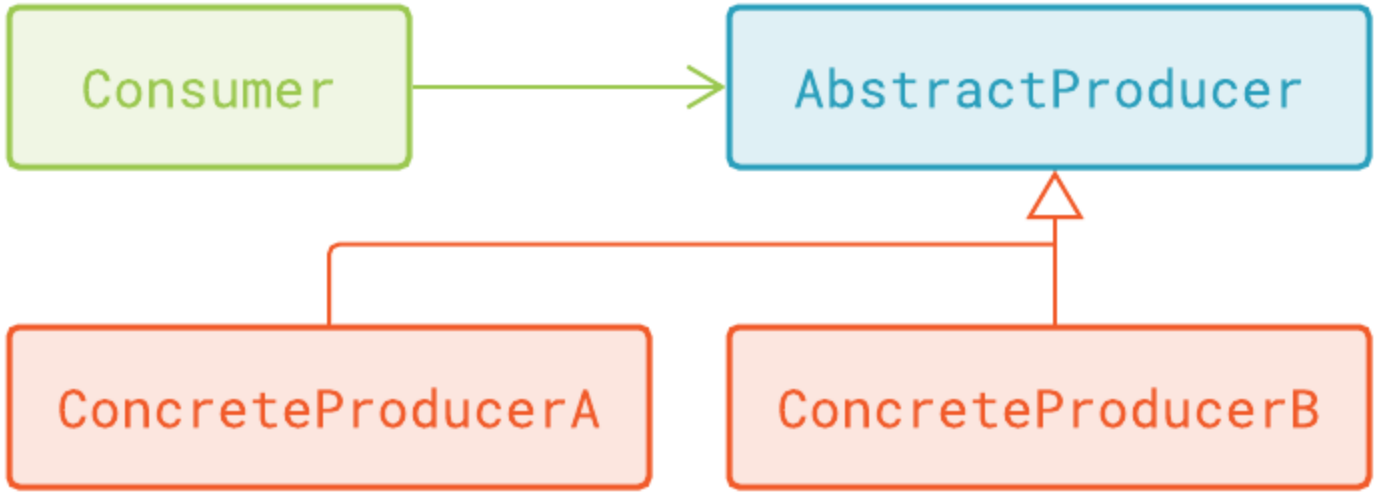
Turning Optional Calls into Calls on Optional Objects

Zoran Horvat

CEO AT CODING HELMET

@zoranh75 http://codinghelmet.com

Polymorphic Call vs. Branching




```
var service = producerA;
service.do();

...

var service = producerB;
```

```
void f(value) {
  if (value > 5)
    executeA();
  else
    executeB();
}
```

```
var selector = useA;

if (selector == useA)
  this.executeA();
else
  this.executeB();
...
var selector = useB;
```

Null Object implementation

```
class MissingPart : Part {
   ...
}
```

```
item.install(Part.MISSING, Warranty.VOID);
```

✗ Wrong!

```
long sensorsCount = items.stream()
  .map(item -> item.getSensor())
  .filter(sensor -> sensor != null)
  .count()
```

```
public Article install(Part sensor, Warranty extendedWarranty) {

}

public Article sensorNotOperational(LocalDate detectedOn) {

}
}
```

Null Object implementation

```
class MissingPart : Part {
   ...
}
```

Design which distinguishes substitute objects from the "real" ones is wrong

Design in which objects know their execution context is wrong

```
    public Article install(Part sensor, Warranty extendedWarranty) {


    }

    public Article sensorNotOperational(LocalDate detectedOn) {

    }
}
```

Summary

Modeling the missing objects

- Not always possible to use a substitute
- No reason to fall back to using null
- Use optional objects instead

Summary

Optional object defined

- It is a proper object
- It may contain another object
- Or it may contain nothing
- Forces you to supply both positive and negative scenarios
- All references remain non-null

Summary

Advanced topics on optional objects

- Optional looks like a stream
- Behaves the same as a stream with zero or one element
- Resulting design is resilient to bugs

Course Summary

Avoiding procedural and imperative coding

- Remove branching over Boolean flags
- Use substitutable objects instead
- The runtime type of an object is the live result of a Boolean test
- Branching becomes a call to a virtual method

Course Summary

Introducing Value Objects

- They are immutable
- They implement value-typed semantic
- Value Objects simplify code
- They help avoid defects

Course Summary

Removing null references

- Null Object and Special Case patterns remove most of the nulls
- `Optional<T>` type models missing objects
- Introduces a new programming model