Marquina Meseguer, Alfredo
Programación Concurrente y Distribuida
Grupo 2, Subgrupo 2
Junio 23/24
Martínez España, Raquel

PROYECTO PROGRAMACIÓN CONCURRENTE Y DISTRIBUIDA

ÍNDICE

1 EJERCICIO 4

1.1 CÓDIGO

1.1.1 PSEUDO CÓDIGO

```
process Controlador(procesosRestantes: int, /** todos los buzones de recibir **/
)

var

	/****** Buzones de recibir ******/

	solicitudCaja: mailbox of mailbox of int;

	// Recibe las solicitudes de los procesos persona que

	// quieren ser asignados a una caja.


	buzonTerminar: mailbox of mailbox of string;

	// Recibe mensajes de los procesos que terminan. Va asociado a la variable

	// procesosRestantes, cada vez que recibe uno, esta se decrementa. Cuando

	// Llegue a cero el proceso controlador debe terminar.


	// Los siguientes buzones son para el control de la exclusión mutua

	// Sin embargo la exclusión mutua de la pantall se controla con un buzón

	// de testigo descentralizado.


	abrirCajaA: mailbox of mailbox of string;

	// Recibe peticiones de procesos persona que quieren obtener la

	// exclusión mútua de la caja A


	liberarCajaA: mailbox of mailbox of string;

	// Recibe mensajes del proceso que tiene la exclusión mútua de la

	// caja A para liberarla.
```

```
      abrirCajaB: mailbox of mailbox of string;
      // Recibe peticiones de procesos persona que quieren obtener la
      // exclusión mútua de la caja B.

      liberarCajaB: mailbox of mailbox of string;
      // Recibe mensajes del proceso que tiene la exclusión mútua de la
      // caja B para liberarla.

      /****** Entero ******/
      procesosRestantes: int;
      // Lleva la cuenta de los procesos que quedan ejecutando.
      // Cuando llega a cero se termina el proceso Controlador.

      /****** Booleanos ******/
      cajaALibre: bool;
      // Variable para el seguimiento de la exclusión mútua de la Caja A.

      cajaBLibre: bool;
      // Variable para el seguimiento de la exclusión mútua de la Caja B.

      escrituraLibre: bool;
      // Variable para el seguimiento de la exclusión mútua de la pantalla.
begin

      cajaALibre = true;
      cajaBLibre = true;
      escrituraLibre = true;

      repeat
            Select
```

```
                        receive(solicitudCaja, buzonRespuesta);
                        // Genera un número entre 1 y 10 que representa el tiempo (en Segundos) a estar en caja
                        tiempoEspera = generarTiempoEspera();
                        if (tiempoEspera ⩾ )
                              send(buzonRespuesta,    crearMensaje(tiempoEspera, cajaA));
                        else
                              send(buzonRespuesta,    crearMensaje(tiempoEspera, cajaB));
                  or
                  when(cajaALibre) ⇒
                        receive(abrirCajaA, buzonRespuesta);
                        cajaALibre = false;
                        send(buzonRespuesta, "ok");
                  or
                  when(!cajaALibre) ⇒
                        receive(liberarCajaA, buzonRespuesta);
                        cajaALibre = true;
                        send(buzonRespuesta, "ok");
                  or
                  when(cajaBLibre) ⇒
                        receive(abrirCajaB, buzonRespuesta);
                        cajaBLibre = false;
                        send(buzonRespuesta, "ok");
                  or
                  when(!cajaBLibre) ⇒
                        receive(liberarCajaB, buzonRespuesta);
                        cajaBLibre = true;
                        send(buzonRespuesta, "ok");
                  or
```

```
                        receive(buzonTerminar, msj);
                        procesosRestantes--;
            end;
      until procesosRestantes == 0 ;
end;

process persona (id, /*Buzones del controlador*/, /*Buzones de la persona*/)
var
      /****** Buzones del controlador ******/
      solicitudCaja: mailbox of mailbox of int;
      abrirCajaA: mailbox of mailbox of string;
      liberarCajaA: mailbox of mailbox of string;
      abrirCajaB: mailbox of mailbox of string;
      liberarCajaB: mailbox of mailbox of string;
      buzonTerminar: mailbox of mailbox of string;

      /****** Buzones de la persona ******/
      respuestaSolicitudCaja: mailbox of string;
      respuestaAbrirCajaA: mailbox of string;
      respuestaLiberarCajaA: mailbox of string;
