1. Person.java

- Concepts: Class, Object, State (Instance Variables), Methods (Behavior)
- Purpose: Demonstrates the creation of a Person class with instance variables (name and age) and a method (greet()) to log a greeting message.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Person {
    private static final Logger logger = LogManager.getLogger(Person.class);

    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
        logger.info("Person created - Name: {}, Age: {}", name, age);
    }

    public void greet() {
        logger.info("Hello, my name is {} and I am {} years old.", name, age);
    }

    // Main method to demonstrate Person class
    public static void main(String[] args) {
        Person person = new Person("Cris", 30);
        person.greet();
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.Person - Person created - Name: Cris, Age: 30
[main] INFO  com.example.javaconcepts.Person - Hello, my name is Cris and I am 30 years old.
```

2. DataTypeExample.java

- Concepts: Basic Data Types, Variable Types, Modifier Types, Final Keyword
- Purpose: Illustrates various data types (int, double, char, boolean), usage of final keyword, and different variable scopes (static, instance).

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class DataTypeExample {
    private static final Logger logger = LogManager.getLogger(DataTypeExample.class);

    private int number = 10;
    private double salary = 50000.0;
    private char grade = 'A';
    private boolean isActive = true;

    private final int finalVariable = 100;

    private static String staticVariable = "I am static";
    private int privateVariable = 200;
    protected int protectedVariable = 300;

    public static void printStatic() {
        logger.info(staticVariable);
    }

    public void printFinal() {
        logger.info("Final Variable: {}", finalVariable);
    }

    public void printPrivate() {
        logger.info("Private Variable: {}", privateVariable);
    }

    public void printProtected() {
        logger.info("Protected Variable: {}", protectedVariable);
    }

    // Main method to demonstrate DataTypeExample
    public static void main(String[] args) {
        printStatic();
```

```
        DataTypeExample example = new DataTypeExample();
        example.printFinal();
        example.printPrivate();
        example.printProtected();
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.DataTypeExample - I am static
[main] INFO  com.example.javaconcepts.DataTypeExample - Final Variable: 100
[main] INFO  com.example.javaconcepts.DataTypeExample - Private Variable: 200
[main] INFO  com.example.javaconcepts.DataTypeExample - Protected Variable: 300
```

3. ConstructorExample.java

- Concepts: Constructors
- Purpose: Shows the usage of constructors in Java to initialize an object (ConstructorExample) and log a message.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class ConstructorExample {
    private static final Logger logger = LogManager.getLogger(ConstructorExample.class);

    public ConstructorExample() {
        logger.info("Message from constructor");
    }

    // Main method to demonstrate ConstructorExample
    public static void main(String[] args) {
        ConstructorExample example = new ConstructorExample();
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.ConstructorExample - Message from constructor
```

4. LoopDecisionExample.java

- Concepts: Loop Control (for loop), Decision Making (if statement)
- Purpose: Demonstrates a for loop and an if statement to log iterations and make a decision based on a condition.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class LoopDecisionExample {
    private static final Logger logger = LogManager.getLogger(LoopDecisionExample.class);

    public void demonstrateLoop() {
        // For loop example
        for (int i = 0; i < 5; i++) {
            logger.info("Iteration: {}", i);
        }

        // Decision making example
        int x = 15;
        if (x > 10) {
            logger.info("x is greater than 10");
        }
    }

    // Main method to demonstrate LoopDecisionExample
    public static void main(String[] args) {
        LoopDecisionExample example = new LoopDecisionExample();
        example.demonstrateLoop();
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.LoopDecisionExample - Iteration: 0
[main] INFO  com.example.javaconcepts.LoopDecisionExample - Iteration: 1
[main] INFO  com.example.javaconcepts.LoopDecisionExample - Iteration: 2
[main] INFO  com.example.javaconcepts.LoopDecisionExample - Iteration: 3
[main] INFO  com.example.javaconcepts.LoopDecisionExample - Iteration: 4
[main] INFO  com.example.javaconcepts.LoopDecisionExample - x is greater than 10
```

5. StringExample.java

- Concepts: Strings
- Purpose: Shows basic usage of String objects and logging a message using strings.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class StringExample {
    private static final Logger logger = LogManager.getLogger(StringExample.class);

    public void demonstrateStrings() {
        String message = "Hello, World!";
        logger.info("Message: {}", message);
    }

