public final class MyEqualityTest

{

public static void main( String args[] )

{

String s1 = new String( "Test" );

String s2 = new String( "Test" );

System.out.println( "\n1 - PRIMITIVES ");

System.out.println( s1 == s2 ); // false

System.out.println( s1.equals( s2 )); // true

A a1 = new A();

A a2 = new A();

System.out.println( "\n2 - OBJECT TYPES / STATIC VARIABLE" );

System.out.println( a1 == a2 ); // false

System.out.println( a1.s == a2.s ); // true

System.out.println( a1.s.equals( a2.s ) ); // true

B b1 = new B();

B b2 = new B();

System.out.println( "\n3 - OBJECT TYPES / NON-STATIC VARIABLE" );

System.out.println( b1 == b2 ); // false

System.out.println( b1.getS() == b2.getS() ); // false

System.out.println( b1.getS().equals( b2.getS() ) ); // true

}

}

final class A

{

// static

public static String s;

A()

{

this.s = new String( "aTest" );

}

}

final class B

{

private String s;

B()

{

this.s = new String( "aTest" );

}

public String getS()

{

return s;

}

}

**1. PRIMITIVES**

String s1 = new String( "Test" );

String s2 = new String( "Test" );

System.out.println( s1 == s2 ); // false

System.out.println( s1.equals( s2 )); // true

* s1 == s2: **false**
  + == so sánh **tham chiếu** (reference) của hai đối tượng.
  + Vì s1 và s2 được tạo bằng new String("Test"), chúng trỏ đến hai đối tượng khác nhau trong bộ nhớ heap, dù nội dung giống nhau.
* s1.equals(s2): **true**
  + .equals() so sánh **nội dung** của hai chuỗi.
  + Chuỗi "Test" có cùng nội dung, nên kết quả là true.

**2. OBJECT TYPES / STATIC VARIABLE**

A a1 = new A();

A a2 = new A();

System.out.println( a1 == a2 ); // false

System.out.println( a1.s == a2.s ); // true

System.out.println( a1.s.equals( a2.s ) ); // true

* **a1 == a2:** **false**
  + a1 và a2 là hai đối tượng khác nhau được tạo từ lớp A, vì vậy tham chiếu của chúng khác nhau.
* **a1.s == a2.s:** **true**
  + s là biến **tĩnh** (static), được chia sẻ chung giữa tất cả các đối tượng của lớp A.
  + Dù hai đối tượng a1 và a2 khác nhau, chúng đều trỏ tới cùng một biến s duy nhất.
* **a1.s.equals(a2.s):** **true**
  + Nội dung của chuỗi s là "aTest", và cả hai đối tượng đều chia sẻ chung nội dung này.

**Ghi chú:** Mặc dù this.s = new String("aTest") trong constructor A(), nhưng vì **s là static, chỉ có một bản sao của s tồn tại.** Bất kỳ đối tượng nào tạo từ A đều sẽ cập nhật cùng biến s.

**3. OBJECT TYPES / NON-STATIC VARIABLE**

B b1 = new B();

B b2 = new B();

System.out.println( b1 == b2 ); // false

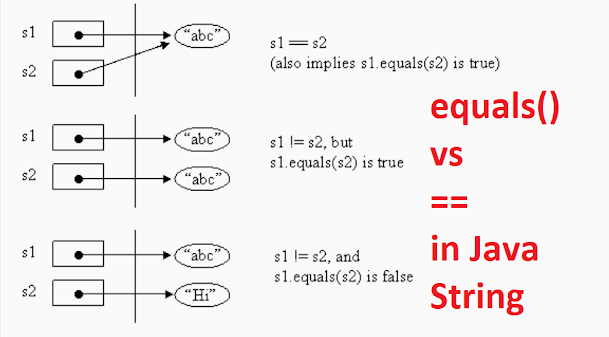
System.out.println( b1.getS() == b2.getS() ); // false

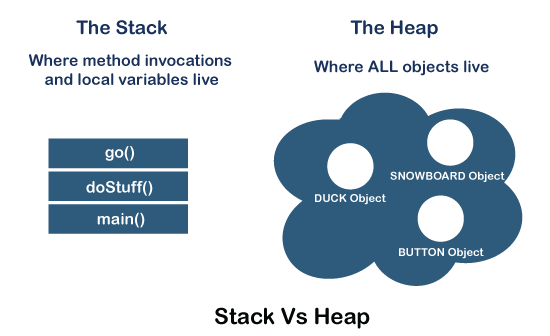
System.out.println( b1.getS().equals( b2.getS() ) ); // true

* **b1 == b2:** **false**
  + b1 và b2 là hai đối tượng khác nhau, nên tham chiếu của chúng không giống nhau.
* **b1.getS() == b2.getS():** **false**
  + Mỗi đối tượng B có biến **không tĩnh** s riêng của mình.
  + Mặc dù cả hai s đều chứa "aTest", chúng là hai đối tượng chuỗi khác nhau trong bộ nhớ heap (do new String("aTest")).
* **b1.getS().equals(b2.getS()):** **true**
  + .equals() so sánh nội dung chuỗi. Vì cả hai chuỗi đều có nội dung "aTest", kết quả là true.

**Tóm tắt**

1. **==**:
   * **So sánh tham chiếu (địa chỉ bộ nhớ)** giữa hai đối tượng.
   * Chỉ trả về **true** nếu **hai biến tham chiếu** cùng **trỏ đến một đối tượng**.
2. **.equals()**:
   * So sánh nội dung của hai đối tượng.
   * Nếu **nội dung** giống nhau, kết quả là true.
3. **Biến tĩnh (static)**:
   * Chỉ có **một bản sao** trong bộ nhớ, **được chia sẻ** giữa tất cả các đối tượng của lớp.
4. **Biến không tĩnh**:
   * **Mỗi đối tượng có bản sao riêng** của biến, không ảnh hưởng lẫn nhau.





Bộ nhớ Stack được dùng để lưu trữ các **biến cục bộ (local variable)** bên trong hàm, vùng nhớ này dùng để chứa giá trị của các tham số khi được gọi đến. Khi kết thúc một hàm, vùng nhớ Stack sẽ được tự động giải phóng. Stack sử dụng theo cấu trúc **LIFO** (vào sau ra trước)

Bộ nhớ Heap được dùng để lưu trữ các **biến toàn cục (global variable)** của chương trình, hoặc lưu trữ các **đối tượng của con trỏ** khi được cấp phát. Vùng nhớ Heap phải được giải phóng thông qua hàm bởi lập trình viên (tuy nhiên ở 1 số ngôn ngữ hiện nay có chức năng tự động thu gom, giải phóng vùng nhớ như GC trong C#).

