|  |
| --- |
|  |
| Prueba de Caja Blanca |
| ***“Gestión de información académica InClass”*** |
|  |
|  |
| **Integrantes:**  Stephen Drouet  Bryan Morales  Alejandro Sarmiento  Jairo Quilumbaquin  **Fecha 18/02/2024** |

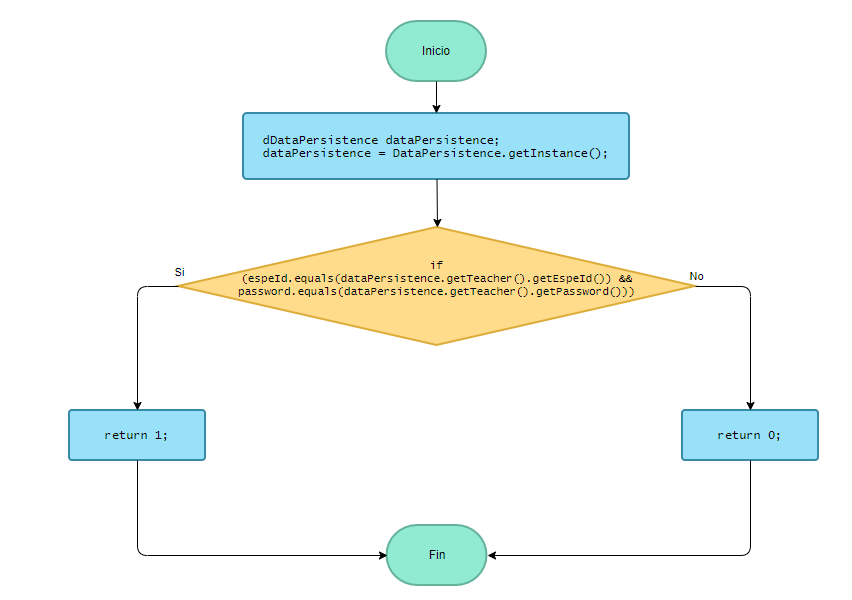
**BUSCAR PROBLEMAS**

**Prueba caja blanca login**

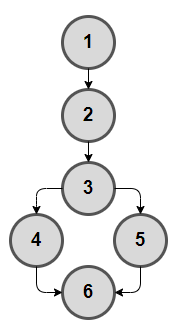
**CÓDIGO FUENTE**

|  |
| --- |
| public static int loginTeacher(String espeId, String password) {  DataPersistence dataPersistence;  dataPersistence = DataPersistence.getInstance();  if (espeId.equals(dataPersistence.getTeacher().getEspeId()) && password.equals(dataPersistence.getTeacher().getPassword())) {  return 1;  } else {  return 0;  }  } |

**DIAGRAMA DE FLUJO**

****

**Grafo**

****

**RUTAS**

**R1:** 1, 2, 3, 4, 6

**R2:** 1, 2, 3, 5, 6

**COMPLEJIDAD CICLOMÁTICA**

Se puede calcular de las siguientes formas:

