UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
Departamento de Computação - DC
Graduação em Ciência da Computação
Disciplina: Programação Concorrente e Distribuída
Prof.: Paulo Henrique Lopes Silva

Prática 1 – Executor de Thread Simples

1. Exemplos em Java.

- Classe ExecutorThreadSimples.java

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ExecutorThreadSimples {
    public static void main(String[] args) {

        //Criando uma thread dentro da thread main
        ExecutorService executor = Executors.newSingleThreadExecutor();

        //Usando execute() para executar um Runnable
        executor.execute(new ThreadSimples());
    }
}
```

- Classe ThreadSimples.java

```java
public class ThreadSimples implements Runnable{

    @Override
    public void run() {
        System.out.println("Oi, eu sou a thread " +
                Thread.currentThread().getName());
    }
}
```

2. Finalizando o executor.

- Classe ExecutorThreadSimples.java

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ExecutorThreadSimples {
```

```
    public static void main(String[] args) {

        //Criando uma thread dentro da thread main
        ExecutorService executor = Executors.newSingleThreadExecutor();

        //Usando execute() para executar um Runnable
        executor.execute(new ThreadSimples());
        /*
         - Inicia uma finalização ordenada de tarefas submetidas.
         - Novas tarefas não são mais aceitas.
         */
        executor.shutdown();
    }
}
```

- Classe ThreadSimples.java.

```
Java
public class ThreadSimples implements Runnable{

    @Override
    public void run() {
        System.out.println("Oi, eu sou a thread " +
                Thread.currentThread().getName());
        for (int i = 0; i < 20; i++) {
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
            System.out.println("Iterador: " + i);
        }
    }
}
```

- Classe ThreadPrimos.java

```
Java
public class ThreadPrimos implements Runnable {
       private final int inicio;
       private final int fim;

       public ThreadPrimos(int start, int end) {
              this.inicio = start;
              this.fim = end;
       }

       private boolean isPrimo(int num) {
              if (num <= 1) return false;
```

```
            for (int i = 2; i <= Math.sqrt(num); i++) {
                if (num % i == 0) {
                    return false;
                }
            }
            return true;
        }

        @Override
        public void run() {
            System.out.println(
                    "Calculando primos entre " + inicio + " e " + fim);
            for (int i = inicio; i <= fim; i++) {
                if (isPrimo(i)) {
                    System.out.println(i + " é primo.");
                }
            }
        }
}
```

3. Terminação forçada.

- Classe ExecutorThreadSimples.java

```
Java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ExecutorThreadSimples {
    public static void main(String[] args) {
        ExecutorService executor = null;
        try {
            //Criando uma thread dentro da thread main
            executor = Executors.newSingleThreadExecutor();

            //Usando execute() para executar um Runnable
            executor.execute(new ThreadSimples());
            executor.execute(new ThreadSimples());
            /*
                - Inicia uma finalização ordenada de tarefas submetidas.
                - Novas tarefas não são mais aceitas.
            */
            //executor.shutdown();

        } catch (Exception e) {
            throw new RuntimeException(e);
        } finally {
            /*
                - Tenta interromper todas as tarefas em execução ativa.
                - Interrompe o processamento de tarefas em espera
                e retorna uma lista das tarefas que estavam aguardando execução.
            */
```

```
            if (executor != null)
                executor.shutdownNow();
        }
    }
}
```

4. Alternativa: usar o método awaitTermination().

- Classe ExecutorThreadSimples.java

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ExecutorThreadSimples {
    public static void main(String[] args) {
        ExecutorService executor = null;
        try {
            //Criando uma thread dentro da thread main
            executor = Executors.newSingleThreadExecutor();

            //Usando execute() para executar um Runnable
            executor.execute(new ThreadSimples());
            executor.execute(new ThreadSimples());
            boolean b = executor.awaitTermination(5, TimeUnit.SECONDS);
        } catch (Exception e) {
            throw new RuntimeException(e);
        } finally {
            if (executor != null)
                executor.shutdownNow();
                //executor.shutdown();
        }
    }
}
```

5. Método submit().

- Classe SubmitThreadSimples.java
    - Thread lançada com submit(). Método isDone() retorna se a tarefa já terminou.
    - Uso combinado de shutdown(), awaitTermination() e shutdownNow().

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;

public class SubmitThreadSimples {
    public static void main(String[] args) {
```

```
        ExecutorService executor = null;
        try {
            //Criando uma thread dentro da thread main
            executor = Executors.newSingleThreadExecutor();

            /*
                - Usando submit() para executar um Runnable
                - Envia uma tarefa Runnable para execução e
                    retorna um Future representando essa tarefa
             */
            Future<?> future = executor.submit(new ThreadSimples());
            System.out.println(future.isDone());
            executor.shutdown();
            boolean b = executor.awaitTermination(60, TimeUnit.SECONDS);
            System.out.println(future.isDone());
        } catch (Exception e) {
            throw new RuntimeException(e);
        } finally {
            if (executor != null)
                executor.shutdownNow();
        }
    }
}
```

- Classe SubmitThreadSimples.java
    - Executando várias tarefas na mesma thread.

