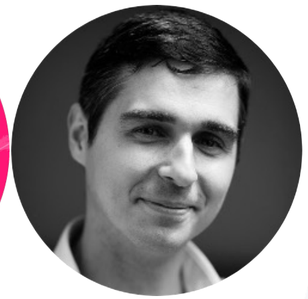
Creating Objects the Right Way



Andrejs Doronins
Software Developer in Test

```
class DayOfWeek {
String MONDAY = “MONDAY”;
String TUESDAY = “TUESDAY”;
// …
}
```

```
enum DayOfWeek {
MONDAY,
TUESDAY,
// …
}
```

class to group related constants

Best practice

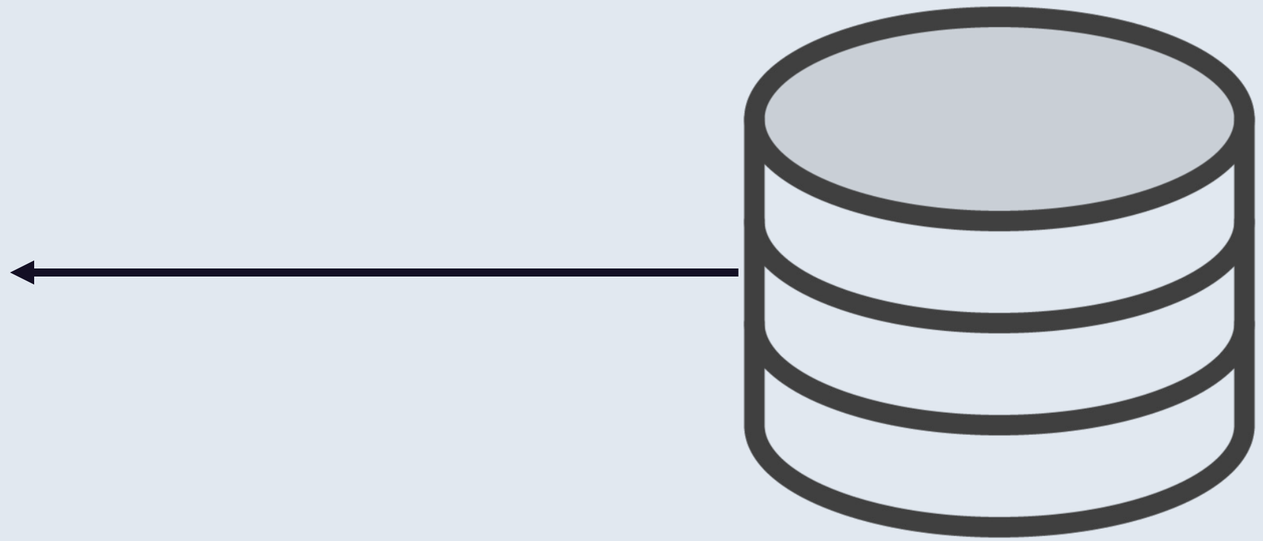
enum

class to only hold data

Best practice

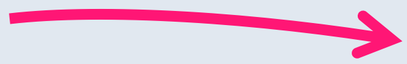
record

```
class Flight {
    Flight(a, b) {}
    from();
    to();
    toString();
    hashCode();
    //…
}
```

