

JUNIT 5

1. Giới thiệu JUnit

JUnit là một framework viết unit test cho ngôn ngữ Java, có vai trò quan trọng trong phát triển các ứng dụng test-driven. Trong tài liệu này trình bày JUnit 5 tương thích các phiên bản Java 8 hoặc mới hơn. JUnit 5 bao gồm ba thành phần quan trọng sau:

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage
--

JUnit Platform định nghĩa TestEngine API để phát triển các framework kiểm thử. **JUnit Jupiter** cung cấp các mô hình lập trình mới để viết test case, bao gồm các annotation mới và hiện thực TestEngine để chạy test case được với các annotation này. **JUnit Vintage** hỗ trợ chạy các test case viết trong JUnit3 và JUnit 4 trên nền JUnit 5.

Để sử dụng JUnit 5 trên các project maven ta thêm dependency của Jupiter Engine.

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.7.0-M1</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-params</artifactId>
  <version>5.7.0-M1</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.platform</groupId>
  <artifactId>junit-platform-runner</artifactId>
  <version>1.7.0-M1</version>
  <scope>test</scope>
  <type>jar</type>
</dependency>
```

Thêm surefire plugin vào pom.xml, plugin này để thực thi unit test case trong quá trình kiểm thử ứng dụng.

<pre><build> <plugins> <plugin></pre>

```
<artifactId>
    maven-surefire-plugin
</artifactId>
    <version>2.22.2</version>
</plugin>
</plugins>
</build>
```

2. JUnit Annotation

JUnit 5 hỗ trợ các annotation để viết test case, các annotation thuộc gói `org.junit.jupiter.api` (**Error! Reference source not found.**).

Các annotation để viết test case trong JUnit5.

@BeforeEach Chạy trước mỗi phương thức test case.
@AfterEach Chạy sau mỗi phương thức test case.
@BeforeAll Chạy trước tất cả các phương thức test case, phương thức có annotation này phải là static.
@AfterAll Chạy sau tất cả các phương thức test case, phương thức có annotation này phải là static.
@Test Phương thức đóng vai trò test case.
@DisplayName Cung cấp tên hiển thị cho phương thức test case hoặc test class.
@Disable Bỏ qua một phương thức test case hoặc test class.
@Nested Dùng tạo các lớp test case lồng nhau.
@Tag Gán nhãn (tag) cho các phương thức test case hoặc test class để dễ tìm kiếm và lọc test case.
@TestFactory Đánh dấu phương thức là test factory cho kiểm thử động.

Ví dụ ta có lớp test như sau:

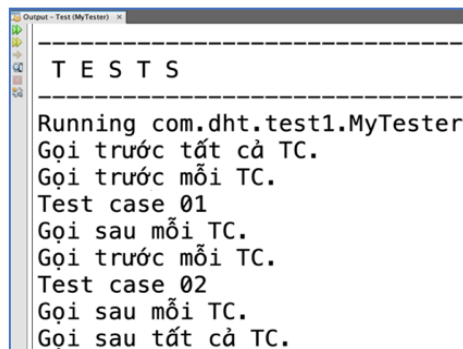
```
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
```

```

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class MyTester {
    @BeforeAll
    public static void setUpClass() {
        System.out.println("Gọi trước tất cả TC.");
    }
    @AfterAll
    public static void tearDownClass() {
        System.out.println("Gọi sau tất cả TC.");
    }
    @BeforeEach
    public void setUp() {
        System.out.println("Gọi trước mỗi TC.");
    }
    @AfterEach
    public void tearDown() {
        System.out.println("Gọi sau mỗi TC.");
    }
    @Test
    public void test1() {
        System.out.println("Test case 01");
    }
    @Test
    public void test2() {
        System.out.println("Test case 02");
    }
}

```