* V(G) = número de nodos predicados(decisiones)+1

V(G)=1+1=2

* V(G) = A – N + 2

V(G)= 6 – 6 + 2 = 2

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

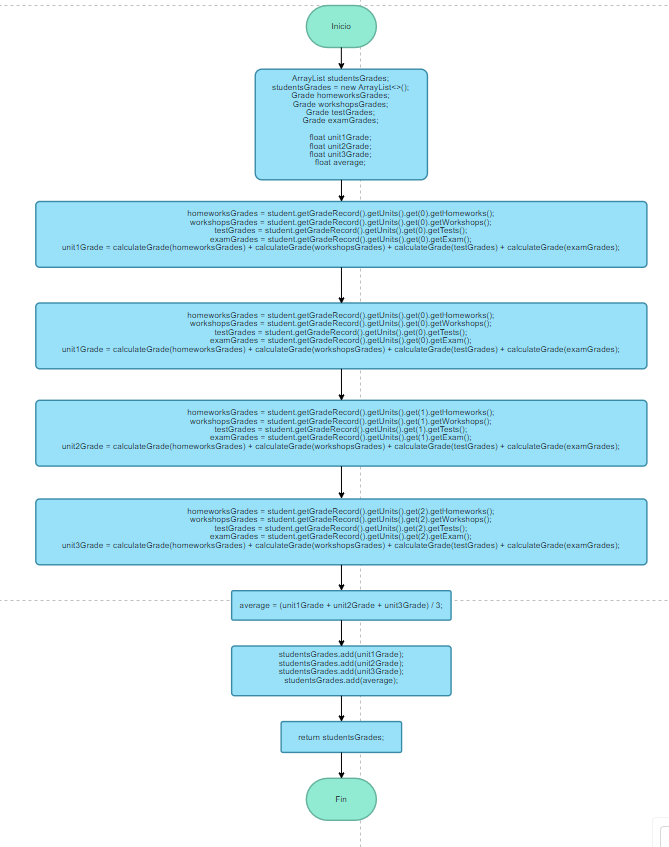
**N:** Número de nodos

**Prueba caja blanca getGrades por unidad**

**CÓDIGO FUENTE**

|  |
| --- |
| public static ArrayList<Float> getGradesUnit(Student student) {  ArrayList<Float> studentsGrades;  studentsGrades = new ArrayList<>();  Grade homeworksGrades;  Grade workshopsGrades;  Grade testGrades;  Grade examGrades;  float unit1Grade;  float unit2Grade;  float unit3Grade;  float average;  homeworksGrades = student.getGradeRecord().getUnits().get(0).getHomeworks();  workshopsGrades = student.getGradeRecord().getUnits().get(0).getWorkshops();  testGrades = student.getGradeRecord().getUnits().get(0).getTests();  examGrades = student.getGradeRecord().getUnits().get(0).getExam();  unit1Grade = calculateGrade(homeworksGrades) + calculateGrade(workshopsGrades) + calculateGrade(testGrades) + calculateGrade(examGrades);  homeworksGrades = student.getGradeRecord().getUnits().get(1).getHomeworks();  workshopsGrades = student.getGradeRecord().getUnits().get(1).getWorkshops();  testGrades = student.getGradeRecord().getUnits().get(1).getTests();  examGrades = student.getGradeRecord().getUnits().get(1).getExam();  unit2Grade = calculateGrade(homeworksGrades) + calculateGrade(workshopsGrades) + calculateGrade(testGrades) + calculateGrade(examGrades);  homeworksGrades = student.getGradeRecord().getUnits().get(2).getHomeworks();  workshopsGrades = student.getGradeRecord().getUnits().get(2).getWorkshops();  testGrades = student.getGradeRecord().getUnits().get(2).getTests();  examGrades = student.getGradeRecord().getUnits().get(2).getExam();  unit3Grade = calculateGrade(homeworksGrades) + calculateGrade(workshopsGrades) + calculateGrade(testGrades) + calculateGrade(examGrades);  average = (unit1Grade + unit2Grade + unit3Grade) / 3;  studentsGrades.add(unit1Grade);  studentsGrades.add(unit2Grade);  studentsGrades.add(unit3Grade);  studentsGrades.add(average);  return studentsGrades;  } |

**DIAGRAMA DE FLUJO**

****

**GRAFO**

**Imagen que contiene Forma

Descripción generada automáticamente**

**RUTAS**

**R1:** 1, 2, 3, 4,5,6,7,8,9,10

**COMPLEJIDAD CICLOMÁTICA**

Se puede calcular de las siguientes formas:

* V(G) = número de nodos predicados(decisiones)+1

V(G)=0+1=1

* V(G) = A – N + 2

V(G)= 9– 10 + 2 = 1

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

**N:** Número de nodos

**Prueba caja blanca calcular porcentaje de asistencias**

**CÓDIGO FUENTE**

|  |
| --- |
| public static float calculateAssistancePersetn(Student student) {  float assistancePersent;  int assistances = 0;  student.getAttendanceRecord().setTotalClassNumber(student. getAttendanceRecord().getAttendance().size());  for (Boolean dayAssistance : student.getAttendanceRecord(). getAttendance()) {  if (dayAssistance) {  assistances++;  }  }  assistancePersent = (float) assistances / (float) student.getAttendanceRecord().getTotalClassNumber();  assistancePersent = Math.round(assistancePersent \* 10000) / 100;  return assistancePersent;  } |