    // Main method to demonstrate StringExample
    public static void main(String[] args) {
        StringExample example = new StringExample();
        example.demonstrateStrings();
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.StringExample - Message: Hello, World!
```

6. ArrayExample.java

- Concepts: Arrays
- Purpose: Illustrates the declaration of an array (int[] numbers) and logs its length.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class ArrayExample {
    private static final Logger logger = LogManager.getLogger(ArrayExample.class);

    public void demonstrateArrays() {
        int[] numbers = {1, 2, 3, 4, 5};
        logger.info("Array Length: {}", numbers.length);
```

```
    }

    // Main method to demonstrate ArrayExample
    public static void main(String[] args) {
        ArrayExample example = new ArrayExample();
        example.demonstrateArrays();
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.ArrayExample - Array Length: 5
```

7. Animal.java and Dog.java

- Concepts: Inheritance, Overriding
- Purpose:
    - Animal.java: Defines an Animal class with a method sound() that logs a generic animal sound.
    - Dog.java: Extends Animal to override the sound() method specifically for a dog's bark.

Animal.java Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Animal {
    private static final Logger logger = LogManager.getLogger(Animal.class);

    public void sound() {
        logger.info("Animal makes a sound");
    }

    // Main method to demonstrate Animal (inherited by Dog)
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.sound();

        Dog dog = new Dog();
        dog.sound();
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.Animal - Animal makes a sound
[main] INFO  com.example.javaconcepts.Dog - Dog barks
```

Dog.java Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Dog extends Animal {
    private static final Logger logger = LogManager.getLogger(Dog.class);

    @Override
    public void sound() {
        logger.info("Dog barks");
    }

    // Main method to demonstrate Dog class
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.sound();
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.Dog - Dog barks
```

8. Shape.java and Circle.java

- Concepts: Abstraction
- Purpose:
    - Shape.java: Defines an abstract Shape class with an abstract method draw() to be implemented by subclasses.
    - Circle.java: Implements Shape to provide a concrete draw() method for drawing a circle.

Shape.java Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public abstract class Shape {
    private static final Logger logger = LogManager.getLogger(Shape.class);
```

```
    public abstract void draw();
}
```

Circle.java Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Circle extends Shape {
    private static final Logger logger = LogManager.getLogger(Circle.class);

    @Override
    public void draw() {
        logger.info("Drawing a Circle");
    }

    // Main method to demonstrate Circle
    public static void main(String[] args) {
        Circle circle = new Circle();
        circle.draw();
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.Circle - Drawing a Circle
```

9. EncapsulatedObject.java

- Concepts: Encapsulation
- Purpose: Encapsulates data (data field) and provides getter and setter methods (getData() and setData()) to access and modify the data, respectively.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class EncapsulatedObject {
    private static final Logger logger = LogManager.getLogger(EncapsulatedObject.class);
```

```
    private String data;

    public EncapsulatedObject(String data) {
        this.data = data;
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }

    // Main method to demonstrate EncapsulatedObject
    public static void main(String[] args) {
        EncapsulatedObject obj = new EncapsulatedObject("Data");
        logger.info("Encapsulated Data: {}", obj.getData());
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.EncapsulatedObject - Encapsulated Data: Data
```

10. Printable.java and Document.java

- Concepts: Interfaces
- Purpose:
    - Printable.java: Declares the Printable interface with an abstract method print().
    - Document.java: Implements Printable to provide a concrete implementation of print() for printing a document.

Printable.java Code:

```
package com.example.javaconcepts;

public interface Printable {
    void print();
}
```

Document.java Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
```

```
public class Document implements Printable {
    private static final Logger logger = LogManager.getLogger(Document.class);

    @Override
    public void print() {
        logger.info("Printing Document");
    }

    // Main method to demonstrate Document
    public static void main(String[] args) {
        Document document = new Document();
        document.print();
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.Document - Printing Document
```

11. ExceptionHandlingExample.java

- Concepts: Exception, Hierarchy of Exception, Handling Exception, Throw vs Throws
- Purpose: Demonstrates exception handling with try-catch blocks and the usage of throw and throws.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class ExceptionHandlingExample {
    private static final Logger logger = LogManager.getLogger(ExceptionHandlingExample.class);

    public static void main(String[] args) {
        try {
            handleException();
        } catch (CustomException e) {
            logger.error("Caught CustomException: {}", e.getMessage());
        }

        try {
            int result = divide(10, 0);
        } catch (ArithmeticException e) {
```

```
            logger.error("Caught ArithmeticException: {}", e.getMessage());
        }
    }

    public static void handleException() throws CustomException {
        throw new CustomException("This is a custom exception");
    }

    public static int divide(int a, int b) {
        return a / b;  // This will throw ArithmeticException if b is 0
    }
}

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

Output:

```
[main] ERROR com.example.javaconcepts.ExceptionHandlingExample - Caught CustomException: This is a custom exception
[main] ERROR com.example.javaconcepts.ExceptionHandlingExample - Caught ArithmeticException: / by zero
```

12. CollectionsExample.java

- Concepts: Collections (List, Map, Set), Collection Operations on Primitives and Custom Objects
- Purpose: Demonstrates basic operations on collections with primitive types and custom objects.