```

Output - Test (MyTester)
-----
TESTS
-----
Running com.dht.test1.MyTester
Gọi trước tất cả TC.
Gọi trước mỗi TC.
Test case 01
Gọi sau mỗi TC.
Gọi trước mỗi TC.
Test case 02
Gọi sau mỗi TC.
Gọi sau tất cả TC.

```

Minh họa sử dụng Test Annotation.

3. JUnit Assertion

Assertion dùng kiểm tra, đánh giá đầu ra mong muốn (Expected Output) và đầu ra thực sự (Actual Output) của test case. JUnit cung cấp các phương thức assertion là các phương thức tĩnh của lớp `org.junit.jupiter.Assertions`.

Các phương thức tĩnh của `org.junit.jupiter.Assertions`.

assertEquals() Kiểm tra hai giá trị bằng nhau.
assertNotEquals() Kiểm tra hai giá trị không bằng nhau.
assertFalse() Kiểm tra một biểu thức điều kiện là false.
assertTrue() Kiểm tra một biểu thức điều kiện là true.
assertNotNull() Kiểm tra một đối tượng khác null.
assertNull() Kiểm tra một đối tượng là null.
assertSame() Kiểm tra hai biến tham chiếu tham chiếu đến cùng đối tượng.
assertNotSame() Kiểm tra hai biến tham chiếu không tham chiếu cùng đối tượng.
assertThrows() Kiểm tra chương trình ném ra ngoại lệ mong muốn hay không.
assertArrayEquals() Kiểm tra hai mảng có bằng nhau không.
assertIterableEquals() Kiểm tra 2 iterable hoàn toàn bằng nhau, trong đó các phần tử cùng vị trí phải bằng nhau, và hai iterable không bắt buộc cùng kiểu dữ liệu Ví dụ kiểm tra hai iterable khác kiểu dữ liệu là <code>LinkedList</code> và <code>ArrayList</code> nếu có các phần tử bằng nhau tại các vị trí thì xem là bằng nhau theo phương thức này.
assertLinesMatch() Kiểm tra hai danh sách chuỗi khớp với nhau. Quá trình so khớp cặp chuỗi (expected, actual) trong hai danh sách tương ứng được thực hiện như sau: <ul style="list-style-type: none">- Kiểm tra nếu <code>expected.equals(actual)</code> là đúng thì so sánh cặp tiếp theo.- Ngược lại, <code>expected</code> được xem như biểu thức chính quy và kiểm tra bằng phương thức <code>matches()</code>. Nếu khớp thì so sánh cặp tiếp theo.- Ngược lại, kiểm tra <code>expected</code> là chuỗi fast-forward (là chuỗi bắt đầu và kết thúc

bằng >> và chứa ít nhất 4 ký tự)

assertTimeout() và assertTimeoutPreemptively()

Kiểm tra nhiệm vụ trong test case được thực thi trong khoảng thời gian (duration) cho phép.

Ví dụ ta có lớp phân số như sau cần viết các test case cho hai phương thức rút gọn và so sánh bằng hai phân số.

```
public class PhanSo {
    private int tuSo;
    private int mauSo;

    public PhanSo(int t, int m) {
        if (m == 0)
            throw new ArithmeticException("Mẫu#0");

        this.tuSo = t;
        this.mauSo = m;
    }

    public void rutGon() {
        int u = ucln(this.tuSo, this.mauSo);
        this.tuSo = this.tuSo / u;
        this.mauSo = this.mauSo / u;
    }

    public static int ucln(int a, int b) {
        while (a != b)
            if (a > b)
                a -= b;
            else
                b -= a;

        return a;
    }

    @Override
    public boolean equals(Object obj) {
        PhanSo p = (PhanSo) obj;
        int t1 = this.tuSo * p.mauSo;
        int t2 = this.mauSo * p.tuSo;

        return t1 - t2 == 0;
    }

    @Override
```

```

    public int hashCode() {
        int hash = 7;
        hash = 97 * hash + this.tuSo;
        hash = 97 * hash + this.mauSo;

        return hash;
    }
}

```

Thiết kế các test case kiểm tra phương thức tìm ước chung lớn nhất hai số nguyên.

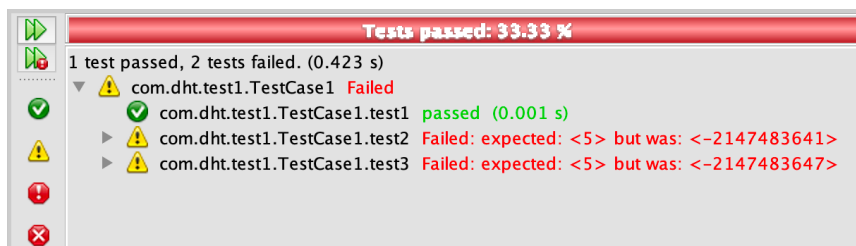
```

package com.dht.test1;

