3. Java Programming for Complete Beginners - Java 16 > Assignments > Problem Statement 1 > FileEncryption.java > ...

```java
import java.io.*;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

public class FileEncryption {

    private static final String ALGORITHM = "AES";

    // Generate AES key of specified size (128 bits here)
    public static SecretKey generateKey() throws Exception {
        KeyGenerator keyGen = KeyGenerator.getInstance(ALGORITHM);
        keyGen.init(128, new SecureRandom());
        return keyGen.generateKey();
    }

    // Encrypt input file and write encrypted bytes to output file
    public static void encryptFile(SecretKey key, File inputFile, File outputFile) throws Exception {
        byte[] inputBytes = readFile(inputFile);
        byte[] outputBytes = doCrypto(Cipher.ENCRYPT_MODE, key, inputBytes);
        writeFile(outputFile, outputBytes);
    }

    // Decrypt encrypted file and write decrypted bytes to output file
    public static void decryptFile(SecretKey key, File encryptedFile, File outputFile) throws Exception {
        byte[] encryptedBytes = readFile(encryptedFile);
        byte[] decryptedBytes = doCrypto(Cipher.DECRYPT_MODE, key, encryptedBytes);
        writeFile(outputFile, decryptedBytes);
    }

    // Helper to perform encryption or decryption
    private static byte[] doCrypto(int cipherMode, SecretKey key, byte[] inputBytes) throws Exception {
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(cipherMode, key);
        return cipher.doFinal(inputBytes);
    }

    // Helper to read file bytes
    private static byte[] readFile(File file) throws IOException {
        try (FileInputStream fis = new FileInputStream(file)) {
            byte[] bytes = new byte[(int) file.length()];
            int read = fis.read(bytes);
            if (read != bytes.length) {
                throw new IOException("Could not read entire file");
            }
            return bytes;
        }
    }

    // Helper to write bytes to file
    private static void writeFile(File file, byte[] bytes) throws IOException {
        try (FileOutputStream fos = new FileOutputStream(file)) {
            fos.write(bytes);
        }
    }
}

```

3. Java Programming for Complete Beginners - Java 16 > Assignments > Problem Statement 1 > Main.java > ...

```java
import java.io.File;
import java.util.Arrays;
import javax.crypto.SecretKey;

public class Main {
    public static void main(String[] args) {
        try {
            // Generate AES key
            SecretKey secretKey = FileEncryption.generateKey();
            System.out.println("AES key generated.");

            // Define files
            File inputFile = new File("input.txt");
            File encryptedFile = new File("encrypted.dat");
            File decryptedFile = new File("decrypted.txt");

            // Encrypt file
            FileEncryption.encryptFile(secretKey, inputFile, encryptedFile);
            System.out.println("File encrypted to " + encryptedFile.getName());

            // Decrypt file
            FileEncryption.decryptFile(secretKey, encryptedFile, decryptedFile);
            System.out.println("File decrypted to " + decryptedFile.getName());

            // Verify integrity by comparing input and decrypted files
            byte[] original = java.nio.file.Files.readAllBytes(inputFile.toPath());
            byte[] decrypted = java.nio.file.Files.readAllBytes(decryptedFile.toPath());

            if (Arrays.equals(original, decrypted)) {
                System.out.println("Decryption verified: files match exactly.");
            } else {
                System.out.println("Decryption verification failed: files differ.");
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

3. Java Programming for Complete Beginners - Java 16 > Assignments > Problem Statement 1 > input.txt

```
This is a test file for AES encryption and decryption.
Line 2 of the file.
End of file.

```

```
PS C:\Users\noelm\Downloads\IBM Consulting\TECHADEMY\Phase 3> cd '.\3. Java Programming for Complete Beginners - Java 16\Assignments\Problem Statement 1\'
PS C:\Users\noelm\Downloads\IBM Consulting\TECHADEMY\Phase 3\3. Java Programming for Complete Beginners - Java 16\Assignments\Problem Statement 1> javac FileEncryption.java Main.java
PS C:\Users\noelm\Downloads\IBM Consulting\TECHADEMY\Phase 3\3. Java Programming for Complete Beginners - Java 16\Assignments\Problem Statement 1> java Main
AES key generated.
File encrypted to encrypted.dat
File decrypted to decrypted.txt
Decryption verified: files match exactly.
```

3. Java Programming for Complete Beginners - Java 16 > Assignments > Problem Statement 2 > Student.java > ...