**DIAGRAMA DE FLUJO**

**Diagrama

Descripción generada automáticamente**

**GRAFO**

**Un dibujo de un reloj

Descripción generada automáticamente con confianza media**

**RUTAS**

**R1:** 1, 2, 3, 4, 7, 8

**R2:** 1, 2, 3, 4, 5, 4, 7, 8

**R3:** 1, 2, 3, 4, 5, 6, 4, 7, 8

**COMPLEJIDAD CICLOMÁTICA**

Se puede calcular de las siguientes formas:

* V(G) = número de nodos predicados(decisiones)+1

V(G)=2+1=3

* V(G) = A – N + 2

V(G)= 9– 8 + 2 = 3

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

**N:** Número de nodos

**Prueba caja blanca verificar si desaprobo por asistencias**

**CÓDIGO FUENTE**

|  |
| --- |
| public static boolean verifyAttendancePesent(float assistancePersent) {  boolean isAproved = true;  if (assistancePersent < 70) {  isAproved = false;  }  return isAproved;  } |

**DIAGRAMA DE FLUJO**

**Diagrama

Descripción generada automáticamente**

**GRAFO**

**Dibujo en blanco y negro de un reloj

Descripción generada automáticamente con confianza media**

**RUTAS**

**R1:** 1, 2, 3, 5, 6

**R2:** 1, 2, 3, 4, 5, 6

**COMPLEJIDAD CICLOMÁTICA**

Se puede calcular de las siguientes formas:

* V(G) = número de nodos predicados(decisiones)+1

V(G)=1+1=2

* V(G) = A – N + 2

V(G)= 6– 6 + 2 = 2

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

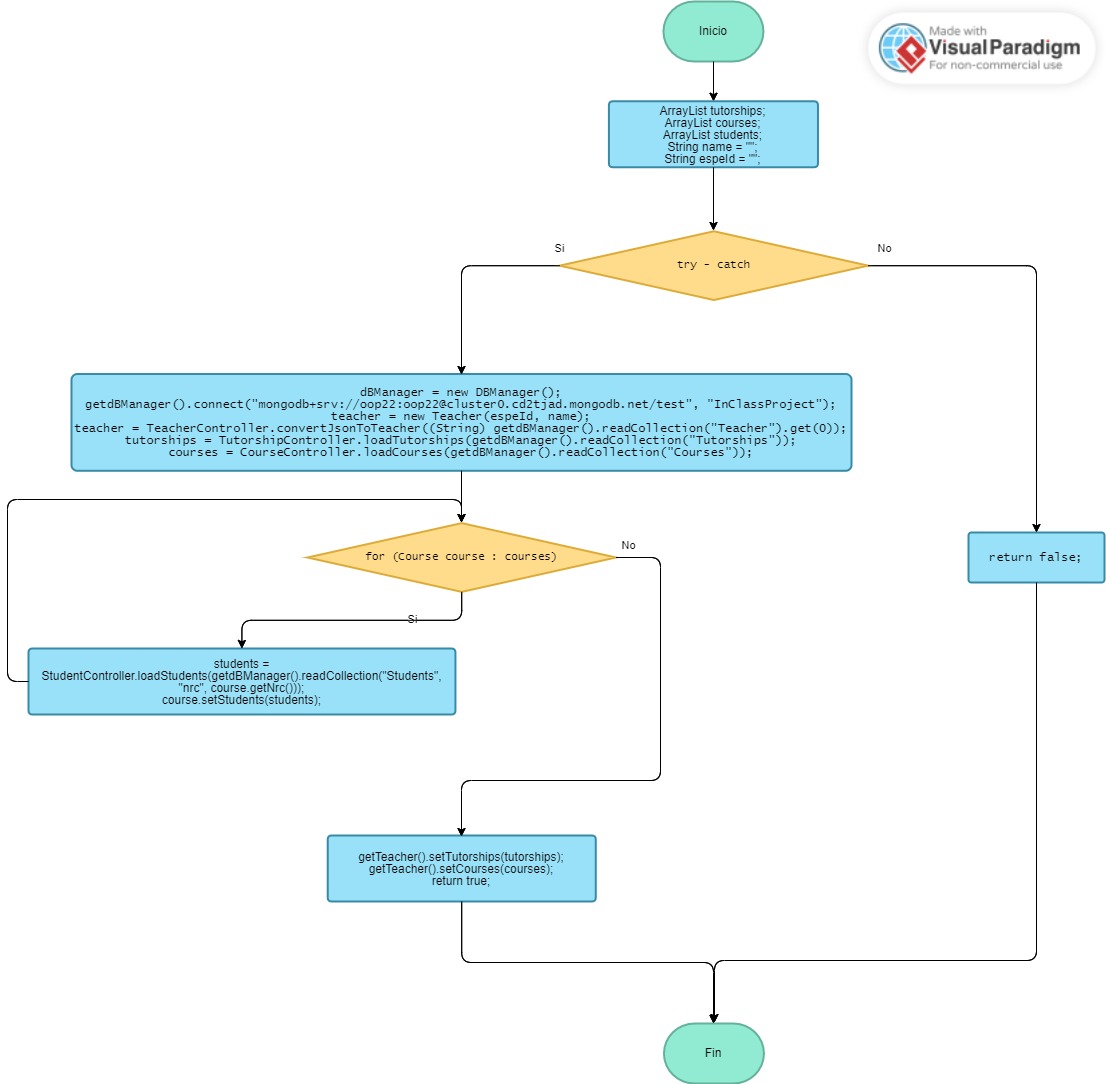
**N:** Número de nodos

**Prueba caja blanca conectar a mongoDB**

**CÓDIGO FUENTE**

|  |
| --- |
| public boolean connectMongoDB() {  ArrayList<Tutorship> tutorships;  ArrayList<Course> courses;  ArrayList<Student> students;  String name = "";  String espeId = "";  try {  dBManager = new DBManager();  getdBManager().connect("mongodb+srv://oop22:oop22@cluster0.cd2tjad.mongodb.net/test", "InClassProject");  teacher = new Teacher(espeId, name);  teacher = TeacherController.convertJsonToTeacher((String) getdBManager().readCollection("Teacher").get(0));  tutorships = TutorshipController.loadTutorships(getdBManager().readCollection("Tutorships"));  courses = CourseController.loadCourses(getdBManager().readCollection("Courses"));  for (Course course : courses) {  students = StudentController.loadStudents(getdBManager().readCollection("Students", "nrc", course.getNrc()));  course.setStudents(students);  }  getTeacher().setTutorships(tutorships);  getTeacher().setCourses(courses);  return true;  } catch (Exception e) {  return false;  }  } |

**DIAGRAMA DE FLUJO**

****

**GRAFO**

**Un dibujo de un reloj

Descripción generada automáticamente con confianza media**

**RUTAS**

**R1:** 1, 2, 3, 5, 9

**R2:** 1, 2, 3, 4, 6, 8, 9

**R3:** 1, 2, 3, 4, 6, 7, 6, 8, 9

**COMPLEJIDAD CICLOMÁTICA**

Se puede calcular de las siguientes formas:

* V(G) = número de nodos predicados(decisiones)+1

V(G)=2+1=3

* V(G) = A – N + 2

V(G)= 10– 9 + 2 = 3

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

**N:** Número de nodos

**Prueba caja blanca conectar buscar cursos**

**CÓDIGO FUENTE**

|  |
| --- |
| public static int findCourse(ArrayList<Course> courses, int nrc) {  int index = -1;    if (courses == null) {  return -1;  }  for (int i = 0; i < courses.size(); i++) {  if (courses.get(i).getNrc() == nrc) {  index = i;  return index;  }  }  return index;  } |

**DIAGRAMA DE FLUJO**

**Diagrama

Descripción generada automáticamente**

**GRAFO**

**Un dibujo de un reloj

Descripción generada automáticamente con confianza media**

**RUTAS**

**R1:** 1, 2, 3, 5, 9

**R2:** 1, 2, 3, 4, 8, 9

**R3:** 1, 2, 3, 4, 6, 7, 8, 9

**R4:** 1, 2, 3, 4, 6, 4, 8, 9

**COMPLEJIDAD CICLOMÁTICA**

Se puede calcular de las siguientes formas:

* V(G) = número de nodos predicados(decisiones)+1

V(G)=3+1=4

* V(G) = A – N + 2

V(G)= 11– 9 + 2 = 4

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

**N:** Número de nodos