PEMROGRAMAN LANJUT

Code Smells: Object Oriented Abuser

Oleh

Tri Hadiah Muliawati

Politeknik Elektronika Negeri Surabaya

2021

Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer

Object Oriented Abuser

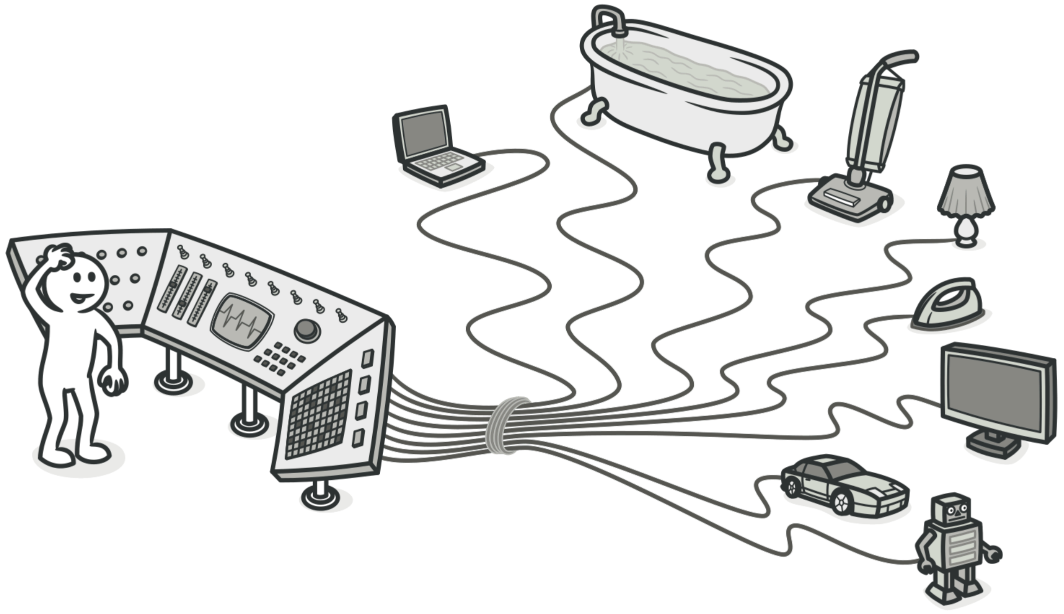
All these smells are incomplete or incorrect application of object-oriented programming principles.

Object Oriented Abuser

- Switch Statements
- Temporary Field
- Refused Bequest
- Alternative Classes with Different Interface

Switch Statements

- You have a complex switch operator or sequence of if statements.
- Code for a single switch can be scattered in different places in the program. When a new condition is added, you have to find all the switch code and modify it



Switch Statements: Refactoring

- **Extract Method** and **Move Method:** To isolate switch and put it in the right class.
- **Replace Conditional with Polymorphism:** If a switch is based on type code to get different behavior or data instead of using subclasses and polymorphism.
- **Replace Parameter with Explicit Methods:** If there aren’t too many conditions in the operator and they all call same method with different parameters.

Case Study: Replace Conditional with Polymorphism

```java
class Bird {
  // ...
  double getSpeed() {
    switch (type) {
      case EUROPEAN:
        return getBaseSpeed();
      case AFRICAN:
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
      case NORWEGIAN_BLUE:
        return (isNailed) ? 0 : getBaseSpeed(voltage);
    }
    throw new RuntimeException("Should be unreachable");
  }
}
```

```java
abstract class Bird {
  // ...
  abstract double getSpeed();
}
```

```java
class European extends Bird {
  double getSpeed() {
    return getBaseSpeed();
  }
}
class African extends Bird {
  double getSpeed() {
    return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
  }
}
class NorwegianBlue extends Bird {
  double getSpeed() {
    return (isNailed) ? 0 : getBaseSpeed(voltage);
  }
}
```

