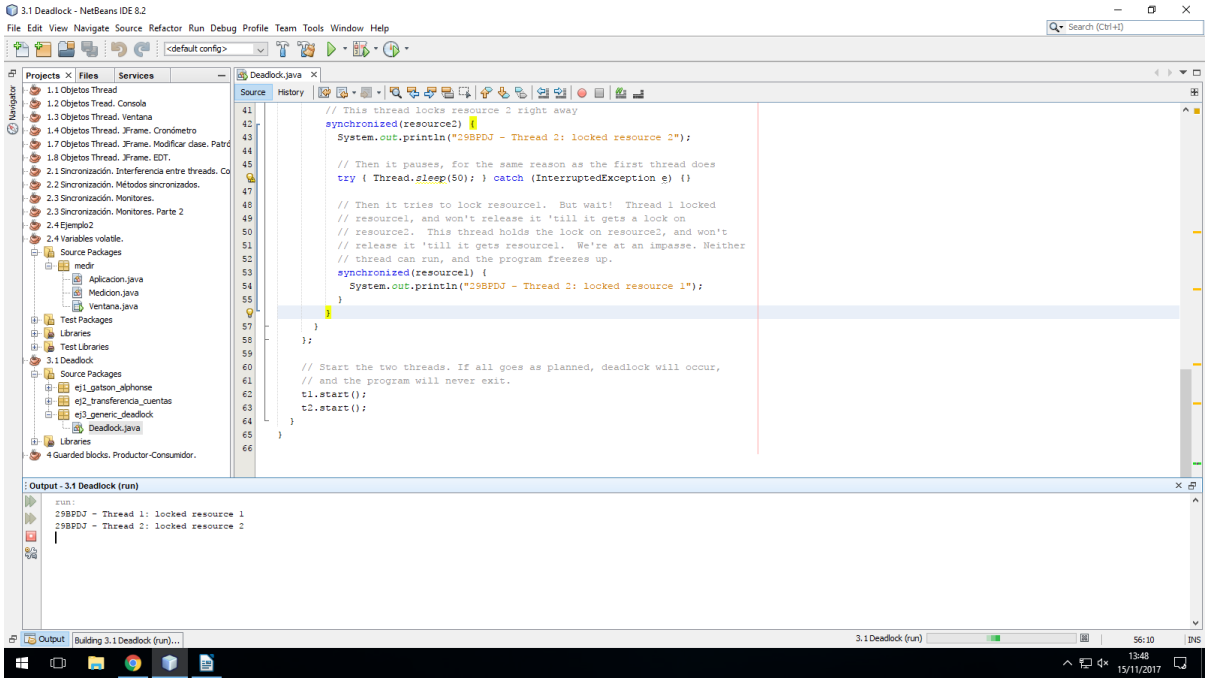
8. Tomar como punto de partida el siguiente ejemplo genérico de deadlock.

```
public class Deadlock {
  public static void main(String[] args) {
    // These are the two resource objects we'll try to get locks for
    final Object resource1 = "resource1";
    final Object resource2 = "resource2";
    // Here's the first thread.  It tries to lock resource1 then resource2
    Thread t1 = new Thread() {
      public void run() {
        // Lock resource 1
        synchronized(resource1) {
          System.out.println("Thread 1: locked resource 1");

          // Pause for a bit, simulating some file I/O or something.
          // Basically, we just want to give the other thread a chance to
          // run.  Threads and deadlock are asynchronous things, but we're
          // trying to force deadlock to happen here...
          try { Thread.sleep(50); } catch (InterruptedException e) {}

          // Now wait 'till we can get a lock on resource 2
          synchronized(resource2) {
            System.out.println("Thread 1: locked resource 2");
          }
        }
```

```
      }
    };

    // Here's the second thread.  It tries to lock resource2 then resource1
    Thread t2 = new Thread() {
      public void run() {
        // This thread locks resource 2 right away
        synchronized(resource2) {
          System.out.println("Thread 2: locked resource 2");

          // Then it pauses, for the same reason as the first thread does
          try { Thread.sleep(50); } catch (InterruptedException e) {}

          // Then it tries to lock resource1.  But wait!  Thread 1 locked
          // resource1, and won't release it 'till it gets a lock on
          // resource2.  This thread holds the lock on resource2, and won't
          // release it 'till it gets resource1.  We're at an impasse. Neither
          // thread can run, and the program freezes up.
          synchronized(resource1) {
            System.out.println("Thread 2: locked resource 1");
          }
        }
      }
    };

    // Start the two threads. If all goes as planned, deadlock will occur,
    // and the program will never exit.
    t1.start();
    t2.start();
  }
}
```

9. Los strings deberán de mostrar el **identificador de alumno**.
10. Capturar el resultado de ejecutar la clase.

David Javier Boquete Pulleiro 29BPDJ



11. Resolver el deadlock.
    a. Sólo deberá modificarse uno de los threads.
    b. No se podrá modificar el orden en el que acceden a los recursos.
12. Copiar y pegar el código de la clase.

