Integer or int?

Sr. No.	Key	int	Integer
1	Type	A int is a data type that stores 32 bit signed two's compliment integer.	On other hand Integer is a wrapper class which wraps a primitive type int into an object.
2	Purpose	int helps in storing integer value into memory.	Integer helps in converting int into object and to convert an object into int as per requirement.
3	Flexibility	int provides less flexibility as compare to Integer as it only allows binary value of an integer in it.	Integer on other hand is more flexible in storing and manipulating an int data.Since Wrapper classes inherit Object class, they can be used in collections with Object reference or generics.
4	Memory allocation	As already mentioned int is a primitive data type and takes 32 bits(4 bytes) to store.	On other hand Integer is an object which takes 128 bits (16 bytes) to store its int value.
5	Casting	In java one canâTM$_t$ assign a string value (containing an integer only) to an int variable directly or even by casting.	In case of Integer we can assign string to an object of Integer type using the Integer(String) constructor or by even use parseInt(String) to convert a String literal to an int value.
6	Direct Conversion to Other base.	In case of int we can't convert its integer value to other base.	However in Integer we can directly convert its integer value to other bases such as Binary, Octal or Hexadecimal format using toBinaryString(), toOctalString() or toHexString() respectively.
7	Allowed operations	int do not allowed any of inbuilt functions to change its value or syntax.	However in Integer we can reverse number or rotate it left or right using reverse(), rotateLeft() and rotateRight() respectively.

Integer vs int performance

- **int** (32 bits of memory), none of the overhead of object allocation that the Integer class requires,

  ➔ **int will run faster than Integer**

- However, it is **time-consuming** in the case
  - JVM converts int data types into Integer instances (in collection classes such as **ArrayList** or **HashMap**).
  - Known as autoboxing, the automatic conversion to an **Integer** and back to an **int** can be very **expensive in terms of both memory and processor usage**.

  ➔ In these cases, the use of an **int** can cause an unexpected performance hit

Truyền tham số cho phương thức

Ví dụ 1:

```
public class ReferenceTest {
    public static void main(String[] args) {
        Dimension d = new Dimension(5, 10);
        ReferenceTest rt = new ReferenceTest();
        System.out.println("Before modify() d.height = " + d.height);
        rt.modify(d);
        System.out.println("After modify() d.height = " + d.height);
    }

    void modify(Dimension dim) {
        dim.height = dim.height + 1;
        System.out.println("dim = " + dim.height);
    }
}
```

Ví dụ 1: Kết quả

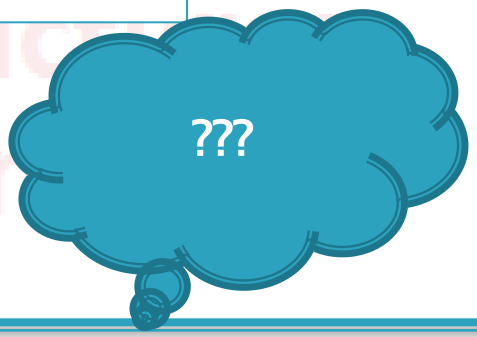
Before modify() d.height = 10

dim.height = 11

After modify() d.height = 11

Ví dụ 2:

```
public class PassingVar extends TestCase {
    public void modify(Student st) {
        st.setName("Tran Thi B");
        st = new Student("Le Van C");
    }

    public void test() {
        Student sv1 = new Student("Nguyen Van A");
        System.out.println("Before modify():" + sv1);
        modify(sv1);
        System.out.println("After modify():" + sv1);
    }
}
```



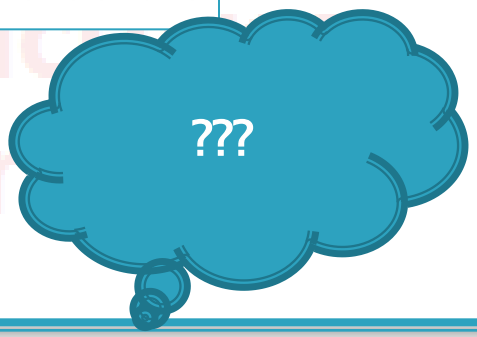
Ví dụ 2: Kết quả

Before modify():Nguyen Van A

After modify():Tran Thi B

Ví dụ 3:

```
public class SwapNumber extends TestCase {
    public void swap(int a, int b) {
        int c = a;
        a = b;
        b = c;
    }

    public void test() {
        int a = 1, b = 2;
        System.out.println("Before swap a: " + a + " , b: " + b);
        swap(a, b);
        System.out.println("After swap a: " + a + " , b: " + b);
    }
}
```



