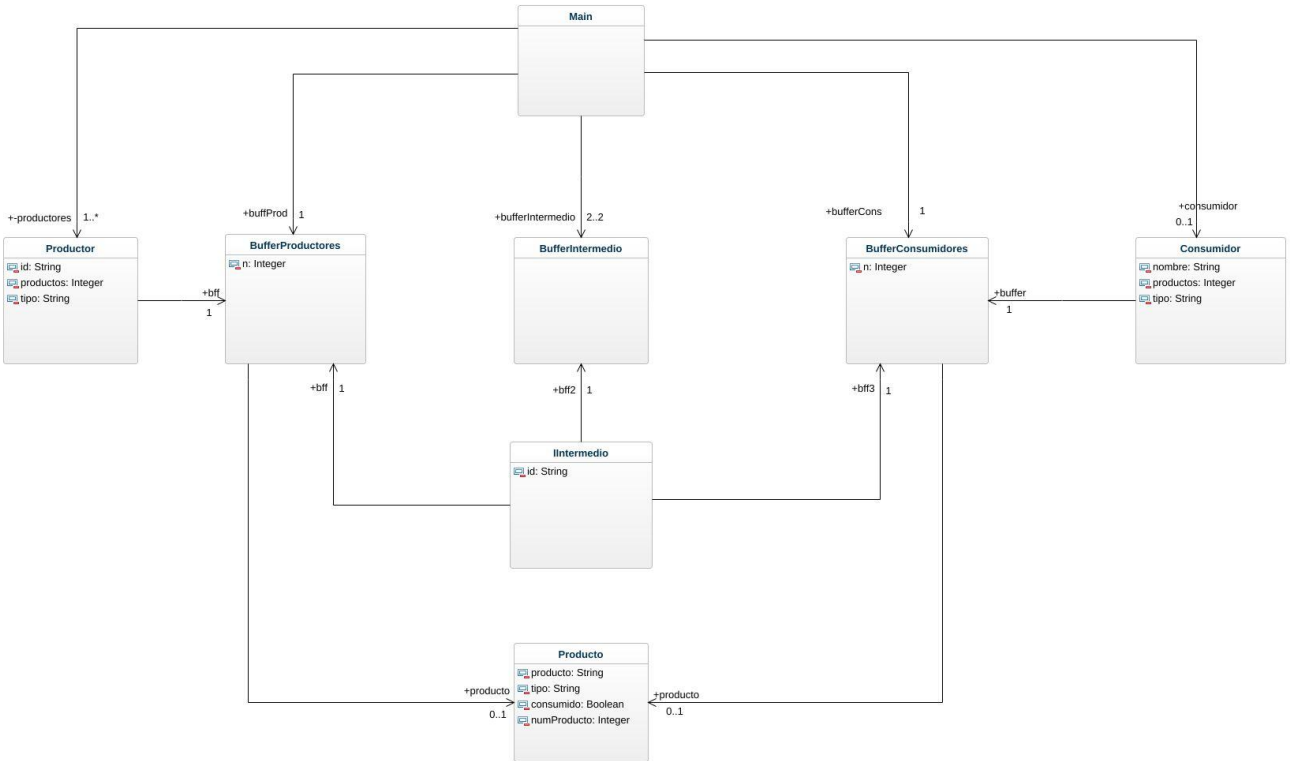
“Caso 1 – Manejo de Concurrencia”

Luis D. Castro Gonzalez, Andrés M. Ochoa Toro
Caso 1 – Infraestructura Computacional
Universidad de los Andes, Bogotá, Colombia
Fecha de presentación: Febrero 22 de 2021

1 Modelo Conceptual

El funcionamiento del caso de manejo de concurrencia se ve representado en el siguiente diagrama de clases. Se parte de la clase principal (clase Main), la cual ejecuta la creación de todas las demás clases y su respectivo inicio. El funcionamiento está ordenado a partir de la creación threads que ejecutan la creación de productos dentro de la clase Productor. Dentro de esta clase, después de crearse un producto se procede a enviarlo e intentar guardarlo dentro del buzón de productores (clase BufferProductores), la cual representa el buzón de los productores. Si no se pueden almacenar en el momento porque el buzón está lleno, se procede a dejar en espera el Thread hasta que se vuelva a tener espacio. El encargado de pasar los productos del buzón de productores al buzón intermedio es la clase Intermedio. Esta clase pasa un único producto a la vez a la clase BufferInteremedio, la cual es capaz de almacenar un único producto a la vez.

Un segundo intermediario es el encargado de pasar los productos del buzón intermedio (clase BufferIntermedio) al buzón de productores (clase BufferConsumidores). De este último buzón es donde los consumidores toman sus productos de manera concurrente. Si sucede el caso de que no existen productos a retirar, el consumidor queda en espera hasta que consiga productos de su tipo.



El proceso de sincronización entre los productores y el intermediario 1 se realiza a partir del buzón de productores (BufferProductores). El buzón es el encargado de recibir productos hasta alcanzar su capacidad máxima y entregarlos al primer intermediario. El cual los entrega a un buzón intermedio.

