Programación Interactiva
Técnicas de Animación en Java usuando Hilos

Escuela de Ingeniería de Sistemas y Computación
Facultad de Ingeniería
Universidad del Valle

Conceptos de Animación

- Existen diferentes maneras para producir efectos de animación en un programa, por ejemplo:
- Crear varias secuencias de una imagen e ir colocándolas sobre el programa una tras de otra Utilizar el API Timer del paquete swing
- Utilizar el concepto de hilo, sobre todo en Applets

- En este capítulo nos enfocaremos a crear animaciones con el API Timer y con el concepto (muy somero aún) sobre Hilos.

Conceptos de Hilos

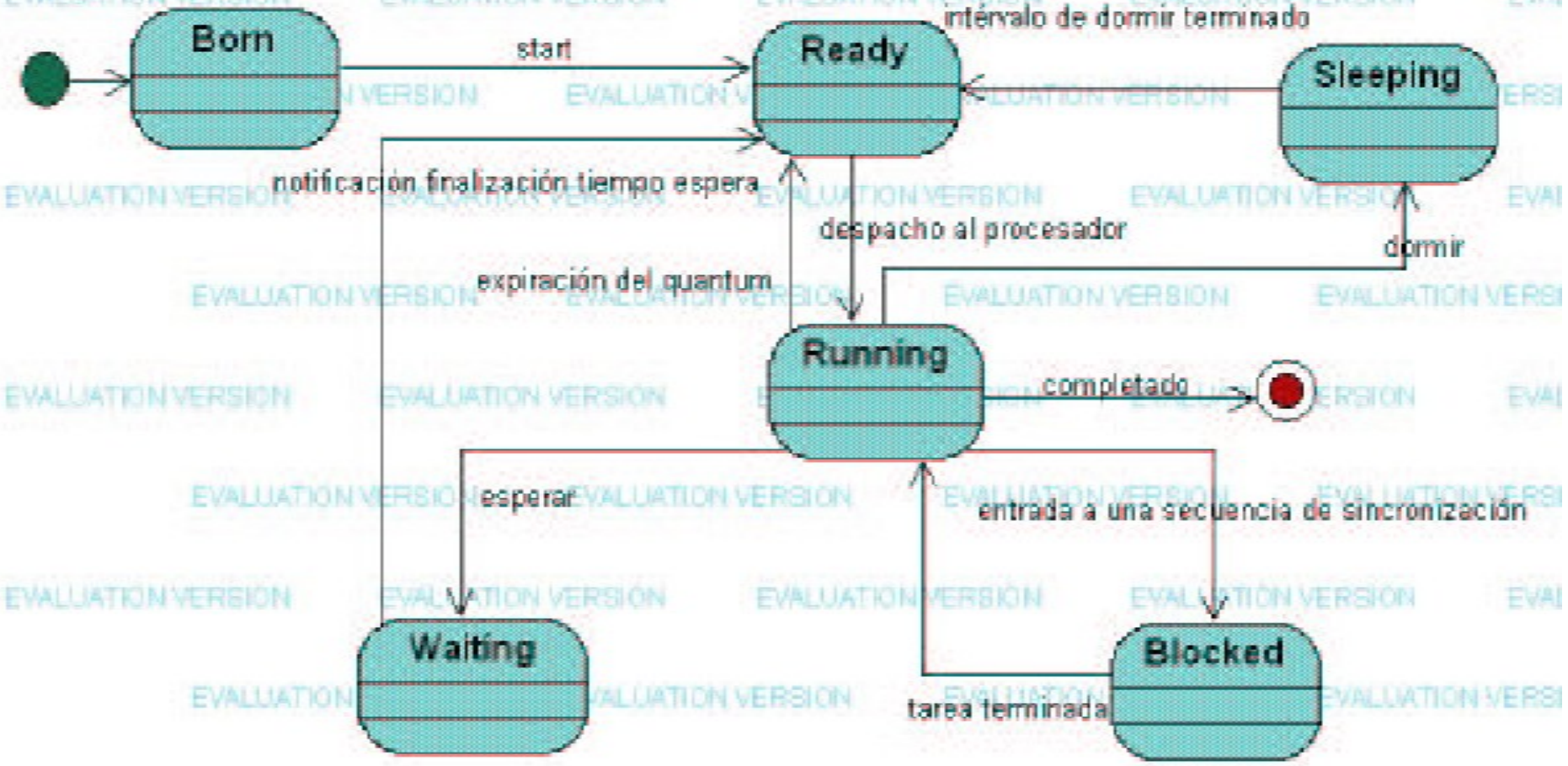
- **Hilo:** es un proceso, actividad o tarea que se ejecuta dentro de un programa en Java.
- La utilización de hilos en un programa permite a éste simular que se están ejecutando al tiempo varias y diferentes tareas.
- Hay que recordar que para Java, un programa en tiempo de ejecución a parte de ser un objeto también es tratado como un hilo.