      respuestaAbrirCajaB: mailbox of string;
      respuestaLiberarCajaB: mailbox of string;
      respuestaSolicitudEscritura: mailbox of string;
      respuestaLiberaEscritura: mailbox of string;

      /****** Buzón control del a exclusión mutua pantalla con testigo ******/
      mutexEscritura: mailbox of mailbox of string;
      // Este buzón sirve para el control de la exclusión mutua mediante paso de
testigo.

      id: int;
```

```
	// El id del proceso

	caja: string; // También puede ser un enum
	// Guardamos la caja utilizada para imprimir en el mensaje más tarde

	tiempoPago: int;
	// El tiempo que pasa la persona en caja.

begin
	for i := 0..N_REPETICIONES do
	begin
		///// 1 Realiza la compra
		sleep(rand());

		///// 2 Solicita una Caja
		send(solicitudCaja, respuestaSolicitudCaja);
		receive(respuestaSolicitudCaja, msj);
		// Como solicitud caja tiene que devolver dos argumentos y el buzon solo puede enviar uno,
		// Se ha elegido codificarlos en un mensaje. Se podrían poner juntos en un struct, pero eso
		// sería más dificil de traducir en Java.
		tiempoPago, caja = decodificarMensaje(msj);

		///// 3 y 4 realiza el pago en una caja y las libera
		// Solicitas exclusión mutua de la caja correspondiente
		if (caja = "A")
		begin
			send(abrirCajaA, respuestaAbrirCajaA);
			receive(respuestaAbrirCajaA, msj);
			sleep(tiempoPago); // Simular realizar pago en caja
```

```
                send(liberarCajaA, respuestaLiberarCajaA);
                receive(respuestaLiberarCajaA, msj);
          end else
          begin
                send(abrirCajaB, respuestaAbrirCajaB);
                receive(respuestaAbrirCajaB, msj);
                sleep(tiempoPago); // Simular realizar pago en caja
                send(liberarCajaB, respuestaLiberarCajaB);
                receive(respuestaLiberarCajaB, msj);
          end;

          ///// 5 Imprime en pantalla información

          receive(mutexEscritura, testigo); // Solicita la exlucisón mutua

          // Imprime por pantalla la información del pago
          // informaciónDelPago forma el mensaje según enunciado
          print(informaciónDelPago(id, caja, tiempoPago);

          send(mutexEscritura, testigo); // Liberar exclusión mutua
    end;

    send(buzonTerminar, "ok");
end;

main()
var
    /****** Buzones del controlador ******/
    solicitudCaja: mailbox of mailbox of int;
    abrirCajaA: mailbox of mailbox of string;
```

```
	liberarCajaA: mailbox of mailbox of string;
	abrirCajaB: mailbox of mailbox of string;
	liberarCajaB: mailbox of mailbox of string;
	buzonTerminar: mailbox of mailbox of string;

	/****** Buzones de la persona ******/
	respuestaSolicitudCaja: array[0..29] of mailbox of string;
	respuestaAbrirCajaA: array[0..29] of mailbox of string;
	respuestaLiberarCajaA: array[0..29] of mailbox of string;
	respuestaAbrirCajaB: array[0..29] of mailbox of string;
	respuestaLiberarCajaB: array[0..29] of mailbox of string;

	/****** Buzón control del a exclusión mutua pantalla con testigo ******/
	mutexEscritura: mailbox of mailbox of string;

	controlador: process Controlador;
	clientes: array[0.. N_CLIENTES-1] of process Persona;
begin
	/*Realiza la inicialización de los buzones aquí*/
	// Se obvia porque es muy simple

	writeln("Definiendo procesos");

	controlador = Controlador(solicitudCaja,
	                abrirCajaA,
	                liberarCajaA,
	                abrirCajaB,
	                liberarCajaB,
	                buzonTerminar,
	                N_CLIENTES);
```

```
for i:= 0..N_CLIENTES-1 do
begin
clientes[i] = Cliente(i,
            solicitudCaja,
            abrirCajaA,
            liberarCajaA,
            abrirCajaB,
            liberarCajaB,
            solicitudEscritura,
            liberaEscritura,
            buzonTerminar,
            respuestaSolicitudCaja[i],
            respuestaAbrirCajaA[i],
            respuestaLiberarCajaA[i],
            respuestaAbrirCajaB[i],
            respuestaLiberarCajaB[i],
            mutexEscritura);
end;

writeln("Comenzando ejecución");

// Se envia el primer testigo desde el main.
send(mutexEscritura, testigo);

// Lanzamos los procesos
cobegin
      controlador.launch();
      for c in clientes do
            c.lauch();
```

```
            end;
        coend;


        writeln("Ejecución terminada");
end;
```

