Fundamentos de Java

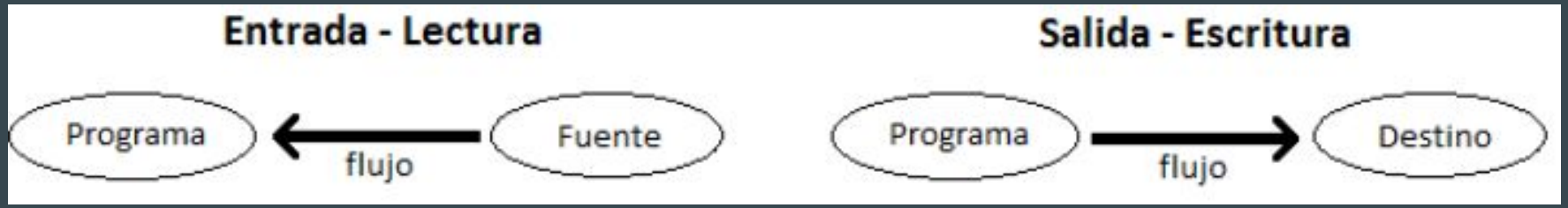
...

Elementos Avanzados

Contenido

- Fundamentos I/O
- Manejo de Archivos (File I/O)
- Concurrencia
- Paralelismo
- El framework Fork-Join
- Parallel Streams

Fundamentos I/O



Flujos binarios – byte (8-bit):

- Tipo de flujo más primitivo y portable.
- Es la base de los flujos ya que todas las operaciones de I/O son flujos de bytes.
- Permite trabajar con datos binarios tales como archivos de imagen, sonido, etc.
- Las clases principales son InputStream y OutputStream

Flujos de caracteres – char (16-bit):

- Tipo de flujo de caracteres en codificación Unicode, ideal para trabajar con texto plano.
- Las clases principales son Reader y Writer.

java.io.IOException

Características:

- Excepción genérica lanzada cuando algún tipo de operación I/O falla
- Debe de ser tratada a través de un bloque de código **try-catch-finally**
- Puede dejarse propagar hacia arriba en la pila de llamadas utilizando **throws IOException**

```
} catch (IOException ex) {
    ex.printStackTrace();
}
```

Manejo de Archivos

Clases necesarias:

- Objeto principal:
  - java.io.File
- Escritura de datos:
  - java.io.PrintWriter
  - java.io.FileWriter
- Lectura de datos:
  - java.io.BufferedReader
  - java.io.FileReader

```
File archivo = new File(nombreArchivo);
try {
    PrintWriter salida = new PrintWriter(new FileWriter(archivo));
    salida.close();
    System.out.println("El archivo se ha creado correctamente");
} catch (IOException ex) {
    ex.printStackTrace();
}
```

Ejemplo

Práctica

Problema:

Crear un programa que permita:

- Consultar la información almacenada en un archivo de texto de sus movimientos bancarios.
- Indicar al usuario la fecha actual.
- Saludar al usuario por su nombre.

Ejemplo:

