MAKERERE UNIVERSITY

COLLEGE OF COMPUTING AND INFORMATION SCIENCES

SCHOOL OF COMPUTING AND INFORMATICS TECHNOLOGY

Course: BSE 3202 – Distributed Systems Development

Java RMI Task Bag Implementation
Documentation Report

BSE1 GROUP MEMBER

TEAM MEMBERS	REGISTRATION NUMBER
Turyahebwa Alex	18/U/23405/Eve
Kitonsa Elvis	20/U/7785/Ps
Odongo Justine	20/U/2050/Eve
Etyang Simon Peter	19/U/9043/Eve
AMADILE MAJID	20/U/23418

Table of Contents

1. Introduction 1
2. System Overview 1
2.1 Objective 1
2.2 Application Use Case 1
3. Task Bag Implementation 2
3.1 Task Bag Operations & Mapping 2
3.2 TaskBag Implementation (TaskBagImpl.java) 2
4. Master Process (Master.java) 3
5. Worker Process (Worker.java) 3
6. Synchronization Approach in Java RMI 3
6.1 Challenges in Synchronization 3
6.2 Implemented Solution 4
7. User Guide 4
7.1 Project Setup 4
7.2 Running Instructions 4
8. Expected Code Printouts 5
8.1 Master Execution 5
8.2 Worker Execution 6
8.3 TaskBag Implementer Execution 7
9. Compliance Checklist: Java RMI Task Bag Implementation 7
9.1 Task Bag (Remote Object) 7
9.2 The Application (Finding Prime Numbers) 8
9.3 Parallel Processing & Synchronization 8
9.4 Monitoring & User Interaction 9
10. Conclusion 9

LIST OF TABLES

Table 1 Task Bag Operations & Mapping 2
Table 2 Task Bag (Remote Object) 7
Table 3 The Application (Finding Prime Numbers) 8
Table 4 Parallel Processing & Synchronization 8
Table 5 Monitoring & User Interaction 9

1. Introduction

This document provides a detailed overview of the Task Bag System, which uses Java Remote Method Invocation (RMI) to distribute tasks among multiple worker processes. The system allows parallel computation of prime numbers using a Master-Worker architecture.

2. System Overview

2.1 Objective

This project aimed to implement a **Task Bag** using **Java Remote Method Invocation (RMI)** for **parallel computation** across multiple workstations. The Task Bag serves as a shared repository where:

- The **Master process** deposits computational tasks.
- The **Worker processes** retrieve and process these tasks in parallel.
- The **Master collects results** once computations are complete.

2.2 Application Use Case

We implemented a **prime number computation task**, where:

1. The **Master** distributes ranges of numbers for prime number checking.
2. **Workers** process each range, find prime numbers, and return results to the Task Bag.
3. The **Master collects and combines the results.**

The system consists of four main components:

1. **TaskBag.java** (Interface) - Defines the remote methods used for task distribution.
2. **TaskBagImpl.java** (Implementation) - Implements the Task Bag logic and stores tasks.
3. **Master.java** - Distributes tasks and collects results.
4. **Worker.java** - Fetches tasks, processes them, and sends results back.

3. Task Bag Implementation

The TaskBag is implemented using Java RMI to facilitate communication between distributed processes. It enables workers to fetch tasks dynamically and ensures tasks are assigned uniquely.

3.1 Task Bag Operations & Mapping

Table 1 Task Bag Operations & Mapping

Operation	Description	Used By
`pairOut(key, value)`	Stores a task or result in the Task Bag.	Master (for tasks), Worker (for results)
`pairIn(key)`	Retrieves and removes a task from the Task Bag.	Worker
`readPair(key)`	Reads a task without removing it.	TaskBag Implementer
`getTaskCount(key)`	Checks the number of available tasks.	Master
`getNextTask()`	Retrieves the next available task index.	Worker
`getMaxValue()`	Retrieves MAX value set by the Master	TaskBag Implementer
`getGranularity()`	Retrieves the GRANULARITY value set by the Master	TaskBag Implementer
`updateNextTask()`	Increments the next task index.	Worker
`setTaskParameters(int max, int granularity)`	Sets MAX and GRANULARITY values dynamically allowing Master to set task distribution parameters.	Master

3.2 TaskBag Implementation (TaskBagImpl.java)

The TaskBagImpl class implements the TaskBag interface using a **ConcurrentHashMap** to store tasks in a thread-safe manner. Workers dynamically fetch and process tasks.