1.1.2 CÓDIGO JAVA

PRINCIPAL

```
import messagepassing.MailBox;

public class Ejercicio4 {
    private static final int N_CLIENTES = 30;

    public static void main(String[] args) {
        /* Buzones de escritura para Controlador */
        MailBox solicitudCaja = new MailBox();
        // Recibe las solicitudes de los procesos persona que
        // quieren ser asignados a una caja.

        MailBox buzonTerminar = new MailBox();
        // Recibe mensajes de los procesos que terminan. Va asociado a la
variable
        // procesosRestantes, cada vez que recibe uno, esta se decrementa.
Cuando
        // Llegue a cero el proceso controlador debe terminar.

        // Los siguientes buzones son para el control de la exclusión mutua
        // Sin embargo la exclusión mutua de la pantall se controla con un buzón
        // de testigo descentralizado.

        MailBox abrirCajaA = new MailBox();
        // Recibe peticiones de procesos persona que quieren obtener la
        // exclusión mútua de la caja A

        MailBox liberarCajaA = new MailBox();
        // Recibe mensajes del proceso que tiene la exclusión mútua de la
        // caja A para liberarla.

        MailBox abrirCajaB = new MailBox();
        // Recibe peticiones de procesos persona que quieren obtener la
        // exclusión mútua de la caja B.

        MailBox liberarCajaB = new MailBox();
        // Recibe mensajes del proceso que tiene la exclusión mútua de la
```

```
        // caja B para liberarla.

        /* Buzones de escritura para Persona */
        // Como en Java la clase MailBox que representa un buzón no se puede
pasar como parámetro por MailBox, se debe tener
        // los buzones respuesta ya presentes en la clase. De esta manera,
simplemente se puede pasar el id del proceso y
        // responder al buzón con la posición igual a este id, que será el de
dicho proceso.

        MailBox[] respuestaSolicitudCaja = new MailBox[N_CLIENTES];
        // Para responder a las peticiones de SolicitudCaja

        MailBox[] respuestaAbrirCajaA = new MailBox[N_CLIENTES];
        // Para responder a las peticiones de AbrirCajaA

        MailBox[] respuestaLiberarCajaA = new MailBox[N_CLIENTES];
        // Para responder a las peticiones de LiberarCajaA;

        MailBox[] respuestaAbrirCajaB = new MailBox[N_CLIENTES];
        // Para responder a las peticiones de AbrirCajaB;

        MailBox[] respuestaLiberarCajaB = new MailBox[N_CLIENTES];
        // Para responder a las peticiones de LiberarCajaB;

        MailBox mutexEscritura = new MailBox();

        /* Clases de procesos */
        Controlador controlador;
        Persona[] clientes;

        System.out.println("Definiendo clases.");

        for (int i = 0; i < N_CLIENTES; i++) {
            respuestaSolicitudCaja[i] = new MailBox();
            respuestaAbrirCajaA[i] = new MailBox();
            respuestaLiberarCajaA[i] = new MailBox();
            respuestaAbrirCajaB[i] = new MailBox();
            respuestaLiberarCajaB[i] = new MailBox();
        }

        controlador = new Controlador(solicitudCaja, buzonTerminar, abrirCajaA,
liberarCajaA, abrirCajaB, liberarCajaB,
                respuestaSolicitudCaja, respuestaAbrirCajaA,
respuestaLiberarCajaA, respuestaAbrirCajaB,
                respuestaLiberarCajaB, N_CLIENTES);

        clientes = new Persona[N_CLIENTES];
        for (int i = 0; i < N_CLIENTES; i++)
            clientes[i] = new Persona(i, solicitudCaja, buzonTerminar,
abrirCajaA, liberarCajaA, abrirCajaB,
```

```
                    liberarCajaB, respuestaSolicitudCaja[i],
respuestaAbrirCajaA[i], respuestaLiberarCajaA[i],
                    respuestaAbrirCajaB[i], respuestaLiberarCajaB[i],
mutexEscritura);


        System.out.println("Comezando Ejecución");
        // Se envia el primer testigo desde el main.
        mutexEscritura.send("ok");
        // Lanzamos los procesos
        controlador.start();
        for (Persona p : clientes)
            p.start();

        try {
            controlador.join();
            for (Persona p : clientes)
                p.join();
        } catch (Exception ex) {
            ex.printStackTrace();
        }

        System.out.println("Ejecución terminada");


    }
}
```

