Task1

Result

```
(base) fzdx@fzdx-PR4910P:~/cjt/CS240/Lab09$ javac ReadersWritersSimulation.java Reader.java Writer.java
(base) fzdx@fzdx-PR4910P:~/cjt/CS240/Lab09$ java ReadersWritersSimulation
Starting Readers-Writers simulation with 5 readers and 2 writers.
Policy: Readers-Priority
------------------------------------------------
Reader 2 acquired read lock.
Reader 2 is reading...
Reader 4 acquired read lock.
Reader 4 is reading...
Reader 1 acquired read lock.
Reader 1 is reading...
Reader 3 acquired read lock.
Reader 3 is reading...
Reader 5 acquired read lock.
Reader 5 is reading...
Reader 4 done, releasing read lock.
Reader 2 done, releasing read lock.
Reader 5 done, releasing read lock.
Reader 3 done, releasing read lock.
Reader 1 done, releasing read lock.
Writer 2 acquired write lock.
Writer 2 is writing...
Writer 2 done, releasing write lock.
Writer 1 acquired write lock.
Writer 1 is writing...
```

Implement the Classs

```
public class ReadersWritersSimulation {

    public static void main(String[] args) {
        // Create a single DataAccessPolicyManager object shared by all threads
        DataAccessPolicyManager lockManager = new DataAccessPolicyManager();

        // Define the number of readers and writers
        int numReaders = 5;
        int numWriters = 2;

        System.out.println("Starting Readers-Writers simulation with " +
numReaders + " readers and " + numWriters + " writers.");
        System.out.println("Policy: Readers-Priority");
        System.out.println("------------------------------------------------")
;


        // Create and start writer threads
        for (int i = 1; i <= numWriters; i++) {
            Writer writer = new Writer(lockManager, i);
```

```
            writer.start();
        }

        // Create and start reader threads
        for (int i = 1; i <= numReaders; i++) {
            Reader reader = new Reader(lockManager, i);
            reader.start();}
```

Implement Reader

```
class Reader extends Thread {
    private DataAccessPolicyManager lockManager;
    private Random random = new Random();
    private int readerId;

    // Constructor: receives a DataAccessPolicyManager instance
    public Reader(DataAccessPolicyManager lockManager, int readerId) {
        this.lockManager = lockManager;
        this.readerId = readerId;
    }
```

```
    @Override
    public void run() {
        while (true) {
            try {
                // Sleep for a random time before starting the next read
                Thread.sleep(random.nextInt(2000) + 1000);
```

```
                lockManager.acquireReadLock();
                System.out.println("Reader " + readerId + " acquired read lock.");
```

```
                // Simulate reading activity
                System.out.println("Reader " + readerId + " is reading...");
                Thread.sleep(random.nextInt(1500) + 500);
```

```
                System.out.println("Reader " + readerId + " done, releasing read
lock.");
```

```
                lockManager.releaseReadLock();


            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```

```
    }
}
```

Implement writter

```
import java.util.Random;
class Writer extends Thread {
    private DataAccessPolicyManager lockManager;
    private Random random = new Random();
    private int writerId;

    // Constructor: receives a DataAccessPolicyManager instance
    public Writer(DataAccessPolicyManager lockManager, int writerId) {
        this.lockManager = lockManager;
        this.writerId = writerId;
    }

    @Override
    public void run() {
        while (true) {
            try {
                // Sleep for a random time before starting the next write
                Thread.sleep(random.nextInt(3000) + 2000);

                lockManager.acquireWriteLock();
                System.out.println("Writer " + writerId + " acquired write lock.");

                // Simulate writing activity
                System.out.println("Writer " + writerId + " is writing...");
                Thread.sleep(random.nextInt(2500) + 1000);
```

```
                System.out.println("Writer " + writerId + " done, releasing write
lock.");
                lockManager.releaseWriteLock();