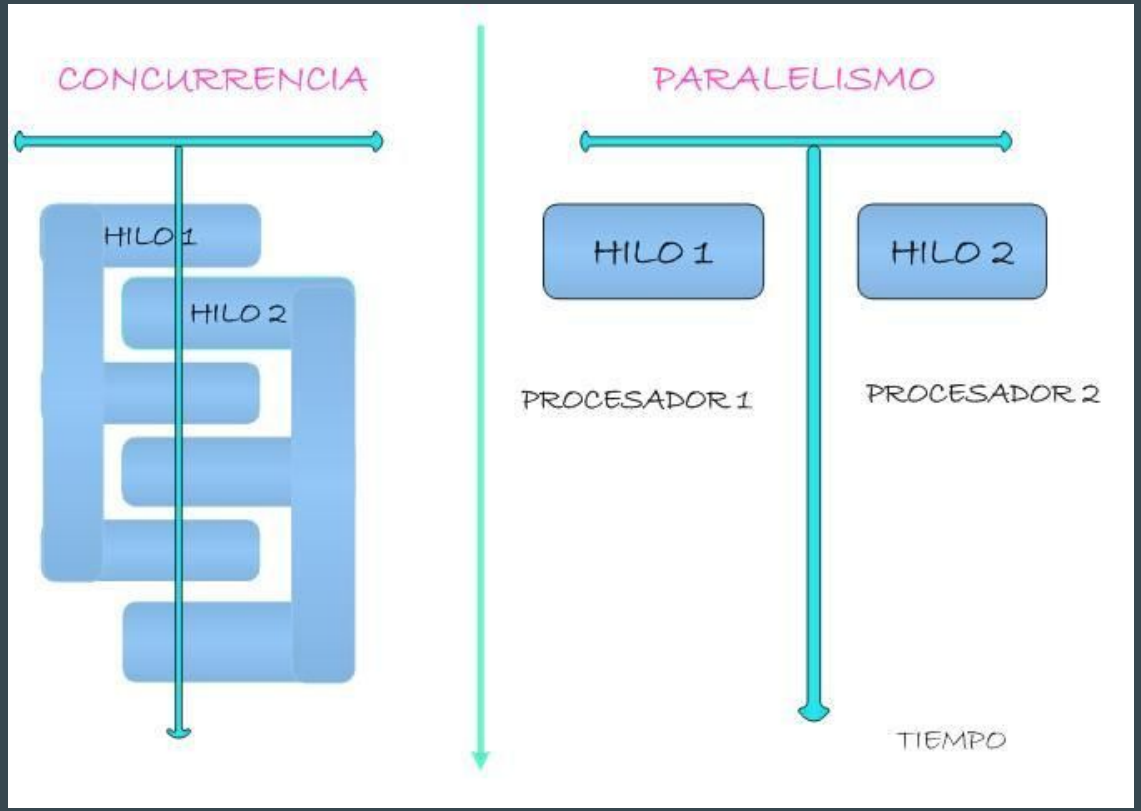
Hola, ***NombreDelUsuario***. Hoy es ***DD/MM/YYYY***

MovimientosBancarios

Indicaciones:

- Crear un proyecto con el nombre **BankAccount.**
- Crear los paquetes:
  - com.main
    - EntryPoint.java
  - com.operation
    - FileControl.java
    - UserAndDateControl.java
  - com.structure
    - I_UserAndDate.java

Concurrencia / Paralelismo



Threads (Concurrencia)

Características:

