**✨ What is Refactoring?**

Refactoring is the art of improving the design of existing code without changing its external behavior. Think of it as spring-cleaning your codebase—no new features, just making it better internally.

As per Martin Fowler:

“Refactoring is a controlled technique for improving the design of an existing code base.”

🔧 **Main goals:**

* Improve code clarity
* Reduce complexity
* Eliminate duplication
* Make code easier to test and extend

🛠️ **Best Practices:**

* Always run tests before and after refactoring
* Use IDE tools (e.g., IntelliJ IDEA, Eclipse) to automate safe refactorings
* Apply small, reversible steps

**👃 What Are Code Smells?**

"Code Smells" are signs that something might be wrong in your code—not bugs, but clues that the design could be cleaner.

🧪 **They hint at:**

* Poor object-oriented design
* Excessive dependencies
* Hard-to-read or duplicate logic

🔍 **Common Smells:**

| **Code Smell** | **What’s Wrong** |
| --- | --- |
| Long Method | Too many responsibilities |
| Duplicate Code | Same logic appears multiple places |
| Large Class | Knows or does too much |
| Primitive Obsession | Over-reliance on basic types |
| Feature Envy | One class heavily depends on another |

**🚀 Why Refactor?**

| **Benefit** | **Description** |
| --- | --- |
| 🧼 Cleaner Code | Easier to read and modify |
| ⚡ Faster Development | Simplifies future changes |
| 🐞 Easier Debugging | Clear structure reveals bugs |
| 💡 Better Design | Prevents design decay (bit rot) |
| 💰 Saves Time Long-Term | Less technical debt |

**⏱️ When to Refactor?**

* Before adding a new feature (to simplify the addition)
* During debugging (to better understand the code)
* After code reviews (to apply suggestions)
* During routine maintenance

**⚠️ Code Smells & Fixes – Java Examples**

**🔴 Long Method**

**Problem:** Doing too much in one place.

java

CopyEdit

public void processOrder() {

validateCustomer();

calculateTax();

sendEmail();

saveToDatabase();

}

✅ **Fix:** Split into smaller methods.

java

CopyEdit

public void processOrder() {

validateCustomer();

calculateTax();

notifyCustomer();

persistOrder();

}

private void notifyCustomer() { sendEmail(); }

private void persistOrder() { saveToDatabase(); }

**🔴 Duplicate Code**

java

CopyEdit

public double discountForStudent() {

return total \* 0.9;

}

public double discountForSenior() {

return total \* 0.9;

}

✅ **Fix:**

java

CopyEdit

public double applyDiscount(double rate) {

return total \* rate;

}

**🔴 Feature Envy**

java

CopyEdit

public class OrderPrinter {

public void print(Order order) {

System.out.println(order.getCustomer().getName());

System.out.println(order.getCustomer().getAddress());

}

}

✅ **Fix:** Let Customer describe itself.

java

CopyEdit

public class Customer {

public String printDetails() {

return name + "\n" + address;

}

}

public class OrderPrinter {

public void print(Order order) {

System.out.println(order.getCustomer().printDetails());

}

}

**🔴 Primitive Obsession**

**Problem:** Using String or int where objects should be.

java

CopyEdit

public class PhoneNumber {

private String areaCode;

private String number;

}

✅ **Fix:** Encapsulate in a value object.

java

CopyEdit

public class PhoneNumber {

private final String value;

public PhoneNumber(String value) {

// Validate format

this.value = value;

}

public String getValue() { return value; }

}

**🧪 Sample Refactoring Before vs After**

**❌ Before: Complex Logic in One Method**

java

CopyEdit

public void addItem(String item) {

if (!readOnly) {

if (items.size() == capacity) {

capacity += 10;

resize();

}

items.add(item);

}

}

**✅ After: Extract Method**

java

CopyEdit

public void addItem(String item) {

if (readOnly) return;

if (shouldResize()) resize();

store(item);

}

private boolean shouldResize() {

return items.size() == capacity;

}

private void resize() {

capacity += 10;

// logic to increase list capacity

}

private void store(String item) {

items.add(item);

}

📌 **Result:** Code becomes self-explanatory, less error-prone, and modular.

**✅ Summary**

| **Refactoring Tool** | **Purpose** |
| --- | --- |
| Extract Method | Break complex methods |
| Rename Variable/Method | Improve readability |
| Introduce Parameter Object | Replace multiple parameters |
| Replace Temp with Query | Remove temporary variables |
| Encapsulate Field | Improve access control |