Key Features:

1. Uses **LinkedBlockingQueue** to store tasks efficiently.
2. Ensures only available tasks are assigned to workers.

3. Dynamically updates task distribution parameters received from the Master.

4. Master Process (Master.java)

The Master process handles:

1. **Task Distribution**
   - Accepts user input for `MAX` and `GRANULARITY`.
   - Assigns tasks in batches and stores them in TaskBag.
2. **Result Collection**
   - Periodically checks the TaskBag for completed results.
   - Retrieves computed prime numbers and displays them.

Execution Flow

1. Prompt user for `MAX` and `GRANULARITY` values.
2. Connect to the TaskBag server.
3. Distribute tasks by adding them to TaskBag.
4. Collect computed prime numbers from workers.

5. Worker Process (Worker.java)

Workers are responsible for fetching and processing tasks.

Execution Flow

1. Connects to the TaskBag server.
2. Retrieves a single available task.
3. Processes numbers and determines primes.
4. Sends results back to the TaskBag.
5. Exits after completing the assigned task.

6. Synchronization Approach in Java RMI

6.1 Challenges in Synchronization

1. Multiple workers accessing tasks concurrently.
2. Avoiding duplication of tasks.
3. Ensuring workers terminate correctly when no tasks remain.

6.2 Implemented Solution

We implemented **synchronized methods** to manage concurrent worker access:

1. `getNextTask()` and `updateNextTask()` ensure **workers get unique tasks**.
2. `pairIn(key)` uses **polling with a timeout**, allowing workers to exit gracefully when no tasks remain.

7. User Guide

7.1 Project Setup

Ensure your project structure is organized as follows:

```
TaskBagProject/
├── src/
│   ├── TaskBag.java
│   ├── TaskBagImpl.java
│   ├── Master.java
│   ├── Worker.java
├── bin/  (Compiled class files will go here)
```

7.2 Running Instructions

1. **Compile all Java files:**

```
javac -d bin -cp src src/*.java
```

2. **Start RMI Registry:**

```
rmiregistry 1099
```

3. **Run TaskBag Server:**

```
java -cp bin TaskBagImpl
```

4. **Run Master Process:**

```
java -cp bin Master
```

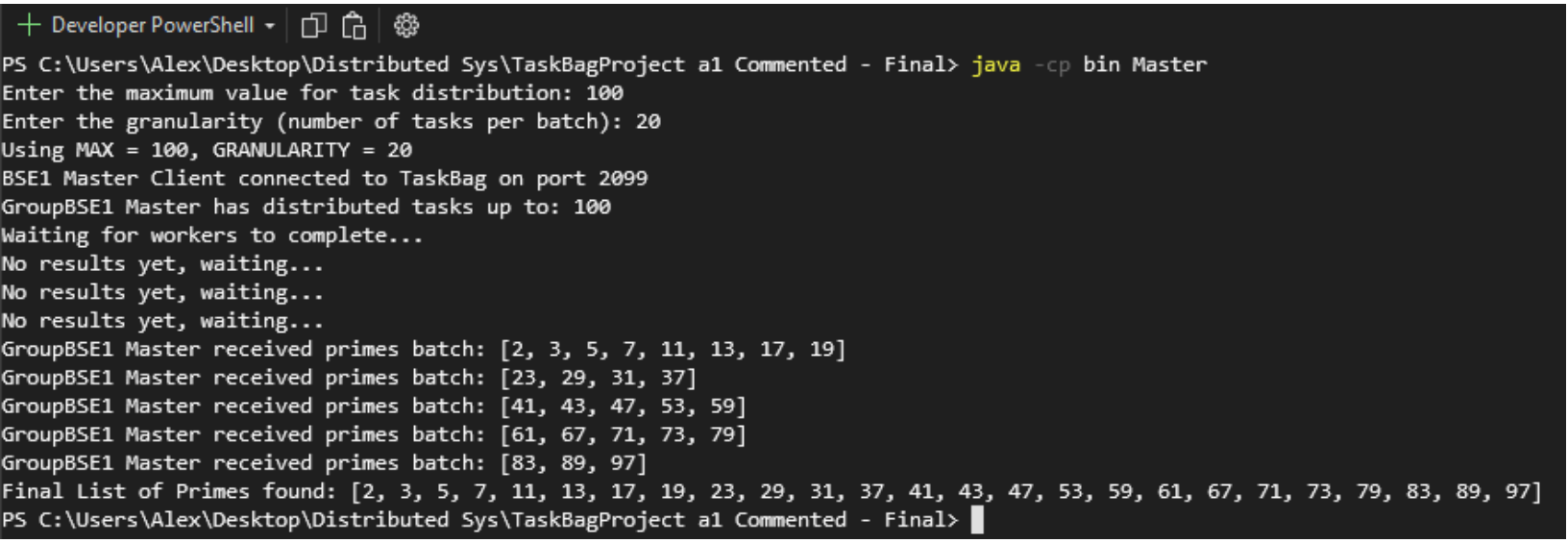
5. **Run Workers (Multiple Terminals):**

```
java -cp bin Worker
```