```java
class Student {
    private int id;
    private String name;

    public Student(int id, String name) {
        if (name == null || name.isEmpty()) {
            throw new IllegalArgumentException("Name can't be null or empty");
        }
        this.id = id;
        this.name = name;
    }

    public int getId() { return id; }
    public String getName() { return name; }

    public void setName(String name) {
        if (name == null || name.isEmpty()) {
            throw new IllegalArgumentException("Name can't be null or empty");
        }
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student{id=" + id + ", name='" + name + "'}";
    }

    @Override
    public int hashCode() {
        return Integer.hashCode(id);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (!(obj instanceof Student)) return false;
        Student other = (Student) obj;
        return this.id == other.id;
    }
}
```

3. Java Programming for Complete Beginners - Java 16 > Assignments > Problem Statement 2 > InvalidStudentDataException.java > ...

```java
public class InvalidStudentDataException extends Exception {
    public InvalidStudentDataException(String message) {
        super(message);
    }
}
```

3. Java Programming for Complete Beginners - Java 16 > Assignments > Problem Statement 2 > CollectionDemo.java > ...

```java
import java.io.*;
import java.util.*;

public class CollectionDemo {
    public static void main(String[] args) {
        // 1. Demonstrate ArrayList, LinkedList, HashMap, HashSet
        ArrayList<Student> arrayList = new ArrayList<>();
        LinkedList<Student> linkedList = new LinkedList<>();
        HashMap<Integer, Student> hashMap = new HashMap<>();
        HashSet<Student> hashSet = new HashSet<>();

        try {
            // Adding students
            Student s1 = new Student(1, "Alice");
            Student s2 = new Student(2, "Bob");
            Student s3 = new Student(3, "Charlie");

            arrayList.add(s1);
            arrayList.add(s2);
            arrayList.add(s3);

            linkedList.addAll(arrayList);
            for (Student s : arrayList) {
                hashMap.put(s.getId(), s);
                hashSet.add(s);
            }

            // CRUD Operations in ArrayList
            System.out.println("Initial ArrayList: " + arrayList);

            // Read by index
            System.out.println("Student at index 1: " + arrayList.get(1));

            // Update
            arrayList.get(1).setName("Bobby");
            System.out.println("After update: " + arrayList);

            // Delete
            arrayList.remove(0);
            System.out.println("After deletion: " + arrayList);

            // Exception handling - accessing invalid index
            try {
                System.out.println(arrayList.get(10));
            } catch (IndexOutOfBoundsException e) {
                System.out.println("Caught Exception: " + e);
            }

            // NullPointerException example
            Student nullStudent = null;
            try {
                System.out.println(nullStudent.getName());
            } catch (NullPointerException e) {
                System.out.println("Caught Exception: " + e);
            }

            // IllegalArgumentException example
            try {
                Student s4 = new Student(4, "");
            } catch (IllegalArgumentException e) {
                System.out.println("Caught Exception: " + e);
            }

            // Custom Exception usage
            try {
                validateStudentData(new Student(5, null));
            } catch (InvalidStudentDataException e) {
                System.out.println("Caught Custom Exception: " + e.getMessage());
            }

            // Reading data from file and handling exceptions
            System.out.println("\nReading students from file:");
            List<Student> fileStudents = readStudentsFromFile("students.txt");
            for (Student s : fileStudents) {
                System.out.println(s);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Method to validate student data using custom exception
    public static void validateStudentData(Student student) throws InvalidStudentDataException {
        if (student.getName() == null || student.getName().isEmpty()) {
            throw new InvalidStudentDataException("Student name is invalid");
        }
    }

    // Read students from file and handle file related exceptions
    public static List<Student> readStudentsFromFile(String filename) {
        List<Student> students = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
            String line;
            int lineNum = 1;
            while ((line = br.readLine()) != null) {
                try {
                    String[] parts = line.split(",");
                    if (parts.length != 2) {
                        System.out.println("Skipping invalid line " + lineNum);
                        lineNum++;
                        continue;
                    }
                    int id = Integer.parseInt(parts[0].trim());
                    String name = parts[1].trim();
                    students.add(new Student(id, name));
                } catch (NumberFormatException e) {
                    System.out.println("Invalid ID format on line " + lineNum);
                }
                lineNum++;
            }
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + filename);
        } catch (IOException e) {
            System.out.println("Error reading file: " + filename);
        }
        return students;
    }
}
```

