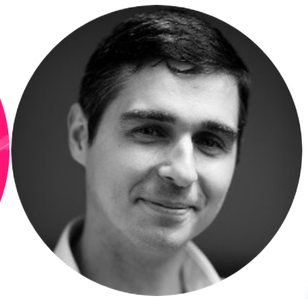
Iterating and Branching - the Good and the Better



Andrejs Doronins
Software Developer in Test

Boolean Checks

```
if (!gateClosed == false)
```

```
if (!gateClosed)
```

```
if (!isGateClosed)
```

```
if (isGateOpen)
```

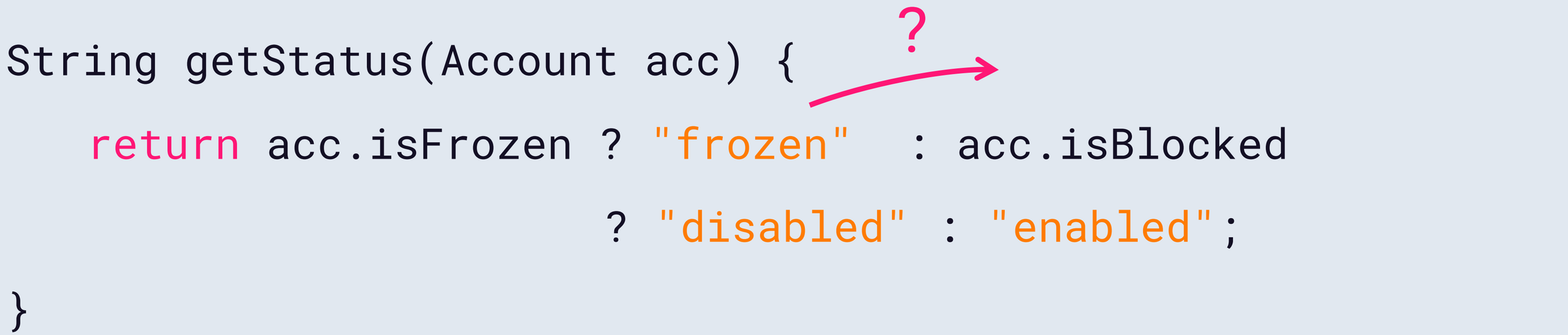
Wrap Conditions into Methods

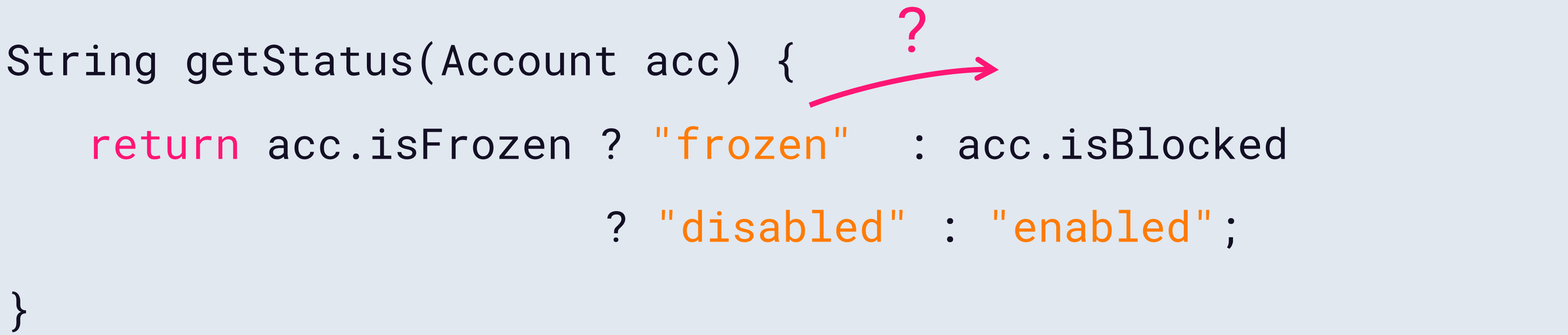
```
                        boolean isDay(int hour) { ... }

if(hour > 6 && hour < 22) {
    // day
}
```

Ternary expressions are great (and also evil)

```
String getStatus(Account acc) {
    return acc.isFrozen ? "disabled" : "enabled";
}
```


String getStatus(Account acc) {
?
return acc.isFrozen ? "frozen" : acc.isBlocked
? "disabled" : "enabled";
}


String getStatus(Account acc) {
?
return acc.isFrozen ? "frozen" : acc.isBlocked
? "disabled" : "enabled";
}

