Nombre: _____
Código: _____

1. Instrucciones para la elaboración del parcial

1) El parcial es en parejas del mismo grupo.

2) El parcial empieza a las 9:00 horas.

3) El parcial termina a las 16:00 horas. El corte en el repositorio se hace hasta las 16:00 horas, cualquier código después de dicha hora no será tenido en cuenta.

4) El parcial no es precencial, puede realizarlo desde cualquier lugar[1].

5) A las 11:00 horas a 12:00 horas, bloque 33-302; los profesores Juan Cardona y Juan Lalinde estará atendiendo preguntas relativas al parcial. También pueden hacer su preguntas a través de Skype.

6) Cree un repositorio tipo *git* en bitbucket privado en el que tenga acceso los miembros del equipo y a su correspodiente profesor (Juan Cardona: usuario en bitbucket `fcardona@eafit.edu.co`), (Juan Lalinde: usuario en bitbucket `jlalinde@eafit.edu.co`).

7) Si no tiene conocimiento sobre *git* en la página Gitimmersion hay un tutorial sobre lo principal

8) Realice un *clone* del repositorio.

9) En el correo que acompaña este mensaje hay un adjunto con un archivo zip. Despliege (descomprima) este archivo en la carpeta donde desplegó su repositorio. Adicione los ficheros al respositorio y haga el primer *commit* y luego *push*.

10) El parcial permite para cada punto soluciones en dos lenguajes: `Java` y `C++`.

11) Se utilizará `C++` versión 11.

12) Se utilizará `Java` version 8.

13) Las soluciones esperadas para `C++` se puede hacer con cualquier tipo de semáforo. Si se añaden bibliotecas estas deben quedar dentro del parcial.

14) En Java los proyectos serán manejados por maven.

15) Las soluciones esperadas para `Java` se espera la implementación de monitores. La compilación y ejecución se hará a través de maven: Tutorial corto de maven.

[1] Obvio acceso a internet

16) En cada punto y lenguaje se indicarán los ficheros que pueden ser modificados y de forma. Generalmente es un único fichero para cada punto y lenguaje.

17) Cualquier modificación para cualquier otro fichero de los indicados sin una indicación previa por parte de los profesores, el punto es considerado inválido y la nota correspondiente es cero.

18) El código será calificado con respecto al uso de soluciones correctas del problema en cuestión. No se esperán soluciones que hagan parecer que el código funciona bien. Por ejemplo, tiempos muertos en los programas para los hilos aparente una solución.

19) Deben trabajar cada problema en un solo lenguaje.

2. Preguntas

(50 %) 1. **Bridge**

Una carreta que recorre el país de Este (*East*) a Oeste (*West*) y vice-versa[2], excepto en un tramo de la carreta que permite solamente un vehículo a la vez pasando por el puente (*Bridge*). Los vehículos de viajan de Oeste (*West*) a Este (*East*), tiene prefencia sobre los que viajan en dirección opuesta; es decir, que sí en un momento hay un vehículo en el puente, y llegan vehículos en ambos lados, el primer vehículo esperando por la orilla Oeste del puente (*West*) tendrá preferencia para pasar; los vehículos que estén en la otra orilla Este (*East*) esperan hasta que el último vehículo esperando en la orilla Oeste (*West*) pase.

C++: La siguiente es la estructura de directorios para el proyecto en C++.

```
punto01/c++
punto01/c++/src
punto01/c++/src/bridge.cpp
punto01/c++/src/bridge.h
punto01/c++/src/main.cpp
punto01/c++/src/makefile
punto01/c++/src/threadArg.cpp
punto01/c++/src/threadArg.h
punto01/c++/src/utils.cpp
punto01/c++/src/utils.h
```

Donde el ficheros que deben ser modificados, si la solución lo requiere, son: `bridge.cpp` y `bridge.h`. Dentro del fichero se debe modificar los siguientes métodos: `Bridge::enterWest`, `Bridge::enterEast`, `Bridge::leaveWest` y `Bridge::leaveEast`. Los métodos que comienzan con `enter` incrementan la variable `nCarsOnBridge` cuando un carro ha sido admitido al puente. Los métodos que terminan con `leave` decrementan la variable `nCarsOnBridge`. **Importante:** el método `Bridge::getNCarsOnBridge` no hace parte de la solución y no puede ser modificado.