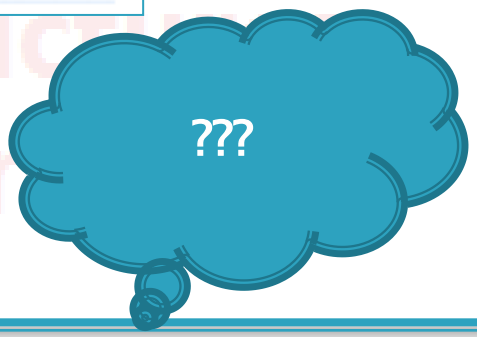
Ví dụ 3: Kết quả

Before swap a: 1 , b: 2

After swap a: 1 , b: 2

Ví dụ 4:

```
public class SwapNumber extends TestCase {
    public void swap(Integer a1, Integer b1) {
        Integer c = a1;
        a1 = b1;
        b1 = c;
    }

    public void test() {
        Integer a = new Integer(1), b = new Integer(2);
        System.out.println("Before swap a: " + a + " , b: " + b);
        swap(a, b);
        System.out.println("After swap a: " + a + " , b: " + b);
    }
}
```

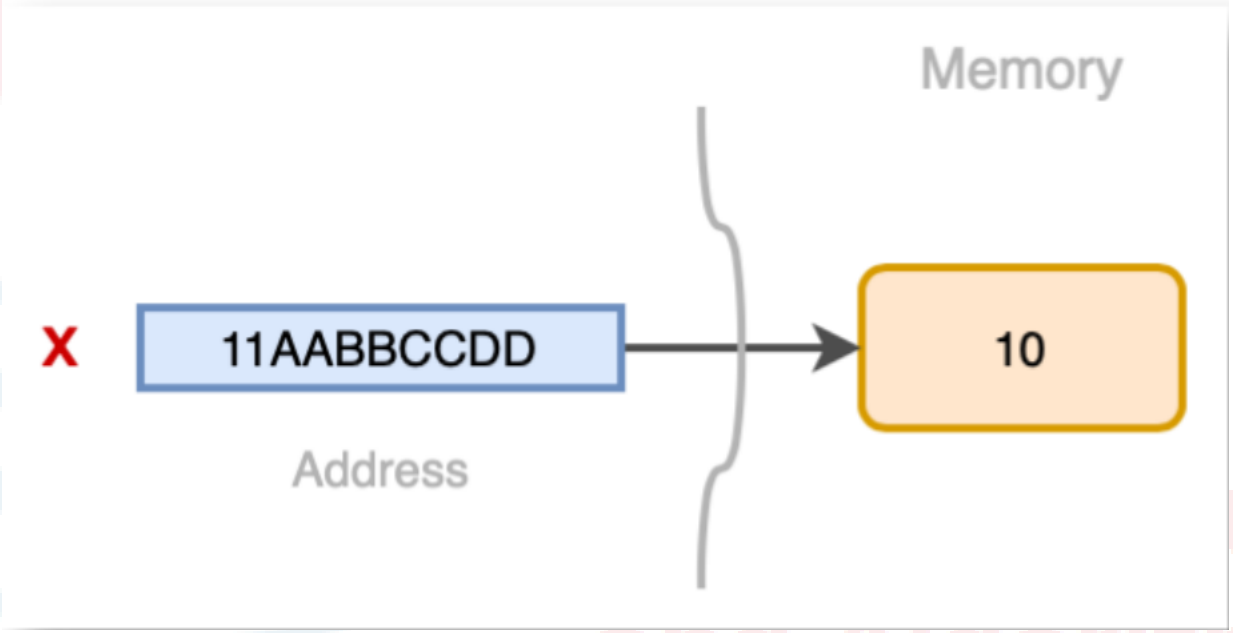


Truyền tham số cho phương thức

- Java hỗ trợ truyền tham số **kiểu tham trị** (*pass-by-value*) **và tham chiếu** (*pass-by-reference*)**?**
  - Đối với kiểu dữ liệu nguyên thủy (primitive data)?
  - Kiểu dữ liệu đối tượng (object)?

Remind: Khai báo biến

- Sơ đồ biểu diễn việc khai báo giá trị cho biến (int x = 10;)



Remind: Truyền tham trị

- Kết quả thực thi đoạn code sau đây là gì?

