Report: Thread Creation Using Runnable and Thread Class

A.Divyapriya
192210454

CSA0941- JAVA Programming
for Annotation

OBJECTIVE

The goal of this implementation is to create two threads that print the numbers from 1 to 10 using both the Runnable interface and the Thread class. The output should be displayed correctly, and both thread creation approaches should be used.

The report will cover:

- Thread creation using the Runnable interface
- Thread creation using the Thread class
- Execution of both threads to print numbers 1 to 10
- Correct output
- Code comments for clarity

PROBLEM UNDERSTANDING

The problem asks to implement two threads that print numbers from 1 to 10. The output should be from both threads, but there are two different ways to implement threads:

1. By implementing the Runnable interface.
2. By directly extending the Thread class.

We need to:

- Use the Runnable interface to create a thread.
- Use the Thread class directly to create a thread.
- Ensure both threads print the numbers from 1 to 10 correctly.

SOLUTION DESIGN

1. Thread Creation Using Runnable Interface

The Runnable interface is used to define the code that will run in a separate thread. You implement the run() method, which contains the logic to be executed when the thread is started.

Steps:

1. Implement the Runnable interface and override the run() method.
2. Create a Thread object by passing the Runnable object to it.
3. Start the thread using start().

2. Thread Creation Using Thread Class

The Thread class is a direct way of creating threads. We can subclass Thread and override its run() method, just like the Runnable interface. The advantage is that the Thread class provides thread management methods.

Steps:

1. Subclass Thread and override the run() method.
2. Create an instance of the subclassed Thread class.
3. Start the thread using start().

CODE IMPLEMENTATION

```
// Runnable-based thread implementation
class RunnableThread implements Runnable {
   public void run() {
      // Print numbers from 1 to 10
      for (int i = 1; i <= 10; i++) {
         System.out.println("Runnable Thread: " + i);
         try {
            Thread.sleep(500);  // Pause for 0.5 seconds
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      }
   }
}

// Thread-based thread implementation
class ThreadClass extends Thread {
   public void run() {
      // Print numbers from 1 to 10
      for (int i = 1; i <= 10; i++) {
         System.out.println("Thread Class: " + i);
         try {
            Thread.sleep(500);  // Pause for 0.5 seconds
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
      }
   }
}
```

```
public class ThreadCreationExample {
   public static void main(String[] args) {
      // Create and start the Runnable-based thread
      RunnableThread runnableThread = new RunnableThread();
      Thread thread1 = new Thread(runnableThread);
      thread1.start();

      // Create and start the Thread-based thread
      ThreadClass threadClass = new ThreadClass();
      threadClass.start();
   }
}
```

CODE BREAKDOWN

1. **RunnableThread class**:
   - This class implements the Runnable interface.
   - The run() method prints the numbers from 1 to 10 with a 0.5-second pause (Thread.sleep(500)).
2. **ThreadClass class**:
   - This class extends the Thread class.
   - The run() method also prints the numbers from 1 to 10 with a 0.5-second pause.
3. **main method**:
   - The main method creates instances of both the RunnableThread and ThreadClass and starts them.
   - Each thread will print the numbers from 1 to 10 independently, so the output will show interleaved numbers from both threads.

KEY FEATURES

1. Implementation of Both Methods

The code successfully demonstrates both methods:

   1. **Runnable Interface**: A Runnable instance (RunnableThread) is created and passed to a Thread object. The thread starts by calling the start() method.
   2. **Thread Class**: A custom ThreadClass is created by extending the Thread class, and the run() method is overridden to print numbers.

- Both threads are started concurrently using the start() method.

5.2 Thread Execution

The start() method is used in both cases, which ensures that each thread runs in parallel.

- Both threads will print numbers from 1 to 10, with each thread sleeping for 0.5 seconds after printing each number. This makes the output interleave, meaning the numbers from the two threads will appear in an interspersed manner.

