**EASYCAB**

SISTEMAS DISTRIBUIDOS

PRÁCTICA 1

Julio Corbalán Moreno

Álvaro Pedreño Rubio

# 

# **ÍNDICE**

[**EC\_Customer** 3](#_zdx493ink0v9)

[Variables Globales 3](#_avlzspbpu75e)

[Funciones 3](#_5elxj73bam93)

[**EC\_Sensor 5**](#_wxv82qaiz7k6)

[Variables Globales 5](#_wzribo4bovm)

[Funciones 5](#_gduej41pvfl3)

[**EC\_DE 7**](#_i52n50o8hxa)

[Variables Globales 7](#_jo51fv8ilblb)

[Funciones 7](#_ajt8bp7bmz2y)

[**EC\_Central 10**](#_no63qqrcd2yn)

[Variables Globales 10](#_rmqguwy4nzn0)

[Funciones 10](#_ehiyakm618dy)

[**EC\_Shared 13**](#_p06rmmgx4mni)

[**Variables Globales 13**](#_t6wge97yhxxq)

[**Funciones 13**](#_don7v5geayxk)

[**EC\_Map 16**](#_dhi1vuhb8lhb)

[**Variables Globales 16**](#_4334hfgqc66j)

[**Clase 16**](#_xjvlpr2qj34t)

[**Map 16**](#_z3bj093rnivk)

[**Métodos 16**](#_l2euc04j1uzs)

[**Funciones 17**](#_khwzqty9g43)

[**Interfaz gráfica 18**](#_ripamqei6bw6)

[**Interfaz ASCII 18**](#_55wy18fvs7jh)

## 

## EC\_Customer

Implementa un cliente para solicitar servicios de transporte a través de un sistema de mensajería basado en Kafka, permitiendo la conexión y comunicación con un Broker y seguimiento del estado del servicio.

#### **Variables Globales**

* BROKER\_IP: Dirección IP del Broker de Kafka. Se obtiene del segundo argumento de línea de comandos.
* BROKER\_PORT: Puerto de conexión al Broker. Se obtiene del tercer argumento de línea de comandos.
* BROKER\_ADDR: Dirección completa (IP y puerto) del Broker en formato <BROKER\_IP>:<BROKER\_PORT>.
* ID: Identificador único del cliente. Se obtiene del cuarto argumento de línea de comandos.
* servicios: Lista de identificadores de servicios que el cliente solicita, cargados desde un archivo JSON.

#### **Funciones**

**comprobarArgumentos(argumentos)** Verifica la cantidad de argumentos de línea de comandos al iniciar el programa.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

Si el número de argumentos es incorrecto, muestra un mensaje de error y finaliza el programa.

**asignarConstantes(argumentos)** Asigna los valores de BROKER\_IP, BROKER\_PORT, BROKER\_ADDR, e ID usando los argumentos de línea de comandos.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

**leerServicios()** Carga los identificadores de servicios desde el archivo JSON EC\_Requests.json y los almacena en la lista servicios.

Si ocurre un error al abrir el archivo, muestra un mensaje de error y finaliza el programa.

**esperarMensaje()** Establece una conexión con el Broker de Kafka como consumidor en el tema TOPIC\_CLIENTES y escucha los mensajes de respuesta del sistema de servicios.

Mientras el cliente está esperando una respuesta, procesa el mensaje recibido para actualizar el estado del servicio (enServicio):

* Mensajes "OK" y "RECOGIDO" indican que el servicio fue aceptado o recogido, respectivamente.
* Mensajes "KO" y "EN\_DESTINO" indican que el servicio fue denegado o que el cliente ha llegado a su destino.

**evaluarMensaje(mensajeRecibido)** Evalúa el contenido de un mensaje recibido para determinar si es una respuesta positiva (True) o negativa (False).

* **Parámetros**:
  + mensajeRecibido: El mensaje a evaluar.

**solicitarServicio(servicio)** Solicita un servicio específico publicando un mensaje en TOPIC\_CLIENTES y llama a esperarMensaje() para gestionar la respuesta.