```
Initial ArrayList: [Student{id=1, name='Alice'}, Student{id=2, name='Bob'}, Student{id=3, name='Charlie'}]
Student at index 1: Student{id=2, name='Bob'}
After update: [Student{id=1, name='Alice'}, Student{id=2, name='Bobby'}, Student{id=3, name='Charlie'}]
After deletion: [Student{id=2, name='Bobby'}, Student{id=3, name='Charlie'}]
Caught Exception: java.lang.IndexOutOfBoundsException: Index 10 out of bounds for length 2
Caught Exception: java.lang.NullPointerException: Cannot invoke "Student.getName()" because "<local8>" is null
Caught Exception: java.lang.IllegalArgumentException: Name can't be null or empty
java.lang.IllegalArgumentException: Name can't be null or empty
        at Student.<init>(Student.java:7)
        at CollectionDemo.main(CollectionDemo.java:66)
```

```
import java.util.*;
import java.util.concurrent.*;

public class ConcurrentCollectionsDemo {

    // Shared collections
    private static ConcurrentHashMap<Integer, String> concurrentMap = new ConcurrentHashMap<>();
    private static HashMap<Integer, String> hashMap = new HashMap<>();

    private static ConcurrentLinkedQueue<Integer> concurrentQueue = new ConcurrentLinkedQueue<>();
    private static LinkedList<Integer> linkedList = new LinkedList<>();

    private static CopyOnWriteArrayList<String> concurrentList = new CopyOnWriteArrayList<>();
    private static ArrayList<String> arrayList = new ArrayList<>();

    private static final int NUM_THREADS = 5;
    private static final int OPERATIONS_PER_THREAD = 10000;

    public static void main(String[] args) throws InterruptedException {
        System.out.println("Starting concurrent collections demo...\n");
```

```
    // Test ConcurrentHashMap vs HashMap with multiple threads
    System.out.println("Testing Map performance:");
    testMapPerformance();

    // Test ConcurrentLinkedQueue vs LinkedList
    System.out.println("\nTesting Queue performance:");
    testQueuePerformance();

    // Test CopyOnWriteArrayList vs ArrayList
    System.out.println("\nTesting List performance:");
    testListPerformance();
}

private static void testMapPerformance() throws InterruptedException {
    // Warmup HashMap (not thread-safe)
    hashMap.clear();
    long startHashMap = System.currentTimeMillis();
    Thread[] threads = new Thread[NUM_THREADS];
    for (int i = 0; i < NUM_THREADS; i++) {
        int threadId = i;
        threads[i] = new Thread(() -> {
            for (int j = 0; j < OPERATIONS_PER_THREAD; j++) {
                // Not synchronized - may cause errors or exceptions
                hashMap.put(threadId * OPERATIONS_PER_THREAD + j, "Val" + j);
            }
        });
        threads[i].start();
    }
    for (Thread t : threads) t.join();
    long endHashMap = System.currentTimeMillis();
    System.out.println("HashMap (non-concurrent) time: " + (endHashMap - startHashMap) + " ms");
    System.out.println("HashMap size: " + hashMap.size());
```

```
    // ConcurrentHashMap
    concurrentMap.clear();
    long startConcurrentMap = System.currentTimeMillis();
    for (int i = 0; i < NUM_THREADS; i++) {
        int threadId = i;
        threads[i] = new Thread(() -> {
            for (int j = 0; j < OPERATIONS_PER_THREAD; j++) {
                concurrentMap.put(threadId * OPERATIONS_PER_THREAD + j, "Val" + j);
            }
        });
        threads[i].start();
    }
    for (Thread t : threads) t.join();
    long endConcurrentMap = System.currentTimeMillis();
    System.out.println("ConcurrentHashMap time: " + (endConcurrentMap - startConcurrentMap) + " ms");
    System.out.println("ConcurrentHashMap size: " + concurrentMap.size());
}

private static void testQueuePerformance() throws InterruptedException {
    // LinkedList (not thread-safe)
    linkedList.clear();
    Thread[] threads = new Thread[NUM_THREADS];
    long startLinkedList = System.currentTimeMillis();
    for (int i = 0; i < NUM_THREADS; i++) {
        int threadId = i;
        threads[i] = new Thread(() -> {
            for (int j = 0; j < OPERATIONS_PER_THREAD; j++) {
                synchronized (linkedList) {
                    linkedList.add(threadId * OPERATIONS_PER_THREAD + j);
                }
            }
```