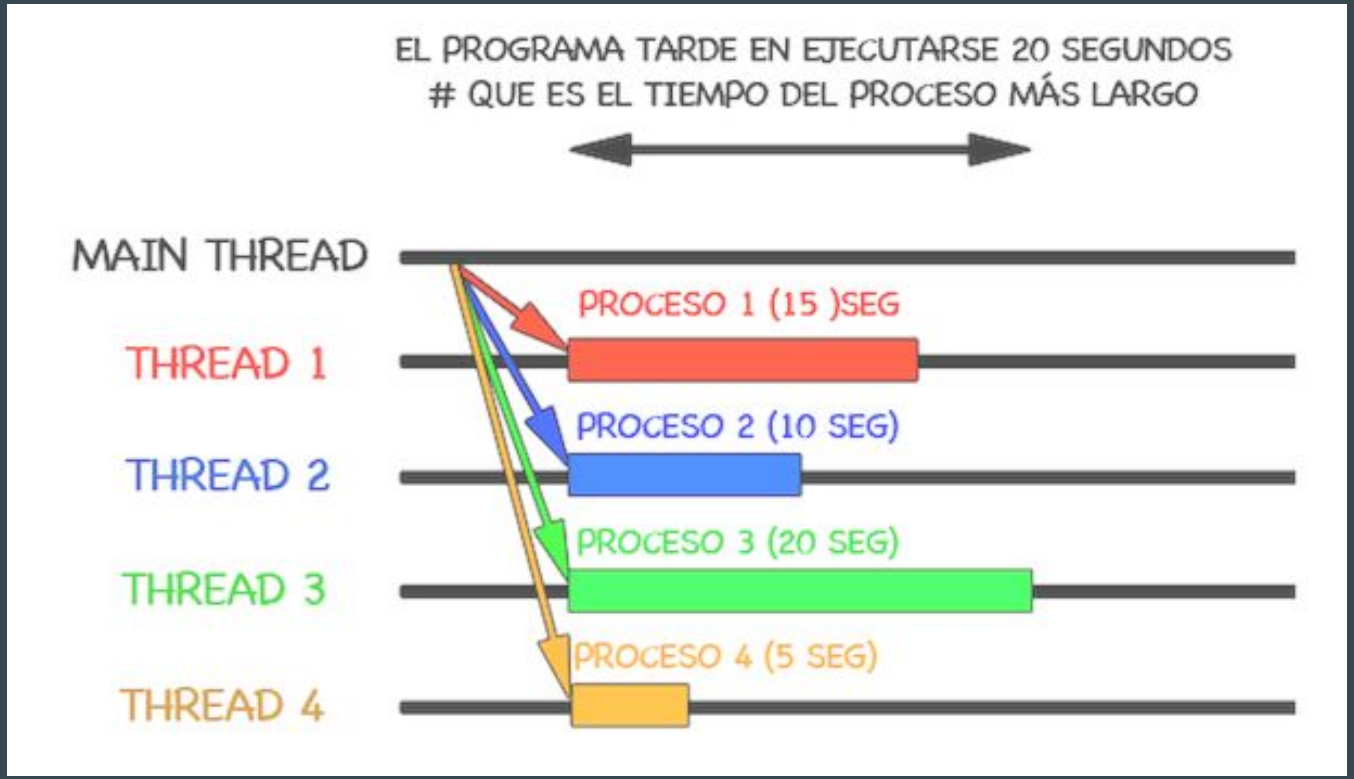
- Programación concurrente de código a través de hilos.
- Un mismo proceso puede tener:
  - Un único hilo -> Monotarea
  - Varios hilos -> Multitarea
- Flujos de ejecución secuencial dentro de un proceso.
- Conocidos también como procesos ligeros: Lightweight Process = LWP
- Se encuentra en el paquete **java.lang.***

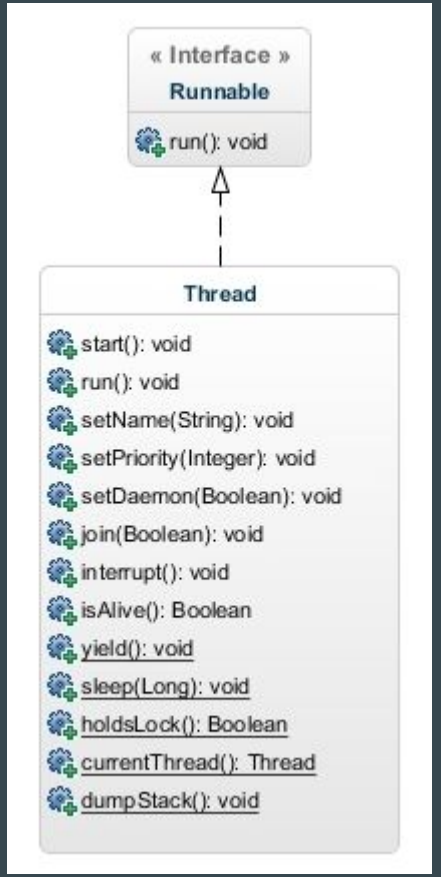
Los hilos de un proceso comparten:

- Espacio de memoria.
- Variables globales.
- Archivos abiertos.
- Procesos hijos.
- Temporizadores.
- Señales y semáforos.
- Contabilidad.

