White Box Testing

Here are test cases for each function in the SubjectGrades class that cover 100% of the following:

1- Statement Coverage
2- Branch Coverage
3- Condition Coverage (when possible)

Function: **setSubjectName()**

test cases (“”) and (“someSubject”) covers 100% Statement Coverage and 100% Branch Coverage.

```
public class SubjectGradesWhiteBoxTest {
    34 usages
    private SubjectGrades subject;

    @BeforeEach()
    void setUp() {
        subject = new SubjectGrades();
    }

    @Test
    void setSubjectName() {
        //----------------Statement Coverage = 100%---------------------
        //------------------Branch Coverage = 100%----------------------
        assertThrows(IllegalArgumentException.class, () -> {
            subject.setSubjectName("");
        }, message: "Shouldn't be empty");
        subject.setSubjectName("someSubject");
        assertEquals( expected: "someSubject", subject.getSubjectName());
    }
```

Similarly in all the functions

```java
@Test
void setSubjectCode() {
    //----------------Statement Coverage = 100%---------------------
    //-----------------Branch Coverage = 100%----------------------
    //----------------Condition Coverage = 100%---------------------
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setSubjectCode("abc12345");
    });
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setSubjectCode("123abc");
    });
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setSubjectCode("abc1d23");
    });
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setSubjectCode("abc123d");
    });
    subject.setSubjectCode("abc123");
    assertEquals( expected: "abc123", subject.getSubjectCode());
    subject.setSubjectCode("abc123s");
    assertEquals( expected: "abc123s", subject.getSubjectCode());
}
```

```java
@Test
void setStudentName() {
    //----------------Statement Coverage = 100%---------------------
    //-----------------Branch Coverage = 100%----------------------
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setStudentName("");
    }, message: "Shouldn't be empty");
    subject.setStudentName("someSubject");
    assertEquals( expected: "someSubject", subject.getStudentName());
}
```

```java
@Test
void setStudentID() {
    //----------------Statement Coverage = 100%---------------------
    //------------------Branch Coverage = 100%---------------------
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setStudentID("abc12345");
    });
    subject.setStudentID("12345678");
    assertEquals( expected: "12345678", subject.getStudentID());
}

@Test
void setOralMark() {
    //----------------Statement Coverage = 100%---------------------
    //------------------Branch Coverage = 100%---------------------
    //----------------Condition Coverage = 100%---------------------
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setOralMark(11);
    });
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setOralMark(-1);
    });
    subject.setOralMark(5);
    assertEquals( expected: 5, subject.getOralMark());
}
```

```java
@Test
void setMidterm() {
    //----------------Statement Coverage = 100%---------------------
    //-----------------Branch Coverage = 100%----------------------
    //----------------Condition Coverage = 100%---------------------
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setMidterm(21);
    });
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setMidterm(-1);
    });
    subject.setMidterm(10);
    assertEquals( expected: 10, subject.getMidterm());
}

@Test
void setFinal() {
    //----------------Statement Coverage = 100%---------------------
    //-----------------Branch Coverage = 100%----------------------
    //----------------Condition Coverage = 100%---------------------
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setFinal(61);
    });
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setFinal(-1);
    });
    subject.setFinal(30);
    assertEquals( expected: 30, subject.getFinal());
}
```

```java
@Test
void setActivities() {
    //----------------Statement Coverage = 100%--------------------
    //-----------------Branch Coverage = 100%---------------------
    //----------------Condition Coverage = 100%--------------------
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setActivities(11);
    });
    assertThrows(IllegalArgumentException.class, () -> {
        subject.setActivities(-1);
    });
    subject.setActivities(5);
    assertEquals( expected: 5, subject.getActivities());
}
```

```java
@Test
void calculateGrade() {
    //----------------Statement Coverage = 100%--------------------
    //-----------------Branch Coverage = 100%---------------------
    //----------------Condition Coverage = 100%--------------------
    assertThrows(IllegalArgumentException.class, () -> {
        SubjectGrades.calculateGrade( mark: -1);
    });
    assertThrows(IllegalArgumentException.class, () -> {
        SubjectGrades.calculateGrade( mark: 101);
    });
    assertEquals( expected: "A+", SubjectGrades.calculateGrade( mark: 98));
    assertEquals( expected: "A", SubjectGrades.calculateGrade( mark: 95));
    assertEquals( expected: "A-", SubjectGrades.calculateGrade( mark: 91));
    assertEquals( expected: "B+", SubjectGrades.calculateGrade( mark: 86));
    assertEquals( expected: "B", SubjectGrades.calculateGrade( mark: 82));
    assertEquals( expected: "B-", SubjectGrades.calculateGrade( mark: 78));
    assertEquals( expected: "C+", SubjectGrades.calculateGrade( mark: 75));
    assertEquals( expected: "C", SubjectGrades.calculateGrade( mark: 72));
    assertEquals( expected: "C-", SubjectGrades.calculateGrade( mark: 68));
    assertEquals( expected: "D+", SubjectGrades.calculateGrade( mark: 66));
    assertEquals( expected: "D", SubjectGrades.calculateGrade( mark: 62));
    assertEquals( expected: "F", SubjectGrades.calculateGrade( mark: 20));
}
```

```java
@Test
void calculateGPA() {
    //----------------Statement Coverage = 100%--------------------
    //------------------Branch Coverage = 100%----------------------
    //----------------Condition Coverage = 100%--------------------
    assertThrows(IllegalArgumentException.class, () -> {
        SubjectGrades.calculateGPA( mark: -1);
    });
    assertThrows(IllegalArgumentException.class, () -> {
        SubjectGrades.calculateGPA( mark: 101);
    });
    assertEquals( expected: 4, SubjectGrades.calculateGPA( mark: 98));
    assertEquals( expected: 4, SubjectGrades.calculateGPA( mark: 95));
    assertEquals( expected: 3.7, SubjectGrades.calculateGPA( mark: 91));
    assertEquals( expected: 3.3, SubjectGrades.calculateGPA( mark: 86));
    assertEquals( expected: 3, SubjectGrades.calculateGPA( mark: 82));
    assertEquals( expected: 2.7, SubjectGrades.calculateGPA( mark: 78));
    assertEquals( expected: 2.3, SubjectGrades.calculateGPA( mark: 75));
    assertEquals( expected: 2, SubjectGrades.calculateGPA( mark: 72));
    assertEquals( expected: 1.7, SubjectGrades.calculateGPA( mark: 68));
    assertEquals( expected: 1.3, SubjectGrades.calculateGPA( mark: 66));
    assertEquals( expected: 1, SubjectGrades.calculateGPA( mark: 62));
    assertEquals( expected: 0, SubjectGrades.calculateGPA( mark: 20));
}
```

And all test cases passed successfully.