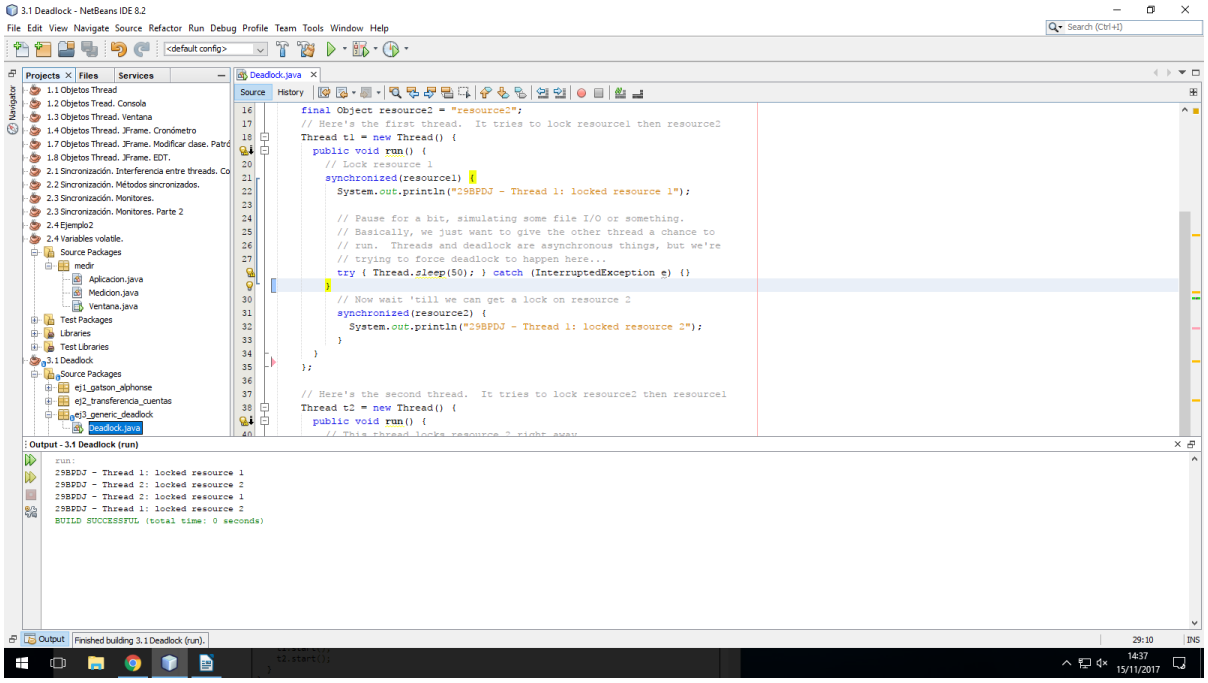
```
  public class Deadlock {
 public static void main(String[] args) {
  // These are the two resource objects we'll try to get locks for
  final Object resource1 = "resource1";
  final Object resource2 = "resource2";
  // Here's the first thread.  It tries to lock resource1 then resource2
  Thread t1 = new Thread() {
    public void run() {
      // Lock resource 1
      synchronized(resource1) {
        System.out.println("29BPDJ - Thread 1: locked resource 1");

        // Pause for a bit, simulating some file I/O or something.
        // Basically, we just want to give the other thread a chance to
        // run.  Threads and deadlock are asynchronous things, but we're
        // trying to force deadlock to happen here...
```

```
        try { Thread.sleep(50); } catch (InterruptedException e) {}
      }
        // Now wait 'till we can get a lock on resource 2
        synchronized(resource2) {
          System.out.println("29BPDJ - Thread 1: locked resource 2");
        }
    }
  };

  // Here's the second thread.  It tries to lock resource2 then resource1
  Thread t2 = new Thread() {
    public void run() {
      // This thread locks resource 2 right away
      synchronized(resource2) {
        System.out.println("29BPDJ - Thread 2: locked resource 2");

        // Then it pauses, for the same reason as the first thread does
        try { Thread.sleep(50); } catch (InterruptedException e) {}

        // Then it tries to lock resource1.  But wait!  Thread 1 locked
        // resource1, and won't release it 'till it gets a lock on
        // resource2.  This thread holds the lock on resource2, and won't
        // release it 'till it gets resource1.  We're at an impasse. Neither
        // thread can run, and the program freezes up.
        synchronized(resource1) {
          System.out.println("29BPDJ - Thread 2: locked resource 1");
        }
      }
    }
  };

  // Start the two threads. If all goes as planned, deadlock will occur,
  // and the program will never exit.
  t1.start();
  t2.start();
```

