Implementing Methods

Andrejs Doronins
TEST AUTOMATION ENGINEER

Module Overview

What methods should return

Parameters

Fail fast & return early

Avoid duplication

Conditionals

Clean Code Concepts

DRY vs. WET

Cyclomatic Complexity

Signal vs. Noise

DRY vs. WET

DRY

Don’t Repeat Yourself

WET

Write Everything Twice

Cyclomatic Complexity (CYC)

A software metric used to simply indicate the complexity of a program

Aim for lower CYC.
Lower complexity often means better code.

Signal vs. Noise

Signal

Clean useful code

Noise

Poor names, high CYC, duplication, bad comments...

```
   public <return type> getCustomerData(args...) {
        // Fail fast & return early
        // Conditionals
}
```

Do Not Return

Null

Special codes (-1, 0, 1 and other)

```
List<String> getSomeData() {
try{ // read from DB }
 catch {
   // operation failed
    return null;
    return Collections.emptyList();
   }
}
```

Leads to either:

```
NullPointerException
or
if (list != null)   +1 CYC
if (list != null)
if (list != null)
if (list != null)
```

Calling Code Is Simplified

```
if(~~list != null &&~~ list.size() != 0)
```

Check for Magic Numbers

```
// resulting balance is -1?
// or does -1 have a special meaning?
if(withdraw(100) == -1)
```

❌

```java
int withdraw(int amount) {
  if (amount > balance) {
    return -1;
  }
  else {
    balance -= amount;
    return 0;
  }
}
```

✓

```java
void withdraw(int amount) throws
InsufficientFundsException {
  if (amount > balance) {
    throw new InsufficientFundsException();
  }
  balance -= amount;
}
```

Generally, fewer method arguments is better.

Number of Arguments

OK	Avoid	Refactor!
0-2	3	4+

Single argument examples:

```
canReadFile("file.csv");

getSalesForYear(2018);

getMedicalRecord(john);
```

Two arguments examples:

```
add(2,3);

assertEquals(a,b);
```

Downsides of Too Many Arguments

Increased complexity

Difficult to read and understand

Methods with 3+ Arguments Might:

Do too many things (split it)

Take too many primitive types (pass a single object)

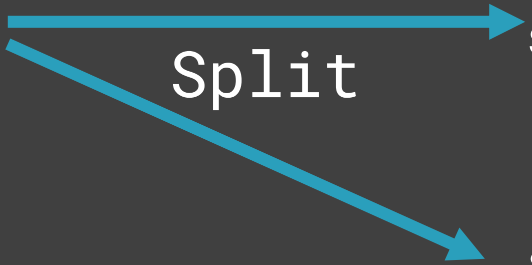
Takes a boolean(flag) argument (remove it)

"Flag arguments are ugly.

It immediately complicates the signature of the method, loudly proclaiming that this function does more than one thing."

Robert Martin

```
switchLights(room, true);
```



```
switchLightsOn(room);

switchLightsOff(room);
```

```
void switchLights(Room room,
                  boolean on){

    if(on){ // ...  }

    else { // ...   }

}
```

Avoid magic numbers. Put them into variables with names.

Fail Fast

Immediately report any failure and let the program fail

Fail Safe

Try to keep the program running

Failing fast frequently means easier troubleshooting

Fail Fast Checks

```
if(someInt == 0) { // ... }

if(someString.isEmpty()) { // ... }

if(someList.isEmpty()) { // ... }
```

Use Libraries

Native Java

- Objects.isNull();

Guava

- Preconditions.checkArgument();

Apache

- ObjectUtils.isNotEmpty();

Boolean Checks

```
if (!doorClosed == false)
```
✗

```
if (!doorClosed)
```
✗

```
if (!isDoorClosed)
```
✗

```
if (isDoorOpen)
```
✓

Avoid nested ternary expressions

Simple Ternary Expression

```
String getTitle(Person p) {
  return p.gender == Person.MALE ? "Mr. " : "Mrs. ";
}
```

Simple Ternary Expression?

```
String getTitle(Person p) {
  return p.gender == Person.MALE ? "Mr.": p.isMarried() ? "Mrs.":"Miss";
}
```

uh...

Refactored Ternary Expression

```
String getTitle(Person p) {
  if (p.gender == Person.MALE) {
    return "Mr.";
  }
  return p.isMarried() ? "Mrs." : "Miss";
}
```

Summary

- ✓ Clean code concepts – DRY, CYC, Signal vs. Noise
- ✓ What to return & number of arguments
- ✓ Fail fast & return early
- ✓ Working with booleans