For the given function, do the following:

1. create a graph (full drawing)
2. find all du-paths that satisfy the criteria for All-Du-Paths Coverage
3. find the minimal test set that achieves Prime Path Coverage and create real Junit tests

\* Analyzes an array of student grades.

\* Returns a summary message based on:

\* - Number of passing grades (>= 50)

\* - If any grade is invalid (< 0 or > 100)

\* - If all students passed

\* @param grades an array of integers representing student grades

\* @return summary message

public static String analyzeGrades(int[] grades) {

boolean hasInvalid = false;

int passCount = 0;

for (int i = 0; i < grades.length; i++) {

int grade = grades[i];

if (grade < 0 || grade > 100) {

hasInvalid = true;

} else if (grade >= 50) {

passCount++;

}

}

if (hasInvalid) {

return "Invalid grades detected.";

} else if (passCount == grades.length && grades.length > 0) {

return "All students passed.";

} else if (passCount == 0) {

return "No students passed.";

} else {

return "Some students passed.";

}

}

1. **Control Flow Graph**

A computer screen shot of a diagram

AI-generated content may be incorrect.

Variables:

* grades
* i
* passCount
* hasInvalid
* grade

1. **Find all du-paths that satisfy the criteria for All-Du-Paths Coverage**

**(Data Flow Graph Coverage)**

DU-Paths => grades

* def: 1
* use: (4), (3, 4), (3, 11), (13, 14), (13, 15)
  + DU-Paths
    - [1, 2, 3, 4]
    - [1, 2, 3, 11]
    - [1, 2, 3, 11, 13, 14]
    - [1, 2, 3, 11, 13, 15]
  + All DU-Path Coverage
    - [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 12]
    - [1, 2, 3, 11, 12]
    - [1, 2, 3, 11, 13, 14]
    - [1, 2, 3, 11, 13, 15, 16]

DU-Paths => i

* def: 2, 10
* use: (10), (3, 4), (3, 11)
  + DU-Paths
    - [2, 3, 4, 5, 6, 10]
    - [2, 3, 4, 5, 7, 8, 10]
    - [2, 3, 4, 5, 7, 9, 10]
    - [2, 3, 4]
    - [2, 3, 11]
    - нема потреба:
    - [10, 2, 3, 4, 5, 6, 10]
    - [10, 2, 3, 4, 5, 7, 8, 10]
    - [10, 2, 3, 4, 5, 7, 9, 10]
    - [10, 3, 4]
    - [10, 3, 11]
  + All DU-Path Coverage
    - [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 12] => покриено од 4-тото по ред
    - [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 12]
    - [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 12]
    - [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 12]
    - [1, 2, 3, 11, 12]

DU-Paths => passCount

* def: 2, 8
* use: (8), (13, 14), (13, 15), (15, 16), (15, 17)
  + DU-Paths
    - [2, 3, 4, 5, 7, 8]
    - [2, 3, 11, 13, 14]
    - [2, 3, 11, 13, 15]
    - [2, 3, 11, 13, 15, 16]
    - [2, 3, 11, 13, 15]
  + All DU-Path Coverage
    - [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 12]
    - [1, 2, 3, 11, 13, 14]
    - [1, 2, 3, 11, 13, 15, 16] => покриено од 4-тото по ред
    - [1, 2, 3, 11, 13, 15, 16]
    - [1, 2, 3, 11, 13, 15, 17]

DU-Paths => hasInvalid

* def: 2, 6
* use: (11, 12), (11, 13)
  + DU-Paths
    - [2, 3, 11, 12]
    - [2, 3, 11, 13]
    - [6, 10, 2, 3, 11, 12]
    - [6, 10, 2, 3, 11, 13]
  + All DU-Path Coverage
    - [1, 2, 3, 11, 12]
    - [1, 2, 3, 11, 13, 14]
    - [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 12]
    - [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 13]

DU-Paths => grade

* def: 4
* use: (5, 6), (5, 7), (7, 8), (7, 9)
  + DU-Paths
    - [4, 5, 6]
    - [4, 5, 7]
    - [4, 5, 7, 8]
    - [4, 5, 7, 9]
  + All DU-Path Coverage
    - [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 12]
    - [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 12] => покриено од 3-тото по ред
    - [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 12]
    - [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 12]

1. **Find the minimal test set that achieves Prime Path Coverage**

**and create real JUnit tests**

**(Graph Coverage)**

Invalid grades detected.