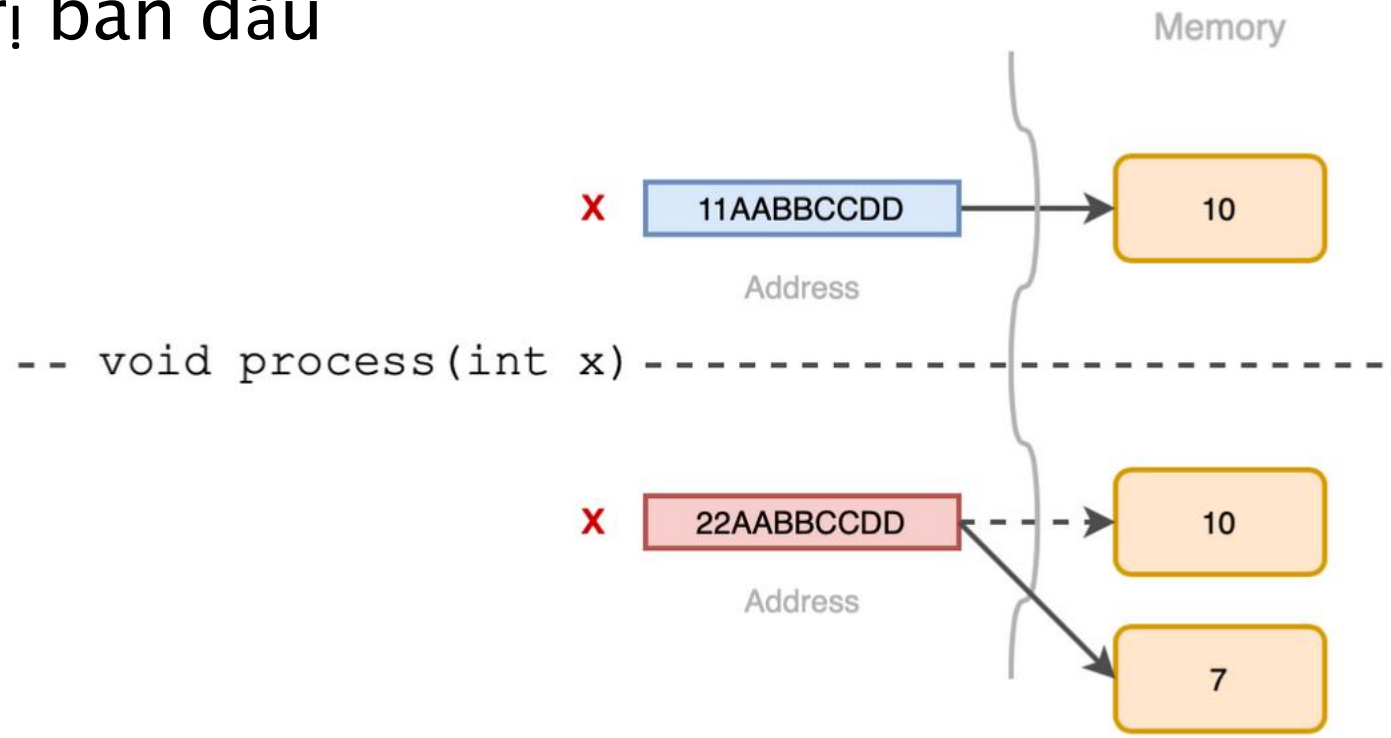
```
public static void main(String[] args) {
    int x = 10;
    System.out.println("Before call process: " + x);
    process(x);
    System.out.println("After call process: " + x);
}

public static void process(int x) {
    x = 7;
}
```

```
Before call process: 10
After call process: 10
```

Sao chép giá trị khi tham số được truyền vào trong một phương thức!!!

Remind: Truyền tham trị

- Việc thay đổi giá trị của biến chỉ có ý nghĩa trong phương thức.
- Khi phương thức kết thúc → giá trị trở lại giá trị ban đầu



Ví dụ:

- Kết quả thực thi đoạn code sau đây là gì?

```
public class SwapNumber extends TestCase {
    public void swap(int a, int b) {
        int c = a;
        a = b;
        b = c;
    }

    public void test() {
        int a = 1, b = 2;
        System.out.println("Before swap a: " + a + " , b: " + b);
        swap(a, b);
        System.out.println("After swap a: " + a + " , b: " + b);
    }
}
```

Before swap a: 1 , b: 2

After swap a: 1 , b: 2

Truyền tham trị!!!