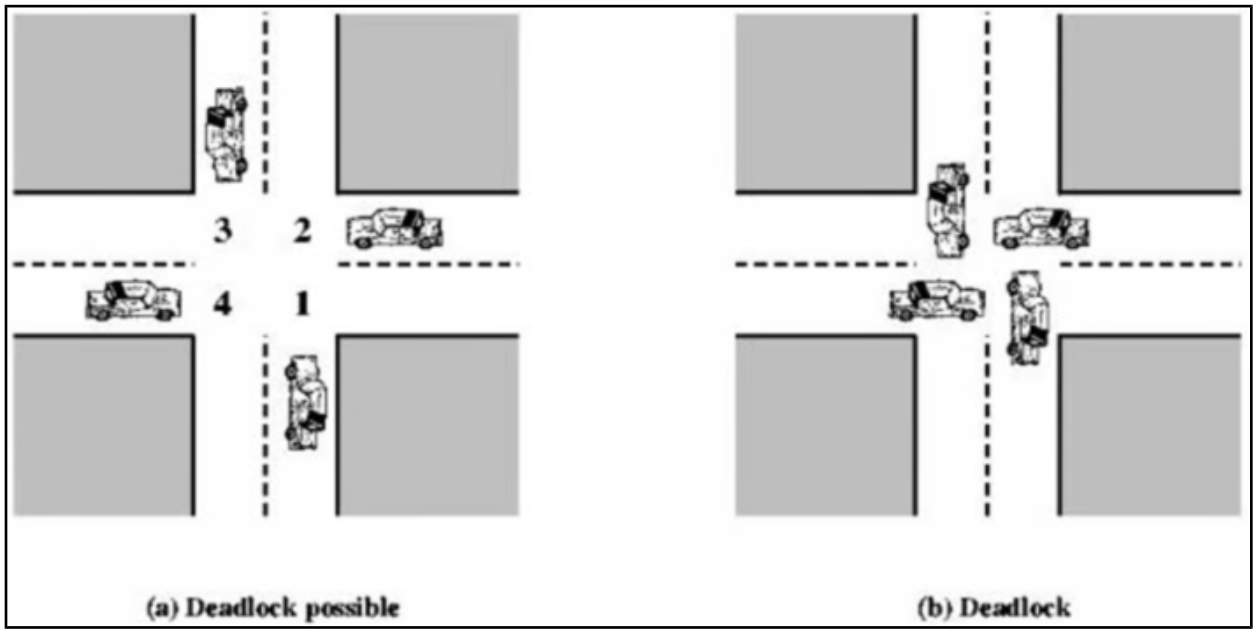
David Javier Boquete Pulleiro 29BPDJ

```
    }
  }
```

13. Capturar el resultado de ejecutar la clase.



Resolver deadlock en un cruce de coches.



David Javier Boquete Pulleiro 29BPDJ

14. Partir del siguiente ejemplo de deadlock.

    a. Clase *Demo0*

```
public class Demo0 {
        public static void main(String[] args) {
        Cars demo1 = new Cars(1);
        Cars demo2 = new Cars(2);
        Cars demo3 = new Cars(3);
        Cars demo4 = new Cars(4);

        new Thread(new Runnable() {
                @Override
                public void run() {
                demo1.waiting(demo2);
                }
        }).start();

        new Thread(new Runnable() {
                @Override
                public void run() {
                demo2.waiting(demo3);
                }
        }).start();

        new Thread(new Runnable() {
                @Override
                public void run() {
                demo3.waiting(demo4);
                }
        }).start();

        new Thread(new Runnable() {
                @Override
                public void run() {
                demo4.waiting(demo1);
                }
        }).start();
        }
}
```