Case Study: Replace Parameter with Explicit Method

```java
void setValue(String name, int value) {
  if (name.equals("height")) {
    height = value;
    return;
  }
  if (name.equals("width")) {
    width = value;
    return;
  }
  Assert.shouldNeverReachHere();
}
```

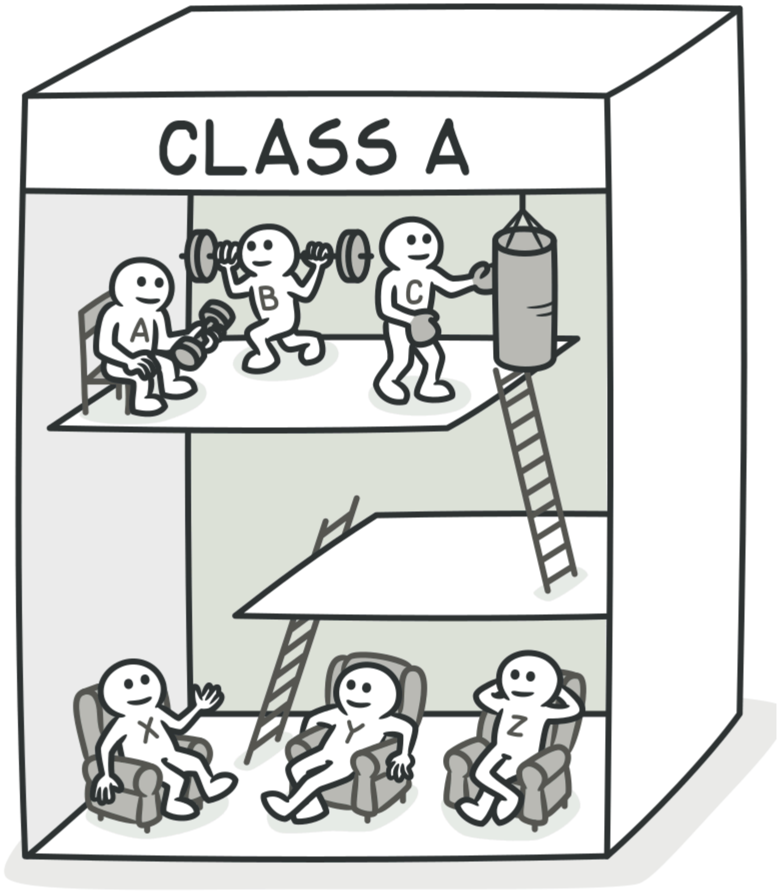
```java
void setHeight(int arg) {
  height = arg;
}
void setWidth(int arg) {
  width = arg;
}
```

Switch Statements: Exception

- When a switch operator performs simple actions, there’s no reason to make code changes.
- Often switch operators are used by factory design patterns (Factory Method or Abstract Factory) to select a created class