Remind: Truyền tham chiếu

- Ví dụ truyền tham chiếu trong C++

```
#include <iostream>

int main() {
    int x = 10;

    cout << "Before call process: " << x << endl;
    process(someValue);
    cout << "After call process: " << x << endl;

    return 0;
}

void process(int& x) {
    x = 7;
}
```

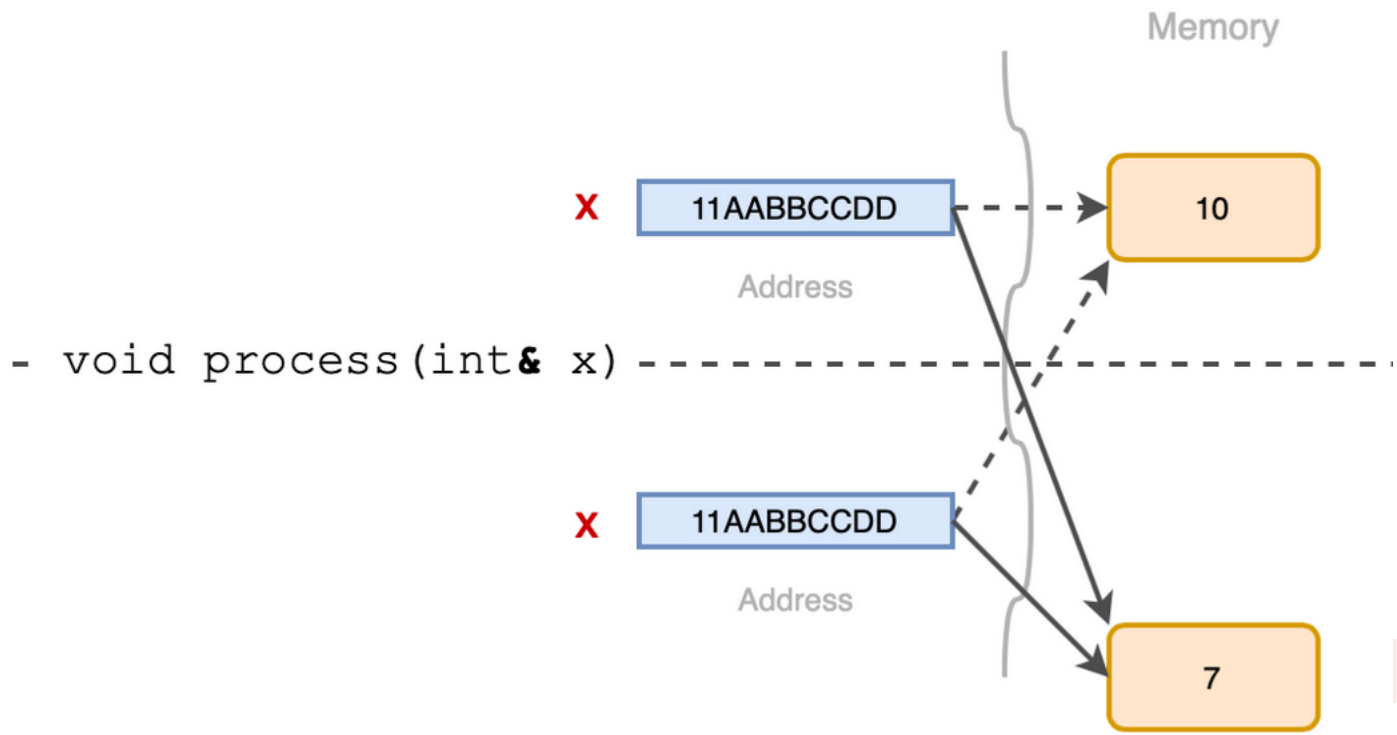
Tham số sẽ truyền theo kiểu *Tham chiếu*

```
Before call process: 10
After call process: 7
```

Truyền tham chiếu: truyền địa chỉ của biến

Remind: Truyền tham chiếu (tt.)

- Sơ đồ mô tả truyền tham chiếu (C++):



Khi đưa vào trong ***process()***, địa chỉ của ***x*** sẽ được truyền vào

Ví dụ 1:

- Cho lớp MyCat như sau:

```
public class MyCat {
    private String name;

    public MyCat(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Ví dụ 1: Trường hợp 1

- Kết quả thực hiện đoạn code sau đây là gì?

```
public static void main(String[] args) {
    MyCat myCat = new MyCat("Kitty");
    System.out.println("Before call process: " + myCat.getName());
    process(myCat);
    System.out.println("After call process: " + myCat.getName());
}

public static void process(MyCat myCat) {
    myCat.setName("Doraemon");
}
```

```
Before call process: Kitty
After call process: Doraemon
```

Thay đổi ***myCat*** bên trong **làm ảnh hưởng** đến ***myCat*** bên ngoài.
➔Truyền ***tham chiếu***?

Ví dụ 1: Trường hợp 2

- Kết quả thực hiện đoạn code sau đây là gì?