Threads (Concurrencia)

Threads (Concurrencia)



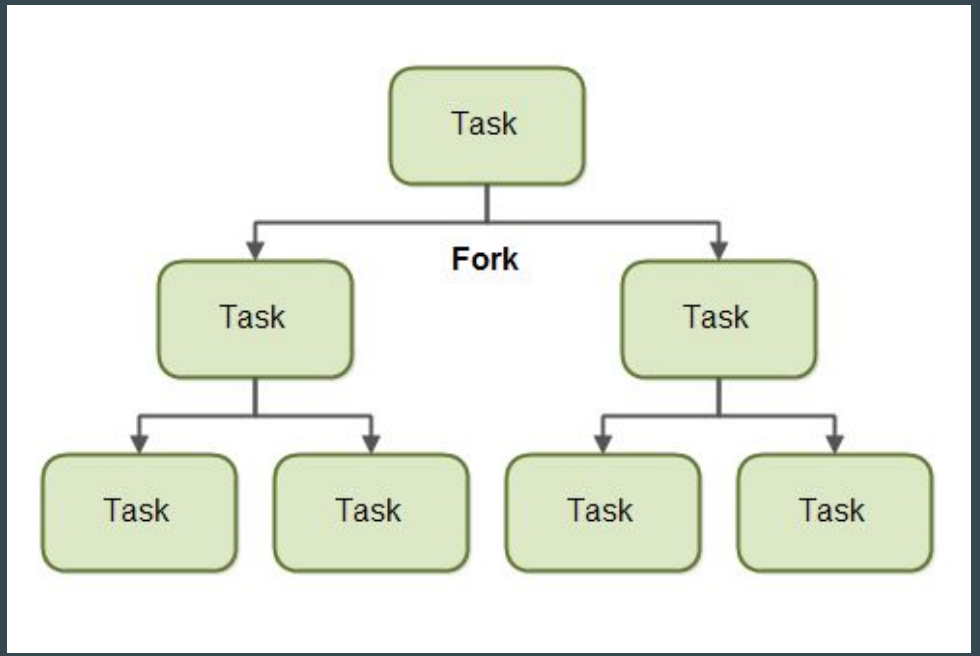


Ejemplo

Fork-Join Framework (Paralelismo)

Características:

- Disponible desde Java 7.
- Ejecuta tareas de forma paralela.
- Aplica el principio de “divide y vencerás”.
- Implementa el algoritmo work-stealing.
- Hay dos operaciones:
  - Dividir una tarea en tareas más pequeñas "fork"
  - Esperar a que las tareas finalicen "join".



Fork-Join Framework (Paralelismo)

- Clases:
  - **RecursiveAction:** No regresa un valor.
  - **RecursiveTask<T>:** Regresa un valor
- Métodos:
  - **compute():** Sobreescribir para la tarea
  - **fork:** tarea
  - **join:** resultado -> error
  - **get:** resultado -> exception
- Se encuentran en el paquete **java.util.concurrent.***

public class MyRecursiveAction extends **RecursiveAction** {}

public class MyRecursiveTask extends **RecursiveTask<T>** {}

Ejemplo

Colecciones Parallel Streams

Características:

- Es un helper para nuestras colecciones.

Métodos:

- parallelStream().sorted()
- parallelStream().filter()
- parallelStream().map()
- parallelStream().limit()
- parallelStream().count()

Buscar en la colección los registros que comiencen con m

```
.parallelStream().filter(x -> x.startsWith("m")).forEach(System.out::println);
```

Ordenar la colección

```
.parallelStream().sorted().forEach(x -> System.out.print(x + " "));
```

Recorrer la colección y realizar alguna operación

```
.parallelStream().map(String::toUpperCase).forEach(x ->
System.out.print(x + " "));
```

Limitar la cantidad de resultados

```
.parallelStream().limit(2));
```

Imprimir la cantidad de registros en la colección

```
System.out.println(.parallelStream().count());
```

Ejemplo