Temporary Field

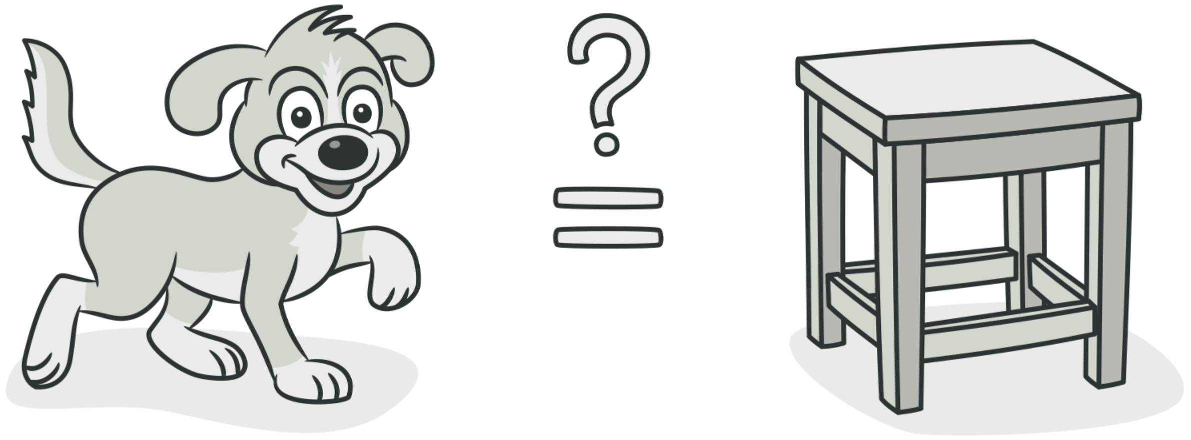
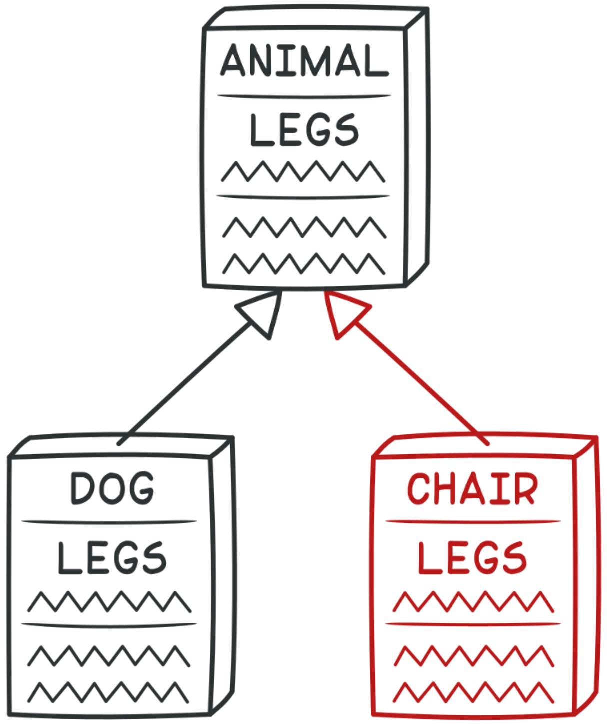
- Temporary fields get their values (and thus are needed by objects) only under certain circumstances. Outside of these circumstances, they're empty.
- Instead of creating a large number of parameters in the method, the programmer decides to create fields for this data in the class. These fields are used only in the algorithm and go unused the rest of the time.
- It contributes to low class cohesion.



Temporary Field: Refactoring

- **Extract Class:** Put temporary fields and all code operating on them into a separate class
- **Refactor into local variable:** If temporary field is used only in a single method