CONTROLADOR

```
import messagepassing.MailBox;
import messagepassing.Selector;

public class Controlador extends Thread {
    private static final int MAX_TIEMPO = 10;
    private static final int MIN_TIEMPO = 1;
    private static final int MIN_TIEMPO_A = 5;
    private static final String OK = "ok";
    public static final String SEPARADOR = ",";
    public static final String CAJA_A = "A";
    public static final String CAJA_B = "B";
    /****** Buzones de recibir ******/
    private final MailBox solicitudCaja;
    // Recibe las solicitudes de los procesos persona que
    // quieren ser asignados a una caja.

    private final MailBox buzonTerminar;
    // Recibe mensajes de los procesos que terminan. Va asociado a la variable
    // procesosRestantes, cada vez que recibe uno, esta se decrementa. Cuando
    // Llegue a cero el proceso controlador debe terminar.
```

```
    // Los siguientes buzones son para el control de la exclusión mutua
    // Sin embargo la exclusión mutua de la pantall se controla con un buzón
    // de testigo descentralizado.

    private final MailBox abrirCajaA;
    // Recibe peticiones de procesos persona que quieren obtener la
    // exclusión mútua de la caja A

    private final MailBox liberarCajaA;
    // Recibe mensajes del proceso que tiene la exclusión mútua de la
    // caja A para liberarla.

    private final MailBox abrirCajaB;
    // Recibe peticiones de procesos persona que quieren obtener la
    // exclusión mútua de la caja B.

    private final MailBox liberarCajaB;
    // Recibe mensajes del proceso que tiene la exclusión mútua de la
    // caja B para liberarla.

    /****** Buzones de respuetas******/
    // Como en Java la clase MailBox que representa un buzón no se puede pasar
como parámetro por MailBox, se debe tener
    // los buzones respuesta ya presentes en la clase. De esta manera,
simplemente se puede pasar el id del proceso y
    // responder al buzón con la posición igual a este id, que será el de dicho
proceso.

    private final MailBox[] respuestaSolicitudCaja;
    // Para responder a las peticiones de SolicitudCaja

    private final MailBox[] respuestaAbrirCajaA;
    // Para responder a las peticiones de AbrirCajaA

    private final MailBox[] respuestaLiberarCajaA;
    // Para responder a las peticiones de LiberarCajaA;

    private final MailBox[] respuestaAbrirCajaB;
    // Para responder a las peticiones de AbrirCajaB;

    private final MailBox[] respuestaLiberarCajaB;
    // Para responder a las peticiones de LiberarCajaB;

    /****** Selector******/
    private final Selector selector;
    // Objeto necesario para implementar un selector que nos permita escuchar
las peticioens de varios buzones de
    // recibir al mismo tiempo

    /****** Entero ******/
    private int procesosRestantes;
```

```
    // Lleva la cuenta de los procesos que quedan ejecutando.
    // Cuando llega a cero se termina el proceso Controlador.

    /****** Booleanos ******/
    private boolean cajaALibre;
    // Variable para el seguimiento de la exclusión mútua de la Caja A.

    private boolean cajaBLibre;
    // Variable para el seguimiento de la exclusión mútua de la Caja B.

    public Controlador(MailBox solicitudCaja, MailBox buzonTerminar, MailBox
abrirCajaA, MailBox liberarCajaA,
                       MailBox abrirCajaB, MailBox liberarCajaB, MailBox[]
respuestaSolicitudCaja,
                       MailBox[] respuestaAbrirCajaA, MailBox[]
respuestaLiberarCajaA, MailBox[] respuestaAbrirCajaB,
                       MailBox[] respuestaLiberarCajaB, int procesosRestantes) {
        this.solicitudCaja = solicitudCaja;
        this.buzonTerminar = buzonTerminar;
        this.abrirCajaA = abrirCajaA;
        this.liberarCajaA = liberarCajaA;
        this.abrirCajaB = abrirCajaB;
        this.liberarCajaB = liberarCajaB;
        this.respuestaSolicitudCaja = respuestaSolicitudCaja;
        this.respuestaAbrirCajaA = respuestaAbrirCajaA;
        this.respuestaLiberarCajaA = respuestaLiberarCajaA;
        this.respuestaAbrirCajaB = respuestaAbrirCajaB;
        this.respuestaLiberarCajaB = respuestaLiberarCajaB;
        this.procesosRestantes = procesosRestantes;
        // Las cajas compartidas estan
        this.cajaALibre = true;
        this.cajaBLibre = true;
        // Definir el selector
        this.selector = new Selector();
        this.selector.addSelectable(this.solicitudCaja, false);
        this.selector.addSelectable(this.abrirCajaA, false);
        this.selector.addSelectable(this.liberarCajaA, false);
        this.selector.addSelectable(this.abrirCajaB, false);
        this.selector.addSelectable(this.liberarCajaB, false);
        this.selector.addSelectable(this.buzonTerminar, false);
    }

    @Override
