Escuela Técnica Superior de Ingeniería

Prácticas de Programación Concurrente y Distribuida

3º Curso de Grado en Ingeniería Informática

PRÁCTICA 3

Sincronización y Creación de *Frames* en JAVA

El objetivo de la práctica es conocer la forma de sincronizar métodos de objetos concurrentes y familiarizarse con la creación de *Frames* y la representación gráfica a través de la clase `Canvas`.

Se tomará como base la pila implementada tras la práctica 2 (`PilaLenta`).

1. Se debe modificar la clase `PilaLenta` para evitar los problemas derivados del acceso concurrente de los hilos, observados en la práctica 2. Para ello se sincronizarán los métodos `Apila` y `Desapila`.

2. Se debe crear un nuevo *Frame* que lance un hilo `productor` y otro hilo `consumidor` y represente gráficamente la evolución de los elementos en la pila, según son insertados o extraidos. Para tal fin, se creará la estructura de clases que representa el diagrama de clases de la figura 1.

La función `main` de la clase `PilaFrame` realizarán las siguientes acciones:

- Crear los objetos `PilaFrame`, `PilaLenta`, `CanvasPila`, y los objetos `productor` y `consumidor`.

- Definirá el tamaño y colores del *Frame* y añadirá el *canvas* al mismo.

- Lanzará los hilos `productor` y `consumidor`.

- Esperará a que los hilos `productor` y `consumidor` finalicen, esperará tres segundos y finalizará la ejecución.

Los métodos de la clase `CanvasPila` realizarán las siguientes acciones:

- `update`: Este método sobrecarga el método `update` de la clase `Canvas`. Es invocado cada vez que se invoca al método `repaint` de la clase `Canvas`. El método `update` de `Canvas` borra el *canvas*, lo repinta con el color del fondo e invoca al método `paint`. Para evitar problemas de rendimiento en la visualización, se debe sobrecargar este método en `CanvasPila` para que invoque al método `paint`. Con esto se consigue que no se borre el *canvas* cada vez que se invoca a `repaint`.

- `paint`: Es el encargado de dibujar en el *canvas*. Lo invoca automáticamente `update` tras una llamada a `repaint`. Para mejorar la representación se usa una imagen a modo de *buffer* sobre la que se dibuja, en lugar de hacerlo sobre el parámetro `g` de tipo `Graphic`. Una vez compuesta la imagen en el *buffer* se dibuja en `g`.

- `avisa`: será invocado cada vez que se desee mostrar el mensaje de "*PILA LLENA*" o "*PILA VACIA*". Tomará como parámetro el mensaje a mostrar. Una vez modificada la información, invocará a `repaint` para actualizar la información del *canvas*.

- `representa`: será invocado cada vez que haya cambios en la pila que deban ser representados en pantalla. Toda la información sobre el estado de la pila será recogida en los parámetros pasados. Una vez modificada la información, invocará a `repaint` para actualizar la información del *canvas*.

La clase `PilaLenta` deberá ser modificada para representar la información cada vez que cambia. Para ello, deberá tener una referencia a la instancia de `CanvasPila` creada en `PilaFrame` e invocar a los métodos `avisa` y `representa` según se *apilen* o *desapilen* elementos de la pila.

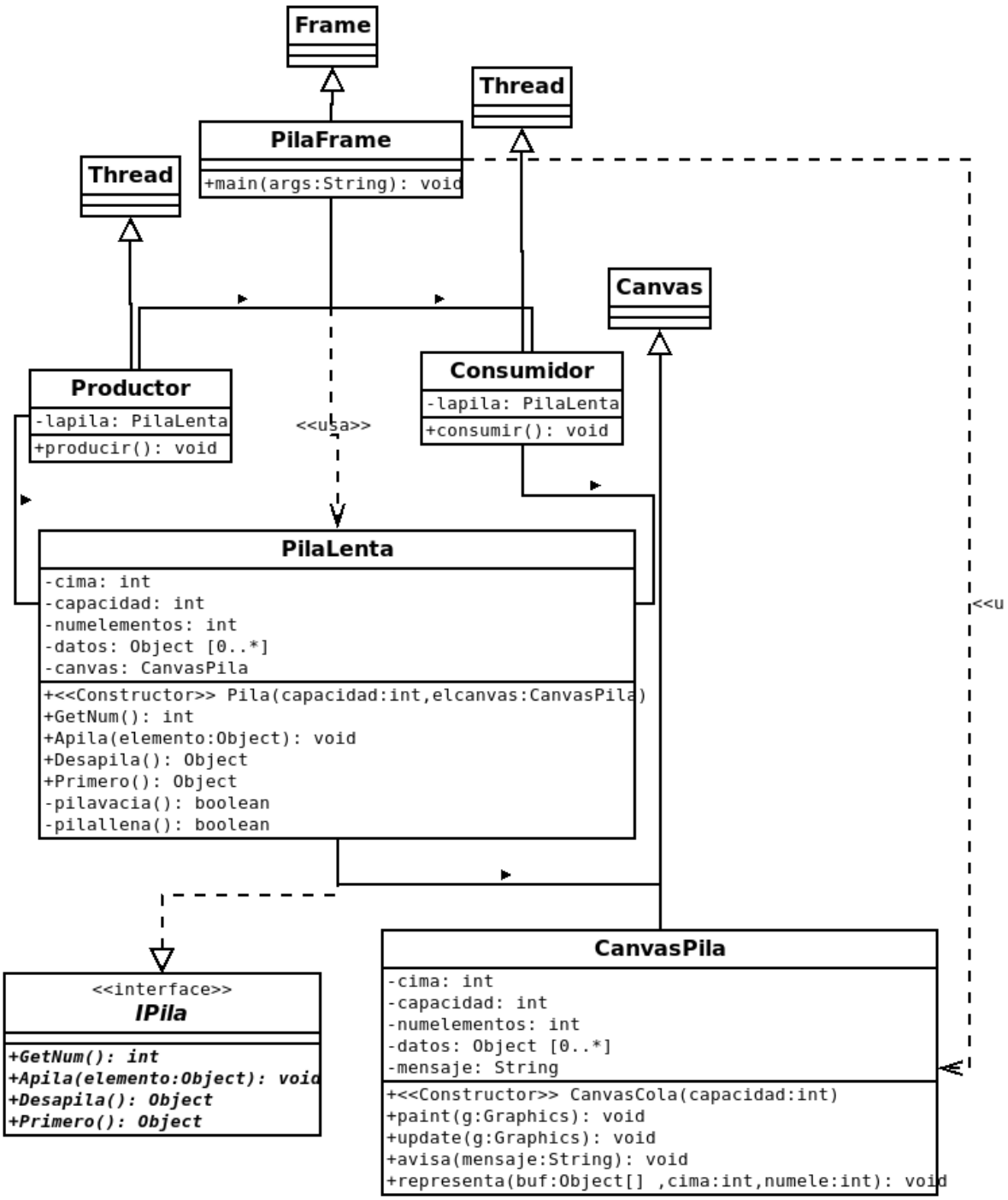

Figura 1: Diagrama de Clases de la práctica 3