CustomObjects.java Code:

```
package com.example.javaconcepts;

public class CustomObject {
    private int id;
    private String name;

    public CustomObject(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }
```

```
    public String getName() {
        return name;
    }

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

CollectionsExample.java Code:

```
package com.example.javaconcepts;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class CollectionOperationsExample {
    private static final Logger logger =
LogManager.getLogger(CollectionOperationsExample.class);

    public static void main(String[] args) {
        // List operations
        List<Integer> numbersList = new ArrayList<>();
        numbersList.add(1);
        numbersList.add(3);
        numbersList.add(2);
        logger.info("List: " + numbersList);

        // Set operations
        Set<String> namesSet = new HashSet<>();
        namesSet.add("Alice");
        namesSet.add("Bob");
        namesSet.add("Alice");
        logger.info("Set: " + namesSet);
```

```
        // Map operations
        Map<Integer, String> studentMap = new HashMap<>();
        studentMap.put(1, "John");
        studentMap.put(2, "Jane");
        studentMap.put(3, "Doe");
        logger.info("Map: " + studentMap);

        // Collection operations on custom objects
        List<CustomObject> customObjects = new ArrayList<>();
        customObjects.add(new CustomObject(1, "John"));
        customObjects.add(new CustomObject(2, "Jane"));
        logger.info("Custom Objects List: " + customObjects);
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.CollectionOperationsExample - List: [1, 3, 2]
[main] INFO  com.example.javaconcepts.CollectionOperationsExample - Set: [Bob, Alice]
[main] INFO  com.example.javaconcepts.CollectionOperationsExample - Map: {1=John, 2=Jane, 3=Doe}
[main] INFO  com.example.javaconcepts.CollectionOperationsExample - Custom Objects List: [CustomObject{id=1, name='John'}, Cu
```

13. SortingExample.java

- Concepts: Sorting Lists using Comparable and Comparator
- Purpose: Demonstrates sorting a list of integers and a list of custom objects using Comparable and Comparator.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.util.*;

public class SortingExample {
    private static final Logger logger = LogManager.getLogger(SortingExample.class);

    public static void main(String[] args) {
        // List of integers
        List<Integer> intList = Arrays.asList(5, 3, 1, 4, 2);
        Collections.sort(intList);
        logger.info("Sorted List of Integers: {}", intList);
```

```
        // List of custom objects
        List<Student> studentList = new ArrayList<>();
        studentList.add(new Student(3, "John"));
        studentList.add(new Student(1, "Jane"));
        studentList.add(new Student(2, "Joe"));
        logger.info("List of Students: {}", studentList);

        // Sorting the list of students using Comparable
        Collections.sort(studentList);
        logger.info("Sorted List of Students by ID: {}", studentList);

        // Sorting the list of students using Comparator
        studentList.sort(Comparator.comparing(Student::getName));
        logger.info("Sorted List of Students by Name: {}", studentList);
    }
}

class Student implements Comparable<Student> {
    private int id;
    private String name;

    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    @Override
    public int compareTo(Student other) {
        return Integer.compare(this.id, other.id);
    }

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

Output:

```
Sorted List of Integers: [1, 2, 3, 4, 5]
List of Students: [Student{id=3, name='John'}, Student{id=1, name='Jane'}, Student{id=2, name='Joe'}]
Sorted List of Students by ID: [Student{id=1, name='Jane'}, Student{id=2, name='Joe'}, Student{id=3, name='John'}]
Sorted List of Students by Name: [Student{id=1, name='Jane'}, Student{id=2, name='Joe'}, Student{id=3, name='John'}]
```

14. MultithreadingExample.java

- **Concepts**: Basic Multithreading
- **Purpose**: Demonstrates creating and running threads.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class MultithreadingExample {
    private static final Logger logger = LogManager.getLogger(MultithreadingExample.class);

    public static void main(String[] args) {
        // Creating threads
        Thread thread1 = new Thread(new MyRunnable("Thread 1"));
        Thread thread2 = new Thread(new MyRunnable("Thread 2"));

        // Starting threads
        thread1.start();
        thread2.start();
    }

    // Runnable implementation
    static class MyRunnable implements Runnable {
        private final String name;

        public MyRunnable(String name) {
            this.name = name;
        }

        @Override
        public void run() {
            for (int i = 0; i < 5; i++) {
                logger.info(name + ": " + i);
                try {
```

```
                Thread.sleep(100); // Simulating some task
            } catch (InterruptedException e) {
                logger.error("Thread interrupted: " + e.getMessage());
            }
        }
    }
  }
}
```