    public void run() {
        // Definimos las variables locales a utilizar en este metodo
        Integer id; // El id del proceso que se está comunicando con el
controlador
        int tiempoEspera; // El tiempo de espera estimado para la solicitud de
caja
        String caja; // Variable auxiliar para contener la caja asignada a la
solicitud de caja
```

```
        do {
            // Denifinimos las guardas del selector, se han puesto en orden del
según su aparición en el switch
            solicitudCaja.setGuardValue(true);              // 1
            abrirCajaA.setGuardValue(cajaALibre);           // 2
            liberarCajaA.setGuardValue(!cajaALibre);        // 3
            abrirCajaB.setGuardValue(cajaBLibre);           // 4
            liberarCajaB.setGuardValue(!cajaBLibre);        // 5
            buzonTerminar.setGuardValue(true);              // 6

            switch (selector.selectOrBlock()) {
                case 1:
                    id = (Integer) solicitudCaja.receive(); // Se obtiene el id
del proceso persona según el mensaje
                    tiempoEspera = generarTiempoEspera(); // Se estima el tiempo
que va a tardar

                    // Se le asigna una caja según el tiempo de pago
                    if (tiempoEspera ⩾ MIN_TIEMPO_A) caja = CAJA_A;
                    else caja = CAJA_B;

                    // Se le responde al proceso que solicitó con los datos
estimados
                    respuestaSolicitudCaja[id].send(crearMensaje(tiempoEspera,
caja));
                    break;
                case 2:
                    id = (Integer) abrirCajaA.receive(); // Se obtiene el id del
proceso persona según el mensaje

                    // Se le asigna el valor false a la cajaA para que ningún
otro proceso pueda pedirla
                    cajaALibre = false;

                    // Se le responde al proceso que preguntó para que sepa que
la operación ha terminado
                    respuestaAbrirCajaA[id].send(OK);
                    break;
                case 3:
                    id = (Integer) liberarCajaA.receive(); // Se obtiene el id
del proceso persona según el mensaje

                    // Se le asigna el valor true a la cajaA para liberla
                    cajaALibre = true;

                    // Se le responde al proceso que preguntó para que sepa que
la operación ha terminado
                    respuestaLiberarCajaA[id].send(OK);
                    break;
                case 4:
```

```
                    id = (Integer) abrirCajaB.receive(); // Se obtiene el id del
proceso persona según el mensaje

                    // Se le asigna el valor false a la cajaB para que ningún
otro proceso pueda pedirla
                    cajaBLibre = false;

                    // Se le responde al proceso que preguntó para que sepa que
la operación ha terminado
                    respuestaAbrirCajaB[id].send(OK);
                    break;
                case 5:
                    id = (Integer) liberarCajaB.receive();// Se obtiene el id
del proceso persona según el mensaje

                    // Se le asigna el valor true a la cajaA para liberla
                    cajaBLibre = true;

                    // Se le responde al proceso que preguntó para que sepa que
la operación ha terminado
                    respuestaLiberarCajaB[id].send(OK);
                    break;
                case 6:
                    buzonTerminar.receive(); // Se recibe un mensaje cuando un
proceso persona termina
                    procesosRestantes--; // Se actualiza el número de procesos
que siguen activos
            }
            // Se termina el proceso controlador cuando el número de procesos
cliente sea cero.
        } while (procesosRestantes != 0);
    }

    /**
     * Codifica los argumentos en un mensaje de String para poder pasar varios
valores por MailBox sin tener que crear
     * un objeto serializable. Este mensaje debe ser "decodificado" por el
proceso que lo recibe.
     *
     * @param tiempoEspera tiempo en segundos que el proceso debe espera en caja
     * @param caja         caja asignada
     * @return un string que contiene el tiempo y la caja asignada segados por
un separador
     */
    private String crearMensaje(Integer tiempoEspera, String caja) {
        return tiempoEspera.toString() + SEPARADOR + caja;
    }

    /**
     * Genera un número aleatorio entre MAX_TIEMPO y MIN_TIEMPO que representa
el tiempo en segundos que un proceso
```

```
     * tardará en pasar por caja
     *
     * @return un número entero aleatorio entre MAX_TIMEPO y MIN_TIEMPO
     */
    public Integer generarTiempoEspera() {
        return (int) (Math.random() * (MAX_TIEMPO - MIN_TIEMPO + 1)) +
MIN_TIEMPO;
    }
}
```