8. Expected Code Printouts

Example Run

8.1 Master Execution



8.2 Worker Execution

```
PS C:\Users\Alex\Desktop\Distributed Sys\TaskBagProject a1 Commented - Final> java -cp bin Worker
Worker Client connected to TaskBag on port 2099
Available tasks in TaskBag: 5
Worker processing task from [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Worker has sent primes to TaskBag: [2, 3, 5, 7, 11, 13, 17, 19]
Worker has completed processing and is exiting...
PS C:\Users\Alex\Desktop\Distributed Sys\TaskBagProject a1 Commented - Final> java -cp bin Worker
Worker Client connected to TaskBag on port 2099
Available tasks in TaskBag: 4
Worker processing task from [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]
Worker has sent primes to TaskBag: [23, 29, 31, 37]
Worker has completed processing and is exiting...
PS C:\Users\Alex\Desktop\Distributed Sys\TaskBagProject a1 Commented - Final> java -cp bin Worker
Worker Client connected to TaskBag on port 2099
Available tasks in TaskBag: 3
Worker processing task from [40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
Worker has sent primes to TaskBag: [41, 43, 47, 53, 59]
Worker has completed processing and is exiting...
PS C:\Users\Alex\Desktop\Distributed Sys\TaskBagProject a1 Commented - Final> java -cp bin Worker
Worker Client connected to TaskBag on port 2099
Available tasks in TaskBag: 2
Worker processing task from [60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79]
Worker has sent primes to TaskBag: [61, 67, 71, 73, 79]
Worker has completed processing and is exiting...
PS C:\Users\Alex\Desktop\Distributed Sys\TaskBagProject a1 Commented - Final> java -cp bin Worker
Worker Client connected to TaskBag on port 2099
Available tasks in TaskBag: 1
Worker processing task from [80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
Worker has sent primes to TaskBag: [83, 89, 97]
Worker has completed processing and is exiting...
PS C:\Users\Alex\Desktop\Distributed Sys\TaskBagProject a1 Commented - Final> java -cp bin Worker
Worker Client connected to TaskBag on port 2099
Available tasks in TaskBag: 0
No more tasks left for processing. Worker is exiting...
PS C:\Users\Alex\Desktop\Distributed Sys\TaskBagProject a1 Commented - Final>
```

8.3 TaskBag Implementer Execution

```
PS C:\Users\Alex\Desktop\Distributed Sys\TaskBagProject a1 Commented - Final> java -cp bin TaskBagImpl
GroupBSE1 TaskBagImplementer is running...
Updated TaskBag parameters: MAX = 100, GRANULARITY = 20
GroupBSE1 TaskBag: Added [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19] under key 'Task'
GroupBSE1 TaskBag: Added [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39] under key 'Task'
GroupBSE1 TaskBag: Added [40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59] under key 'Task'
GroupBSE1 TaskBag: Added [60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79] under key 'Task'
GroupBSE1 TaskBag: Added [80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99] under key 'Task'
TaskBag: Retrieved [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19] under key 'Task'
GroupBSE1 TaskBag: Added [2, 3, 5, 7, 11, 13, 17, 19] under key 'Primes'
TaskBag: Retrieved [2, 3, 5, 7, 11, 13, 17, 19] under key 'Primes'
TaskBag: Retrieved [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39] under key 'Task'
GroupBSE1 TaskBag: Added [23, 29, 31, 37] under key 'Primes'
TaskBag: Retrieved [23, 29, 31, 37] under key 'Primes'
TaskBag: Retrieved [40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59] under key 'Task'
GroupBSE1 TaskBag: Added [41, 43, 47, 53, 59] under key 'Primes'
TaskBag: Retrieved [41, 43, 47, 53, 59] under key 'Primes'
TaskBag: Retrieved [60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79] under key 'Task'
GroupBSE1 TaskBag: Added [61, 67, 71, 73, 79] under key 'Primes'
TaskBag: Retrieved [61, 67, 71, 73, 79] under key 'Primes'
TaskBag: Retrieved [80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99] under key 'Task'
GroupBSE1 TaskBag: Added [83, 89, 97] under key 'Primes'
TaskBag: Retrieved [83, 89, 97] under key 'Primes'
```

