```
/**
 * DailyOperationsDashboard.java
 *
 * A naive, hardcoded implementation of the Daily Operations Dashboard decision logic.
 * This class simulates how a developer might implement a decision table directly
 * with nested if-else statements.
 *
 *
 */

import java.util.ArrayList;
import java.util.List;

public class DailyOperationsDashboard {

   /**
    * Evaluates project status and determines which actions to trigger
    * based on the decision table rules.
    *
    * @param projectActive      Whether the project is active
    * @param taskOverdue        Whether any task is overdue
    * @param kpiBreach          Whether there is a KPI breach
    * @param dependencyBlocked  Whether any dependency is blocked
    * @return List of actions to perform
    */
   public static List<String> processProjectDecision(boolean projectActive, boolean
taskOverdue,
                                      boolean kpiBreach, boolean dependencyBlocked) {

      List<String> actions = new ArrayList<>();

      // Rule 4: Project not active
      if (!projectActive) {
         actions.add("Log Event & Notify Project Owner");
         return actions;
      }

      // Rule 1: KPI Breach (highest priority)
      if (kpiBreach) {
         actions.add("Flag Project as AtRisk");
         actions.add("Notify Manager");
         actions.add("Escalate to Executive");
         return actions;
      }
```

```
    // Rule 2: Operational Issues (Task Overdue or Dependency Blocked)
    if ((taskOverdue && !kpiBreach) || (dependencyBlocked && !kpiBreach)) {
        actions.add("Notify Manager");

        if (taskOverdue) {
            actions.add("Send Reminder to Assignee(s)");
        }

        if (dependencyBlocked) {
            actions.add("Create Dependency Alert");
        }

        return actions;
    }

    // Rule 3: Normal Execution (Active, no issues)
    if (projectActive && !taskOverdue && !kpiBreach && !dependencyBlocked) {
        actions.add("No Action – Continue Monitoring");
        return actions;
    }

    return actions; // Fallback (should never reach)
}


/**
 * Demonstration of the naive decision logic.
 */
public static void main(String[] args) {
    List<TestCase> testCases = List.of(
        new TestCase("Active with KPI Breach", true, false, true, false),
        new TestCase("Active with Task Overdue", true, true, false, false),
        new TestCase("Active with Dependency Blocked", true, false, false, true),
        new TestCase("Active and Nominal", true, false, false, false),
        new TestCase("Project Not Active", false, true, false, true)
    );

    for (TestCase test : testCases) {
        List<String> result = processProjectDecision(
                test.projectActive,
                test.taskOverdue,
                test.kpiBreach,
                test.dependencyBlocked
```

```
            );

            System.out.println("\n" + test.description + ":");
            System.out.println(" -> " + result);
        }
    }

    /**
     * Simple record-like structure to group test data.
     */
    static class TestCase {
        String description;
        boolean projectActive;
        boolean taskOverdue;
        boolean kpiBreach;
        boolean dependencyBlocked;

        TestCase(String description, boolean projectActive, boolean taskOverdue,
                boolean kpiBreach, boolean dependencyBlocked) {
            this.description = description;
            this.projectActive = projectActive;
            this.taskOverdue = taskOverdue;
            this.kpiBreach = kpiBreach;
            this.dependencyBlocked = dependencyBlocked;
        }
    }
}
```