* **Parámetros**:
  + servicio: El identificador del servicio a solicitar.

## EC\_Sensor

Implementa un sensor de estado para un taxi, permitiendo la conexión y comunicación continua con el servidor central de taxis. El sensor envía un mensaje indicando su estado actual y permite al usuario cambiar dicho estado.

#### **Variables Globales**

* estado: Representa el estado del sensor. True indica que el sensor está en "OK" y False indica que está en "KO".
* TAXI\_IP: Dirección IP del taxi con el cual el sensor se conecta.
* TAXI\_PORT: Puerto de conexión del taxi.
* TAXI\_ADDR: Dirección completa del taxi en formato (TAXI\_IP, TAXI\_PORT).

#### **Funciones**

**comprobarArgumentos(argumentos)** Verifica la cantidad de argumentos de línea de comandos al iniciar el programa.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

Si el número de argumentos es incorrecto, muestra un mensaje de error y finaliza el programa.

**asignarConstantes(argumentos)** Asigna los valores de TAXI\_IP, TAXI\_PORT, y TAXI\_ADDR usando los argumentos de línea de comandos.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

**gestionarConexionTaxi()** Crea una conexión de socket con el taxi y envía continuamente el estado actual del sensor (OK o KO) al servidor central.

1. Intenta establecer una conexión con TAXI\_ADDR.
2. En un bucle continuo, envía el mensaje con el estado del sensor ("OK" o "KO" según la variable estado), cada segundo.
3. Si la conexión se interrumpe, muestra un mensaje de advertencia y reintenta la conexión después de 3 segundos.

**cambiar\_estado()** Permite al usuario cambiar el estado del sensor manualmente mediante un menú interactivo.

1. Muestra un menú con dos opciones:
   * Cambiar el estado del sensor (OK a KO o viceversa).
   * Salir del programa.
2. Actualiza el valor de estado y muestra el nuevo estado en pantalla.

## EC\_DE

Sistema de comunicación entre taxis y una central (EC\_Central) utilizando sockets y un gestor de colas (Kafka). A través de diversas funciones, permite que los taxis reciban servicios de clientes, se muevan hacia la ubicación del cliente y luego al destino. Este flujo se gestiona mediante una comunicación bidireccional entre el taxi, la central y los sensores.

#### **Variables Globales**

* CENTRAL\_IP: Dirección IP del servidor de mensajes (broker).
* CENTRAL\_PORT: Puerto del broker.
* CENTRAL\_ADDR: Dirección completa del broker en formato (BROKER\_IP,
* BROKER\_IP: Dirección IP del servidor de mensajes (broker).
* BROKER\_PORT: Puerto del broker.
* BROKER\_ADDR: Dirección completa del broker en formato (BROKER\_IP, BROKER\_PORT).
* LISTEN\_PORT:Puerto en el que el sistema escucha las conexiones entrantes
* ID: ID específico del taxi.
* posX: Posición actual del taxi en el eje X.
* posY: Posición actual del taxi en el eje Y.
* cltX: Posición actual del cliente a recoger en el eje X.
* cltY: Posición actual del cliente a recoger en el eje Y.
* destX: Posición actual del destino del cliente en el eje X.
* destY: Posición actual del destino del cliente en el eje Y.
* clienteARecoger: ID del cliente asignado a recoger.
* clienteRecogido: Estado que indica si el cliente ha sido recogido (True = “Si”, False = “No”)

#### **Funciones**

**comprobarArgumentos(argumentos)** Verifica la cantidad de argumentos de línea de comandos al iniciar el programa.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

Si el número de argumentos es incorrecto, muestra un mensaje de error y finaliza el programa.

**asignarConstantes(argumentos)** Asigna los valores de TAXI\_IP, TAXI\_PORT, y TAXI\_ADDR usando los argumentos de línea de comandos.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

**modificarSensoresConectados(valor)** Ajusta el número de sensores conectados en función del valor proporcionado (incremento o decremento).