PERSONA

```
import messagepassing.MailBox;

public class Persona extends Thread {
    private static final int N_REPETICIONES = 5;
    public static final int MILIS_A_SEGUNDOS = 1000;
    public static final String OK = "ok";
    /****** Buzones de enviar ******/
    private final MailBox solicitudCaja;
    // Recibe las solicitudes de los procesos persona que
    // quieren ser asignados a una caja.

    private final MailBox buzonTerminar;
    // Recibe mensajes de los procesos que terminan. Va asociado a la variable
    // procesosRestantes, cada vez que recibe uno, esta se decrementa. Cuando
    // Llegue a cero el proceso controlador debe terminar.

    // Los siguientes buzones son para el control de la exclusión mutua
    // Sin embargo la exclusión mutua de la pantall se controla con un buzón
    // de testigo descentralizado.

    private final MailBox abrirCajaA;
    // Recibe peticiones de procesos persona que quieren obtener la
    // exclusión mútua de la caja A

    private final MailBox liberarCajaA;
    // Recibe mensajes del proceso que tiene la exclusión mútua de la
    // caja A para liberarla.

    private final MailBox abrirCajaB;
    // Recibe peticiones de procesos persona que quieren obtener la
    // exclusión mútua de la caja B.

    private final MailBox liberarCajaB;
    // Recibe mensajes del proceso que tiene la exclusión mútua de la
    // caja B para liberarla.

    /****** Buzones de recibir******/
    // Como en Java la clase MailBox que representa un buzón no se puede pasar
    // como parámetro por MailBox, se debe tener
```

```
    // los buzones respuesta ya presentes en la clase. De esta manera,
    // simplemente se puede pasar el id del proceso y
    // responder al buzón con la posición igual a este id, que será el de dicho
    // proceso.

    private final MailBox respuestaSolicitudCaja;
    // Para responder a las peticiones de SolicitudCaja

    private final MailBox respuestaAbrirCajaA;
    // Para responder a las peticiones de AbrirCajaA

    private final MailBox respuestaLiberarCajaA;
    // Para responder a las peticiones de LiberarCajaA;

    private final MailBox respuestaAbrirCajaB;
    // Para responder a las peticiones de AbrirCajaB;

    private final MailBox respuestaLiberarCajaB;
    // Para responder a las peticiones de LiberarCajaB;

    /****** Buzón control del a exclusión mutua pantalla con testigo ******/
    private final MailBox mutexEscritura;
    // Este buzón sirve para el control de la exclusión mutua mediante paso de
    // testigo.

    private final Integer id;
    // El id del proceso

    private String caja; // También puede ser un enum
    // Guardamos la caja utilizada para imprimir en el mensaje más tarde

    private Integer tiempoPago;
    // El tiempo que pasa la persona en caja.


    public Persona(Integer id, MailBox solicitudCaja, MailBox buzonTerminar,
    MailBox abrirCajaA, MailBox liberarCajaA, MailBox abrirCajaB, MailBox
    liberarCajaB, MailBox respuestaSolicitudCaja, MailBox respuestaAbrirCajaA,
    MailBox respuestaLiberarCajaA, MailBox respuestaAbrirCajaB, MailBox
    respuestaLiberarCajaB, MailBox mutexEscritura) {
        this.id = id;
        this.solicitudCaja = solicitudCaja;
        this.buzonTerminar = buzonTerminar;
        this.abrirCajaA = abrirCajaA;
        this.liberarCajaA = liberarCajaA;
        this.abrirCajaB = abrirCajaB;
        this.liberarCajaB = liberarCajaB;
        this.respuestaSolicitudCaja = respuestaSolicitudCaja;
        this.respuestaAbrirCajaA = respuestaAbrirCajaA;
        this.respuestaLiberarCajaA = respuestaLiberarCajaA;
        this.respuestaAbrirCajaB = respuestaAbrirCajaB;
```

```
        this.respuestaLiberarCajaB = respuestaLiberarCajaB;
        this.mutexEscritura = mutexEscritura;
    }

