Poznan University of Technology

Object Oriented Programming

Jakub Piotr Hamerliński, M.Eng.

Object Oriented Programming

Agenda

- Inheritance Is Evil
- Solution To Inheritance
- Task

Inheritance Is Evil

Object Oriented Programming

Inheritance Is Evil

We all agree that composition should take precedence over inheritance, but do we truly grasp why? In almost every article on the topic, authors have warned against using inheritance since it can affect your code. Does this imply that there are occasions when inheritance makes sense? David West stated that inheritance should not be used at all in OOP. Perhaps Dr. West is correct, and we ought to completely disregard the *extends* keyword in Java, for instance.

Object Oriented Programming

Inheritance Is Evil

It’s not only because with inheritance we introduce unnecessary coupling. "Inherit", as an English verb, means: "Derive (a quality, characteristic, or predisposition) genetically from one’s parents or ancestors." Deriving a characteristic from another object is a great idea, and it’s called subtyping.

Object Oriented Programming

Inheritance Is Evil

It perfectly fits into OOP and actually enables polymorphism: An object of class *Article* inherits all characteristics of objects in class *Manuscript* and adds its own. For example, it inherits an ability to print itself and adds an ability to submit itself to a conference:

```
1  interface Manuscript {
2    void print(Console console);
3  }
4  interface Article extends Manuscript {
5    void submit(Conference cnf);
6  }
```

Object Oriented Programming

Inheritance Is Evil

This is subtyping, and it's a perfect technique; whenever a manuscript is required, we can provide an article and nobody will notice anything, because type *Article* is a subtype of type *Manuscript*. But what does copying methods and attributes from a parent class to a child one have to do with "deriving characteristics?" Implementation inheritance is exactly that—copying—and it has nothing to do with the meaning of the word "inherit" I quoted above.

Object Oriented Programming

Inheritance Is Evil

Implementation inheritance is much closer to a different meaning: "Receive (money, property, or a title) as an heir at the death of the previous holder." Who is dead, you ask? An object is dead if it allows other objects to inherit its encapsulated code and data. This is implementation inheritance:

```
class Manuscript {
  protected String body;
  void print(Console console) {
    console.println(this.body);
  }
}
class Article extends Manuscript {
  void submit(Conference cnf) {
    cnf.send(this.body);
  }
}
```

Object Oriented Programming

Inheritance Is Evil

Implementation inheritance was developed as a tool for code reuse and has no place in OOP. Yes, it may appear practical at first, but in terms of object thinking, it couldn't be further from the truth. Implementation inheritance converts objects into data and procedure containers, much way getters and setters do. Of course, to avoid duplicating code, it's helpful to replicate part of the data and processes to a new object. However, this is not the point of things. They are still alive and not dead.

Solution To Inheritance

Object Oriented Programming

Solution To Inheritance

All classes should be final. A final class is one that can’t be extended via inheritance. Simply put, a class should say, "You can never break me; I’m a black box for you".

Object Oriented Programming
Solution To Inheritance

The only way to extend such a final class is through **decoration** of his children. Let's say I have the class *HTTPStatus*, and I don't like him.

```
1   class HTTPStatus implements Status {
2     private URL page = new URL("http://localhost");
3     @Override
4     public int read() throws IOException {
5       return HttpURLConnection.class.cast(
6         this.page.openConnection()
7       ).getResponseCode();
8     }
9   }
```

Object Oriented Programming

Solution To Inheritance

Well, I like him, but he's not powerful enough for me. I want him to throw an exception if HTTP status is over 400. I want his method, *read()*, to do more that it does now. A traditional way (the bad way!) would be to extend the class and overwrite his method:

```
1   class OnlyValidStatus extends HTTPStatus {
2     public OnlyValidStatus(URL url) {
3       super(url);
4     }
5     @Override
6     public int read() throws IOException {
7       int code = super.read();
8       if (code >= 400) {
9         throw new RuntimeException("Unsuccessful HTTP code");
10      }
11      return code;
12    }
13  }
```

Object Oriented Programming

Solution To Inheritance

Why is this wrong? It is very wrong because we risk breaking the logic of the entire parent class by overriding one of his methods. Remember, once we override the method *read()* in the child class, all methods from the parent class start to use his new version. We're literally injecting a new "piece of implementation" right into the class. Philosophically speaking, this is an offense!

Object Oriented Programming

Solution To Inheritance

On the other hand, to extend a final class, you have to treat him like a black box and decorate him with your own implementation (i.e. Decorator Pattern):

```
1   final class OnlyValidStatus implements Status {
2     private final Status origin;
3     public OnlyValidStatus(Status status) {
4       this.origin = status;
5     }
6     @Override
7     public int read() throws IOException {
8       int code = this.origin.read();
9       if (code >= 400) {
10        throw new RuntimeException("Unsuccessful HTTP code");
11      }
12      return code;
13    }
14  }
```

Object Oriented Programming

Solution To Inheritance

Make sure that this class is implementing the same interface as the original one: Status. The instance of HTTPStatus will be passed into him through the constructor and encapsulated. Then every call will be intercepted and implemented in a different way, if necessary. In this design, we treat the original object as a black box and never touch his internal logic. If you don't use that final keyword, anyone (including yourself) will be able to extend the class and... offend him :(So a class without final is a bad design.

Task

Object Oriented Programming

Task

This challenge is an English twist on the Japanese word game Shiritori. The basic premise is to follow two rules:

- First character of next word must match last character of previous word.
- The word must not have already been said.

Below is an example of a Shiritori game:

- ["word", "dowry", "yodel", "leader", "righteous", "serpent"] // valid!
- ["motive", "beach"] // invalid! - beach should start with "e"
- ["hive", "eh", "hive"] // invalid! - "hive" has already been said

Object Oriented Programming

Task

Write a Shiritori class that has two instance properties:

- words: an array of words already said.
- game_over: a boolean that is true if the game is over.

and three instance methods:

- play: a method that takes in a word as an argument and checks if it is valid (the word should follow rules #1 and #2 above). If it is valid, it adds the word to the words array, and returns the words array. If it is invalid (either rule is broken), it returns "game over" and sets the game_over boolean to true.
- restart: a method that sets the words array to an empty one [] and sets the game_over boolean to false. It should return "game restarted".
- printWords: a method that returns the words array.

Object Oriented Programming

Task

Example: my_shiritori = Shiritori.new()
my_shiritori.play("apple") → ["apple"]
my_shiritori.play("ear") → ["apple", "ear"]
my_shiritori.play("rhino") → ["apple", "ear", "rhino"]
my_shiritori.play("corn") → "game over" // prints "corn does not start with an 'o'".
my_shiritori.printWords → ["apple", "ear", "rhino"]
my_shiritori.restart() → "game restarted"
my_shiritori.printWords → []
my_shiritori.play("hostess") → ["hostess"]
my_shiritori.play("stash") → ["hostess", "stash"]
my_shiritori.play("hostess") → "game over" // prints "hostess has already been used".

Sources

Object Oriented Programming

Based on:

- https://www.yegor256.com/2014/11/20/seven-virtues-of-good-object.html
- https://www.yegor256.com/2016/09/13/inheritance-is-procedural.html
- Task created by E. Clark