public class TestCase1 {
    @Test
    @Tag("important")
    public void test1() {
        assertEquals(5, PhanSo.ucln(15, 25));
    }
    @Test
    @Tag("important")
    public void test2() {
        assertEquals(5, PhanSo.ucln(-15, -25));
    }
    @Test
    public void test3() {
        assertEquals(5, PhanSo.ucln(15, -25));
    }
}

```

Khi thực thi ta sẽ thấy test2() và test3() sẽ bị failed do kết quả thật sự và kết quả mong muốn khác nhau (phương thức tìm ước chung lớn nhất có số nguyên âm chưa được xử lý).



Kết quả thực thi test case mẫu trên NetBeans IDE 11.2.

Thiết kế một số test case kiểm tra phương thức rút gọn phân số và so sánh hai phân số.

- Phương thức test1() kiểm tra ném ngoại lệ khi truyền mẫu số là 0.
- Phương thức test2() kiểm tra rút gọn phân số với tử số và mẫu số là số dương.
- Phương thức test3() và test4() kiểm tra 2 hai phân số bằng nhau.
- Phương thức test5() kiểm tra 2 mảng phân số bằng nhau.
- Phương thức test6() kiểm tra thực hiện rút gọn phân số thực hiện trong tối đa 2 giây.

```
package com.dht.test2;

public class TestCase2 {
    @BeforeAll
    public static void setUpClass() { }
    @AfterAll
    public static void tearDownClass() { }
    @BeforeEach
    public void setUp() { }
    @AfterEach
    public void tearDown() { }
    @Test
    @DisplayName("Kiểm tra ném ngoại lệ khi mẫu = 0")
    public void test1() {
        assertThrows(ArithmeticException.class, ()->{
            new PhanSo(2, 0);
        });
    }

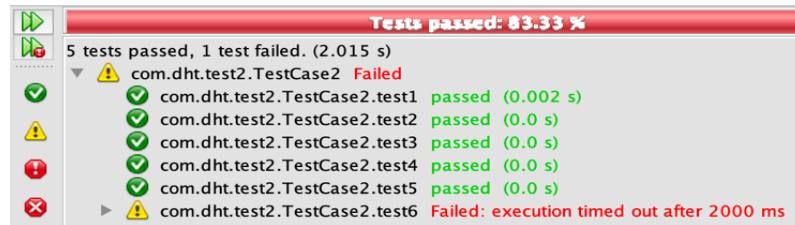
    @Test
    @Tag("important")
    @DisplayName("Kiểm tra rút gọn phân số")
    public void test2() {
        PhanSo p = new PhanSo(8, 36);
        p.rutGon();
        assertEquals(p.getTuSo(), 2);
        assertEquals(p.getMauSo(), 9);
    }
    @Test
    @Tag("important")
    @DisplayName("Kiểm tra hai phân số bằng nhau")
    public void test3() {
```

```

        PhanSo p1 = new PhanSo(-1, 2);
        PhanSo p2 = new PhanSo(4, -8);
        assertTrue(p1.equals(p2)) ;
    }
    @Test
    @DisplayName("Kiểm tra 2 phân số khác nhau")
    public void test4() {
        PhanSo p1 = new PhanSo(-1, 2);
        PhanSo p2 = new PhanSo(-4, -8);
        assertFalse(p1.equals(p2)) ;
    }
    @Test
    @DisplayName("Kiểm tra hai mảng phân số")
    public void test5() {
        PhanSo[] p1 = {new PhanSo(2, -4),
                        new PhanSo(8, 28)};
        PhanSo[] p2 = {new PhanSo(-1, 2),
                        new PhanSo(-2, -7)};
        assertArrayEquals(p1, p2);
    }
    @Test
    @Tag("important")
    @DisplayName("Lặp vô hạn khi tử hoặc mẫu âm")
    public void test6() {
        assertTimeoutPreemptively(
            Duration.ofSeconds(2), () -> {
                PhanSo p1 = new PhanSo(-8, 36);
                p1.rutGon();
            });
    }
}

```

Phương thức test case thứ 6 sẽ bị failed do truyền tử số là số âm, phương thức tìm ước chung lớn nhất hiện tại sẽ rơi vào vòng lặp vô hạn nên vượt quá thời gian timeout cho phép.



Hình Error! No text of specified style in document..1. Kết quả thực thi test case PhanSo.

Assumptions cung cấp các phương thức tĩnh giúp thực thi test dựa trên điều kiện nào đó, khi assumptions trả về kết quả kiểm tra thất bại thì ngoại lệ `TestAbortedException` sẽ được ném ra và phương thức test sẽ dừng lại. Các phương thức thông dụng:

- `assumeTrue()` kiểm tra giả định đưa ra là đúng, ngược lại test sẽ dừng lại.
- `assumeFalse()` kiểm tra giả định đưa ra là sai, ngược lại test sẽ dừng lại.

Ví dụ kiểm tra môi trường phát triển đang là DEV mới thực thi test case, ngược lại không thực thi test case.