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```

Implement DadaAccessPolicyManager

```
class DataAccessPolicyManager {
    private int readerCount;
    private Semaphore mutex; // To protect readerCount
    private Semaphore wrt;   // To control write access

    public DataAccessPolicyManager() {
        readerCount = 0;
        mutex = new Semaphore(1);
        wrt = new Semaphore(1);
    }
```

```
    public void acquireReadLock() {
        mutex.acquire();
        readerCount++;
        if (readerCount == 1) { // If this is the first reader
            wrt.acquire();      // it needs to acquire the write lock to block
writers
        }
        mutex.release();
    }
```

```
    public void releaseReadLock() {
        mutex.acquire();
        readerCount--;
        if (readerCount == 0) { // If this is the last reader
            wrt.release();      // it needs to release the write lock to allow
writers
        }
        mutex.release();
```

```
    }

    public void acquireWriteLock() {
        wrt.acquire();
    }

    public void releaseWriteLock() {
        wrt.release();
    }
}
```

Task2

Result

```
^C(base) fzdx@fzdx-PR4910P:~/cjt/CS240/Lab09$ javac ReadersWritersSimulation.java Reader.java Writer.java
(base) fzdx@fzdx-PR4910P:~/cjt/CS240/Lab09$ java ReadersWritersSimulation
Starting Readers-Writers simulation with 5 readers and 2 writers.
Policy: Writers-Priority
------------------------------------------------
Reader 1 acquired read lock.
Reader 1 is reading...
Reader 2 acquired read lock.
Reader 2 is reading...
Reader 3 acquired read lock.
Reader 3 is reading...
Reader 5 acquired read lock.
Reader 5 is reading...
Reader 1 done, releasing read lock.
Reader 4 acquired read lock.
Reader 4 is reading...
Reader 4 done, releasing read lock.
Reader 3 done, releasing read lock.
Reader 2 done, releasing read lock.
Reader 5 done, releasing read lock.
Writer 1 acquired write lock.
Writer 1 is writing...
Writer 1 done, releasing write lock.
Writer 2 acquired write lock.
Writer 2 is writing...
Writer 2 done, releasing write lock.
Writer 1 acquired write lock.
Writer 1 is writing...
```

Implement DadaAccessPolicyManager2

Then change the Class implementation **DadaAccessPolicyManager2 within other java codes**

```
class DataAccessPolicyManager2 {
    private int readerCount;
    private int writerCount;
    private Semaphore mutex1;     // Protects readerCount
    private Semaphore mutex2;     // Protects writerCount
    private Semaphore wrt;        // Common for readers and writers
    private Semaphore readLock;   // For blocking readers

    public DataAccessPolicyManager2() {
        readerCount = 0;
        writerCount = 0;
        mutex1 = new Semaphore(1);
        mutex2 = new Semaphore(1);
        wrt = new Semaphore(1);
        readLock = new Semaphore(1);
    }

    public void acquireReadLock() {
        readLock.acquire(); // Lock to prevent readers if a writer is waiting
        mutex1.acquire();
        readerCount++;
        if (readerCount == 1) { // First reader
            wrt.acquire();      // Lock out writers
        }
        mutex1.release();
        readLock.release(); // Release the read lock for other readers
    }

    public void releaseReadLock() {
        mutex1.acquire();
        readerCount--;
        if (readerCount == 0) { // Last reader
            wrt.release();      // Allow writers
        }
        mutex1.release();
    }

    public void acquireWriteLock() {
        mutex2.acquire();
        writerCount++;
        if (writerCount == 1) { // First writer
```

```
            readLock.acquire(); // Lock out readers
        }
        mutex2.release();
        wrt.acquire();
    }

    public void releaseWriteLock() {
        wrt.release();
        mutex2.acquire();
        writerCount--;
        if (writerCount == 0) { // Last writer
            readLock.release(); // Allow readers
        }
        mutex2.release();
    }
}
```