* **Parámetros**:
  + valor: Valor a asignar.

**gestionarSocketSensores()** Configura un socket servidor para escuchar conexiones de sensores.

Acepta conexiones solo si el límite de sensores no ha sido alcanzado, y lanza un nuevo hilo para cada sensor conectado.

**gestionarSensor(conexion, direccion)** Gestor de un sensor específico. Recibe mensajes, valida su contenido y actualiza el estado del sensor.

Cambia el estado del sensor y publica el cambio en un tópico de Kafka si el estado es modificado.

* **Parámetros**:
  + conexion: Instancia de la conexión por sockets con el sensor.
  + direccion: Dirección utilizada por el sensor para la conexión.

**recibirMapaLogin(socket)** Recibe el mapa de la central al iniciar sesión. Carga y despliega el estado del mapa en el objeto mapa del taxi.

* **Parámetros**:
  + valor: Valor a asignar.

**gestionarConexionCentral()** Establece y gestiona la conexión con la central. Envía la autenticación y recibe mensajes de servicio, lo que activa el movimiento del taxi.

* **Parámetros**:
  + valor: Valor a asignar.

**gestionarBroker()** Conecta al broker de Kafka y escucha mensajes de servicio de la central. Actualiza el mapa y, en caso de recibir un servicio, registra la ubicación del cliente y el destino.

**obtenerPosicion(id, cliente)** Busca y retorna la posición (x, y) de un cliente o destino específico en el mapa. Realiza una búsqueda en el mapa basado en el id y tipo de localización.

* **Parámetros**:
  + id: ID del elemento a buscar.
  + cliente: Boolean, si es verdadero busca un cliente, de lo contrario una buscará localización

**mover(x, y)** Realiza el movimiento del taxi a la posición (x, y) si está dentro de los límites de movimiento. Publica el movimiento en el tópico de Kafka para que la central esté al tanto.

* **Parámetros**:
  + x: Nueva posición eje X
  + y: Nueva posición eje Y.

**calcularMovimientos(X, Y, destX, destY)** Calcula el siguiente paso (x, y) para el taxi en dirección a la ubicación de destino (destX, destY). Retorna las nuevas coordenadas calculadas.

* **Parámetros**:
  + X: Actual posición del eje X.
  + Y: Actual posición del eje Y
  + destX: Posición deseada del eje X
  + destY: Posición deseada del eje Y

**manejarMovimientos()** Controla el movimiento general del taxi para recoger clientes y llevarlos a su destino.

Espera hasta alcanzar la posición de destino o al cliente, y publica el estado en el tópico de Kafka.

## EC\_Central

Implementa un servidor central que gestiona la autenticación de taxis, la asignación de clientes y la actualización del estado de los taxis en un sistema de transporte. Utiliza sockets para la comunicación con los taxis y Kafka para la mensajería entre componentes, permitiendo la recepción de información de ubicación y el control del flujo de servicios solicitados, así como la gestión del mapa de localizaciones activas.

#### **Variables Globales**

* DATABASE: Ruta de la base de datos SQLite utilizada en el sistema para almacenar datos de ubicaciones y estados de taxis.
* LISTEN\_PORT: Puerto de escucha del servidor.
* THIS\_ADDR: Dirección completa del servidor en formato (HOST, LISTEN\_PORT).
* BROKER\_IP: Dirección IP del servidor de mensajes (broker).
* BROKER\_PORT: Puerto del broker.
* BROKER\_ADDR: Dirección completa del broker en formato (BROKER\_IP, BROKER\_PORT).
* taxisConectados: Lista de IDs de taxis actualmente conectados.
* taxisLibres: Lista de IDs de taxis disponibles para asignación de servicios.
* mapa: Instancia de la clase Map, que gestiona las ubicaciones y el estado de los taxis y clientes.

#### **Funciones**

**comprobarArgumentos(argumentos)** Verifica que la cantidad de argumentos de línea de comandos sea la correcta para iniciar el programa.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

Si el número de argumentos es incorrecto, muestra un mensaje de error y finaliza el programa.