9. Compliance Checklist: Java RMI Task Bag Implementation

9.1 Task Bag (Remote Object)

Table 2 Task Bag (Remote Object)

Requirement	Status	Implementation Details
Implements a Remote Object using Java RMI	Yes	`TaskBagImpl` extends `UnicastRemoteObject` and is bound to Registry at port **2099**
Stores tasks as key-value pairs	Yes	Implemented in `TaskBagImpl` using `ConcurrentHashMap<String, LinkedBlockingQueue<List<Integer>>>`
Supports `pairOut()` to add tasks	Yes	Adds tasks using `computeIfAbsent()` and `.offer(value)`
Supports `pairIn()` to retrieve and remove tasks	Yes	Uses `.poll(1000, TimeUnit.MILLISECONDS)` for **blocking behavior** (ensures workers wait if no tasks are available)

Supports `readPair()` to check without removing	Yes	Uses `.peek()` to read a task without removing it
Handles multiple values with the same key (not a Set)	Yes	Uses `LinkedBlockingQueue<>` which allows multiple values under the same key
Implements `getTaskCount()` for monitoring	Yes	`TaskBagImpl` implements `getTaskCount()` to return the number of tasks available
Implements `getNextTask()` and `updateNextTask()`	Yes	Ensures **workers get unique task batches**
Supports `setTaskParameters(max, granularity)`	Yes	Allows **dynamic configuration** of `MAX` and `GRANULARITY`

9.2 The Application (Finding Prime Numbers)

Table 3 The Application (Finding Prime Numbers)

Requirement	Status	Implementation Details
Computes prime numbers	Yes	`Worker.java` checks for **prime numbers** using `isPrime()`
Divides the problem into identical subtasks	Yes	The **Master distributes tasks** in batches using `GRANULARITY`
Workers process a batch of numbers and return results	Yes	Workers retrieve tasks, compute **primes**, and store them in `TaskBag` under `"Primes"`
Master collects and combines results	Yes	`Master.java` retrieves prime number batches from `TaskBag`
Supports **different numbers of workers**	Yes	The program works with **1, 10, or 1000 workers**
Ensures **results do not overwrite each other**	Yes	Uses **multiple values under `"Primes"`** key to prevent overwriting

9.3 Parallel Processing & Synchronization

Table 4 Parallel Processing & Synchronization

Requirement	Status	Implementation Details
Uses a **Master-Worker** model	Yes	`Master.java` distributes tasks, `Worker.java` processes them
Workers fetch tasks dynamically	Yes	Workers repeatedly call `pairIn("Task")`
Ensures **synchronization**	Yes	`pairIn()` uses `.poll()` (timeout-based polling), preventing **race conditions**
Handles **multiple workers in parallel**	Yes	Uses `ConcurrentHashMap<>` to allow **concurrent task access**
Supports **task completion detection**	Yes	Workers exit when `pairIn("Task")` **returns null**

Prevents **CPU wastage via polling**	Yes	Uses `.poll(1000, TimeUnit.MILLISECONDS)` instead of **infinite loops**

9.4 Monitoring & User Interaction

Table 5 Monitoring & User Interaction

Requirement	Status	Implementation Details
Master allows **dynamic input** for `MAX` and `GRANULARITY`	Yes	Uses `Scanner` to get **user input**
Master provides **feedback on received results**	Yes	**Prints** received **prime numbers** from workers
Workers log received tasks and results	Yes	**Prints** received task **batches** and **computed primes**
Allows checking the **number of tasks left (`getTaskCount()`)**	Yes	Implemented but **not actively used** by Master
Supports **graceful termination of workers**	Yes	Workers **stop automatically** when `pairIn("Task")` returns `null`

10. Conclusion

This system efficiently distributes computational tasks among multiple worker processes using Java RMI. It dynamically assigns tasks, collects results, and ensures efficient execution through distributed processing.