Documentación y Tutorial del Framework

Introducción:

Este documento describe el framework desarrollado en la Práctica 2, incluyendo sus APIs, puntos de extensión, ciclo de vida, validación, testeo, instalación, configuración, ejemplos de funciones y código MapReduce, e implementaciones en Java y Scala.

APIs y Contratos:

- Invoker:
    - execute(action: Action, input: Object): Ejecuta una acción con la entrada proporcionada y devuelve el resultado.
    - subscribe(observer: Observer): Suscribe un observador para recibir notificaciones de eventos.
    - unsubscribe(observer: Observer): Anula la suscripción de un observador.
- Controller:
    - registerAction(name: String, action: Action, memory: Int): Registra una acción con su nombre, la función a ejecutar y la memoria requerida.
    - registerInvoker(invoker: Invoker): Registra un invoker en el sistema.
    - invoke(name: String, input: Object): Invoca la acción con el nombre especificado y la entrada proporcionada, y devuelve el resultado.
    - getInvokers: Obtiene la lista de invokers registrados.
    - getActions: Obtiene la lista de acciones registradas.
    - showMetrics: Muestra las métricas de ejecución recopiladas.
    - analyzeMetrics: Analiza las métricas de ejecución y muestra información útil.
- Action:
    - execute(input: Object): Define la lógica de la acción a ejecutar.
- PolicyManager:
    - selectInvoker(controller: Controller, action: Action): Invoker: Selecciona un invoker adecuado para ejecutar la acción.
- Observer:
    - updateMetrics(metrics: Metrics): Recibe notificaciones de las métricas de ejecución.

Puntos de Extensión y Ciclo de Vida:

- Puntos de Extensión:
    - Se pueden crear nuevas políticas de selección de invokers extendiendo la clase PolicyManager.
    - Se pueden implementar nuevas acciones extendiendo la clase Action.
    - Se pueden agregar nuevos observadores extendiendo la clase Observer.
- Ciclo de Vida:
    - Las acciones se ejecutan en un ciclo de vida que incluye las etapas de inicialización, ejecución y finalización.
    - Los invokers pueden notificar a los observadores en diferentes puntos del ciclo de vida de la acción.

Validación y Testeo de Servicios:

```
INVOKES:
InvokerSelect: 1
Metric received: Metrics(invokerId=1, executionTime=3, memoryUsage=5)
Invoker 1 notified
InvokerSelect: 2
Metric received: Metrics(invokerId=2, executionTime=0, memoryUsage=5)
Invoker 2 notified
Result of action 'add': 20
Result of action 'add': 160
```

```
OBSERVER:
ArrayBuffer(Metrics(invokerId=1, executionTime=3, memoryUsage=5), Metrics(invokerId=2, executionTime=0, memoryUsage=5))
```

```
Metrics:
    Average time: 1.0 ms
    Maximum time: 3 ms
    Minimum time: 0 ms
    Total execution time: 3 ms
```

Ejemplos de Funciones y Código MapReduce:

- countWords(text: String): Int: Cuenta el número de palabras en un texto.
- wordCount(text: String): Map[String, Int]: Genera un mapa con la frecuencia de cada palabra en un texto.

```
val countWords: Function[String, Int] = (input: Object) => {
  val stringText: String = input.asInstanceOf[String]
  if (stringText == null || stringText.isEmpty) {
    0
  } else {
    stringText.split("\\s+").length
  }
}

var wordCount: Function[AnyRef, AnyRef] = (text: AnyRef) => {
  val inputText: String = text.asInstanceOf[String]
  val wordCounts: util.Map[String, Integer] = new util.HashMap[String, Integer]
  val words: Array[String] = inputText.split("\\s+")
  for (word <- words) {
    if (word.nonEmpty) wordCounts.put(word, wordCounts.getOrDefault(word, 0) + 1)
  }
  wordCounts
}
```

```
MAP REDUCE:
Action countWords Registered
Action wordCounts Registered
InvokerSelect: 1
Metric received: Metrics(invokerId=1, executionTime=10, memoryUsage=55)
Invoker 1 notified
ResultCountWords: 9231
InvokerSelect: 2
Metric received: Metrics(invokerId=2, executionTime=24, memoryUsage=55)
Invoker 2 notified
ResultWordCounts: {please!"=1, together,=1, come—they=1, half=2, #72454]=1, don't=21, spoke=4,
```

Ejemplos del Marco en Java y Scala:

```java
public <T, R> R invokeAction(Action<T, R> action, Object input) throws Exception {
    long start = System.currentTimeMillis();
    int memory = action.getMemory();
    R result;
    if ((usedMemory + memory) <= totalMemory) {
        usedMemory += memory;
        try{
            result = action.run((T) input);
        } catch (Exception e){
            throw new Exception("Error when executing this "+ action +" : "+ e.getMessage(), e);
        } finally {
            long end = System.currentTimeMillis();
            Metrics metrics = new Metrics(id, time: end-start, usedMemory);
            this.notify(metrics);
        }
    } else {
        throw new RuntimeException("Not enough resources for this action.");
    }
    return result;
}
```

```scala
def invoke[T, R](id: String, input: Object): R = {
  val executeAction = actionsList(id).asInstanceOf[Action[T, R]]
  val invoker = policy.selectInvoker(this, executeAction)
  val result = invoker.invokeAction(executeAction, input)
  //println(s"Action $id invoked in ${invoker.getId}")
  result
}
```

```java
public void analyzeMetrics() {
    System.out.println("\nMetrics:");
    // Calculate and display the average execution time of all actions
    double averageTime = metricsList.stream().mapToDouble(Metrics::getExecutionTime).average().orElse( other: 0.0);
    System.out.println("\tAverage time: " + averageTime + " ms");
    // Calculates and displays the maximum and minimum execution time of all actions
    long maximumTime = metricsList.stream().mapToLong(Metrics::getExecutionTime).max().orElse( other: 0L);
    System.out.println("\tMaximum time: " + maximumTime + " ms");
    long minimumTime = metricsList.stream().mapToLong(Metrics::getExecutionTime).min().orElse( other: 0L);
    System.out.println("\tMinimum time: " + minimumTime + " ms");
    // Calculates and displays the total execution time of all actions
    long totalTime = metricsList.stream().mapToLong(Metrics::getExecutionTime).sum();
    System.out.println("\tTotal time: " + totalTime + " ms");
    // Calculates and displays the total memory utilization of each Invoker
    Map<Integer, Integer> memoryInvoker = metricsList.stream().collect(Collectors.groupingBy(Metrics::getId,Collectors.summingInt(Metrics::getUsedMemory)));
    memoryInvoker.forEach((invokerId, avgMemoryUsage) -> System.out.println("\tInvoker: " + invokerId + " -> Total memory: " + avgMemoryUsage + " MB"));
}
```

```scala
def analyzeMetrics(): Unit = {
  println("\nMetrics:")
  val averageTime: Double = metricsList.map(_.getExecutionTime).sum / metricsList.length
  println("\tAverage time: " + averageTime + " ms")
  val maximumTime: Long = metricsList.maxBy(_.getExecutionTime).executionTime
  println("\tMaximum time: " + maximumTime + " ms")
  val minimumTime: Long = metricsList.minBy(_.getExecutionTime).executionTime
  println("\tMinimum time: " + minimumTime + " ms")
  val totalTime: Long = metricsList.map(_.getExecutionTime).sum
  println("\tTotal execution time: " + totalTime + " ms")
}
```