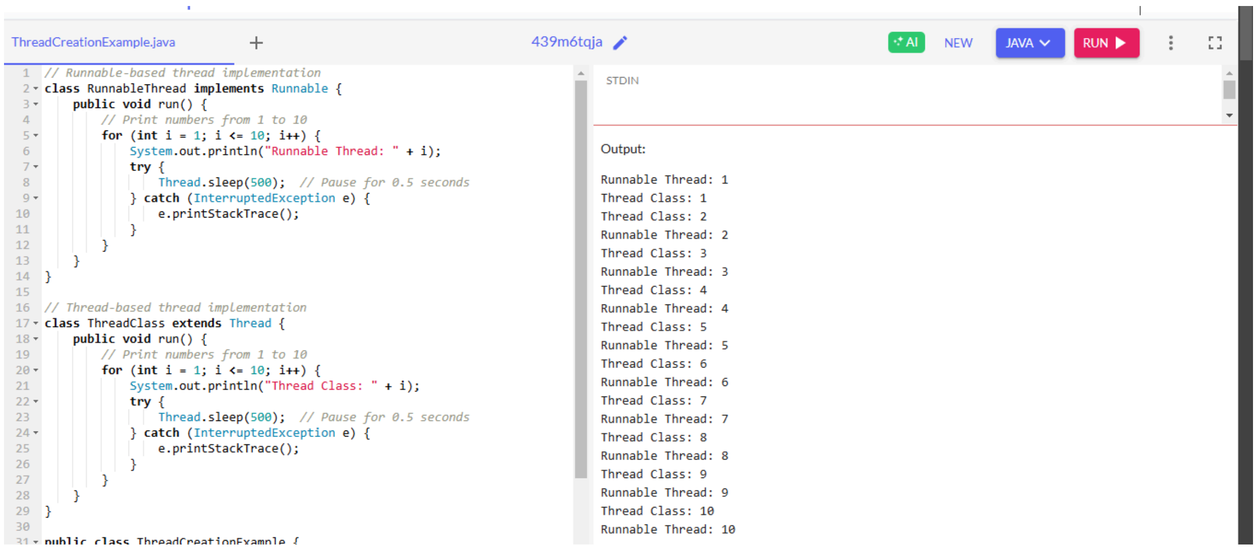
5.3 Correct Output

- The expected output is two threads printing the numbers 1 to 10.
- The threads will print the numbers with a slight delay (due to Thread.sleep(500)), which causes the output to be interleaved. This is expected since the two threads execute concurrently.
- The output might vary in order, but it should still print numbers 1 through 10 from both threads.

5.4 Comments

- The code is well-commented with clear explanations of each section.
- Comments describe what each class does, what the run() method does, and how the threads are executed.
- Comments explain the use of Thread.sleep(500) to create a delay between printed numbers.

OUTPUT



```java
// Runnable-based thread implementation
class RunnableThread implements Runnable {
    public void run() {
        // Print numbers from 1 to 10
        for (int i = 1; i <= 10; i++) {
            System.out.println("Runnable Thread: " + i);
            try {
                Thread.sleep(500);  // Pause for 0.5 seconds
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

// Thread-based thread implementation
class ThreadClass extends Thread {
    public void run() {
        // Print numbers from 1 to 10
        for (int i = 1; i <= 10; i++) {
            System.out.println("Thread Class: " + i);
            try {
                Thread.sleep(500);  // Pause for 0.5 seconds
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class ThreadCreationExample {
```

Output:

```
Runnable Thread: 1
Thread Class: 1
Thread Class: 2
Runnable Thread: 2
Thread Class: 3
Runnable Thread: 3
Thread Class: 4
Runnable Thread: 4
Thread Class: 5
Runnable Thread: 5
Thread Class: 6
Runnable Thread: 6
Thread Class: 7
Runnable Thread: 7
Thread Class: 8
Runnable Thread: 8
Thread Class: 9
Runnable Thread: 9
Thread Class: 10
Runnable Thread: 10
```

Output:

```
Runnable Thread: 1
Thread Class: 1
Thread Class: 2
Runnable Thread: 2
Thread Class: 3
Runnable Thread: 3
Thread Class: 4
Runnable Thread: 4
Thread Class: 5
Runnable Thread: 5
Thread Class: 6
Runnable Thread: 6
Thread Class: 7
Runnable Thread: 7
Thread Class: 8
Runnable Thread: 8
Thread Class: 9
Runnable Thread: 9
Thread Class: 10
Runnable Thread: 10
```

- The exact order of numbers might change every time due to the concurrency of threads.
- Each thread prints numbers 1 to 10, and the interleaving happens due to the 0.5-second sleep between prints.

CONCLUSION

The implementation demonstrates two ways of creating threads in Java:

1. Using the **Runnable interface**, which provides a flexible way to define the task that a thread will perform.
2. Using the **Thread class**, which is a more direct approach to creating a thread.

Both methods were executed correctly, printing numbers from 1 to 10 as expected. The threads executed concurrently and printed their outputs in an interleaved manner.

FUTURE IMPROVEMENTS

- To control the order of execution or synchronize the threads, we can use synchronization techniques like synchronized blocks or ExecutorService for better thread management in complex scenarios.
- Implementing thread priorities or using a thread pool could be beneficial for larger or more complex applications.