Refused Bequest

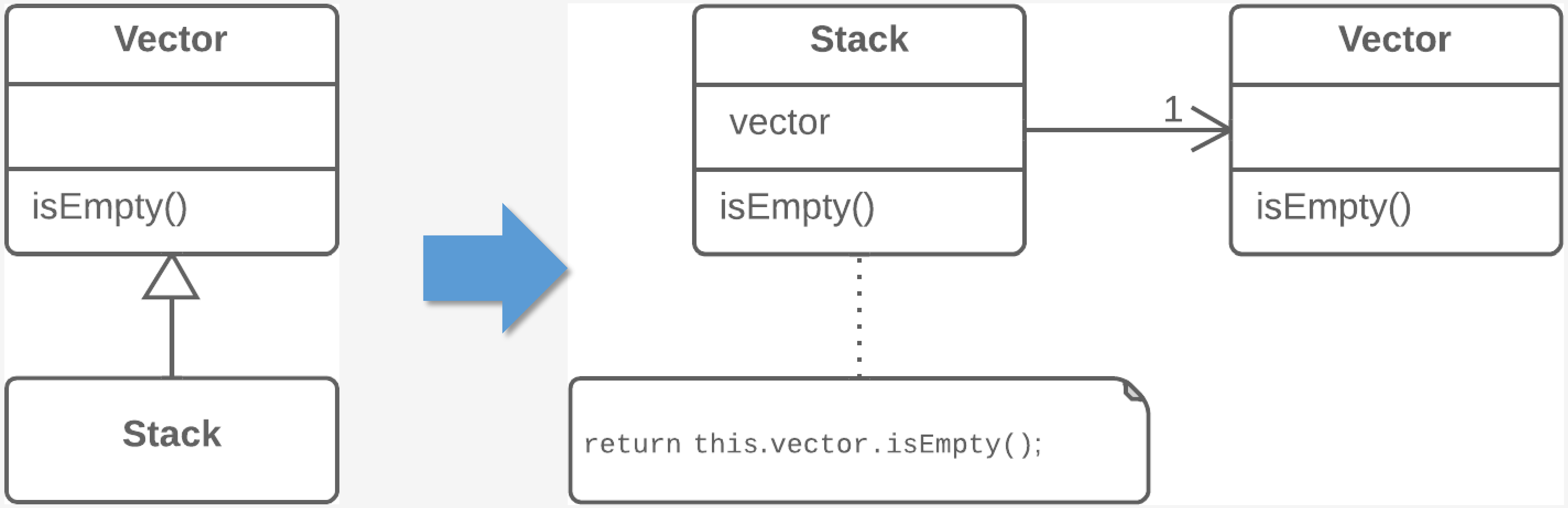



If a subclass uses only some of the methods and properties inherited from its parents, the hierarchy is off-kilter.

Refused Bequest: Refactoring

- **Replace Inheritance with Delegation:** If inheritance makes no sense and the subclass really does have nothing in common with the superclass or it’s not possible to inherit superclass data.
- **Extract Superclass:** If inheritance is appropriate, extract all fields and methods needed by the subclass from the parent class. Put them in a new subclass, and set both classes to inherit from it.

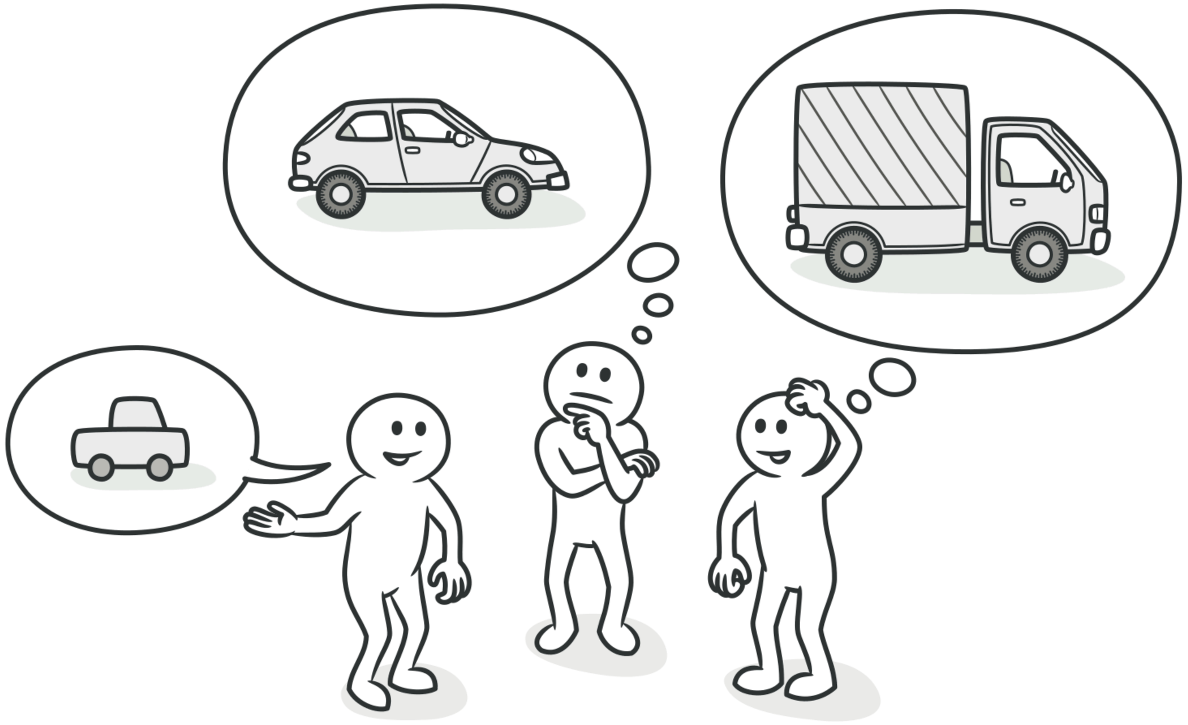
Case Study: Replace Inheritance with Delegation



Create a field and put a superclass object in it, delegate methods to the superclass object, and get rid of inheritance.