```
        });
        threads[i].start();
    }
    for (Thread t : threads) t.join();
    long endLinkedList = System.currentTimeMillis();
    System.out.println("LinkedList with synchronization time: " + (endLinkedList - startLinkedList) + " ms");
    System.out.println("LinkedList size: " + linkedList.size());

    // ConcurrentLinkedQueue (thread-safe)
    concurrentQueue.clear();
    long startConcurrentQueue = System.currentTimeMillis();
    for (int i = 0; i < NUM_THREADS; i++) {
        int threadId = i;
        threads[i] = new Thread(() -> {
            for (int j = 0; j < OPERATIONS_PER_THREAD; j++) {
                concurrentQueue.add(threadId * OPERATIONS_PER_THREAD + j);
            }
        });
        threads[i].start();
    }
    for (Thread t : threads) t.join();
    long endConcurrentQueue = System.currentTimeMillis();
    System.out.println("ConcurrentLinkedQueue time: " + (endConcurrentQueue - startConcurrentQueue) + "
ms");
    System.out.println("ConcurrentLinkedQueue size: " + concurrentQueue.size());
  }

  private static void testListPerformance() throws InterruptedException {
    // ArrayList (not thread-safe)
    arrayList.clear();
    Thread[] threads = new Thread[NUM_THREADS];
    long startArrayList = System.currentTimeMillis();
    for (int i = 0; i < NUM_THREADS; i++) {
```

```
        int threadId = i;
        threads[i] = new Thread(() -> {
            for (int j = 0; j < OPERATIONS_PER_THREAD; j++) {
                synchronized (arrayList) {
                    arrayList.add("Val" + (threadId * OPERATIONS_PER_THREAD + j));
                }
            }
        });
        threads[i].start();
    }
    for (Thread t : threads) t.join();
    long endArrayList = System.currentTimeMillis();
    System.out.println("ArrayList with synchronization time: " + (endArrayList - startArrayList) + " ms");
    System.out.println("ArrayList size: " + arrayList.size());

    // CopyOnWriteArrayList (thread-safe but costly on writes)
    concurrentList.clear();
    long startCopyOnWrite = System.currentTimeMillis();
    for (int i = 0; i < NUM_THREADS; i++) {
        int threadId = i;
        threads[i] = new Thread(() -> {
            for (int j = 0; j < OPERATIONS_PER_THREAD; j++) {
                concurrentList.add("Val" + (threadId * OPERATIONS_PER_THREAD + j));
            }
        });
        threads[i].start();
    }
    for (Thread t : threads) t.join();
    long endCopyOnWrite = System.currentTimeMillis();
    System.out.println("CopyOnWriteArrayList time: " + (endCopyOnWrite - startCopyOnWrite) + " ms");
    System.out.println("CopyOnWriteArrayList size: " + concurrentList.size());
}
```

}

```
Starting concurrent collections demo...

Testing Map performance:
HashMap (non-concurrent) time: 25 ms
HashMap size: 40794
ConcurrentHashMap time: 20 ms
ConcurrentHashMap size: 50000

Testing Queue performance:
LinkedList with synchronization time: 10 ms
LinkedList size: 50000
ConcurrentLinkedQueue time: 16 ms
ConcurrentLinkedQueue size: 50000

Testing List performance:
ArrayList with synchronization time: 22 ms
ArrayList size: 50000
CopyOnWriteArrayList time: 1080 ms
CopyOnWriteArrayList size: 50000
```

```
import java.util.*;
import java.util.concurrent.*;

public class ConcurrentCollectionsTest {

    // Collections to test
    private static ConcurrentHashMap<Integer, String> concurrentMap = new ConcurrentHashMap<>();
    private static HashMap<Integer, String> hashMap = new HashMap<>();

    private static ConcurrentLinkedQueue<Integer> concurrentQueue = new ConcurrentLinkedQueue<>();
    private static LinkedList<Integer> linkedList = new LinkedList<>();

    private static CopyOnWriteArrayList<String> concurrentList = new CopyOnWriteArrayList<>();
```

```
private static ArrayList<String> arrayList = new ArrayList<>();

private static final int THREADS = 4;
private static final int OPERATIONS = 50000;

public static void main(String[] args) throws InterruptedException {
    System.out.println("Concurrent Collections Performance Test\n");

    testMaps();
    testQueues();
    testLists();
}

private static void testMaps() throws InterruptedException {
    System.out.println("Testing Map:");