```
public class TestCase2 {
    @Test
    public void test1() {
        String env = System.getProperty("ENV");
        Assumptions.assumeTrue("DEV".equals(env)) ;
        // phần thực thi test case
    }
}
```

4. JUnit Test Suite

Test Suite cho phép thực thi test case thuộc nhiều test class và nhiều package khác nhau. JUnit 5 cung cấp các annotation hỗ trợ thực thi test suite.

@SelectPackages

Chỉ định tên các gói được chọn chạy trong test suite thông qua `@RunWith(JUnitPlatform.class)`.

@SelectClasses

Chỉ định tên các lớp được chọn chạy trong test suite thông qua `@RunWith(JUnitPlatform.class)`.

@IncludePackages và **@ExcludePackages**

Annotation `@SelectPackages` sẽ tìm test class trong tất cả các gói chỉ định và các gói con của nó, nếu

<ul style="list-style-type: none"> - Nếu chỉ muốn sử dụng vài gói con trong gói chỉ định sử dụng <code>@IncludePackages</code>. - Nếu muốn bỏ qua vài gói con trong gói chỉ định sử dụng <code>@ExcludePackages</code>.
@IncludeClassNamePatterns và @ExcludeClassNamePatterns Chỉ định các lớp được sử dụng hoặc bỏ qua khi thực thi test suite bằng mẫu biểu thức chính quy.
@IncludeTags và @ExcludeTags Chỉ định các phương thức test case được sử dụng hoặc bỏ qua khi thực thi test suite sử dụng tag.

Ví dụ

```
import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.SelectClasses;
import org.junit.platform.suite.api.SelectPackages;
import org.junit.runner.RunWith;

@RunWith(JUnitPlatform.class)
@SelectPackages({"com.dht.test2", "com.dht.test3"})
@SelectClasses(com.dht.test1.TestCase1.class)
public class TestCase {

}
```

5. Parameterized Test

Parameterized Test cung cấp cơ chế thực thi một phương thức test nhiều lần với các tham số khác nhau.

Các cách thức truyền đối số cho phương thức test.

- **@ValueSource**: chỉ định mảng giá trị là danh sách các đối số lần lượt được truyền phương thức kiểm thử.
- **@MethodSource**: chỉ định các phương thức của lớp kiểm thử hoặc từ lớp ngoài, phương thức này phải là phương thức tĩnh (static).
- **@CsvSource**: chỉ định danh sách các phần tử, mỗi phần tử là chuỗi gồm nhiều giá trị cách nhau bằng dấu phẩy tương ứng là các đối số truyền vào phương thức kiểm thử.
- **@CsvFileSource**: đọc các đối số từ tập tin CSV, mỗi cột của từng dòng tương ứng

là danh sách các đối số truyền vào phương thức kiểm thử.

```
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import static
org.junit.jupiter.params.provider.Arguments.arguments;
import org.junit.jupiter.params.provider.CsvFileSource;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.MethodSource;
import org.junit.jupiter.params.provider.ValueSource;

public class TestCase3 {
    @ParameterizedTest
    @ValueSource(ints = {2, 3, 5, 7, 11, 13})
    public void testNguyenTo(int n) {
        assertTrue(Tester.ktNguyenTo(n));
    }
    @ParameterizedTest
    @ValueSource(ints = {1, 4, 6, 10, 15})
    public void testKhongNguyenTo(int n) {
        assertFalse(Tester.ktNguyenTo(n));
    }
    @ParameterizedTest
    @CsvSource({"2,true", "4,false", "7,true"})
    public void testCSV(int n, boolean expected) {
        assertEquals(Tester.ktNguyenTo(n), expected);
    }
    @ParameterizedTest
    @CsvFileSource(resources = "/data/data.csv",
        numLinesToSkip = 1)
    public void testCSVFile(int n,
        boolean expected) {
        assertEquals(Tester.ktNguyenTo(n), expected);
    }