**asignarConstantes(argumentos)** Asigna los valores de HOST, LISTEN\_PORT, THIS\_ADDR, BROKER\_IP, BROKER\_PORT, y BROKER\_ADDR utilizando los argumentos de línea de comandos.

* **Parámetros**:
  + argumentos: Lista de argumentos de línea de comandos.

Actualiza las variables globales con los valores correspondientes de los argumentos.

**leerConfiguracionMapa()** Carga las ubicaciones iniciales de taxis y clientes desde un archivo JSON.Lee el archivo EC\_locations.json y actualiza el mapa con las ubicaciones de taxis y clientes. Si el archivo no está disponible, muestra un mensaje de error y cierra el programa.

**leerBBDD()** Carga las ubicaciones de taxis y clientes desde la base de datos SQLite. Conecta a la base de datos, recupera las posiciones de los taxis y clientes, y actualiza las ubicaciones en mapa. Luego, cierra la conexión a la base de datos.

**ejecutarSentenciaBBDD(sentencia)** Ejecuta una sentencia SQL en la base de datos. Devuelve de la ejecución de la sentencia, o None si ocurre un error.

* **Parámetros**:
  + sentencia: String con la sentencia SQL a ejecutar.

**ejecutarScriptBBDD(script)** Ejecuta un script SQL completo en la base de datos. Lee y ejecuta cada comando del script en la base de datos.

* **Parámetros**:
  + script: Ruta del archivo SQL a ejecutar.

**comprobarTaxi(idTaxi)** Verifica si el taxi con el ID dado está en la base de datos y si ya está conectado. True si el taxi existe y no está conectado; False en caso contrario.

* **Parámetros**:
  + idTaxi: ID del taxi a verificar.

**gestionarBrokerClientes()** Conecta al broker como consumidor de mensajes de clientes y gestiona las solicitudes de servicio.

Escucha los mensajes del broker en el tópico de clientes, procesa las solicitudes, asigna taxis disponibles a clientes y actualiza la base de datos y el mapa.

**gestionarBrokerTaxis()** Conecta al broker como consumidor de mensajes de taxis y gestiona las actualizaciones de estado y ubicación de los taxis.

Escucha los mensajes del broker en el tópico de taxis, actualiza el estado y la posición de los taxis en mapa y la base de datos, y envía las actualizaciones al broker.

## EC\_Shared

Implementan funciones comunes que usarán distintas partes del código. Utiliza funciones para abrir sockets de servidor y cliente, enviar y recibir mensajes, y conectar a un broker de Kafka para la publicación y suscripción a temas relacionados con taxis y clientes. Incluye métodos para manejar la codificación de mensajes y proporcionar información sobre las operaciones realizadas, como la apertura de sockets y la publicación de mensajes en Kafka.

#### Variables Globales

* HEADER: Tamaño del encabezado para el envío de mensajes a través de sockets.
* FORMAT: Formato de codificación para los mensajes (UTF-8).
* TOPIC\_TAXIS: Nombre del tema en Kafka para los mensajes relacionados con los taxis.
* TOPIC\_CLIENTES: Nombre del tema en Kafka para los mensajes relacionados con los clientes.
* TOPIC\_ERRORES: Nombre del tema en Kafka para los mensajes de error.

#### **Funciones**

**printInfo(mensaje)**: Imprime un mensaje informativo con una marca de tiempo.

* **Parámetros**:
  + mensaje: El mensaje a imprimir.

**printWarning(mensaje)**: Imprime un mensaje de advertencia con una marca de tiempo.

* **Parámetros**:
  + mensaje: El mensaje a imprimir.

**printError(mensaje)**: Imprime un mensaje de error con una marca de tiempo.

* **Parámetros**:
  + mensaje: El mensaje a imprimir.

**abrirSocketServidor(socket\_addr)**: Crea un socket de servidor en la dirección especificada. Devuelve la conexión socket.