Java: La siguiente es la estructura de directorios en Java:

```
punto01/java
punto01/java/bridge
punto01/java/bridge/pom.xml
punto01/java/bridge/src
punto01/java/bridge/src/main
punto01/java/bridge/src/main/java
punto01/java/bridge/src/main/java/co
punto01/java/bridge/src/main/java/co/eafit
punto01/java/bridge/src/main/java/co/eafit/dis
punto01/java/bridge/src/main/java/co/eafit/dis/st0257
punto01/java/bridge/src/main/java/co/eafit/dis/st0257/s20181
punto01/java/bridge/src/main/java/co/eafit/dis/st0257/s20181/bridge
punto01/java/bridge/src/main/java/co/eafit/dis/st0257/s20181/bridge/App.java
punto01/java/bridge/src/main/java/co/eafit/dis/st0257/s20181/bridge/Bridge.java
punto01/java/bridge/src/main/java/co/eafit/dis/st0257/s20181/bridge/Utils.java
```

[2]La vía es bidireccional

El fichero que debe ser modificado es `Bridge.java`, usted puede modificar el código de los los métodos: `enterWest`, `enterEast`, `leaveWest` y `leaveEast`. Los métodos que empiezan con `enter` muestra el carro que está entrando cuando la solución lo permita e incrementa el número de carros en el puente. Los métodos que empiezan con `leave` muestra el carro que está saliendo del puente y decrementa el número de carros en el puente. Como Java implementa un cierto tipo de monitores, se puede utilizar el módificador `synchronized` donde sea necesario, excepto en los constructores y en el método `getNCarsOnBridge`. **Importante:** el método `getNCarsOnBridge` no puede ser modificado.

(50 %) 2. **Batch Operating System**

Un simple sistema operativo de lotes *Batch Operating System* puede ser descrito por tres procesos interactuando como siguen:

```
class Card;
class Line;

const int N = n;

Card input_buffer[N];
Line output_buffer[N];

Card* readCard();
void  printLine(Line*);
Line* transformCardToLine(Card*);
void* processReader(void* args);
void* processExecuter(void *args);
void* processPrinter(void *args);
void  deployCardInputBuffer(Card*);
Card* getCardFromInputBuffer();
void  deployLineOutputBuffer(Line*);
Line* getLineFromOutputBuffer();

int
main(int argc, char *argv[]) {
 ...
}

void*
processReader(void * args) {
  for (;;) {
    Card *card = readCard();
    deployCardInputBuffer(card);
  }
  return NULL;
}

void*
processExecuter(void *args) {
  for (;;) {
     Card *card = getCardFromInputBuffer();
```

```
38      Line* line = transformCardToLine(card);
39      deployLineOutputBuffer(line);
40    }
41    return NULL;
42  }
43
44  void*
45  processPrinter(void *args) {
46    for (;;) {
47      Line* line = getLineFromOutputBuffer();
48      printLine(line);
49    }
50    return NULL;
51  }
```

Ustedes debe implementar las funciones si lo hacen en C++: `deployCardInputBuffer`, `getCardFromInputBuffer`, `deployLineOutputBuffer` y `getLineFromOutputBuffer`. Esto se debe hacer utilizando semáforos e implementando una solución que permita al sistema *batch*.

En Java: debe expandir dos clase `BufferInput` y `BufferOutput`; en la primera clase de debe implementar los métodos: `deployCard` y `fetchCard`; en la segunda clase se debe implementar los métdos: `deployLine` y `fetchLine`. La solución en Java debe utilizar monitores.

C++: La siguiente es la jerarquía de directorios de C++:

```
punto02/c++
punto02/c++/src
punto02/c++/src/batch.cpp
punto02/c++/src/card.cpp
punto02/c++/src/card.h
punto02/c++/src/concurrency.cpp
punto02/c++/src/concurrency.h
punto02/c++/src/line.cpp
punto02/c++/src/line.h
punto02/c++/src/makefile
punto02/c++/src/util.cpp
punto02/c++/src/util.h
```

El fichero que pueden ser modificados para implementar la solución son: `concurrency.cpp` y `concurrency.h`. Los métodos que deben ser modificados para implementar la solución son: `deployCardInputBuffer`, `getCardFromInputBuffer`, `deployLineOutputBuffer` y `getLineFromOutputBuffer`.

Java: La siguiente es la jerarquía de java.

```
punto02/java
punto02/java/batch
```

```
punto02/java/batch/pom.xml
punto02/java/batch/src
punto02/java/batch/src/main
punto02/java/batch/src/main/java
punto02/java/batch/src/main/java/co
punto02/java/batch/src/main/java/co/edu
punto02/java/batch/src/main/java/co/edu/eafit
punto02/java/batch/src/main/java/co/edu/eafit/st0257
punto02/java/batch/src/main/java/co/edu/eafit/st0257/s20181
punto02/java/batch/src/main/java/co/edu/eafit/st0257/s20181/App.java
punto02/java/batch/src/main/java/co/edu/eafit/st0257/s20181/Card.java
punto02/java/batch/src/main/java/co/edu/eafit/st0257/s20181/InputBuffer.java
punto02/java/batch/src/main/java/co/edu/eafit/st0257/s20181/Line.java
punto02/java/batch/src/main/java/co/edu/eafit/st0257/s20181/OutputBuffer.java
punto02/java/batch/src/main/java/co/edu/eafit/st0257/s20181/Util.java
```

El ficheros a modificar son `InputBuffer.java` y `OutputBuffer.java`. En primero, los métodos a modificar son: `deployCard` y `fetchCard`. En el segundo, los métodos a modificar son: `deployLine` y `fetchLine`. Como Java implementa un cierto tipo de monitores, se puede utilizar el módificador `synchronized` donde sea necesario, excepto en los constructores