    public Integer tiempoPaseoPorLaTienda() {
        return ((int) (Math.random() * 11)) * 1000;
    }

    @Override
    public void run() {
        for (int i = 0; i < N_REPETICIONES; i++) {

            ///// 1 Realiza la compra
            try {
                // Este tiempo de espera simula a la persona paseando por la
                // tienda o algo
                Thread.sleep(tiempoPaseoPorLaTienda());
            } catch (Exception ignored) {}

            ///// 2 Solicita una Caja
            solicitudCaja.send(id); // Se hace una solicitud de caja
            // Se espera la respuesta que es un mensaje que contiene el tiempo
            // de espera en caja y la caja asignada
            String respuestaSolicitud = respuestaSolicitudCaja.
                                        receive().toString();

            // Este Método separa el mensaje escrito en String e inserta los
            // valores en las variables tiempoPago y caja
            decodificarMensaje(respuestaSolicitud);

            ///// 3 y 4 realiza el pago en una caja y las libera
            // Solicitas exclusión mutua de la caja correspondiente
            if (caja.equals(Controlador.CAJA_A)) {
                abrirCajaA.send(id); // Solicita exclusión mutua
                respuestaAbrirCajaA.receive(); // Se espera la respuesta
                try {
                    Thread.sleep(tiempoPago * MILIS_A_SEGUNDOS); // Se espera el
                                                                 // tiempo de pago
                } catch (Exception ignored) {}

                liberarCajaA.send(id); // Se envia petición de liberaciópn de la
                                       // exclusión mutua
                respuestaLiberarCajaA.receive(); // Se recibe un mensaje cuando
                                                 // se libera exitosamente
            } else {
                abrirCajaB.send(id); // Solicita exclusión mutua
                respuestaAbrirCajaB.receive(); // Se espera la respuesta
                try {
                    Thread.sleep(tiempoPago * MILIS_A_SEGUNDOS); // Se espera el
                                                                 // tiempo de pago
                } catch (Exception ignored) {}
```

```
                liberarCajaB.send(id); // Se envia petición de liberaciópn de la
                                       // exclusión mutua
                respuestaLiberarCajaB.receive(); // Se recibe un mensaje cuando
                                                 // se libera exitosamente
            }

            ///// 5 Imprime en pantalla información
            mutexEscritura.receive(); // Solicita la exlucisón mutua

            // Imprime por pantalla la información del pago
            // informaciónDelPago forma el mensaje según enunciado
            System.out.println(informacionDelPago());

            mutexEscritura.send(OK); // Solicita la exlucisón mutua
        }
        buzonTerminar.send(OK); // Se informa al controlador cuando se termina
                                // de el proceso
    }

    private String informacionDelPago() {
        return "\nPersona " + id + " ha usado la caja " + caja + "\nTiempo de
pago = " + tiempoPago + "\n\tThread.sleep(" + tiempoPago + ")\n" + "Persona " +
id + " liberando la caja " + caja;
    }

    private void decodificarMensaje(String respuestaSolicitud) {
        // Se separa los elementos por el separador
        String[] elementos = respuestaSolicitud.split(Controlador.SEPARADOR);
        // Se castea el primero a integer, este representa el tiempo de pago
        this.tiempoPago = Integer.valueOf(elementos[0]);
        this.caja = elementos[1]; // Este valor representa la caja a la que se
                                  // le ha asignado el proceso
    }
}
```