```
public static void main(String[] args) {
    MyCat myCat = new MyCat("Kitty");
    System.out.println("Before call process: " + myCat.getName());
    process(myCat);
    System.out.println("After call process: " + myCat.getName());
}

public static void process(MyCat myCat) {
    myCat = new MyCat("Doraemon");
}
```

```
Before call process: Kitty
After call process: Kitty
```

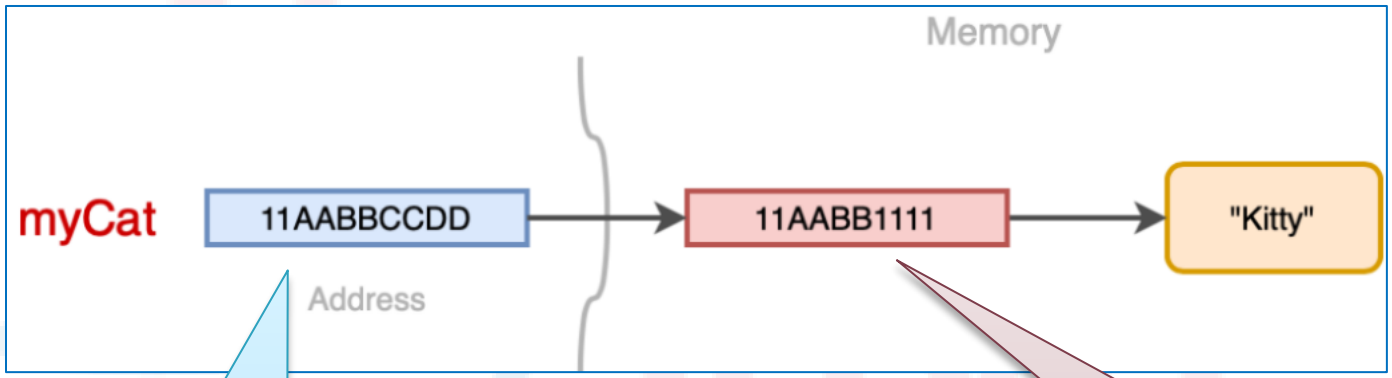
Thay đổi ***myCat*** bên trong **không làm ảnh hưởng** đến ***myCat*** bên ngoài.

➔Truyền ***tham trị***?

Java hỗ trợ truyền tham chiếu hay không?

Reconsider: Ví dụ 1

- MyCat myCat = new MyCat("Kitty");



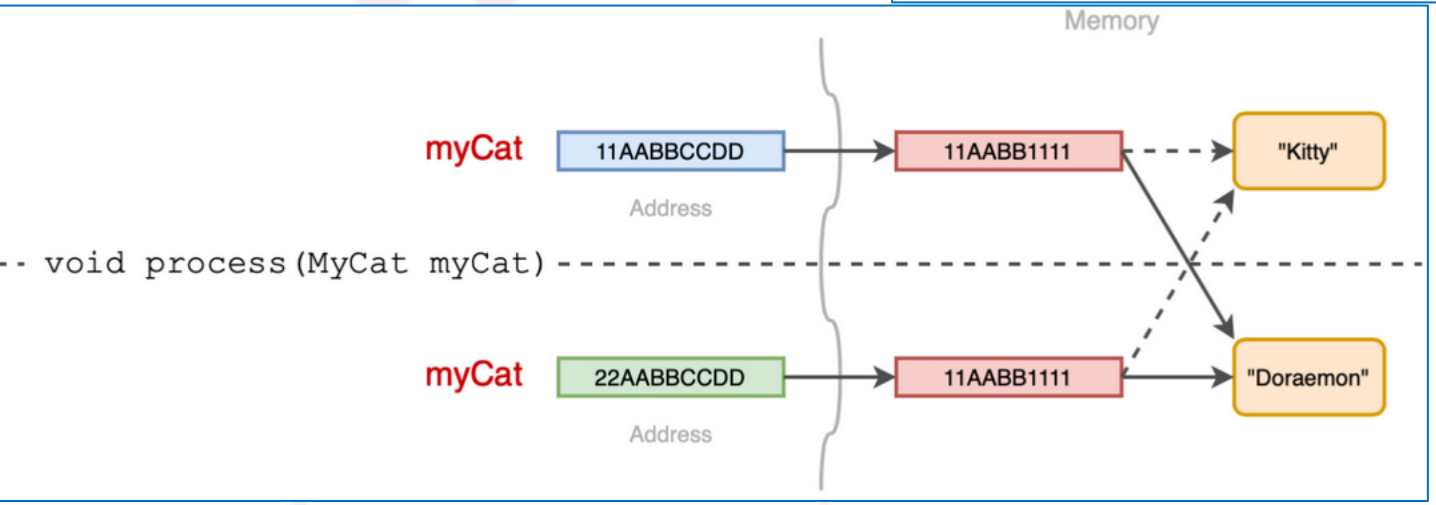
Địa chỉ của biến **myCat**

Địa chỉ chỉ đến thuộc tính của **MyCat**

Reconsider:Trường hợp 1

myCat bên ngoài và bên trong khác nhau về địa chỉ, nhưng cùng giá trị chính là địa chỉ đến ***name***.