La creación del producto está a cargo del Productor. Este crea un producto a partir de su id y su tipo, y se los asigna al producto. Esto con el objetivo de tenerlo como identificador.

```
synchronized public void run(){
	if(this.tipo=="a") {
		for (int i = 0; i < productos; i++) {
			Producto producto = new Producto(id, i,"a");
			bff.almacenar(producto);
		}
	}
	if(this.tipo=="b") {
		for (int i = 0; i < productos; i++) {
			Producto producto = new Producto(id, i,"b");
			bff.almacenar(producto);
		}
	}

	System.out.println("\n--------------------\nEl Productor " + id + " terminó de
	producir.\n--------------------");
}
```

Al decirle al buzón que reciba un producto, este intenta guardarlo. Si no es posible guardarlo, se procede a dejar los Threads de los productores en espera activa hasta que se libere un espacio para ser almacenado. Por último, avisa sobre el objeto vacío con el fin de permitir que se intenten agregar más productos.

```
public void almacenar ( Producto m ){
	boolean continuar = true;
	while(continuar) {
		synchronized(this) {
			if(buff.size()<n) {
				buff.add(m);
				continuar = false;
			}
		}
		if(continuar) {
				Thread.yield();
		}
	}
	synchronized(vacio) {
		try { vacio.notify();}
		catch (Exception e) {e.printStackTrace();}
	}
}
```

De manera sincrónica, un intermediario está intentando obtener productos del buzón de productores para dejarlo en un buzón intermedio. Intentar retirar un producto se realiza con espera pasiva en el caso de que no se puedan retirar productos. Solo se procederá volver a intentar retirar productos cuando el intermediario sea notificado de que se agregaron nuevos productos.

```
synchronized public void run(){
	while(true) {
		if(id=="intermedio1") {
			Producto producto = bff.retirar("c");
			bff2.almacenar(producto);
		}
		if(id=="intermedio2") {
			Producto producto = bff2.retirar();
			bff3.almacenar(producto);
		}
	}
}


public Producto retirar (String pTipo){
	boolean continuar = true;
	boolean hay = false;
	int posicion = 0;
	Producto m =null;
	while(continuar) {
		synchronized(this) {
			if(pTipo=="c") {
				hay = true;
				posicion = 0;
			}
			else{
				for(int i = 0; i<buff.size() ; i++) {
					if(buff.get(i).getTipo() == pTipo){
						hay = true;
						posicion = i;
						break;
					}
				}
			}
			if(buff.size()>0 && hay == true) {
				m = buff.remove(posicion);
				continuar = false;
			}
		}
		if(continuar) {
			synchronized(vacio) {
				try{vacio.wait();}
				catch (Exception e) { e.printStackTrace(); }
			}
		}
	}
	return m;
}
```

Después de pasar el producto al buzón intermedio, se intenta guardarlo. Este buzón es de tamaño 1, por lo cual en el caso de que este se encuentre lleno, se hace esperar al intermediario con espera pasiva hasta que este buzón sea desocupado, donde intentará volver a agregar el producto.

```
public void almacenar ( Producto m ){
	boolean continuar = true;
```

```
		while(continuar) {
			synchronized(this) {
				if(buff.size()<n) {
					buff.add(m);
					continuar = false;
				}
			}
			if(continuar) {
				synchronized(lleno) {
					try{lleno.wait(); }
					catch (Exception e) { e.printStackTrace(); }
				}
			}
		}
		synchronized(vacio) {
			try { vacio.notify();}
			catch (Exception e) {e.printStackTrace();}
		}
	}


	public Producto retirar (){
		boolean continuar = true;
		Producto m =null;
		while(continuar) {
			synchronized(this) {
				if(buff.size()>0) {
					m = buff.remove(0);
					continuar = false;
				}
			}
			if(continuar) {
				synchronized(vacio) {
					try{vacio.wait();}
					catch (Exception e) { e.printStackTrace(); }
				}
			}
		}
		synchronized(lleno) {
			try{ lleno.notify();}
			catch (Exception e) { e.printStackTrace(); }
		}
		return m;
	}
```

En este buzón se repite el proceso de pasar el producto del buzón intermedio al buzón de los consumidores. Este proceso es similar al caso de pasar del buzón de consumidores al siguiente buzón. La diferencia en este escenario es que en lugar de implementar espera semi-activa, se hace espera pasiva al intentar agregar un producto sobre el buzón.

Por último, los consumidores intentan obtener productos de su buzón. El intento de que los consumidores tomen productos puede que no sea complete, porque el buzón se encuentre vacío o porque no hay productos del mismo tipo del consumidor, razón por la cual va a quedar en espera semi-activa. Mientras el consumidor no logre obtener la cantidad de productos esperada, seguirá accediendo al buzón para tomar más productos.

```
public void run(){
	for (int i = 0; i < productos; i++) {
		if(this.tipo == "a") {
			Producto producto = buffer.retirar("a");
			System.out.println(producto.getProducto() + " entregado al
			consumidor " + nombre+" de tipo "+ tipo);
			producto.cambiarConsumido();
		}
		if(this.tipo == "b"){
			Producto producto = buffer.retirar("b");
			System.out.println(producto.getProducto() + " tomado por el
			consumidor " + nombre +" tipo "+tipo);
			producto.cambiarConsumido();
		}
	}

System.out.println(" \n--------------------\nEl Consumidor " + nombre + " ya tomó
todos sus productos \n--------------------");
}
```