Conceptos de Hilos

- *Cómo ayuda entonces el concepto de hilo en la realización de un programa que dentro de sus funciones tenga asociada una animación*?

- Un hilo permite que un programa en Java pueda manejar una tarea (de animación en este caso) que depende de algunas funciones del programa, permite comezarla, terminarla, actualizarla, detenerla durante un tiempo, es decir, permite manipularla dependiendo de las acciones mismas del programa.

Ciclo de Vida de un Hilo



Ciclo de Vida de un Hilo

- Un hilo siempre permanece en alguno de los estados de la gráfica anterior durante su ciclo de vida en un programa.

- El diagrama anterior es un diagrama de estados de UML, para la clase Thread.

Descripción de los Estados de un Hilo

Estado Born o de Creación:

- Este estado representa la fase del ciclo de vida del hilo en la cual, se ha creado una instancia del hilo.

- Ejemplo:
  1. Para el API Timer:

     Timer t = new Timer();
  2. Para la clase Thread:

     Thread hilo = new Thread();

Descripción de los Estados de un Hilo

Estado Ready o de Inciación de la ejecución:

- Este estado representa la fase del ciclo de vida del hilo en la cual se le ha dado la orden al hilo de que comience la ejecución de su tarea-

Ejemplo:

1. Para el API Timer:

   t.start();

2. Para la clase Thread:

   hilo.start();

Descripción de los Estados de un Hilo

Estado Running o de Ejecución:

- Este estado representa la fase del ciclo de vida del hilo en la cual inmediatamente después del estado start, el hilo continúa su ejecución de manera indeterminada.

Ejemplo:

Tanto para Timer como para Thread se llega a este estado cuando se ha invocado al método start() casi inmediatamente.

Descripción de los Estados de un Hilo

Estado Sleeping o Durmiendo:

- En este estado el hilo va a "descansar" de su ejecución natural, es decir, aparentemente va a estar suspendido durante unos cuantos milisegundos, luego de los cuales continuará con su ejecución invocando al método start() y llegando de nuevo al estado Ready.

Ejemplo:

1. Para el API Timer: es controlado por el actionPerformed asociado al timer
2. Para la clase Thread:

```
hilo.sleep(numMilisegundos);
```

Método sleep de la clase Thread

- Este método arroja una exception de tipo InterruptedException, en caso tal que abruptamente el hilo termine su ciclo de vida en tiempo de ejecución. Es por eso que la invocación de éste método va dentro de un bloque try-catch.

Ejemplo:

```
try{
   hilo.sleep(numMilisegundos);
}catch(InterruptedException ex){...}
```

Descripción de los Estados de un Hilo

Estado Blocked o Bloqueado:

- Este estado permite que el hilo quede bloqueado mientras que la tarea que está realizando pueda terminarse, puesta que hay en ejecución una solcitud de I/O que no permite que sea terminada dicha tarea.

Ejemplo:

2. Para la clase Thread:

```
synchronized(this);
```

Descripción de los Estados de un Hilo

Estado Waiting o de Espera:

Este estado el hilo espera a que una condición esté satisfecha para continuar. Una vez así sea, el objeto encargado de información que la condición está satisfecha realiza una invocación al método notify() o notifyAll() para que el hilo regrese al estado Ready.

Ejemplo:

2. Para la clase Thread:

```
hilo.wait();
```

Formas de Producir Animación

- Existen diferentes maneras de producir animación, pero en este caso se trabajará con:

  1. El API Timer
  2. Hilos en Applets

Clase Timer

- Lanza una o más eventos de acción después de una demora específica. Por ejemplo, una animación puede usar un Timer para mostrar animadamente varios cuadros de dibujo de un componente

Clase Timer

Pasos para la utilización de un Timer:

1. Crear un objeto de tipo Timer
2. Registrar uno o más action listeners en él
3. Inicializar su ejecución con el método start().