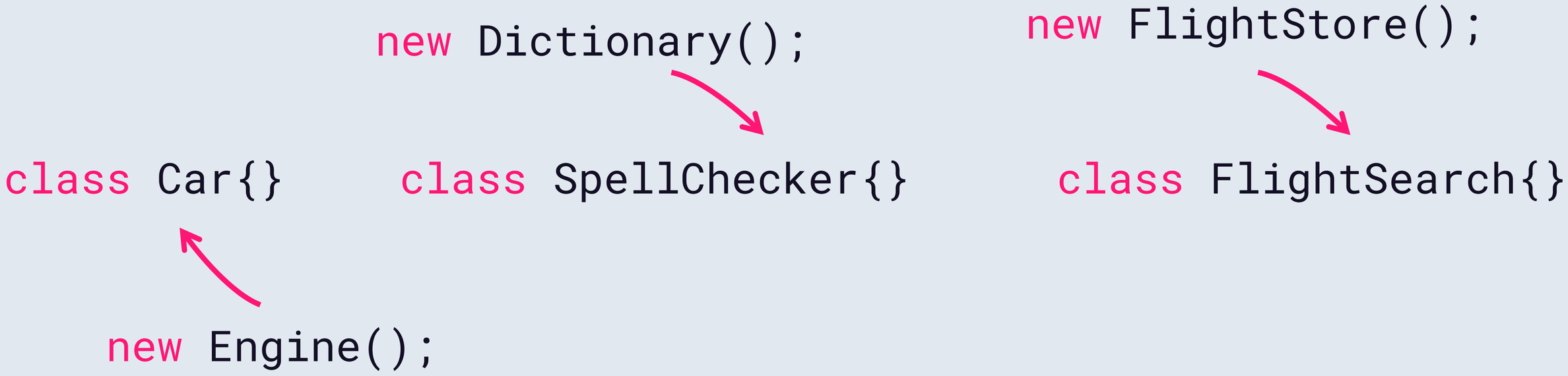


Less code = fewer bugs

```
class Flight {
    Flight(a, b) {}
    from();
    to();
    toString();
    hashCode();
    //…
}
```



```
record Flight(a, b) {


    // all boilerplate done for
    you


}
```


new Dictionary();
new FlightStore();
class Car{}
class SpellChecker{}
class FlightSearch{}
new Engine();

```
class FlightSearch {
  FlightRepo repo;

  FlightSearch() {
    this.repo = new FlightRepo();
  }
}

var search = new FlightSearch();
```

can't swap with

can't swap with JSON

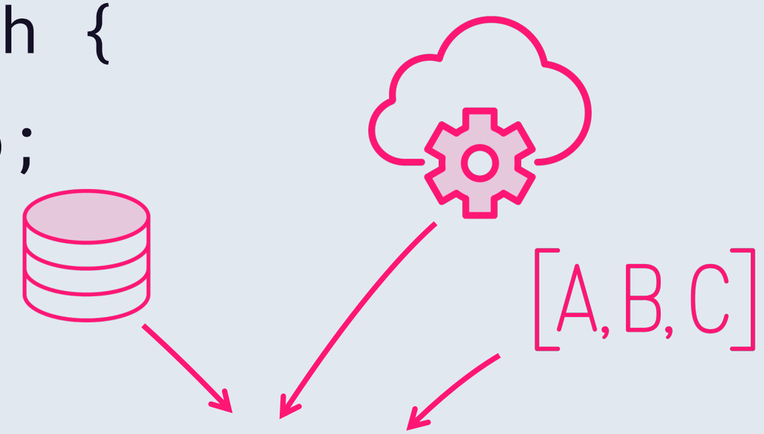
```
@Test
void searchTest() {
   var search = new FlightSearch();
   // act
   // assert
}
```

Dependency Injection

```
class FlightSearch {
  FlightRepo repo;


  FlightSearch() {
    this.repo = new FlightRepo();
  }
}
```

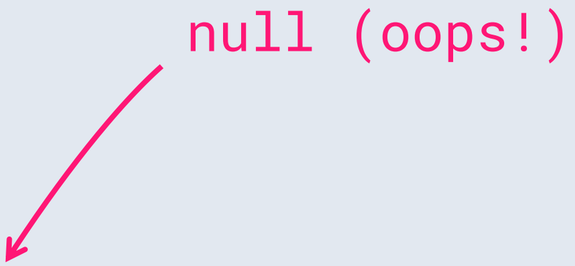
Dependency Injection

```
class FlightSearch {
  FlightRepo repo;

  FlightSearch(FlightRepo repo) {
    this.repo = repo;
  }
}
```

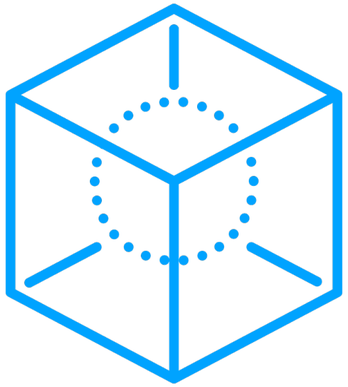


Dependency Injection

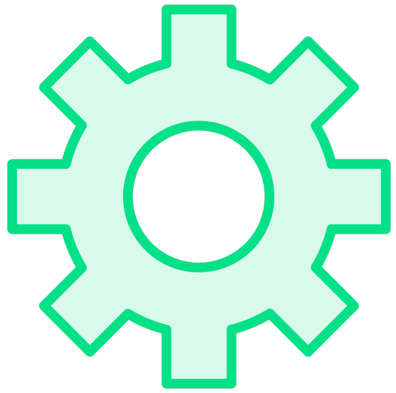
```
class FlightSearch {
  FlightRepo repo;


  FlightSearch(FlightRepo repo) {
    this.repo = repo;
  }
}
```