Alternative Classes with Different Interface

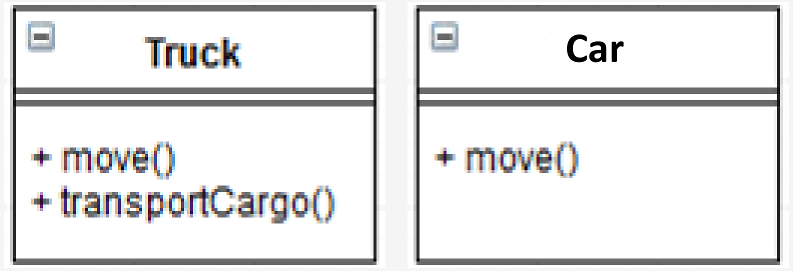
- Two classes perform identical functions but have different method names.
- The programmer who created one of the classes probably didn't know that a functionally equivalent class already existed.

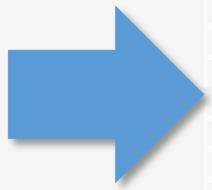


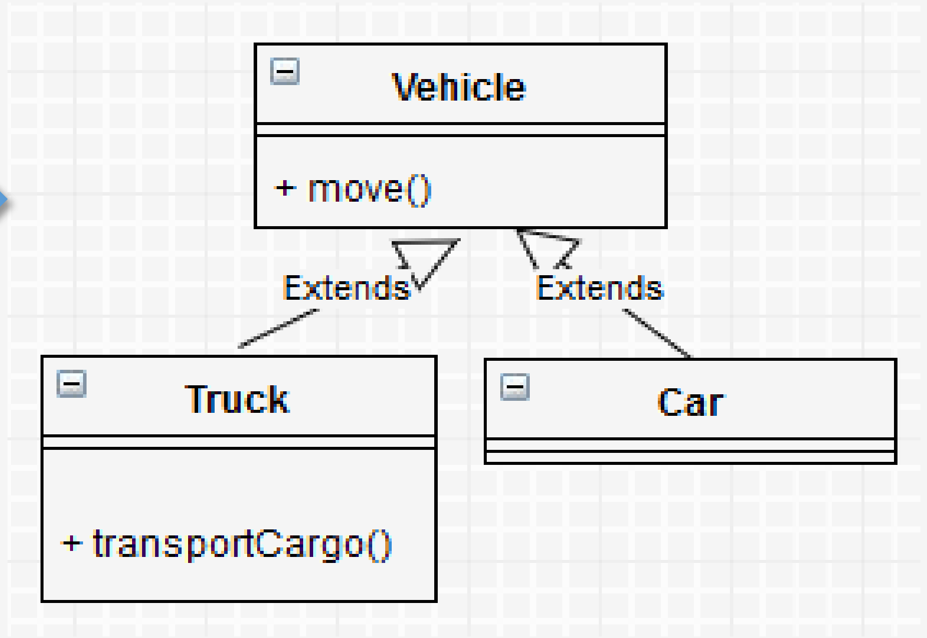
Alternative Classes with Different Interface: Refactoring

- **Rename Method:** To make them identical in all alternative classes.
- **Move Method, Add Parameter,** and **Parameterize Method:** To make the signature and implementation of methods the same.
- **Extract Superclass:** If only part of the functionality of the classes is duplicated

Case Study: Extract Superclass







References

- Fowler, Martin. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999.
- https://refactoring.guru/

bridge to the future

http://www.eepis-its.edu