Master record In Java

The Cleanest Way to Model Data or Create POJO

The Problem :

You need to create a Model class or Pojo

I'm sure many of you are still doing that in below way !

```
public class User {
    private final String name;
    private final int age;

    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String name() { return name; }
    public int age() { return age; }

    // equals(), hashCode(), toString()
}
```

😖 **20+ lines for a simple data holder**

→ Do it in 1 line

Use **record** : Feature introduced in Java 16+

```
public record User(String name, int age) { }
```

Just 1 line 😊

Constructor + Getters + equals()
+ hashCode() + toString()

🚀 All auto-generated

Know more about it →

Records are :

- Immutable by default
    - Record fields are final, meaning their values can’t be changed after creation
- Transparent: values are exposed
    - Record components automatically generate getter-like methods with no hiding of internal state.
- Ideal for DTOs, config, events
    - Records are perfect for carrying data without behavior, making them great for simple data transfer.

Avoid Records If

Don’t use Records when:

- You need mutability
    - You need to change field values after creation, they’re not suitable.
- You want to extend a class (records can’t)
    - Records implicitly extend java.lang.Record and cannot inherit from other classes.
- You need full control over object identity
    - Records auto-generate equals() and hashCode() based on all fields, it will not allow you to add custom identity logic

Bonus Tip

Bonus Tip -1

🔥 Bonus: Records with Validation

```
public record User(String name, int age) {
    public User {
        if (age < 0) throw new
            IllegalArgumentException("Age cannot be
                                          negative");
    }
}
```

💡 You can add constructors, methods, and validations!

Bonus tip →

Bonus Tip -2

🧪 Records + Sealed Classes = Perfect for modeling state machines or API responses

```
sealed interface Response permits Success, Error { }


record Success(String message) implements
Response {}


record Error(int code, String message)
implements Response {}
```

Swipe one more time — we saved the best tip for last!

Golden words

🚀 Clean Code is a feature.

Records make your data modeling modern, concise, and readable.

The End