```
public static void main(String[] args) {
    MyCat myCat = new MyCat("Kitty");
    System.out.println("Before call process: " + myCat.getName());
    process(myCat);
    System.out.println("After call process: " + myCat.getName());
}

public static void process(MyCat myCat) {
    myCat.setName("Doraemon");
}
```



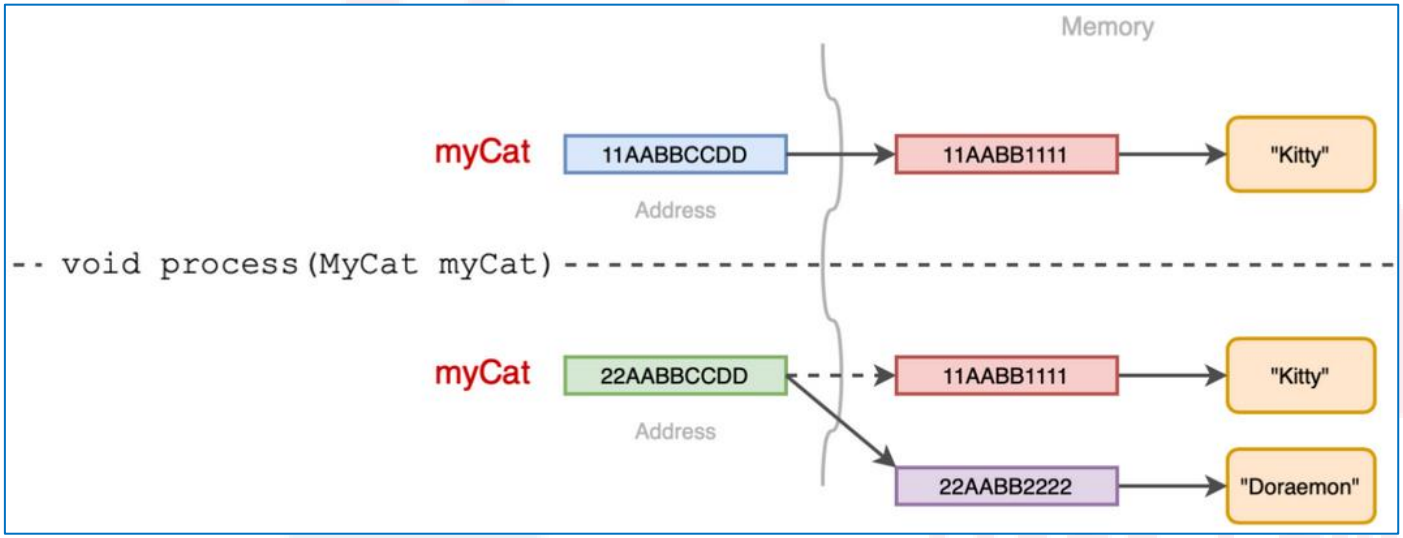
Thay đổi ***name*** đối với ***myCat*** ➔ thay đổi cả ***name*** của ***myCat*** trước khi truyền vào ***process()***

Truyền tham trị!!!

Reconsider:Trường hợp 2

myCat truyền vào trong ***process()***, việc sao chép giá trị (chính là địa chỉ của ***name***) vào trong phương thức vẫn diễn ra

```
public static void main(String[] args) {
    MyCat myCat = new MyCat("Kitty");
    System.out.println("Before call process: " + myCat.getName());
    process(myCat);
    System.out.println("After call process: " + myCat.getName());
}