* **Parámetros**:
  + socket\_addr: Tupla que contiene la dirección IP y el puerto donde el servidor escuchará.

**abrirSocketCliente(socket\_addr)**: Crea un socket de cliente y se conecta a la dirección especificada. Devuelve la conexión socket.

* **Parámetros**:
  + socket\_addr: Tupla que contiene la dirección IP y el puerto del servidor.

**enviarMensajeServidor(conexion, mensaje)**: Envía un mensaje al servidor a través de la conexión especificada.

* **Parámetros**:
  + conexion: Conexión socket al servidor.
  + mensaje: El mensaje a enviar.

**recibirMensajeServidor(conexion)**: Recibe un mensaje del servidor a través de la conexión especificada. Devuelve el mensaje recibido.

**Parámetros**:

conexion: Conexión socket al servidor.

**recibirMensajeServidorSilent(conexion)**: Recibe un mensaje del servidor de manera silenciosa (sin imprimir información). Devuelve el mensaje recibido

**Parámetros**:

conexion: Conexión socket al servidor.

**enviarMensajeCliente(socket, mensaje)**: Envía un mensaje al cliente a través del socket especificado.

**Parámetros**:

socket: Socket del cliente.

mensaje: El mensaje a enviar.

**recibirMensajeCliente(conexion)**: Recibe un mensaje del cliente a través de la conexión especificada.

**Parámetros**:

conexion: Conexión socket al cliente.

**recibirMensajeClienteSilent(conexion)**: Recibe un mensaje del cliente de manera silenciosa (sin imprimir información). Devuelve el mensaje recibido.

**Parámetros**:

conexion: Conexión socket al cliente.

**conectarBrokerConsumidor(broker\_addr, topic)**: Establece una conexión al broker de Kafka como consumidor del tema especificado. Devuelve el consumidor de Kafka

* **Parámetros**:
  + broker\_addr: Dirección del broker de Kafka.
  + topic: Nombre del tema del que se suscribirá.

**publicarMensajeEnTopic(mensaje, topic, broker\_addr)**: Publica un mensaje en el tema especificado del broker de Kafka.

* **Parámetros**:
  + mensaje: El mensaje a publicar.
  + topic: Nombre del tema donde se publicará el mensaje.
  + broker\_addr: Dirección del broker de Kafka.

## EC\_Map

Implementa un sistema para visualizar la ubicación y el estado de los taxis en un mapa, permitiendo la representación gráfica de los taxis, clientes y localizaciones. Utiliza una interfaz gráfica con Tkinter y se integra con un sistema de mensajes para recibir actualizaciones sobre el estado de los taxis, también se muestra el mapa por pantalla en ASCII.

#### **Variables Globales**

* SIZE: Tamaño del mapa, representando la cantidad de filas y columnas.
* TILE\_SIZE: Tamaño de cada celda en la cuadrícula del mapa.
* diccionarioPosiciones: Diccionario que almacena las posiciones de los taxis, clientes y localizaciones.
* taxisActivos: Lista que contiene los IDs de los taxis que están activos en el mapa.

#### **Clase**

##### Map

Clase que gestiona el mapa de taxis y proporciona métodos para su visualización y manipulación.

#### **Métodos**

**print()** Imprime el mapa en la consola con los elementos correspondientes a sus posiciones.

**draw\_on\_canvas(canvas)** Dibuja el mapa en un Canvas de Tkinter, mostrando los taxis, clientes y localizaciones. Actualiza la representación gráfica cada segundo.

**clear()** Limpia las posiciones y taxis activos en el mapa.

**exportJson()** Exporta el diccionario de posiciones a formato JSON.

**exportActiveTaxis()** Exporta la lista de taxis activos en un formato específico.

**loadJson(jsonData)** Carga posiciones en el mapa desde un string JSON.

**loadActiveTaxis(jsonData)** Carga taxis activos desde un string JSON.

**move(key, x, y)** Actualiza la posición de un elemento en el mapa.

* Parámetros:
  + key: Identificador del elemento a mover.
  + x: Nueva coordenada X.
  + y: Nueva coordenada Y.