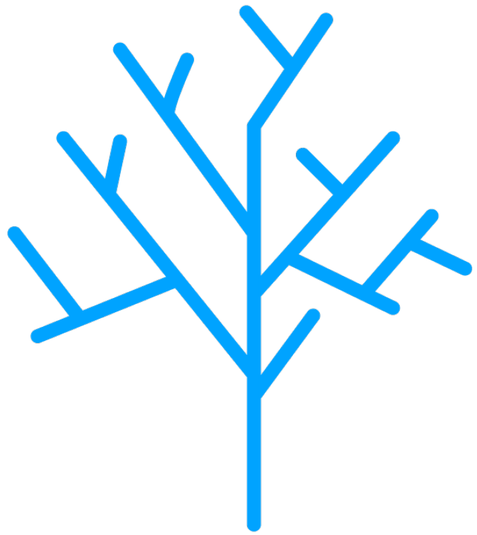
We should avoid bugs

Thanks, captain obvious!

```
// what can be simpler, right?

if() else {}
```

Ways to Branch



Terrible

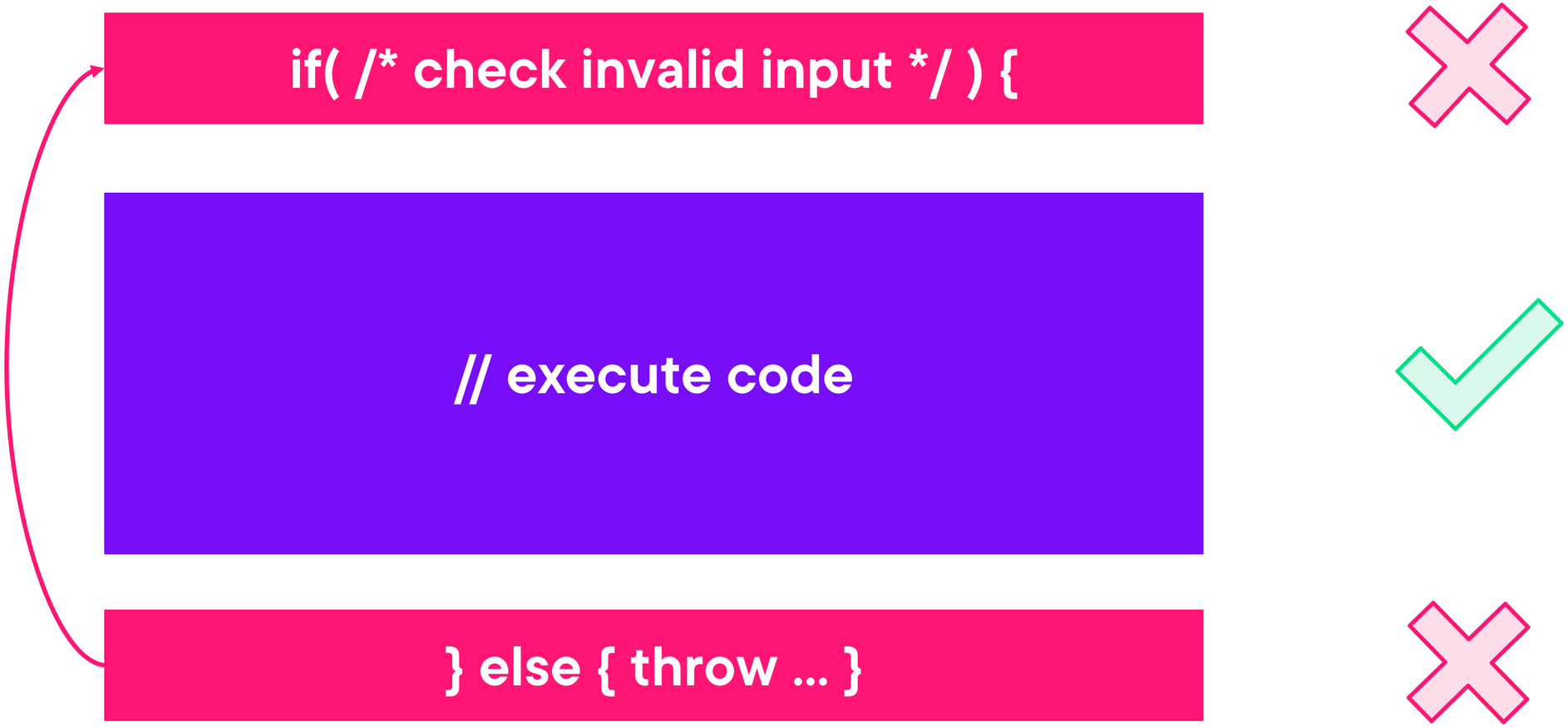
- Arrow anti-pattern

Less terrible

- if(input != null) { … }

Good

- fail fast & return early


if(/* check invalid input */) {
// execute code
} else { throw ... }

```
if( /* check invalid input*/ ) {
throw ...
}
```

```
// execute code
```

Safe beyond this point


Specific but lengthy
Generic but short

Prefer streams in simple cases

or other special-case
control of looping

```
for (int i = 0; i < someList.size(); i = i + 2) {
    // ...
}

for (int num: someList) {
    // ...
}
```

Imperative vs. Declarative

Imperative (for loops)

- How to achieve it, step by step?

Declarative (streams)

- What do you want to achieve?

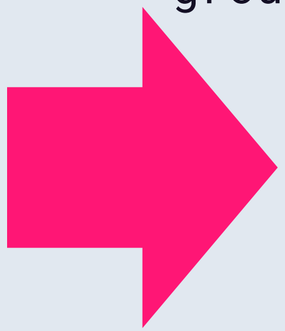
```
someList.stream()
        .filter(...)
        .map(transform)
        .sorted()
        .collect(...);
```

Prefer streams

Also, avoid streams

Prefer streams

Avoid overly complex streams



```
words.collect(
      groupingBy(word -> word.chars().sorted()
                .collect(StringBuilder::new,
                   (sb, c) -> sb.append((char) c),
                   StringBuilder::append).toString()))
     .values().stream()
     .filter(group -> group.size() >= minGroupSize)
     .map(group -> group.size() + ": " + group)
     .forEach(System.out::println);
```

```
list.stream()
    .filter(e -> complexCheck(e))
    .toList();


boolean complexCheck(String e) {
    //...
}
```

```
if(!isOpen == false) { }
```

```
if(isOpen) { }
```

```
if(isOpen && someInt > 10) { }
```

```
if(isSomeCheck()) { }
```

```
if(isValidInput1) {
    if(isValidInput2) {
        // ...
    }
}
```

```
if(isInputInvalid) {
    // throw, return, handle
}

// safely process the input
someCollection.stream()
// ...
```

Up Next:

Handling Exceptions Gracefully