Removing Dispensables

Andrejs Doronins
TEST AUTOMATION ENGINEER

Dispensables

Code that is not needed and can be removed

Dispensables

- Bad Comments
- Dead Code
- Duplicate Code
- Speculative Generality
- Lazy Class & Data Class

Comments

Misused or misplaced comments

Bad Comments Issues

Compensate for bad code

Clutter

Explanatory Comments

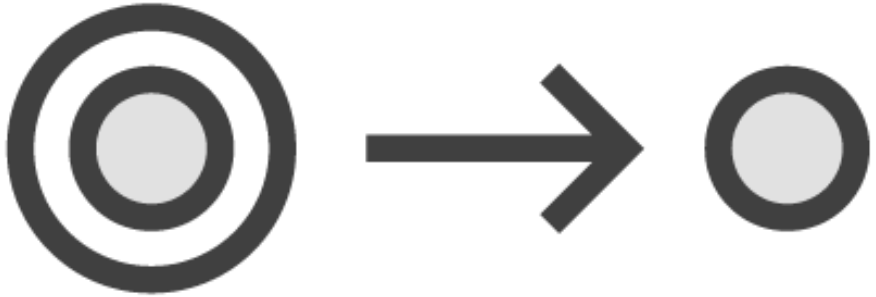
```
public String someComplexMethod(int i, String b){
    // check if...
    if( i < 21 && b.equals(...)) {
        // ....
    }
}
```

Comments should not compensate for bad code

Refactor code so that you don’t need comments

How to Fix Dispensables

Remove



Add functionality

(Applies to classes)

Bad Kind of Comments

- Redundant comments
- Wiki and VCS comments
- Misleading comments
- Misplaced comments
- Commented out code

Dead Code

Unused code

Dead Code Issues

Clutter – takes up space

Costs to read and understand

What if someone will need it tomorrow?

Not used, so let's remove it. We can revert with our VCS.

Duplicate code

Same code written multiple times in several places.

DRY
Don’t Repeat Yourself

Duplicate Code within a Method

```
void someMethod(){
    boolean isGold =

    if(order.getCustomer().isGoldMember()) {
        // ....
    }
    checkAnotherThing(order.getCustomer().isGoldMember());
}
```

Duplicate Code Across Methods

```
void someMethod(){
    if(order.getCustomer().isGoldMember()) { //...    }
 }
void anotherMethod(){
    int points = getBonusPoints(order.getCustomer().isGoldMember());
    //...
}
void isGoldMember(Order o){
}
```

Duplicate Code in Constructors

```
class Order{
    // ...
    Order(String v){
    this.voucher = v;
    }
    Order(String voucher, Customer c) {
    this.voucher = voucher;
    this.customer = c;
    }
}
```

Speculative Generality

Code created "just in case" to support anticipated future features that never get implemented