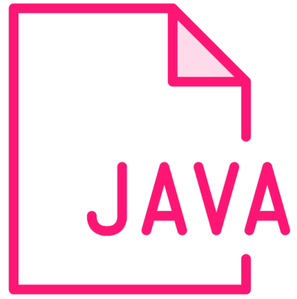
What Is Spring?



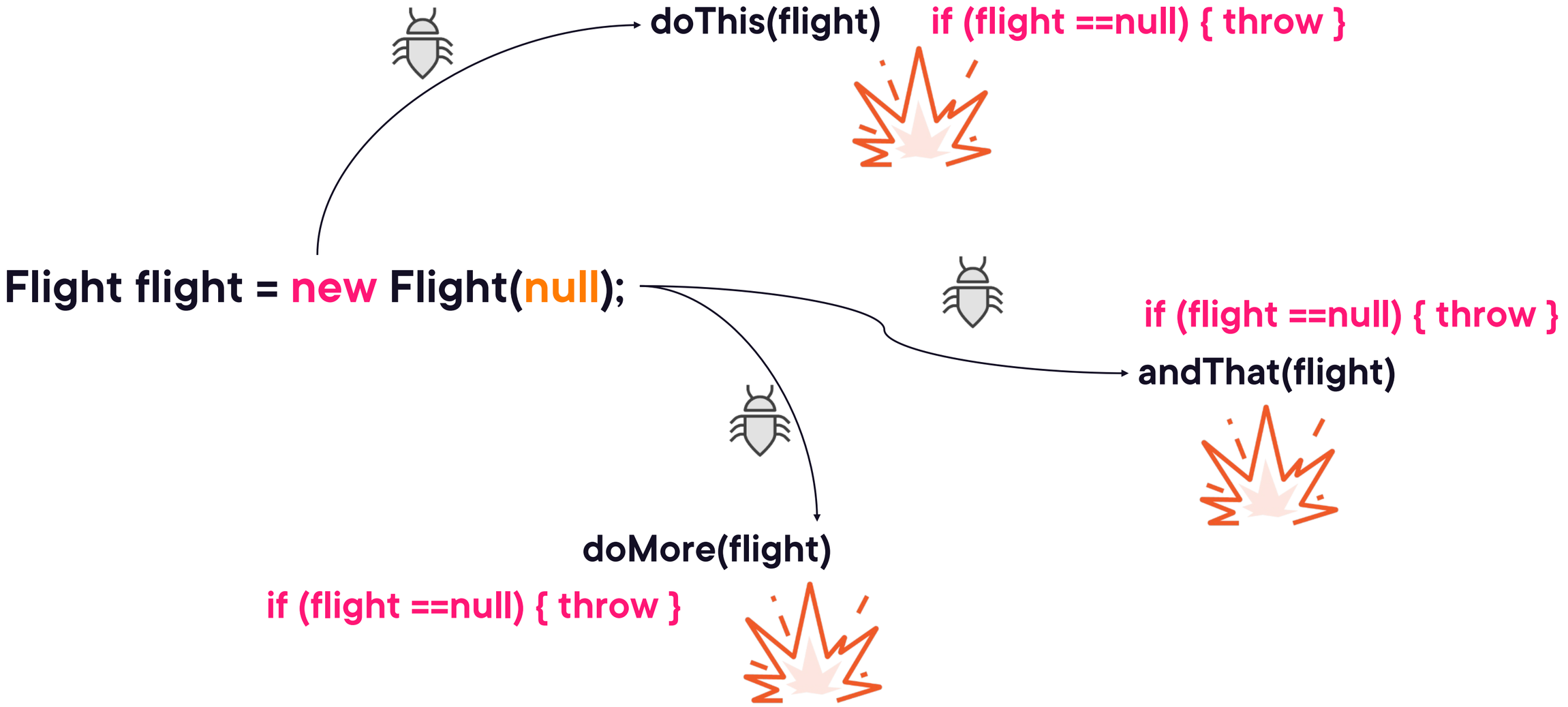
Inversion of Control Container

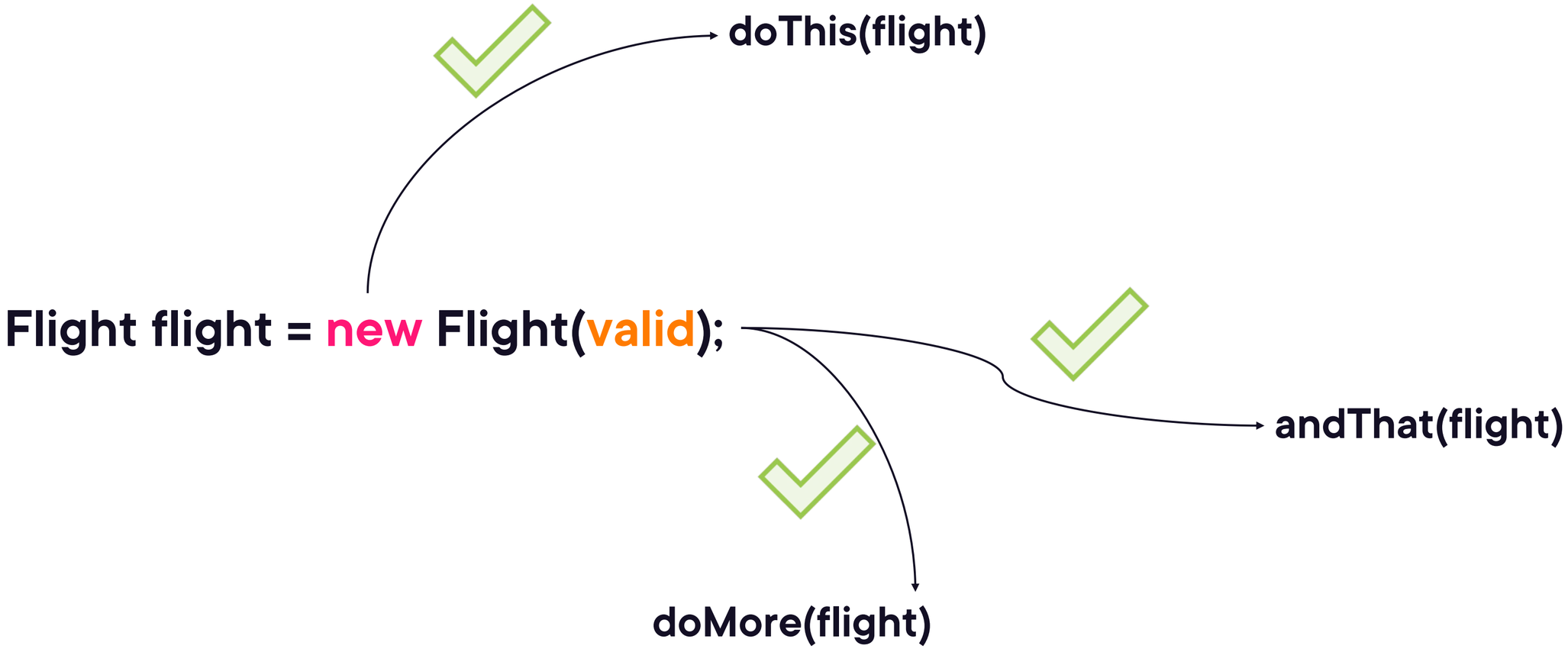


Dependency Injection



Java without Enterprise JavaBeans (EJBs)


doThis(flight)
if (flight ==null) { throw }
Flight flight = new Flight(null);
if (flight ==null) { throw }
andThat(flight)
doMore(flight)
if (flight ==null) { throw }


doThis(flight)
Flight flight = new Flight(valid);
andThat(flight)
doMore(flight)

Class Invariant

A property that always remains true for all instances of a class no matter what happens

