

Concurrencia

Concurrencia

- **Concurrencia.** Capacidad para realizar varios cálculos al mismo tiempo.
- Ejemplos:
- Aplicaciones que se ejecutan a la vez en una computadora.
- Un programa que corre en distintos cores de un procesador.
- Un servidor web que atiende a varios usuarios a la vez.
- Una aplicación móvil que realiza parte de su procesamiento en un servidor (“en la nube”).



Concurrencia

- Una aplicación que realiza una tarea larga en un segundo plano para no bloquear el programa principal.
- Un medio ambiente de programación (IDEA, Netbeans, etc.) que compila el programa mientras se modifica.



Procesos e hilos

- Hay dos tipos de módulos concurrentes: procesos e hilos.

Procesos e hilos

- **Proceso.** Programa en ejecución independiente y aislado de otros procesos.
- **Hilo.** Segmento de código que se ejecuta de manera concurrente a otros hilos y que es parte de un proceso.
- Un proceso puede tener varios hilos ejecutándose al mismo tiempo.
- Los hilos y los procesos tiene su propia pila, su propio contador de programa y sus propias variables locales.

Diferencias

Parámetro	Procesos	Hilos
Tiempo de terminación	Necesitan más tiempo	Necesitan menos tiempo
Tiempo de creación	Toman más tiempo	Toman menos tiempo
Comunicación entre pares	Necesitan más tiempo	Necesitan menos tiempo
Context switching	Necesitan más tiempo	Necesitan menos tiempo
Tratamiento por el sistema operativo	Procesos diferentes son tratados por separado	Todos los hilos al mismo nivel son tratados como una sola tarea
Memoria	No comparten memoria	Comparten memoria
Datos	No comparten datos	Comparten datos



Procesos en Java

- La mayoría de las implementaciones de la máquina virtual de Java se ejecutan como un solo proceso.
- Una aplicación Java puede crear procesos adicionales utilizando un objeto `ProcessBuilder`.
- Las aplicaciones multiproceso están más allá del alcance de este curso.

Hilos en Java

- Cada hilo está asociado con una instancia de la clase Thread.
- Hay dos estrategias básicas para usar objetos Thread para crear una aplicación concurrente.
 1. Crear un objeto Thread.
 2. Crear un objeto Executor.
- La opción 1 permite tener control sobre la creación y administración de los hilos.
- La opción 2 permite separar la administración de hilos del resto de la aplicación y es la preferida.

Interfaces Executor

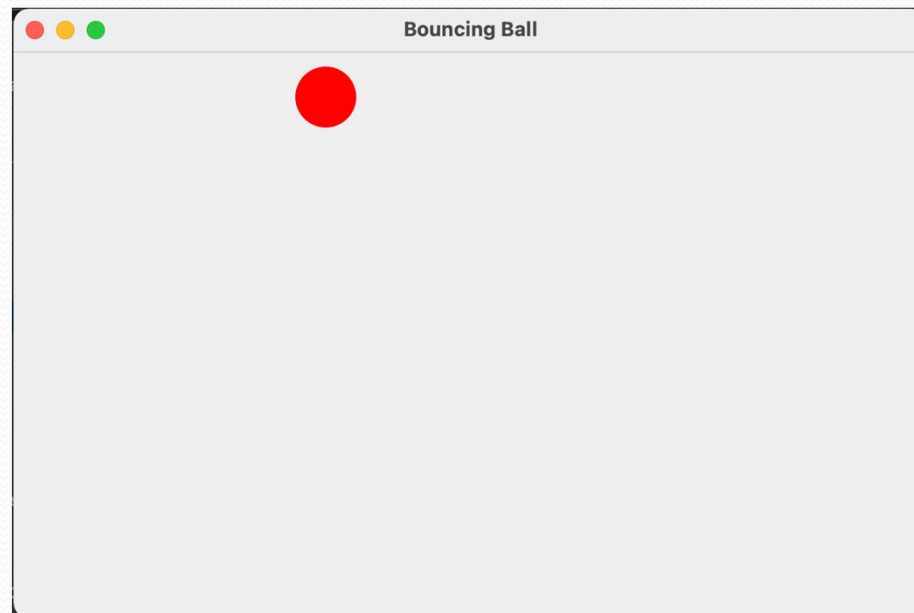
- El paquete `java.util.concurrent` define tres interfaces Executor:
 1. `Executor`. Una interface simple que permite el lanzamiento de nuevas tareas.
 2. `ExecutorService`. Agrega características que ayudan a administrar el ciclo de vida, tanto de las tareas individuales como del propio ejecutor.
 3. `ScheduledExecutorService`. Permite la ejecución futura y / o periódica de tareas.
- El tutorial de Java recomienda utilizar las interfaces `ExecutorService` o `ScheduledExecutorService`.