1.2 Recursos No Compartibles y Secciones Críticas

Los recursos no compartibles en este algoritmo son las cajas A y B y las pantalla. Los dos primeros lo hacen con buzones centralizados en el controlador, mientras que la pantalla se hace a través de paso de testigo.

En las cajas tenemos primero una simulación de una cola inicial de solicitud de caja donde se le asigna un tiempo de espera y una caja según este tiempo de espera al proceso Persona. En el pseudocódigo esto se consigue teniendo un buzon centralizado de solicitudes donde las Personas envían sus buzones donde esperan respuesta, de esta manera el controlador puede ir asignando y enviando los resultados a los buzones respuesta recibidos a través del buzón de solictudes centralizado. De la misma manera se

utilizan los buzones de solicitud y liberación de caja. En el de solicitud de caja que solo se lee si la variable booleana de caja dice que esta se encuentra libre, se le da el estado ocupado a la caja, dandole así la exclusión mutua al proceso Persona que envió el buzón de respuesta. Por el contrario, en el buzón liberar solo se lee cuando la caja correspondiente se encuentra ocupada y solo le puede enviar mensajes el proceso Persona que actualmente tiene la exclusión mutua.

Cuando los procesos Persona tienen la exclusión mutua pueden ejecutar la sección critica que se encuentra entre las llamadas de abrir caja y liberar caja que garantizan la exclusión mutua. Como se trata de una simulación solo se duerme el proceso en estas.

El recurso no compartible pantalla asegura la exclusión mutua mediante paso de testigos. El proceso main envía el primer mensaje. Desde entonces, los procesos Persona solo tieen que ejecutar la sección criti entre las llamadas de recibir mensjae y enviar otro para pasar el testigo al siguiente proceso.