Speculative Generality Issues

Hard to understand and over-engineered code

YAGNI

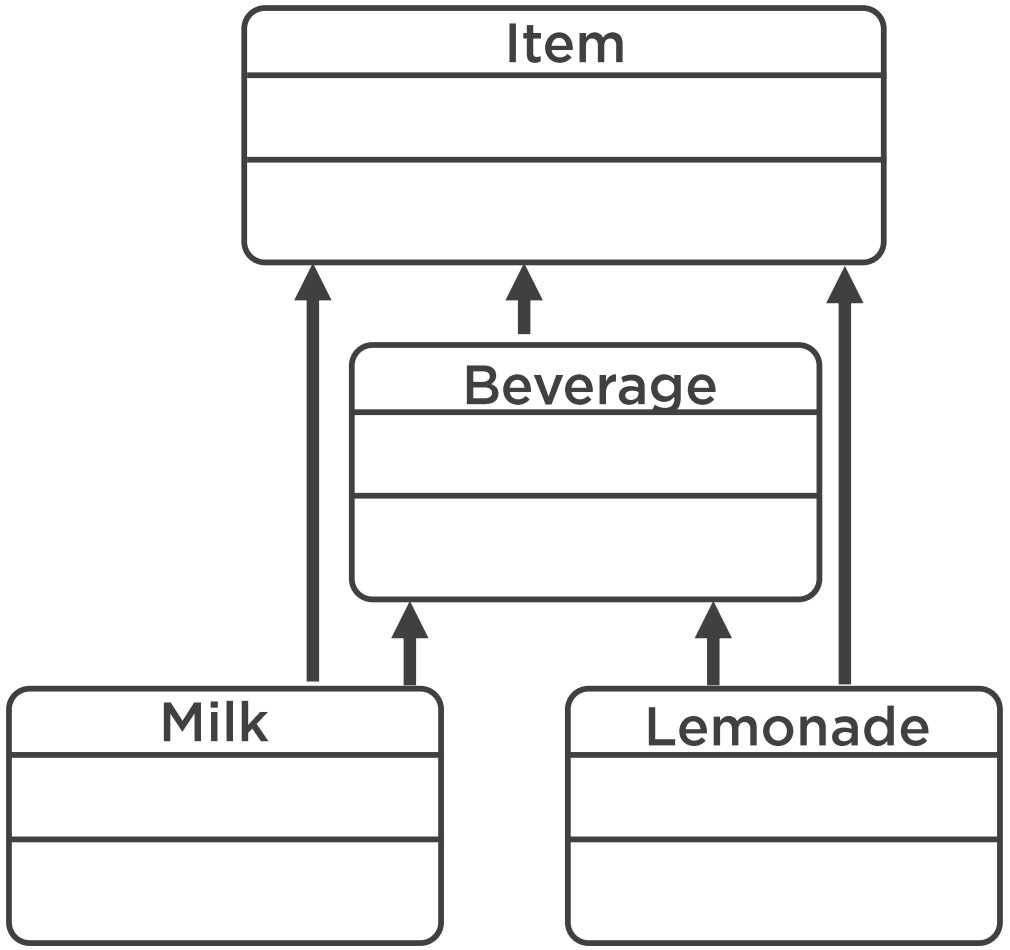
You ain’t gonna need it

Speculative Generality

```
void someMethod(){
    if(printToPdf) {
    // implementation
    } else if(printToWordDoc){
    // should implement soon
    }
}
```

Says a 2-year-old comment...

Remove Redundant Classes



Benefits Achieved

Simpler code is faster to work with

Lazy Class

A class that doesn't do enough to have a reason to exist

Class with Little Functionality

```
class OrderHelper {
    displayItems(Order o){// ...};
    doThings(){// ...};
    doOtherThings(){// ...};
    }
```