Fail Fast

Check the value (input) for validity and throw an exception immediately

```
LocalDate date = LocalDate.of(2022, null, 10);
```

```
Exception in thread "main" java.lang.NullPointerException: month
    at java.base/java.util.Objects.requireNonNull(Objects.java:233)
    at java.base/java.time.LocalDate.of(LocalDate.java:251)
```

```
var thing = new Thing(new A());
```

```
var thing = new Thing(new A(), new B(),
                      new C(), new D());
```

```
var thing =  Thing.get(); // complexity hidden
```

```
var dbSearch =  FlightSearch.newDbSearch();
var apiSearch =  FlightSearch.newApiSearch();
var fileSearch =  FlightSearch.newFileSearch();
```

```
FlightSearch newDbSearch() {
   return new FlightSearch(a);
}
```

```
var dbSearch =  FlightSearch.newDbSearch();
```

```
FlightSearch newDbSearch() {
   return new FlightSearch(a, b ,c);
}
```

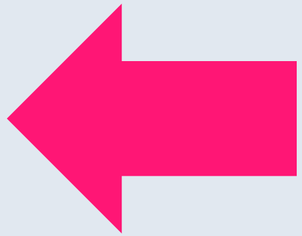

```
var dbSearch =  FlightSearch.newDbSearch();
```

Item 1: Consider static factory methods instead of constructors

Joshua Bloch, Effective Java, 3rd edition

Static Factory Methods in Java

```
LocalDate.now();

Optional.empty();

String.valueOf(true);

Collections.unmodifiableCollection(...);
```



Break naming conventions
(and that’s OK)

Improving Object Creation

Encapsulate with Static Factory Methods

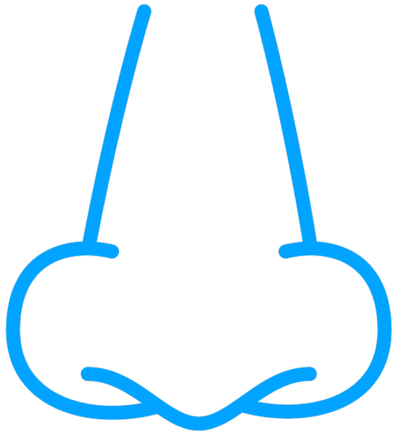
DRY with constructor chaining

```
Thing(String a, String b);

Thing(String a, String b, String c, int d);
        class NewThing {
          String a, b, c;
        }
```

Primitive obsession

You are using primitive types too much instead of custom objects

Code Smells



- Bloaters
- OO Abusers
- Change Preventers
- Couplers
- Dispensables

Fixing code smells

Java Refactoring: Best Practices

Andrejs Doronins

Constructor Telescoping

```
Pizza(int size) { ... }

Pizza(int size, boolean cheese) { ... }

Pizza(int size, boolean cheese, boolean ham) { ... }

Pizza(int size, boolean cheese, boolean ham, boolean
mushroom) { ... }
```

Builder Pattern

```
Flight flight = FlightBuilder()
        .from(“NYK”)
        .to(“LDN”)
        .capacity(150)
        .build();
```

Course: Java Design Patterns

```
record Flight(String from, String to) {
}
```

```
class FlightSearch {
  FlightRepo repo;

  FlightSearch(){
    this.repo = new FlightRepo();
  }
}
```

```
class FlightSearch {
  FlightRepo repo;

  FlightSearch(FlightRepo repo){
    this.repo = repo;
  }
}
```

```
class FlightSearch {
  FlightRepo repo;

  FlightSearch(FlightRepo repo){
    this.repo = requireNonNull(repo);
  }
}
```

```
class FlightSearch {
  FlightRepo repo;

  private FlightSearch(FlightRepo repo){
    this.repo = requireNonNull(repo);
  }
  public static FlightSearch newSearch(){…}
}
```

```
class FlightSearch {
   FlightRepo repo;
   Thing thing;
   FlightSearch(FlightRepo repo){
      this(repo, new Thing());
   }
   FlightSearch(FlightRepo repo, Thing thing){
      // all assignment here
   }

}
```

```
class Flight() {

  Flight(String from, String to, String etc) {…}
}
```

```
class Airport {

  String a, b;

}
```

Up Next:

Implementing Methods Best Practices