Output:

```
[Thread-0] INFO  com.example.javaconcepts.MultithreadingExample - Thread 1: 0
[Thread-1] INFO  com.example.javaconcepts.MultithreadingExample - Thread 2: 0
[Thread-0] INFO  com.example.javaconcepts.MultithreadingExample - Thread 1: 1
[Thread-1] INFO  com.example.javaconcepts.MultithreadingExample - Thread 2: 1
[Thread-0] INFO  com.example.javaconcepts.MultithreadingExample - Thread 1: 2
[Thread-1] INFO  com.example.javaconcepts.MultithreadingExample - Thread 2: 2
[Thread-0] INFO  com.example.javaconcepts.MultithreadingExample - Thread 1: 3
[Thread-1] INFO  com.example.javaconcepts.MultithreadingExample - Thread 2: 3
[Thread-0] INFO  com.example.javaconcepts.MultithreadingExample - Thread 1: 4
[Thread-1] INFO  com.example.javaconcepts.MultithreadingExample - Thread 2: 4
```

15. ThreadPoolExecutorExample.java

- **Concepts**: Thread Pool Executor
- **Purpose**: Demonstrates using a thread pool executor to manage multiple threads.

Code:

```
package com.example.javaconcepts;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class ThreadPoolExecutorExample {
   private static final Logger logger =
LogManager.getLogger(ThreadPoolExecutorExample.class);

   public static void main(String[] args) {
      ExecutorService executorService = Executors.newFixedThreadPool(2);

      // Submitting tasks to thread pool
      for (int i = 1; i <= 5; i++) {
         executorService.submit(new MyTask("Task " + i));
```

```
        }

        // Shutting down the executor
        executorService.shutdown();
    }

    static class MyTask implements Runnable {
        private final String name;

        public MyTask(String name) {
            this.name = name;
        }

        @Override
        public void run() {
            logger.info(name + " started");
            try {
                Thread.sleep(100); // Simulating some task
            } catch (InterruptedException e) {
                logger.error("Task interrupted: " + e.getMessage());
            }
            logger.info(name + " completed");
        }
    }
}
```

Output:

```
[pool-2-thread-2] INFO  com.example.javaconcepts.ThreadPoolExecutorExample - Task 2 started
[pool-2-thread-1] INFO  com.example.javaconcepts.ThreadPoolExecutorExample - Task 1 started
[pool-2-thread-2] INFO  com.example.javaconcepts.ThreadPoolExecutorExample - Task 2 completed
[pool-2-thread-1] INFO  com.example.javaconcepts.ThreadPoolExecutorExample - Task 1 completed
[pool-2-thread-2] INFO  com.example.javaconcepts.ThreadPoolExecutorExample - Task 3 started
[pool-2-thread-1] INFO  com.example.javaconcepts.ThreadPoolExecutorExample - Task 4 started
[pool-2-thread-1] INFO  com.example.javaconcepts.ThreadPoolExecutorExample - Task 4 completed
[pool-2-thread-2] INFO  com.example.javaconcepts.ThreadPoolExecutorExample - Task 3 completed
[pool-2-thread-1] INFO  com.example.javaconcepts.ThreadPoolExecutorExample - Task 5 started
[pool-2-thread-1] INFO  com.example.javaconcepts.ThreadPoolExecutorExample - Task 5 completed
```

16. VolatileExample.java

- **Concepts**: Volatile Keyword
- **Purpose**: Demonstrates the use of the volatile keyword for visibility of shared variables among threads.

Code:

```
package com.example.javaconcepts;
```

```java
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class VolatileExample {
    private static final Logger logger = LogManager.getLogger(VolatileExample.class);

    private static volatile boolean flag = false;

    public static void main(String[] args) throws InterruptedException {
        Thread writerThread = new Thread(() -> {
            logger.info("Starting writer thread");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            flag = true;
            logger.info("Flag set to true");
        });

        Thread readerThread = new Thread(() -> {
            logger.info("Starting reader thread");
            while (!flag) {
                // Busy wait until flag is true
            }
            logger.info("Flag is now true");
        });

        writerThread.start();
        readerThread.start();

        writerThread.join();
        readerThread.join();

        logger.info("End of main method");
    }
}
```

Output:

```
[Thread-1] INFO  com.example.javaconcepts.VolatileExample - Starting reader thread
[Thread-0] INFO  com.example.javaconcepts.VolatileExample - Starting writer thread
[Thread-1] INFO  com.example.javaconcepts.VolatileExample - Flag is now true
[Thread-0] INFO  com.example.javaconcepts.VolatileExample - Flag set to true
[main] INFO  com.example.javaconcepts.VolatileExample - End of main method
```