Interface ScheduledExecutorService

- Tiene los siguientes métodos:
- `void execute(Runnable r)`. Ejecuta el objeto `r`.
- `Future<?> submit(Runnable/Callable r)`. Ejecuta el objeto `r` y regresa una referencia para administrarlo.
- `ScheduleFuture<?> schedule(Runnable/Callable r, long delay, TimeUnit unit)`. Ejecuta el objeto `r` después de un retraso.
- `ScheduleFuture<?> scheduleAtFixedRate(Runnable r, long initialDelay, long period, TimeUnit unit)`. Ejecuta el objeto `r` después de un retraso inicial y luego lo repite con el periodo dado.

Ejemplo

- Hacer un programa con una pelota animada que rebote contra las paredes de la ventana.



Clase Ball

```
public class Ball {  
    private final Point center;      // Centro de la pelota  
    private int vX, vY;              // Velocidad  
    private final int radius;        // Radio  
    private Color color;             // Color  
    private final Image image;       // Imagen  
    // Constructor  
    public Ball(Point center, int vX, int vY, int radius, Color color, Image image)
```


Clase Ball

// Métodos

```
public void move(int width, int height)
```

```
public void draw(Graphics2D g2)
```

Clase MainPanel

```
public class MainPanel extends JPanel {  
    private static final int FPS = 60;  
    private final Ball ball;  
    public MainPanel() {  
        super();  
        ball = new Ball(new Point(100, 100), 5, 2, 20, Color.red, null);  
        Runnable r = this::updateGame;           // Referencia a método  
        ScheduledExecutorService executor =  
Executors.newSingleThreadScheduledExecutor();  
        executor.scheduleAtFixedRate(r, 10, 1000 / FPS, TimeUnit.MILLISECONDS);  
    }  
}
```

Clase MainPanel

```
public void paint(Graphics g) {  
    Graphics2D g2 = (Graphics2D) g;  
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,  
                        RenderingHints.VALUE_ANTIALIAS_ON);  
    ball.draw(g2);  
}  
private void updateGame() {  
    ball.move(getWidth(), getHeight());  
    repaint();  
}}
```


Modificación

- En lugar de `Executors.newSingleThreadScheduledExecutor()` se puede utilizar `Executors.newScheduledThreadPool(1)` con los mismos resultados.
- Agregar que la pelota se detenga después de 10 segundos.
- Los únicos cambios son en el constructor de `MainPanel`.

Ejemplo modificado

```
public MainPanel()
{
    ...
    ScheduledExecutorService executor =
        Executors.newScheduledThreadPool(1);
    ScheduledFuture<?> handle = executor.scheduleAtFixedRate(r, 10,
        1000 / FPS, TimeUnit.MILLISECONDS);
    Runnable canceller = () -> handle.cancel(false);
    executor.schedule(canceller, 10, TimeUnit.SECONDS);
}
```

Juegos en 2D

- Los caracteres y el campo de juego utilizan dos dimensiones (x y y).
- Ejemplos:
 - Juegos de mesa: Gato, Memoria, Reversi, Bejeweled, etc.
 - Juegos de arcade: Pac-Man, Invasores del espacio, Digger, etc.
- No requieren cámara virtual ni proyecciones.