    @ParameterizedTest
    @MethodSource(value = "primeData")
    public void testMethod(int n,
        boolean expected) {
        assertEquals(Tester.ktNguyenTo(n), expected);
    }
}
```

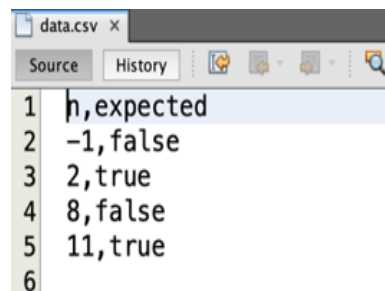
```

    }

    static Stream<Arguments> primeData() {
        return Stream.of(
            arguments(4, false),
            arguments(6, true)
        );
    }
}

```

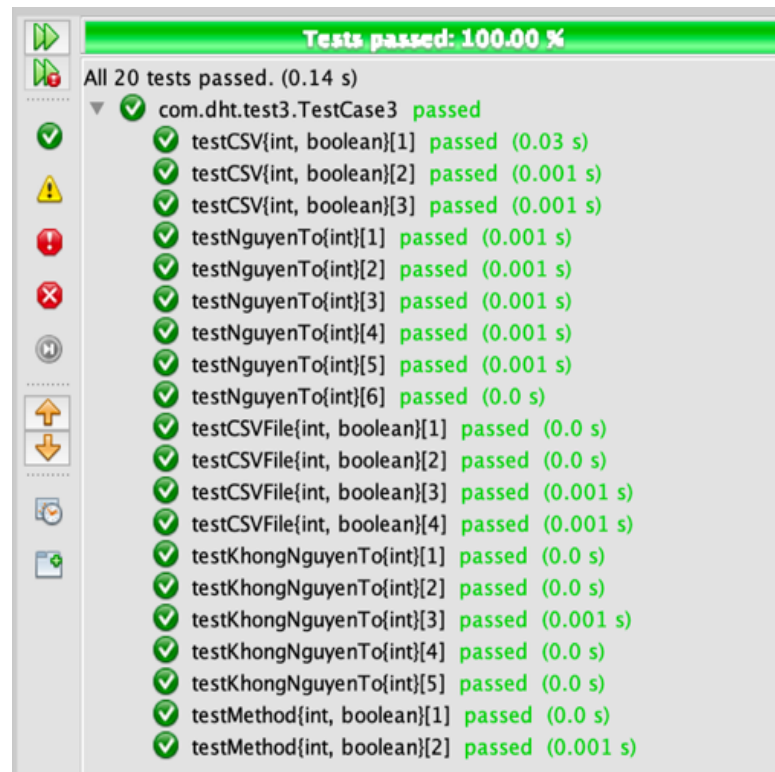
Trong đó tập tin data.csv nằm trong thư mục test/resources/data (xem **Error! Reference source not found.**).



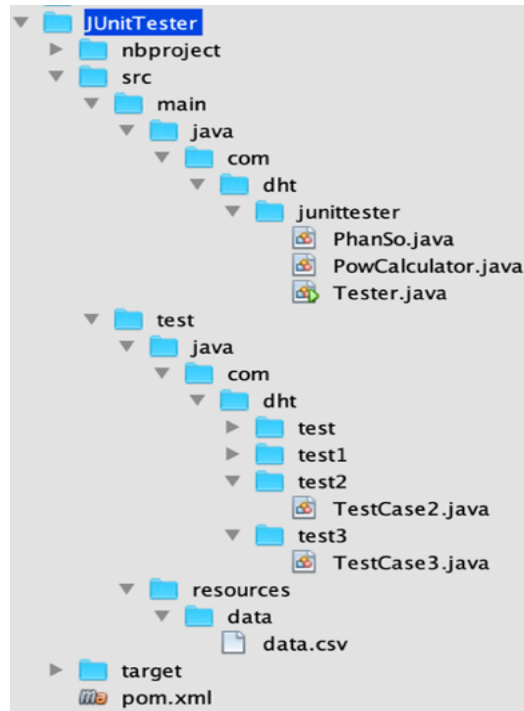
```

1 h,expected
2 -1,false
3 2,true
4 8,false
5 11,true
6

```



Kết quả thực thi Parameterized Test.



Minh họa cấu trúc project kiểm thử với junit.

6. Bài tập

Bài 1: Cho hàm tính x^n bằng đệ quy, trong đó x là số thực và n là số nguyên bất kỳ, biết x^n được tính như sau:

$$x^n = \begin{cases} 1 & \text{nếu } n = 0 \\ x^{n-1} \times x & \text{nếu } n > 0 \\ \frac{x^{n+1}}{x} & \text{nếu } n < 0 \end{cases}$$