Lazy Class Issues

Maintaining a lazy class is not worth the effort

Data Class

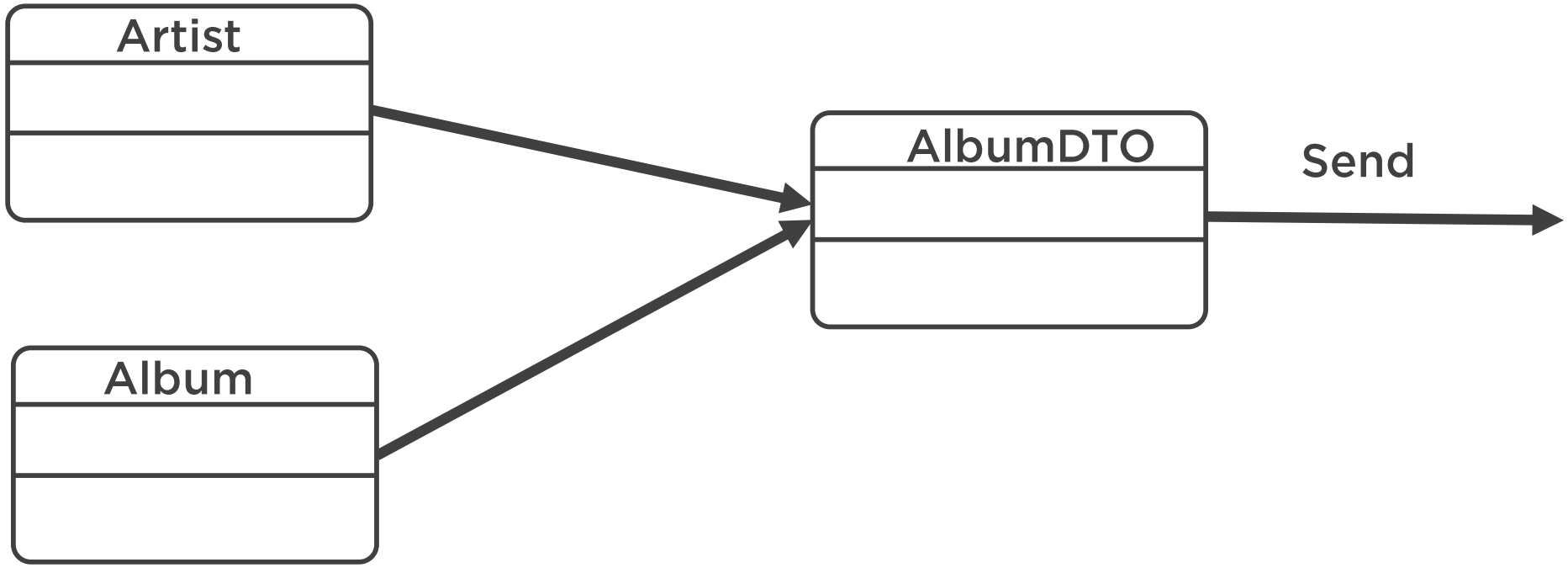
A class with getters and setters, and maybe an explicit constructor. No other methods. Used as containers for data.

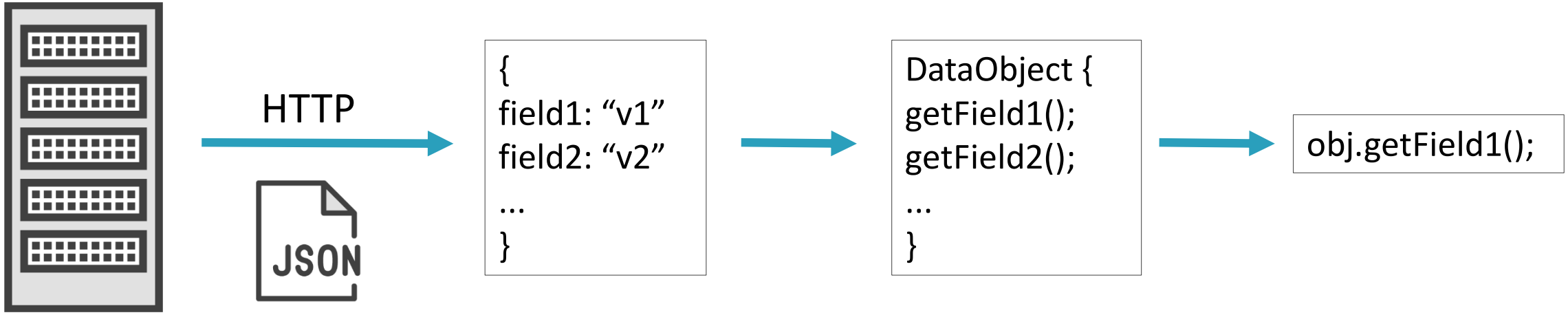
Data Classes are required by Data Transfer Object (DTO) pattern.

Data Transfer Object

A DTO is an object that carries data between two processes or two subsystems of in order to reduce the number of method calls

Data Transfer Object




HTTP
JSON
{
field1: “v1”
field2: “v2”
...
}
DataObject {
getField1();
getField2();
...
}
obj.getField1();

Don’t remove Data Classes
“just because”

Summary

- Remove Dispensables
- Don’t code for the future
- Remove or “promote” Lazy Classes. Keep Data Classes in some cases