17. SynchronizationExample.java

- **Concepts**: Synchronized Methods and Blocks
- **Purpose**: Demonstrates thread safety using synchronized methods and blocks.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class SynchronizationExample {
    private static final Logger logger = LogManager.getLogger(SynchronizationExample.class);

    private static int counter = 0;

    public static void main(String[] args) {
        // Using synchronized method
        Thread thread1 = new Thread(() -> {
            for (int i = 0; i < 5; i++) {
                synchronizedMethod();
            }
        });

        // Using synchronized block
        Thread thread2 = new Thread(() -> {
            for (int i = 0; i < 5; i++) {
                synchronizedBlock();
            }
        });

        thread1.start();
        thread2.start();
    }

    public synchronized static void synchronizedMethod() {
        logger.info("Synchronized Method - Counter: " + (++counter));
```

```
    }

    public static void synchronizedBlock() {
        synchronized (SynchronizationExample.class) {
            logger.info("Synchronized Block - Counter: " + (++counter));
        }
    }
}
```

Output:

```
[Thread-1] INFO  com.example.javaconcepts.SynchronizationExample - Synchronized Block - Counter: 1
[Thread-1] INFO  com.example.javaconcepts.SynchronizationExample - Synchronized Block - Counter: 2
[Thread-1] INFO  com.example.javaconcepts.SynchronizationExample - Synchronized Block - Counter: 3
[Thread-1] INFO  com.example.javaconcepts.SynchronizationExample - Synchronized Block - Counter: 4
[Thread-1] INFO  com.example.javaconcepts.SynchronizationExample - Synchronized Block - Counter: 5
[Thread-0] INFO  com.example.javaconcepts.SynchronizationExample - Synchronized Method - Counter: 6
[Thread-0] INFO  com.example.javaconcepts.SynchronizationExample - Synchronized Method - Counter: 7
[Thread-0] INFO  com.example.javaconcepts.SynchronizationExample - Synchronized Method - Counter: 8
[Thread-0] INFO  com.example.javaconcepts.SynchronizationExample - Synchronized Method - Counter: 9
[Thread-0] INFO  com.example.javaconcepts.SynchronizationExample - Synchronized Method - Counter: 10
```

18.InstanceThreadSafetyExample.java

Thread Safety with Instance Variables

- **Concepts**: Synchronized Methods and Blocks
- **Purpose**: Demonstrates thread safety using synchronized methods and instance variables.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class InstanceThreadSafetyExample {
    private static final Logger logger =
LogManager.getLogger(InstanceThreadSafetyExample.class);

    private int counter = 0;

    public static void main(String[] args) {
        InstanceThreadSafetyExample example = new InstanceThreadSafetyExample();
        example.runThreads();
    }

    public void runThreads() {
```

```
        Thread thread1 = new Thread(this::incrementCounter);
        Thread thread2 = new Thread(this::incrementCounter);

        thread1.start();
        thread2.start();
    }

    public synchronized void incrementCounter() {
        for (int i = 0; i < 5; i++) {
            counter++;
            logger.info("Counter: {}", counter);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```

Output:

```
[Thread-0] INFO  com.example.javaconcepts.InstanceThreadSafetyExample - Counter: 1
[Thread-0] INFO  com.example.javaconcepts.InstanceThreadSafetyExample - Counter: 2
[Thread-0] INFO  com.example.javaconcepts.InstanceThreadSafetyExample - Counter: 3
[Thread-0] INFO  com.example.javaconcepts.InstanceThreadSafetyExample - Counter: 4
[Thread-0] INFO  com.example.javaconcepts.InstanceThreadSafetyExample - Counter: 5
[Thread-1] INFO  com.example.javaconcepts.InstanceThreadSafetyExample - Counter: 6
[Thread-1] INFO  com.example.javaconcepts.InstanceThreadSafetyExample - Counter: 7
```

Thread Safety with Local Variables

- **Concepts**: Local Variables
- **Purpose**: Demonstrates that local variables are inherently thread-safe because each thread has its own stack.

LocalThreadSafetyExample.java

Code:

```
package com.example.javaconcepts;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class LocalThreadSafetyExample {
```

```
    private static final Logger logger = LogManager.getLogger(LocalThreadSafetyExample.class);
    public static void main(String[] args) {
        LocalThreadSafetyExample example = new LocalThreadSafetyExample();
        example.runThreads();
    }
    public void runThreads() {
        Thread thread1 = new Thread(this::printNumbers);
        Thread thread2 = new Thread(this::printNumbers);
        thread1.start();
        thread2.start();
    }

    public void printNumbers() {
        for (int i = 0; i < 5; i++) {
            int localCounter = i;
            logger.info("Thread: {}, Local Counter: {}", Thread.currentThread().getName(),
localCounter);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```