Java

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;

public class SubmitThreadSimples {
    public static void main(String[] args) {
        ExecutorService executor = null;
        try {
            //Criando uma thread dentro da thread main
            executor = Executors.newSingleThreadExecutor();
            executor.execute(new ThreadSimples());
            executor.execute(new ThreadSimples());
            Future<?> future = executor.submit(new ThreadSimples());
            System.out.println(future.isDone());
            executor.shutdown();
            boolean b = executor.awaitTermination(60, TimeUnit.SECONDS);
            System.out.println(future.isDone());
        } catch (Exception e) {
            throw new RuntimeException(e);
        } finally {
            if (executor != null)
                executor.shutdownNow();
        }
    }
}
```

6. Exemplo com executores de thread única operando sobre a sequência de Fibonacci.

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.Arrays;

public class FibonacciStreamExecutor {
	public static void main(String[] args) {
		// Criando um executor de uma única thread
		ExecutorService executor = Executors.newSingleThreadExecutor();

		executor.submit(() -> {
			// Gerando os 20 primeiros termos de Fibonacci
			int[] fibonacci = new int[20];
			fibonacci[0] = 0;
			fibonacci[1] = 1;

			for (int i = 2; i < 20; i++) {
				fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];
			}

			System.out.println("Sequência de Fibonacci:");
			System.out.println(Arrays.toString(fibonacci));

			// Usando Streams para mapear e filtrar os valores
			System.out.println(
				"\nQuadrados dos números pares da sequência:");

			Arrays.stream(fibonacci)
			.map(n -> n * n)
			.filter(n -> n % 2 == 0)
			.forEach(System.out::println);
		});

	// Finaliza o executor após a execução da tarefa
	executor.shutdown();
	}
}
```

7. Exemplo com executores de thread única (virtual) operando sobre a sequência de Fibonacci.

```java
import java.util.Arrays;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class FibonacciVirtualExecutor {
	public static void main(String[] args) {
		// Executor que cria uma thread virtual para cada tarefa
		try (ExecutorService executor =
			Executors.newVirtualThreadPerTaskExecutor()) {
			executor.submit(() -> {
				int[] fibonacci = new int[20];
```

```
                        fibonacci[0] = 0;
                        fibonacci[1] = 1;

                        for (int i = 2; i < 20; i++) {
                                fibonacci[i] =
                                        fibonacci[i - 1] + fibonacci[i - 2];
                        }

                        System.out.println("Sequência de Fibonacci:");
                        System.out.println(Arrays.toString(fibonacci));

                        System.out.println(
                                "\nQuadrados dos números pares da sequência:");
                        Arrays.stream(fibonacci)
                                .map(n -> n * n)
                                .filter(n -> n % 2 == 0)
                                .forEach(System.out::println);
                });
        } // executor é fechado automaticamente aqui
    }
}
```

8. Exemplo com executores de thread única (virtual) operando sobre a sequência de Fibonacci.

Java

```
import java.util.Arrays;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class FibonacciVirtualExecutorComShutdown {
    public static void main(String[] args) {

        ExecutorService executor =
                Executors.newVirtualThreadPerTaskExecutor();

        // Submetendo a tarefa
        executor.submit(() -> {
                int[] fibonacci = new int[20];
                fibonacci[0] = 0;
                fibonacci[1] = 1;

                for (int i = 2; i < 20; i++) {
                        fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];
                }

                System.out.println("Sequência de Fibonacci:");
                System.out.println(Arrays.toString(fibonacci));

                System.out.println(
                        "\nQuadrados dos números pares da sequência:");
                Arrays.stream(fibonacci)
                        .map(n -> n * n)
                        .filter(n -> n % 2 == 0)
```

```
                        .forEach(System.out::println);
                });

                executor.shutdown();

                try {
                        if (!executor.awaitTermination(5, TimeUnit.SECONDS)) {
                        System.out.println(
                                "Ainda existem tarefas em execução após 5 segundos.");
                        }
                } catch (InterruptedException e) {
                        System.out.println("Aguardando término foi interrompido.");
                        Thread.currentThread().interrupt();
                }
        }
}
```

9. Adapte o código a seguir para usar o serviço de executores.

Java

```
import java.util.List;

public class EcommerceSimulacao {

        public static void main(String[] args) {

                List<Integer> pedidos =
                        List.of(1001, 1002, 1003, 1004,
                                1005, 2001, 2002, 2003, 2004, 2005);

                for (Integer pedidoId : pedidos) {
                        Thread.startVirtualThread(() -> processarPedido(pedidoId));
                }

                try {
                        Thread.sleep(2000);
                } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                }
        }

        private static void processarPedido(int pedidoId) {

                System.out.println("Pedido "
                        + pedidoId
                        + " recebido (Thread: " + Thread.currentThread() + ")");

                try {
                        Thread.sleep(300);
                        System.out.println("Pedido " + pedidoId + " validado");

                        Thread.sleep(400);
                        System.out.println("Pagamento do pedido "
                                + pedidoId
```

```
                        + " aprovado");

                Thread.sleep(300);
                System.out.println("Pedido " + pedidoId + " enviado");

            } catch (InterruptedException e) {
                System.out.println("Erro ao processar o pedido " + pedidoId);
                Thread.currentThread().interrupt();
            }
        }
}
```