**getPosition(key)** Devuelve la posición de un elemento dado su identificador.

* Parámetros:
  + key: Identificador del elemento.

**activateTaxi(idTaxi)** Activa un taxi en el mapa, añadiéndolo a la lista de taxis activos.

* Parámetros:
  + idTaxi: ID del taxi a activar.

**deactivateTaxi(idTaxi)** Desactiva un taxi en el mapa, removiéndolo de la lista de taxis activos.

* Parámetros:
  + idTaxi: ID del taxi a desactivar.

#### **Funciones**

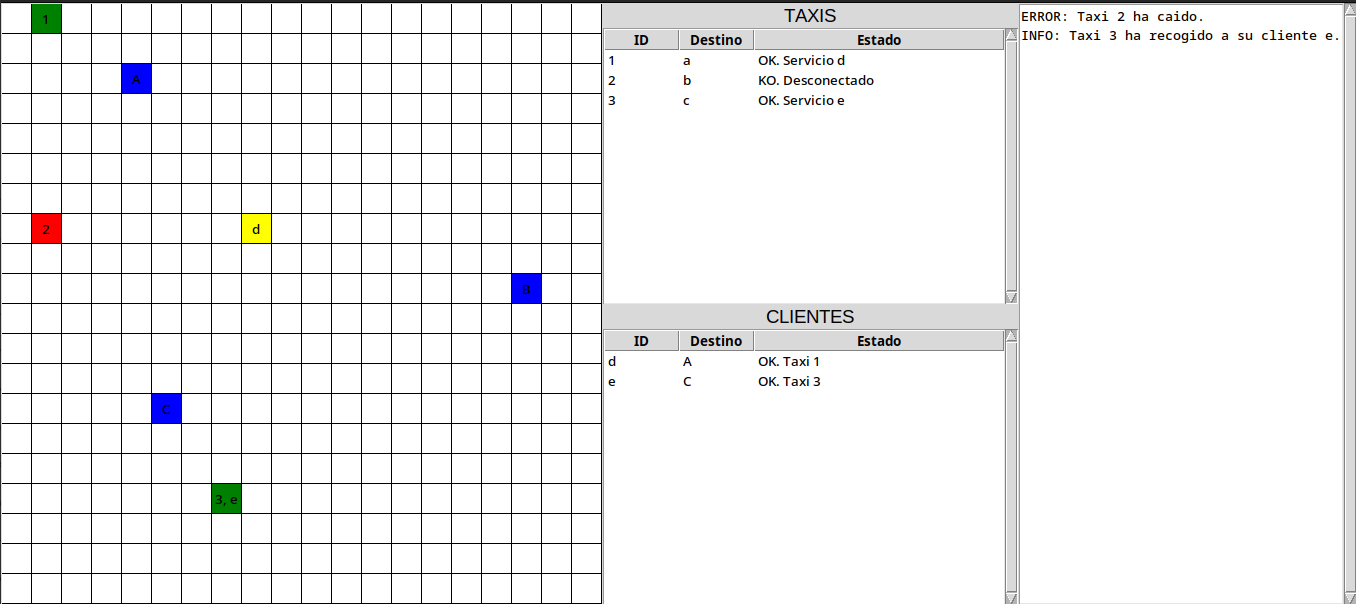
**consumidorErrores(topic, broker\_addr, add\_error\_callback)** Función en segundo plano que lee mensajes de errores desde un tema específico en Kafka.

* Parámetros:
  + topic: Tema de Kafka a suscribirse.
  + broker\_addr: Dirección del broker de Kafka.
  + add\_error\_callback: Callback para manejar mensajes de error.

**create\_window(map\_instance)** Crea la ventana principal de la aplicación con el mapa de taxis y las tablas de taxis y clientes.

* Parámetros:
  + map\_instance: Instancia de la clase Map que se utiliza para dibujar el mapa.

#### **Interfaz gráfica**



#### **Interfaz ASCII**