Output:

```
[Thread-0] INFO  com.example.javaconcepts.LocalThreadSafetyExample - Thread: Thread-0, Local Counter: 0
[Thread-1] INFO  com.example.javaconcepts.LocalThreadSafetyExample - Thread: Thread-1, Local Counter: 0
[Thread-0] INFO  com.example.javaconcepts.LocalThreadSafetyExample - Thread: Thread-0, Local Counter: 1
[Thread-1] INFO  com.example.javaconcepts.LocalThreadSafetyExample - Thread: Thread-1, Local Counter: 1
[Thread-0] INFO  com.example.javaconcepts.LocalThreadSafetyExample - Thread: Thread-0, Local Counter: 2
[Thread-1] INFO  com.example.javaconcepts.LocalThreadSafetyExample - Thread: Thread-1, Local Counter: 2
[Thread-0] INFO  com.example.javaconcepts.LocalThreadSafetyExample - Thread: Thread-0, Local Counter: 3
[Thread-1] INFO  com.example.javaconcepts.LocalThreadSafetyExample - Thread: Thread-1, Local Counter: 3
[Thread-0] INFO  com.example.javaconcepts.LocalThreadSafetyExample - Thread: Thread-0, Local Counter: 4
[Thread-1] INFO  com.example.javaconcepts.LocalThreadSafetyExample - Thread: Thread-1, Local Counter: 4
```

Thread Safety with Method Synchronization

- **Concepts**: Method Synchronization
- **Purpose**: Demonstrates thread safety using synchronized methods.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class MethodSynchronizationExample {
    private static final Logger logger =
LogManager.getLogger(MethodSynchronizationExample.class);

    private int counter = 0;

    public static void main(String[] args) {
        MethodSynchronizationExample example = new MethodSynchronizationExample();
        example.runThreads();
    }

    public void runThreads() {
        Thread thread1 = new Thread(this::incrementCounter);
        Thread thread2 = new Thread(this::incrementCounter);

        thread1.start();
        thread2.start();
    }

    public synchronized void incrementCounter() {
        for (int i = 0; i < 5; i++) {
            counter++;
            logger.info("Counter: {}", counter);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```

Output:

```
[Thread-0] INFO  com.example.javaconcepts.MethodSynchronizationExample - Counter: 1
[Thread-0] INFO  com.example.javaconcepts.MethodSynchronizationExample - Counter: 2
[Thread-0] INFO  com.example.javaconcepts.MethodSynchronizationExample - Counter: 3
[Thread-0] INFO  com.example.javaconcepts.MethodSynchronizationExample - Counter: 4
[Thread-0] INFO  com.example.javaconcepts.MethodSynchronizationExample - Counter: 5
[Thread-1] INFO  com.example.javaconcepts.MethodSynchronizationExample - Counter: 6
[Thread-1] INFO  com.example.javaconcepts.MethodSynchronizationExample - Counter: 7
```

Thread Safety with Block Synchronization

- **Concepts**: Block Synchronization
- **Purpose**: Demonstrates thread safety using synchronized blocks.

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class BlockSynchronizationExample {
    private static final Logger logger =
LogManager.getLogger(BlockSynchronizationExample.class);

    private int counter = 0;

    public static void main(String[] args) {
        BlockSynchronizationExample example = new BlockSynchronizationExample();
        example.runThreads();
    }

    public void runThreads() {
        Thread thread1 = new Thread(this::incrementCounter);
        Thread thread2 = new Thread(this::incrementCounter);

        thread1.start();
        thread2.start();
    }

    public void incrementCounter() {
        for (int i = 0; i < 5; i++) {
            synchronized (this) {
                counter++;
                logger.info("Counter: {}", counter);
```

```
            }
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```

Output:

```
[Thread-0] INFO  com.example.javaconcepts.BlockSynchronizationExample - Counter: 1
[Thread-1] INFO  com.example.javaconcepts.BlockSynchronizationExample - Counter: 2
[Thread-0] INFO  com.example.javaconcepts.BlockSynchronizationExample - Counter: 3
[Thread-1] INFO  com.example.javaconcepts.BlockSynchronizationExample - Counter: 4
[Thread-0] INFO  com.example.javaconcepts.BlockSynchronizationExample - Counter: 5
[Thread-1] INFO  com.example.javaconcepts.BlockSynchronizationExample - Counter: 6
[Thread-0] INFO  com.example.javaconcepts.BlockSynchronizationExample - Counter: 7
[Thread-1] INFO  com.example.javaconcepts.BlockSynchronizationExample - Counter: 8
```

19.Final Keyword

- **Concepts**: Final Class, Final Method, Final Variable
- **Purpose**: Demonstrates the usage of the `final` keyword in different contexts.

