Python vs Java: Class Properties, Methods & Static Concepts

1. Python: Instance vs Class vs Static

Python has three kinds of methods and two kinds of properties.

Instance Property & Instance Method

Instance Property

- Defined inside `__init__`
- Belongs to each object
- Every instance gets its own value

```python
class A:
    def __init__(self):
        self.x = 10   # instance property
```

Instance Method

- First parameter: `self`
- Can access instance and class properties

```python
class A:
    def show(self):
        print(self.x)
```

Class Property & Class Method

Class Property

- Defined directly inside the class
- Shared by all objects

```python
class A:
    count = 0     # class property
```

Class Method

- Declared with `@classmethod`
- First parameter: `cls` → refers to the class
- Can modify class properties

```python
class A:
    count = 0

    @classmethod
    def inc(cls):
        cls.count += 1
```

Important: `cls` is dynamic and refers to whichever class calls the method. Therefore, `classmethod` is polymorphic.

Static Method

Static Method

- Declared with `@staticmethod`
- Receives no `self` or `cls`
- Utility function placed inside a class

```python
class A:
    @staticmethod
    def add(a, b):
        return a + b
```

Static methods in Python:

- Not tied to class or instance
- Not polymorphic
- Same as Java static method

Python Summary Table

Type	Has self?	Has cls?	Shared?	Polymorphic?
Instance property	✓	✗	✗	✓ (via instance methods)
Class property	✗	✓	✓	✓
Instance method	✓	✗	✗	✓

Type	Has self?	Has cls?	Shared?	Polymorphic?
Class method	❌	✓	✓	✓
Static method	❌	❌	❌	❌

---

2. Java: Instance vs Static

Instance Variables & Methods

- Belong to objects
- Support method overriding
- Support runtime polymorphism

```java
class A {
    int x = 10;

    void show() {
        System.out.println(x);
    }
}
```

Static Variables & Static Methods

Static Variable

- Shared by the entire class

```java
class A {
    static int count = 0;
}
```

Static Method

- Belongs to the class, not the object
- Does not receive `this`

```java
```

```
class A {
    static void inc() {
        count++;
    }
}
```

Java static methods:

- Resolved at compile-time
- Not polymorphic
- Cannot be overridden (only hidden)

Is Java static same as Python classmethod?

❌ **No. Absolutely not.**

- **Java static ≈ Python staticmethod**
- Java has no equivalent to Python classmethod

But both can access class properties:

- Java static method can access static variable
- Python classmethod can access class variable

However, the mechanism is different:

- Java static → no class object passed
- Python classmethod → class object (`cls`) is passed dynamically

---

3. Why Java Static Methods Do NOT Override

Consider this code:

java

```
class A {
    static int count = 0;
    static void inc() { count++; }
}

class B extends A {
    static int count = 100;
}
```

Calling `B.inc()` uses `A.inc()`, not `B.inc()`.

Why?

Static methods belong to the class, not the object.

Overriding requires:

- A real object
- Runtime polymorphism
- Dynamic dispatch

Static methods:

- Are resolved at compile time
- According to reference type, not object type
- Therefore cannot participate in polymorphism

They are hidden, not overridden.

Example: Hiding vs Overriding

Static → Hiding

```java
A obj = new B();
obj.inc();   // calls A.inc()
```

Instance → Overriding

```java
A obj = new B();
obj.show();  // calls B.show() if overridden
```

---

4. Why Python classmethod Behaves Differently

Because:

- It receives `cls`
- `cls` refers to the actual class calling the method
- So child classes get their own dynamic version

Example:

```python
class A:
    count = 0
    @classmethod
    def inc(cls):
        cls.count += 1

class B(A):
    count = 100

B.inc()
print(B.count)  # 101
```

This is **REAL overriding and polymorphism**.

Java static methods do not work this way.

Final Summary

Python

- `@classmethod` = dynamic, polymorphic, receives class
- `@staticmethod` = no class/instance, same as Java static

Java

- `static` ≠ `classmethod`
- `static` = `staticmethod`
- Static methods do NOT override (compile-time)
- Instance methods DO override (runtime)

Why Java static cannot override?

Because: **Static methods belong to class, not object → no runtime dispatch → no overriding**