* [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 12]
* [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 12]
* [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 12]
* [1, 2, 3, 11, 12]

All students passed.

* [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 13, 14]
* [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 13, 14]
* [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 13, 14]
* [1, 2, 3, 11, 13, 14]

No students passed.

* [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 13, 15, 16]
* [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 13, 15, 16]
* [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 13, 15, 16]
* [1, 2, 3, 11, 13, 15, 16]

Some students passed.

* [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 13, 15, 17]
* [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 13, 15, 17]
* [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 13, 15,17]
* [1, 2, 3, 11, 13, 15, 17]

**JUnit Tests**

package com.example.laboratoryexercises.JUnit.LaboratoryExercies02;  
  
import org.junit.Test;  
  
import static org.junit.Assert.*assertEquals*;  
  
public class GradeAnalyzerTest {  
  
 *// ---- "Invalid grades detected." paths ----* @Test  
 public void testSingleInvalidGrade() {  
 *// Path: [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 12]  
 // A path where an invalid grade is found during iteration* int[] grades = {-5};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("Invalid grades detected.", result);  
 }  
  
 @Test  
 public void testPassingThenInvalid() {  
 *// Path: [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 12]  
 // Path where we have valid passing grades before finding an invalid one* int[] grades = {75, 101};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("Invalid grades detected.", result);  
 }  
  
 @Test  
 public void testFailingThenInvalid() {  
 *// Path: [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 12]  
 // Path where we have valid failing grades before finding an invalid one* int[] grades = {45, 101};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("Invalid grades detected.", result);  
 }  
  
 *// Note: Path [1, 2, 3, 11, 12] with empty array actually leads to "No students passed"  
  
 // ---- "All students passed." paths ----* @Test  
 public void testAllStudentsPassed() {  
 *// Path: [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 13, 14]  
 // Path where all grades are passing (≥50)* int[] grades = {50, 75, 90};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("All students passed.", result);  
 }  
  
 @Test  
 public void testSingleStudentPassed() {  
 *// Similar to Path: [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 13, 14] but with one iteration  
 // Single passing grade* int[] grades = {75};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("All students passed.", result);  
 }  
  
 *// Note: Paths [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 13, 14],  
 // [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 13, 14], and [1, 2, 3, 11, 13, 14]  
 // are logically impossible for "All students passed"  
  
 // ---- "No students passed." paths ----* @Test  
 public void testAllStudentsFailed() {  
 *// Path: [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 13, 15, 16]  
 // Path where all grades are failing (<50)* int[] grades = {30, 45, 49};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("No students passed.", result);  
 }  
  
 @Test  
 public void testEmptyArray() {  
 *// Path: [1, 2, 3, 11, 13, 15, 16]  
 // Path for an empty array where no students passed* int[] grades = {};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("No students passed.", result);  
 }  
  
 @Test  
 public void testSingleStudentFailed() {  
 *// Similar to Path: [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 13, 15, 16] but with one iteration  
 // Single failing grade* int[] grades = {45};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("No students passed.", result);  
 }  
  
 *// Note: Paths [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 13, 15, 16] and  
 // [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 13, 15, 16] are logically impossible for "No students passed"  
  
 // ---- "Some students passed." paths ----* @Test  
 public void testSomeStudentsPassedPassingFirst() {  
 *// Path: [1, 2, 3, 4, 5, 7, 8, 10, 2, 3, 11, 13, 15, 17]  
 // Path with mix of passing and failing grades (some passing)* int[] grades = {75, 45};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("Some students passed.", result);  
 }  
  
 @Test  
 public void testSomeStudentsPassedFailingFirst() {  
 *// Path: [1, 2, 3, 4, 5, 7, 9, 10, 2, 3, 11, 13, 15, 17]  
 // Path with mix of passing and failing grades (first failing)* int[] grades = {45, 75};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("Some students passed.", result);  
 }  
  
 @Test  
 public void testMultipleMixedGrades() {  
 *// Combined path with multiple iterations through both passing and failing branches* int[] grades = {45, 75, 30, 80, 49};  
 String result = GradeAnalyzer.*analyzeGrades*(grades);  
 *assertEquals*("Some students passed.", result);  
 }  
  
 *// Note: Paths [1, 2, 3, 4, 5, 6, 10, 2, 3, 11, 13, 15, 17] and  
 // [1, 2, 3, 11, 13, 15, 17] are logically impossible for "Some students passed"*}