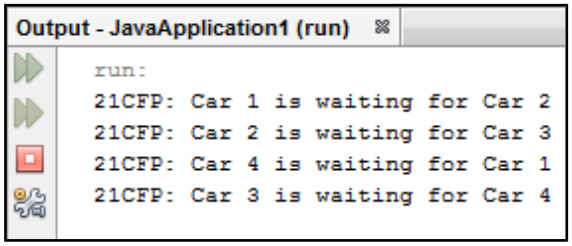
    b. Clase *Cars*

```
public class Cars {
        private int number;

        public Cars(int t) {
        number = t;
        }

        public int getNumber() {
        return number;
        }

        public synchronized void waiting(Cars t) {
        System.out.format("Car " + number + " is waiting for Car " + t.getNumber() + "\n");
```

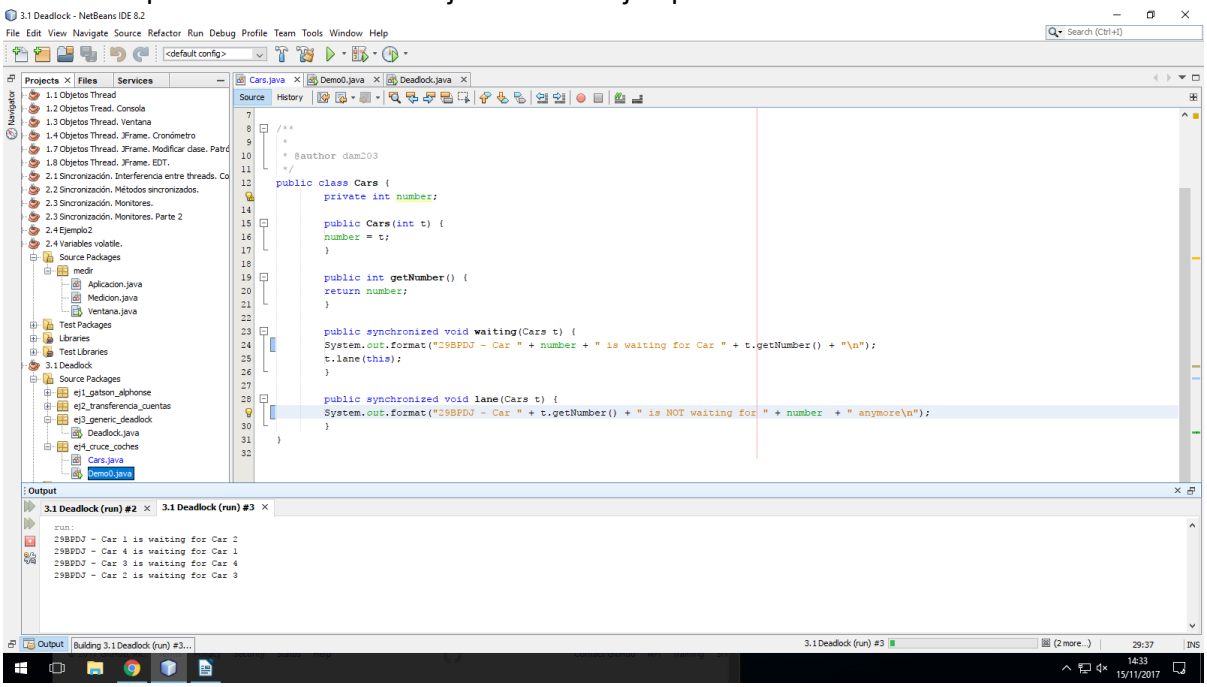
David Javier Boquete Pulleiro 29BPDJ

```
        t.lane(this);
        }

        public synchronized void lane(Cars t) {
        System.out.format("Car " + t.getNumber() + " is NOT waiting for " + number  + "
anymore\n");
        }
}
```

c. Al ejecutar el programa se produce un deadlock.



15. Implementar ambas clases.
    a. Los nombres de clase deberán comenzar por mayúscula.
    b. Los strings que muestra deberán de incorporar el **identificador de alumno**.
16. Capturar el resultado de ejecutar este ejemplo.



17. Resolver el deadlock. Será necesario inspirarse en la solución de los filósofos cenando para evitar que los coches choquen.
18. Copiar y pegar el código de la clase que se ha modificado.

public void waiting(Cars t) {

David Javier Boquete Pulleiro 29BPDJ

```
        if (this.number == 1) {
            synchronized (this) {
                System.out.format("Car " + number + " is waiting for Car " + t.getNumber() + "\n");
            }
            t.lane(this);
        } else{
            synchronized (this) {
                System.out.format("Car " + number + " is waiting for Car " + t.getNumber() + "\n");
                t.lane(this);
            }
        }
}
```

19. Capturar el resultado de ejecutar este ejemplo.