FinalKeywordExample.java

Code:

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class FinalKeywordExample {
    private static final Logger logger = LogManager.getLogger(FinalKeywordExample.class);

    // Final instance variable
    private final int finalVariable = 100;

    public static void main(String[] args) {
        FinalKeywordExample example = new FinalKeywordExample();
        example.demoFinalVariable();
        example.demoFinalMethod();
    }
```

```
    public void demoFinalVariable() {
        logger.info("Final Variable: {}", finalVariable);
        // finalVariable = 200; // This will cause a compilation error
    }

    // Final method
    public final void demoFinalMethod() {
        logger.info("This is a final method and cannot be overridden.");
    }
}

// Final class
final class FinalClass {
    // Class implementation
}

// The following class definition will cause a compilation error
// because FinalClass cannot be subclassed
// class SubClass extends FinalClass {}
```

Output:

```
[main] INFO  com.example.javaconcepts.FinalKeywordExample - Final Variable: 100
[main] INFO  com.example.javaconcepts.FinalKeywordExample - This is a final method and cannot be overridden.
```

20.Polymorphism

- **Concepts**: Method Overriding, Upcasting, and Dynamic Method Dispatch
- **Purpose**: Demonstrates polymorphism through method overriding and dynamic method dispatch.

PolymorphismExample.java

Code:

```
package com.example.javaconcepts;

public class PolymorphismExample {
    public static void main(String[] args) {
        Bike bike1 = new MountainBike();
        Bike bike2 = new RoadBike();

        bike1.ride(); // Outputs: Mountain bike is riding on rough terrain
```

```
        bike2.ride(); // Outputs: Road bike is riding on smooth roads
    }
}
```

Bike.java

<u>Code:</u>

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Bike {
    private static final Logger logger = LogManager.getLogger(Bike.class);

    public void ride() {
        logger.info("Generic bike is riding");
    }
}
```

MountainBike.java

<u>Code:</u>

```
package com.example.javaconcepts;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class MountainBike extends Bike {
    private static final Logger logger = LogManager.getLogger(MountainBike.class);

    @Override
    public void ride() {
        logger.info("Mountain bike is riding on rough terrain");
    }
}
```

RoadBike.java

<u>Code:</u>

```
package com.example.javaconcepts;
```

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class RoadBike extends Bike {
    private static final Logger logger = LogManager.getLogger(RoadBike.class);

    @Override
    public void ride() {
        logger.info("Road bike is riding on smooth roads");
    }
}
```

Output:

```
[main] INFO  com.example.javaconcepts.MountainBike - Mountain bike is riding on rough terrain
[main] INFO  com.example.javaconcepts.RoadBike - Road bike is riding on smooth roads
```

JSON Execution

Code:

```
package com.example.json;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

public class JSONParsingExample {
    public static void main(String[] args) {
        String jsonString = "{\"name\": \"John\", \"age\": 30}";

        JSONParser parser = new JSONParser();
        try {
            JSONObject json = (JSONObject) parser.parse(jsonString);
            String name = (String) json.get("name");
            long age = (Long) json.get("age");

            System.out.println("Name: " + name);
            System.out.println("Age: " + age);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```java
package com.example.json;

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

public class JSONParsingExample {
    public static void main(String[] args) {
        String jsonString = "{\"name\": \"John\", \"age\": 30}";

        JSONParser parser = new JSONParser();
        try {
            JSONObject json = (JSONObject) parser.parse(jsonString);
            String name = (String) json.get("name");
            long age = (Long) json.get("age");

            System.out.println("Name: " + name);
            System.out.println("Age: " + age);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
}
```

Problems Javadoc Declaration Console Servers

```
<terminated> JSONParsingExample [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe
Name: John
Age: 30
```

XML Execution:

Code:

```java
package com.example.xml;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

public class XMLParsingExample {
    public static void main(String[] args) {
        try {
            // Step 1: Create a DocumentBuilderFactory
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

            // Step 2: Create a DocumentBuilder
            DocumentBuilder builder = factory.newDocumentBuilder();
```

```
        // Step 3: Parse the XML file
        Document document = builder.parse("example.xml");

        // Optional: Normalize the XML structure (optional but recommended)
        document.getDocumentElement().normalize();

        // Step 4: Get the root element
        Element root = document.getDocumentElement();

        // Step 5: Get NodeList of 'employee' elements
        NodeList employeeList = root.getElementsByTagName("employee");

        // Step 6: Iterate over 'employee' elements
        for (int i = 0; i < employeeList.getLength(); i++) {
            Element employeeElement = (Element) employeeList.item(i);