```

static double Power(double x, int n)
{
    if (n == 0)
        return 1.0;
    else if (n > 0)
        return n * Power(x, n - 1);
    else
        return Power(x, n + 1) / x;
}

```

Viết unit test kiểm thử hàm Power với đặc tả yêu cầu như trên.

Bài 2: Một chương trình tính giá trị đa thức tại một giá trị x nào đó. Chương trình nhập vào số nguyên n là bậc đa thức và $n + 1$ số nguyên a_i ($0 \leq i \leq n$), với a_i là hệ số của x^i và giá trị biến nguyên x . Nếu người dùng nhập không đủ $n + 1$ hệ số cho đa thức hoặc nhập n âm thì ném ra ngoại lệ `ArgumentException`, với nội dung lỗi là “Invalid Data”. Viết Unit Test kiểm tra đoạn chương trình hiện thực hóa yêu cầu trên.

```
class Polynomial
{
    private int n;
    private List<int> a;

    public Polynomial(int n, List<int> a)
    {
        if (a.Count() != n + 1)
            throw new ArgumentException("Invalid Data");

        this.n = n;
        this.a = a;
    }

    public int Cal(double x)
    {
        int result = 0;
        for (int i = 0; i <= this.n; i++)
        {
            result += (int)(a[i] * Math.Pow(x, i));
        }

        return result;
    }
}
```

Bài 3: Cho chương trình chuyển đổi số nguyên dương cơ số 10 sang cơ số nguyên k bất kỳ (với $2 \leq k \leq 16$). Viết các Unit Test kiểm thử đoạn chương trình này.

```
public class Radix
{
    private int number;

    public Radix(int number)
    {
        if (number < 0)
            throw new ArgumentException("Incorrect Value");

        this.number = number;
    }

    public string ConvertDecimalToAnother(int radix = 2)
    {
        int n = this.number;

        if (radix < 2 || radix > 16)
            throw new ArgumentException("Invalid Radix");

        List<string> result = new List<string>();
        while (n > 0)
        {
            int value = n % radix;
            if (value < 10)
                result.Add(value.ToString());
            else
            {
                switch (value)
                {
                    case 10: result.Add("A"); break;
                    case 11: result.Add("B"); break;
                    case 12: result.Add("C"); break;
                    case 13: result.Add("D"); break;
                    case 14: result.Add("E"); break;
                    case 15: result.Add("F"); break;
                }
            }
            n = n / radix;
        }
        return string.Join("", result.Reverse());
    }
}
```

```

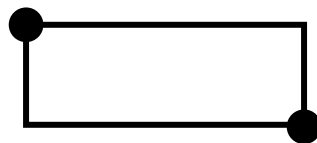
        {
            case 10: result.Add("A"); break;
            case 11: result.Add("B"); break;
            case 12: result.Add("C"); break;
            case 13: result.Add("D"); break;
            case 14: result.Add("E"); break;
            case 15: result.Add("F"); break;
        }
    }
    n /= radix;
}

result.Reverse();
return String.Join("", result.ToArray());
}
}

```

Bài 4: Viết lớp `Diem` để thao tác với điểm trong không gian hai chiều bao gồm hai thuộc tính hoành độ và tung độ. Lớp `HinhChuNhat` chứa thông tin của một hình chữ nhật, biết một hình chữ nhật được xác định bởi 2 điểm là tọa độ điểm trên bên trái và tọa độ điểm dưới bên phải:

Điểm trên bên trái



Điểm dưới bên phải

Lớp `HinhChuNhat` có hai phương thức thực hiện các chức năng sau:

- Tính diện tích hình chữ nhật
- Kiểm tra hai hình chữ nhật có giao nhau hay không?

Viết Unit Test để kiểm thử các chức năng của chương trình trên.

Bài 5: Một trung tâm gia sư cần quản lý thông tin học viên, một học viên bao gồm thông tin: mã số học viên, họ tên, quê quán, điểm của ba môn học chính. Vào cuối khoá học, trung tâm muốn tìm ra một số học viên có thành tích học tập tốt để trao học bổng khuyến khích. Một học viên được đánh giá là tốt nếu điểm trung bình ba môn học chính từ 8.0 trở lên và không có môn nào trong ba môn chính điểm dưới 5.

- a) *Viết chương trình cho phép nhập danh sách học viên và xác định danh sách học viên có thể nhận học bổng.*
- b) *Viết các Unit Test để kiểm thử các chức năng của chương trình trên.*