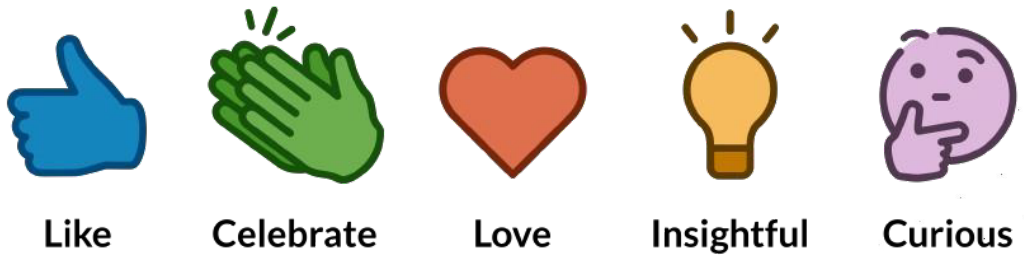
shift EduTech

Bad vs Good

Java Clean Code Tips!



Java Clean Code

1. Keep Classes Small

Small, single-purpose classes adhere to the Single Responsibility Principle (SRP) and make your code easier to understand and maintain.

Bad

```
public class OrderManager {
    // Order logic
    // Payment logic
    // Shipping logic
}
```

Good

```
public class OrderService { /* Order logic */ }
public class PaymentService { /* Payment logic */ }
public class ShippingService { /* Shipping logic */ }
```

shift EduTech

Java Clean Code

2. Prefer Enum for Constants

Enums provide a type-safe way to define constants and avoid issues with hard-coded strings. Use them wherever possible for better readability.

Bad

```
public static final String STATUS_ACTIVE = "ACTIVE";
public static final String STATUS_INACTIVE = "INACTIVE";
```

Good

```
public enum Status {
    ACTIVE, INACTIVE;
}
```

shift EduTech

Java Clean Code

3. Avoid NullPointerException

Reversing string comparisons *("constant".equals(variable))* prevents NPE. Use Optional to avoid null checks altogether and write safer code.

Bad

```
if (user.getName().equals("John")) {
    // Do something
}
```

Good

```
if ("John".equals(user.getName())) {
    // Do something
}
```

```
Optional.ofNullable(user.getName())
        .ifPresent(name -> {
            if (name.equals("John")) {
                // Do something
            }
        });
```

shift EduTech

Java Clean Code

4. Use Dependency Injection (DI)

Dependency injection decouples classes and improves testability. Use frameworks like Spring to inject dependencies instead of creating them manually.

Bad

```java
public class OrderService {
    private UserService userService = new UserService();
}
```

Good

```java
public class OrderService {
    private final UserService userService;

    public OrderService(UserService userService) {
        this.userService = userService;
    }
}
```

shift EduTech

Java Clean Code

5. Meaningful Variable Names

Avoid cryptic or single-letter variable names. Use names that reveal intent.

Bad

```
int d = 30; // What is 'd'?
```

Good

```
int daysUntilDeadline = 30; // Clear and self-explanatory
```

shift EduTech

Java Clean Code

6. Avoid Long Methods

Break down long methods into smaller, focused methods.

Bad

```
public void processData() {
    // 100+ lines of code
}
```

Good

```
public void processData() {
    validateInput();
    transformData();
    saveToDatabase();
}
private void validateInput() { /*...*/ }
private void transformData() { /*...*/ }
private void saveToDatabase() { /*...*/ }
```

shift EduTech

Java Clean Code

7. Use Constants for Magic Number

Replace magic numbers with constants for readability.

Bad

```
if (status == 5) {
    // Do something
}
```

Good

```
private static final int STATUS_COMPLETE = 5;
if (status == STATUS_COMPLETE) {
    // Do something
}
```

shift EduTech

Java Clean Code

8. Handle Exception Properly

Log meaningful errors and rethrow exceptions if needed.

Bad

```
try {
    // code
} catch (Exception e) {
    e.printStackTrace();
}
```

Good

```
try {
    // code
} catch (IOException e) {
    logger.error("Error processing file", e);
    throw new CustomException("File processing failed", e);
}
```

shift EduTech

Java Clean Code

9. Use Streams Wisely

Use Java Streams for concise and readable code.

Bad

```
List<String> names = new ArrayList<>();
for (User user : users) {
    if (user.isActive()) {
        names.add(user.getName());
    }
}
```

Good

```
List<String> names = users.stream()
    .filter(User::isActive)
    .map(User::getName)
    .collect(Collectors.toList());
```

Follow Me

Follow us on

Instagram

@ShiftEduTech

Follow us on

facebook

@ShiftEduTech

Follow us on

YouTube

@ShiftEduTech

Follow us on

LinkedIn

@ShiftEduTech

Like

Celebrate

Love

Insightful

Curious