    // Non-concurrent HashMap (not thread-safe)
    hashMap.clear();
    Thread[] threads = new Thread[THREADS];
    long start = System.currentTimeMillis();
    for (int i = 0; i < THREADS; i++) {
        int tid = i;
        threads[i] = new Thread(() -> {
            for (int j = 0; j < OPERATIONS; j++) {
                synchronized(hashMap) {
                    hashMap.put(tid * OPERATIONS + j, "Val" + j);
                }
            }
        });
        threads[i].start();
    }
    for (Thread t : threads) t.join();
```

```
        long end = System.currentTimeMillis();
        System.out.printf("HashMap with synchronization time: %d ms, size: %d%n", (end - start), hashMap.size());

        // ConcurrentHashMap
        concurrentMap.clear();
        for (int i = 0; i < THREADS; i++) {
            int tid = i;
            threads[i] = new Thread(() -> {
                for (int j = 0; j < OPERATIONS; j++) {
                    concurrentMap.put(tid * OPERATIONS + j, "Val" + j);
                }
            });
            threads[i].start();
        }
        for (Thread t : threads) t.join();
        end = System.currentTimeMillis();
        System.out.printf("ConcurrentHashMap time: %d ms, size: %d%n\n", (end - start), concurrentMap.size());
    }

    private static void testQueues() throws InterruptedException {
        System.out.println("Testing Queue:");

        // LinkedList with synchronization
        linkedList.clear();
        Thread[] threads = new Thread[THREADS];
        long start = System.currentTimeMillis();
        for (int i = 0; i < THREADS; i++) {
            int tid = i;
            threads[i] = new Thread(() -> {
                for (int j = 0; j < OPERATIONS; j++) {
                    synchronized(linkedList) {
                        linkedList.add(tid * OPERATIONS + j);
```

```
            }
        }
    });
    threads[i].start();
}
for (Thread t : threads) t.join();
long end = System.currentTimeMillis();
System.out.printf("LinkedList with synchronization time: %d ms, size: %d%n", (end - start), linkedList.size());

// ConcurrentLinkedQueue
concurrentQueue.clear();
for (int i = 0; i < THREADS; i++) {
    int tid = i;
    threads[i] = new Thread(() -> {
        for (int j = 0; j < OPERATIONS; j++) {
            concurrentQueue.add(tid * OPERATIONS + j);
        }
    });
    threads[i].start();
}
for (Thread t : threads) t.join();
end = System.currentTimeMillis();
System.out.printf("ConcurrentLinkedQueue time: %d ms, size: %d%n%n", (end - start), concurrentQueue.size());
}

private static void testLists() throws InterruptedException {
    System.out.println("Testing List:");

    // ArrayList with synchronization
    arrayList.clear();
    Thread[] threads = new Thread[THREADS];
```

```
        long start = System.currentTimeMillis();
        for (int i = 0; i < THREADS; i++) {
            int tid = i;
            threads[i] = new Thread(() -> {
                for (int j = 0; j < OPERATIONS; j++) {
                    synchronized(arrayList) {
                        arrayList.add("Val" + (tid * OPERATIONS + j));
                    }
                }
            });
            threads[i].start();
        }
        for (Thread t : threads) t.join();
        long end = System.currentTimeMillis();
        System.out.printf("ArrayList with synchronization time: %d ms, size: %d%n", (end - start), arrayList.size());

        // CopyOnWriteArrayList
        concurrentList.clear();
        for (int i = 0; i < THREADS; i++) {
            int tid = i;
            threads[i] = new Thread(() -> {
                for (int j = 0; j < OPERATIONS; j++) {
                    concurrentList.add("Val" + (tid * OPERATIONS + j));
                }
            });
            threads[i].start();
        }
        for (Thread t : threads) t.join();
        end = System.currentTimeMillis();
        System.out.printf("CopyOnWriteArrayList time: %d ms, size: %d%n", (end - start), concurrentList.size());
    }
}
```

```
Concurrent Collections Performance Test

Testing Map:
HashMap with synchronization time: 78 ms, size: 200000
ConcurrentHashMap time: 158 ms, size: 200000

Testing Queue:
LinkedList with synchronization time: 31 ms, size: 200000
ConcurrentLinkedQueue time: 99 ms, size: 200000

Testing List:
ArrayList with synchronization time: 72 ms, size: 200000
```