public static void process(MyCat myCat) {
    myCat = new MyCat("Doraemon");
}
```



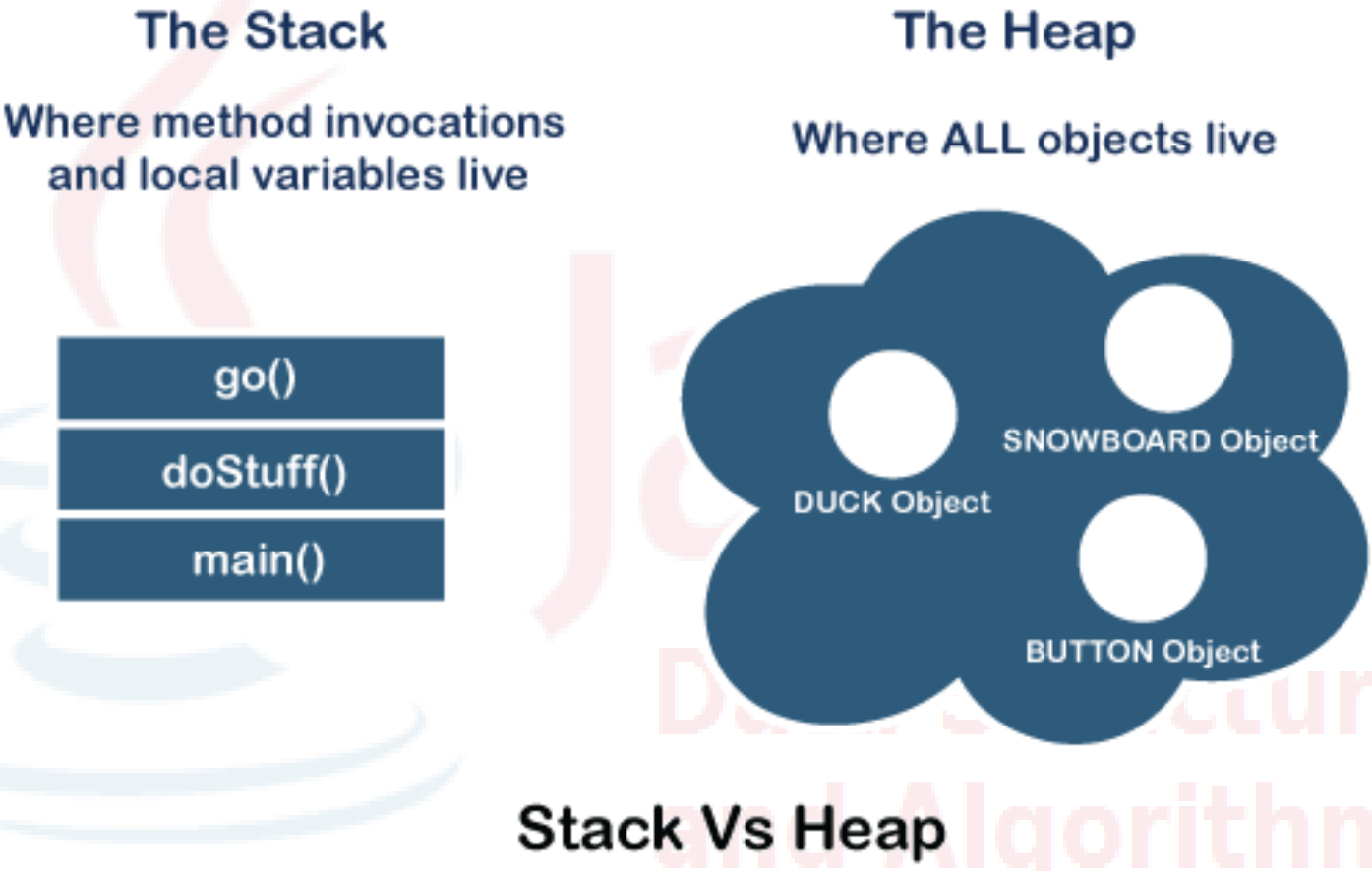
Trong ***process(), myCat*** được khởi tạo lại thành một đối tượng mới thông qua toán tử ***new*** ➔ giá trị mới của thuộc tính name cũng thay đổi theo

Truyền tham trị!!!

Truyền tham số cho phương thức

- Java chỉ hỗ trợ truyền tham số **kiểu tham trị** (*pass-by-value*):
  - Đối với kiểu dữ liệu nguyên thủy (primitive data);
    - việc thay đổi giá trị của biến chỉ có ý nghĩa trong phương thức.
    - Khi phương thức kết thúc → giá trị trở lại giá trị ban đầu
  - Kiểu dữ liệu đối tượng (object):
    - **thay đổi giá trị thuộc tính của đối tượng** bên trong phương thức sẽ ảnh hưởng tới đối tượng dùng làm tham số.
    - Thay đổi bằng phép gán (=) chỉ có ý nghĩa trong phương thức.

Stack vs Heap Java



Stack Vs Heap

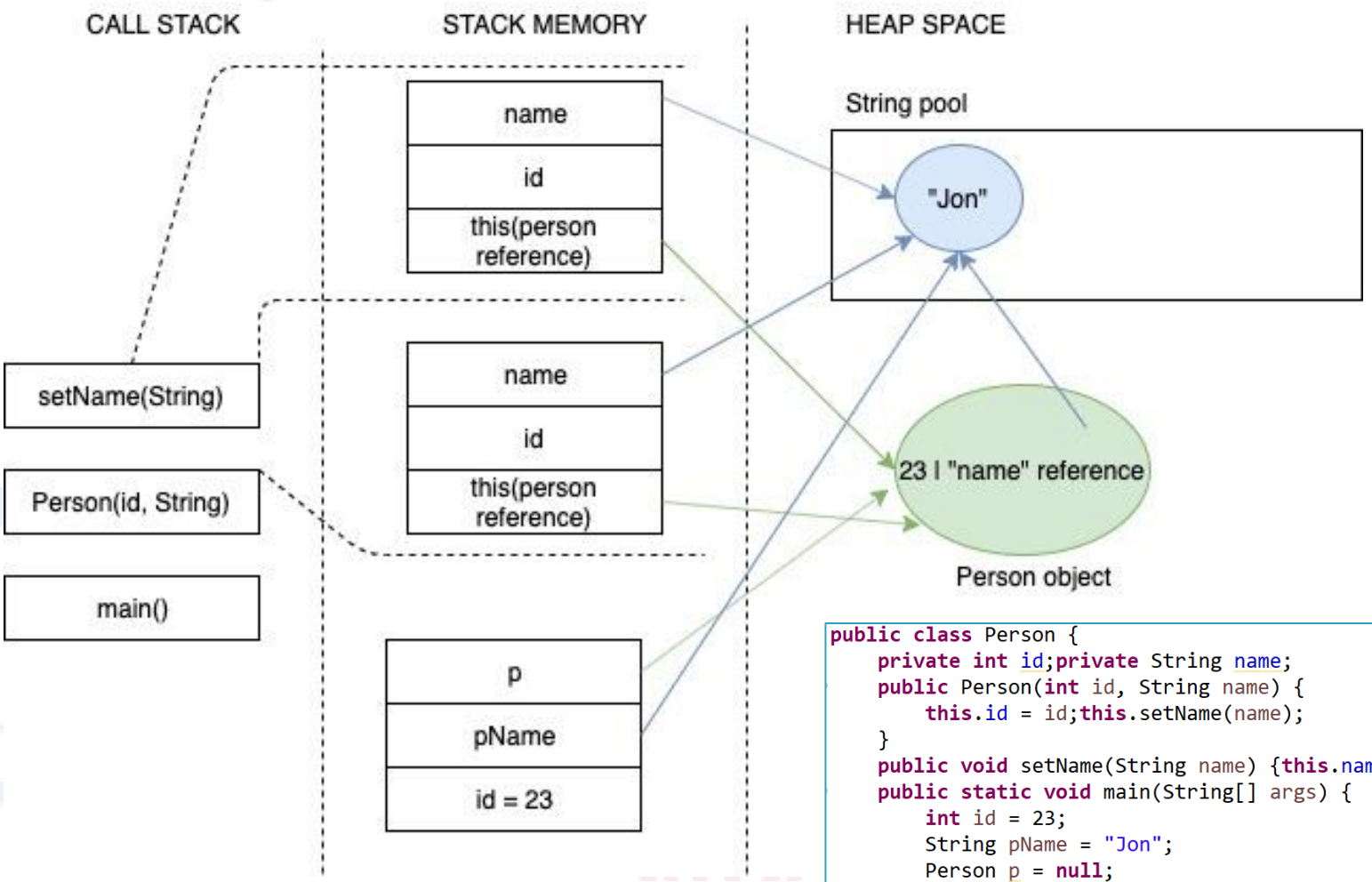
Stack vs Heap Java (cont.)

Example:

```
public class Person {
    private int id;
    private String name;

    public Person(int id, String name) {
        this.id = id;
        this.setName(name);
    }

    public void setName(String name) {
        this.name = name;
    }

    public static void main(String[] args) {
        int id = 23;
        String pName = "Jon";
        Person p = null;
        p = new Person(id, pName);
    }
}
```

https://www.baeldung.com/java-stack-heap

Stack vs Heap Java (cont.)




```
public class Person {
    private int id;private String name;
    public Person(int id, String name) {
        this.id = id;this.setName(name);
    }
    public void setName(String name) {this.name = name;}
    public static void main(String[] args) {
        int id = 23;
        String pName = "Jon";
        Person p = null;
        p = new Person(id, pName);
    }
}
```