Proceso de diseño

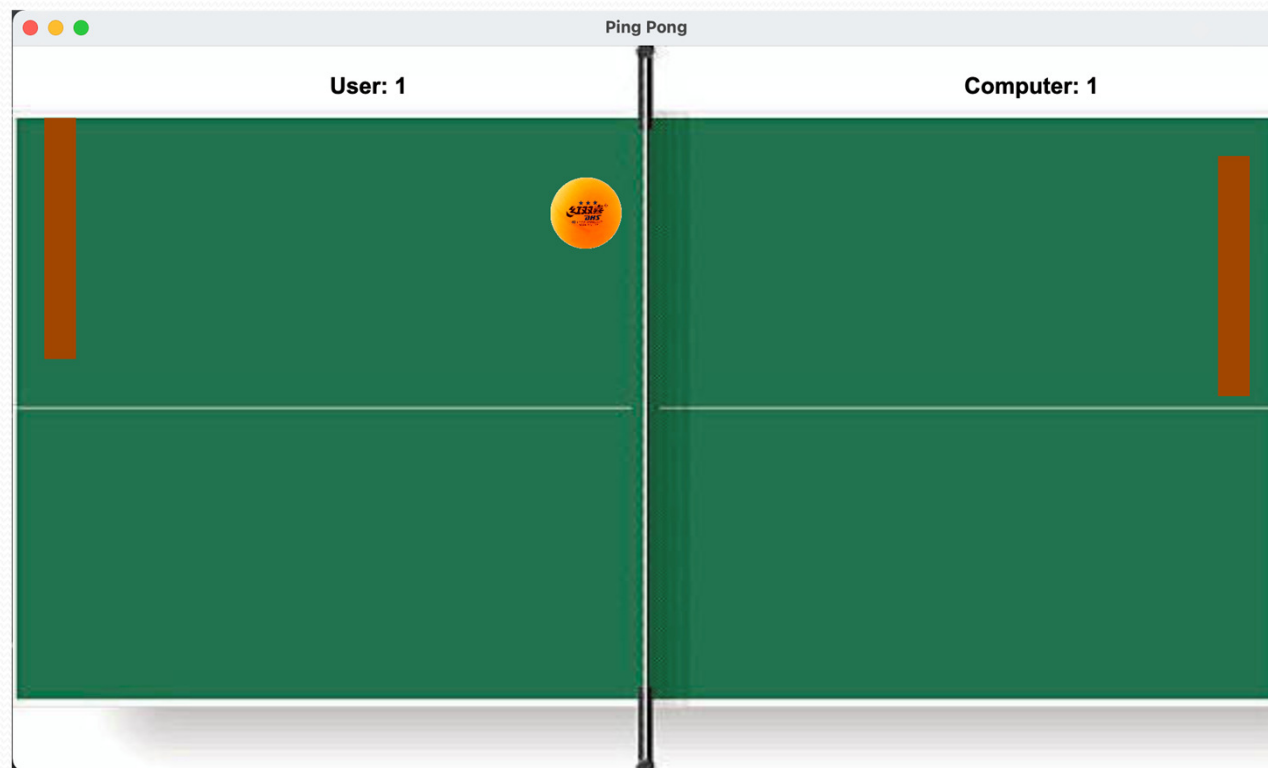
- Componentes del juego:
 - Objetos: pelotas, fantasmas, Mario, dragones, etc.
 - Campo de juego: tablero, espacio sideral, paisaje, etc.
- Reglas:
 - Objetivos.
 - Objetos autónomos y objetos controlados por el jugador.
 - Movimiento.
 - Puntaje.
 - Condiciones de victoria.

Game loop (ciclo del juego)

- Corre en un hilo aparte del hilo principal.

```
while (!termine) {  
    revisar input del usuario  
    ejecutar AI si existe  
    mover los objetos autónomos  
    resolver colisiones  
    revisar condiciones de victoria o derrota  
    actualizar las gráficas  
    tocar música  
}
```


Ejemplo: ping-pong



Ciclo de juego

```
private void update()
```

```
{
```

```
    if (ball.isMovingUp()) {
```

```
        racket2.moveUp();
```

```
    }
```

```
    else {
```

```
        racket2.moveDown(height);
```

```
    }
```

```
// Mueve la pelota y revisa si rebota con las raquetas
```

```
ball.move(height, racket1, racket2);
```

```
// La raqueta 2 sigue a la pelota
```

Ciclo de juego

```
if (ball.getX() < 0) {           // Punto para la computadora
    goals2++;
    ball.moveToCenter(width, height);
}
if (ball.getX() > width) {       // Punto para el jugador
    goals1++;
    ball.moveToCenter(width, height);
}
repaint();
}
```