            // Step 7: Get specific child elements by tag name
            String id = employeeElement.getElementsByTagName("id").item(0).getTextContent();
            String name =
employeeElement.getElementsByTagName("name").item(0).getTextContent();
            String salary =
employeeElement.getElementsByTagName("salary").item(0).getTextContent();

            // Step 8: Print employee details
            System.out.println("Employee ID: " + id);
            System.out.println("Employee Name: " + name);
            System.out.println("Employee Salary: " + salary);
            System.out.println("--------------------");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

Output:

```
Employee ID: 1
Employee Name: Cris James
Employee Salary: 50000
--------------------
Employee ID: 2
Employee Name: Adam Smith
Employee Salary: 60000
--------------------
```

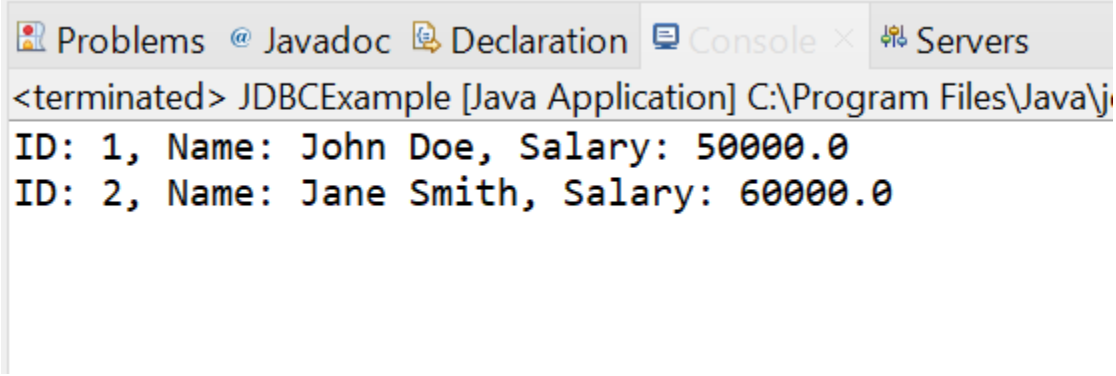
JDBC Execution

Code:

```
package com.example.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JDBCExample {
    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        String dbURL = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "root";
        String password = "Vin@@123";

        try {
            // Step 1: Register JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Step 2: Open a connection
            conn = DriverManager.getConnection(dbURL, username, password);

            // Step 3: Execute a query
            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT * FROM employees");

            // Step 4: Process the result set
            while (rs.next()) {
```

```
            int id = rs.getInt("id");
            String name = rs.getString("name");
            double salary = rs.getDouble("salary");

            System.out.println("ID: " + id + ", Name: " + name + ", Salary: " + salary);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // Step 5: Close resources
        try {
            if (rs != null) rs.close();
            if (stmt != null) stmt.close();
            if (conn != null) conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
  }
}
```

Output:



File Read Write Execution:

Code:

```
package com.example.fileio;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
```

```java
import java.util.Scanner;

public class FileReadWriteExample {

    private static final String FILE_PATH = "data.txt";

    public static void main(String[] args) {
        // Create file and write data
        createFileAndWriteData();

        // Read data from file
        readDataFromFile();

        // Update data in file
        updateDataInFile();

        // Read updated data from file
        readDataFromFile();

        // Delete data from file
        deleteFile();
    }

    // Method to create a new file and write data to it
    public static void createFileAndWriteData() {
        try {
            FileWriter writer = new FileWriter(FILE_PATH);
            writer.write("Hello, World!\n");
            writer.write("This is a test file.\n");
            writer.close();
            System.out.println("File created and data written successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Method to read data from the file
    public static void readDataFromFile() {
        try {
            File file = new File(FILE_PATH);
            Scanner scanner = new Scanner(file);
            System.out.println("Reading data from file:");
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
```

```
            System.out.println(line);
        }
        scanner.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// Method to update data in the file
public static void updateDataInFile() {
    try {
        FileWriter writer = new FileWriter(FILE_PATH, true); // true for append mode
        writer.write("Additional line added for update.\n");
        writer.close();
        System.out.println("Data updated successfully.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// Method to delete the file
public static void deleteFile() {
    File file = new File(FILE_PATH);
    if (file.delete()) {
        System.out.println("File deleted successfully.");
    } else {
        System.out.println("Failed to delete the file.");
    }
}
}
```

Output:

```
File created and data written successfully.
Reading data from file:
Hello, World!
This is a test file.
Data updated successfully.
Reading data from file:
Hello, World!
This is a test file.
Additional line added for update.
File deleted successfully.
```