Clase Timer

Componentes de un Timer:

- uno o más *action listeners* un *delay* (tiempo entre cada una de las acciones).

Cómo trabaja?

- Cuando los milisegundos especficados en el atributo *delay* han pasado, el Timer lanza un action event a sus listeners. Por defecto éste ciclo se repite hasta que el método stop es llamado.

Clase Timer

Aspectos sobre cómo trabaja el Timer

- Para que el Timer sólo realice la acción una vez y no la repita, se puede invocar al método setRepeats(false).
- Para hacer el tiempo incial diferente del tiempo de acción entre los diferentes eventos se puede usar el método setInitialDelay().

Clase Timer

- Si se crean varios objeto de tipo Timer en un programa, todos ejecutan su estado de waiting usando un solo hilo compartido creado por el pimer objeto Timer que se ejecuta.
- Los manejadores de eventos para los Timers se ejecutan en otro hilo (el que despacha los eventos). Esto significa que los manejadores de eventos para los Timers pueden ejecutar operaciones de manera segura e los componentes Swing.

Clase Timer: Atributos

protected EventListenerList

listaEscuchas

- Un objeto de esta clase contiene una lista de actionListeners.

Clase Timer: Constructor

Timer (int delay, ActionListener escucha)

- Crea un objeto de tipo Timer que notifica a su(s) escucha(s) cada delay milisegundos. Si escucha no es nulo, éste se registra como el action listener del timer.

- Dónde: delay es un entero que indica cada cuánto es notificado el action listener. escucha es el actionListener que describe lo que el Timer va a ejecutar cada delay milisegundos

Clase Timer: Métodos

restart

- Reincia al Timer para que vuelva y comience su ejecución, cancelando actividades pendientes y causando que inicie con su tiempo - delay definido al principio.

Sintaxis:

public void **restart**()

Clase Timer: Métodos

setInitialDelay

- Asigna al Timer un delay inicial, el cual por defecto es el mismo entre cada uno de los eventos producidos. Este delay es sólo usado para la primera ejecución del Timer.
- *Parámetros:* delayInicial: definido en milisegundos, es el tiemo que va a demorar su ejecución entre la invocación de start() y la primera activación del action event.

Sintaxis:

public void setInitialDelay(int delayInicial)

Clase Timer: Métodos

start

- Comienza el timer produciendo el envío de un action event al ActionListener.

Sintaxis:

public void **start()**

Clase Timer: Métodos

stop

- Para la ejecución del Timer,causando en éste una acción de stop que es mandada como actionEvent a lo(s) listener(s).

Sintaxis:

public void **stop()**

Un Timer dentro de una Animación (Pasos)

1. Importar el API Timer en la sección de atributos import javax.swing.Timer;
2. Declarar el timer en la sección de atributos Timer relojBolas;
3. Declarar el objeto de tipo ActionListener que va a controlar al timer ActionListener escuchaReloj;

Un Timer dentro de una Animación (Pasos)

4. Crear el objeto de tipo ActionListener incluyendo la sobreescritura del método actionPerformed

```
escuchaReloj= new ActionListener(){
    public void
        actionPerformed(ActionEvent e){
                areaBolas.moverBolas();
                areaBolas.repaint();
        }
```

El método actionPerformed se incluye en la creación del objeto ActionListener y dentro de el se incluyen las instrucciones que se quiere el timer ejecute cada delay milisegundos

Un Timer dentro de una Animación (Pasos)

5. Crear el objeto de tipo Timer con el valor de milisegundos que le va a permitir cada cuánto activarse y el objeto actionListener creado en el punto 3.

```
relojBolas = new
Timer(delay,escuchaReloj);
```

Un Timer dentro de una Animación (Pasos)

6. Invocar al método start() del timer para que inicie su ejecución

```
relojBolas.start();
```

Esta invocación debe ir en el lugar del código desde dónde se quiere controlar la animación.

Ejemplo. El actionPerformed de un botón de Inicio

Ver Baloto