Algorithms

History, State, Behavior, Enemies of OOP

YEGOR BUGAYENKO

Lecture #1 out of 8 90 minutes

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as website. Copyright belongs to their respected authors.

History

Original Intent

Object Thinking vs. Algorithms

Enemies of Object Thinking

How to Pass the Exam?

Read and Watch

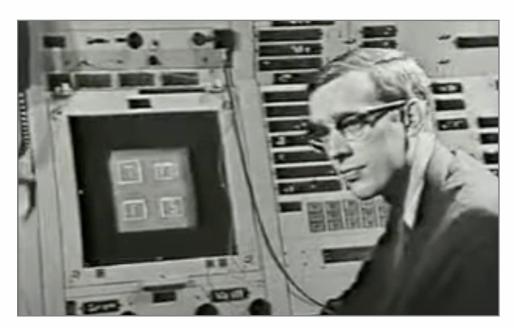
WARNING!

In the pursuit of academic enlightenment within this course, it is paramount to caution that the doctrines disseminated may present a potentially hazardous venture if employed in real-life software projects. This inherent risk arises from the potential incongruity with the broadly accepted canon of object-oriented programming and recognized best programming practices. If one remains resolute in their decision to adapt their coding methodologies to align with the principles propagated in this course, it would be prudent to employ a certain degree of foresight. A humorous, yet sincere suggestion, would be to secure alternate employment prior to a possible premature termination of one's current professional engagement.

Written by me, edited by ChatGPT

Chapter #1:
History

Who started it?



Ivan Sutherland's seminal **Sketchpad** application was an early inspiration for OOP, created between 1961 and 1962 and published in his Sketchpad Thesis in 1963. Any object could become a "master," and additional instances of the objects were called "occurrences". Sketchpad's masters share a lot in common with JavaScript's prototypal inheritance.

(c) Wikipedia

Who invented Objects, Classes, and Inheritance?



Simula was developed in the 1965 at the Norwegian Computing Center in Oslo, by Ole-Johan Dahl and Kristen Nygaard. Like Sketchpad, Simula featured objects, and eventually introduced classes, class inheritance, subclasses, and virtual methods. (c) Wikipedia

Simula-67: Sample Code

```
1 Class Figure;
   Virtual: Real Procedure square Is Procedure square;;
3 Begin
 End;
5 Figure Class Circle (c, r);
   Real c, r;
 Begin
   Real Procedure square;
   Begin
   square := 3.1415 * r * r;
   End;
11
12 End;
```

Who coined the "OOP" term?



Smalltalk was created in the 1970s at Xerox PARC by Learning Research Group (LRG) scientists, including Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Diana Merry, and Scott Wallace. (c) Wikipedia

Smalltalk: Sample Code

```
Object subclass: Account [
       balance
      Account class >> new [
           r
          r := super new. r init. ^r
      init [ balance := 0 ]
 Account extend [
      deposit: amount [ balance := balance + amount ]
11
12 a := Account new
a deposit: 42
```



"Everyone will be in a favor of OOP. Every manufacturer will promote his products as supporting it. Every manager will pay lip service to it. Every programmer will practice it (differently). And no one will know just what it is."

Tim Rentsch,Object Oriented Programming,ACM SIGPLAN Notices 17.9, 1982

Who made it all popular?



C++ was created by Danish computer scientist Bjarne Stroustrup in 1985, by enhancing C language with Simula-like features. C was chosen because it was general-purpose, fast, portable and widely used.

You may enjoy watching this one-hour dialog of Dr. Stroustrup and me.

History Intent O.T. Enemies Exam Literature

[Sketchpad Objects Simula-67 OOP Smalltalk Stroustrup C++ Languages Features]

C++: Sample Code

```
class Figure {
  virtual float square() = 0;
};
class Circle : public Figure {
  Circle(float c, float r) : c(c), r(r) {};
  float square() { return 3.1415 * r * r; };

private:
  float c, r;
};
```



"There are as many definitions of OOP as there papers and books on the topic"

Ole Lehrmann Madsen et al.,
 What Object-Oriented Programming May Be—And What
 It Does Not Have to Be, ECOOP'89

History Intent O.T. Enemies Exam Literature

[Sketchpad Objects Simula-67 OOP Smalltalk Stroustrup C++ Languages Features]



"I made up the term 'object-oriented,' and I can tell you I didn't have C++ in mind"

Alan Kay, OOPSLA'97

There was an interesting debate between Alan Kay and a few readers of my blog, in the comments section under this blog post: Alan Kay Was Wrong About Him Being Wrong

What happened later?

C++ was released in 1985. And then...

Erlang 1986

Eiffel 1986

Self 1987

Perl 1988

Haskell 1990

Python 1991

Lua 1993

JavaScript 1995

Ruby 1995

Java 1995

Go 1995

PHP3 1998

C# 2000

Rust 2010

Swift 2014



"There is no uniformity or an agreement on the set of features and mechanisms that belong in an OO language as the paradigm itself is far too general"

Oscar Nierstrasz,A Survey of Object-Oriented Concepts, 1989

Incomplete list of OOP features, so far:

Polymorphism

Nested Objects

Traits

Templates

Generics

Invariants

Classes

NULL

Exceptions

Operators

Methods

Static Blocks

Virtual Tables

Coroutines

Monads

Algebraic Types

Annotations

Interfaces

Constructors

Destructors

Lifetimes

Volatile Variables

Synchronization

Macros

Inheritance

Overloading

Tuple Types

Closures

Access Modifiers

Pattern Matching

Enumerated Types

Namespaces

Modules

Type Aliases

Decorators

Lambda Functions

Type Inference

Properties

Value Types

Multiple Inheritance

Events

Callbacks

NULL Safety

Streams

Buffers

Iterators

Generators

Aspects

Anonymous Objects

Anonymous Functions

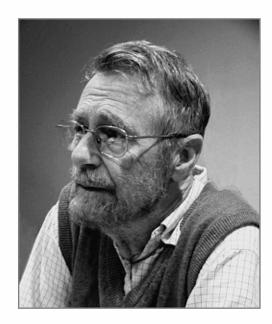
Reflection

Type Casting

Lazy Evaluation

Garbage Collection

Immutability



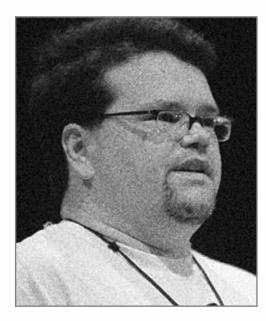
"Object oriented programs are offered as alternatives to correct ones... Object-oriented programming is an exceptionally bad idea which could only have originated in California."

Edsger W. Dijkstra, 1989



"C++ is a horrible language... C++ leads to really, really bad design choices... In other words, the only way to do good, efficient, and system-level and portable C++ ends up to limit yourself to all the things that are basically available in C."

Linus Torvalds, 2007Creator of Linux



"OO seems to bring at least as many problems to the table as it solves"

Jeff Atwood, 2007Co-founder of Stack Overflow



"I think that large objected-oriented programs struggle with increasing complexity as you build this large object graph of mutable objects. You know, trying to understand and keep in your mind what will happen when you call a method and what will the side effects be."

Rich Hickey, 2010Creator of Clojure

The <u>complexity</u> of object-oriented code remains its primary drawback



"Reading an OO code you can't see the big picture and it is often impossible to review all the small functions that call the one function that you modified"

Asaf Shelly, 2015Flaws of Object Oriented Modeling



"Object oriented programming promotes ease in designing reusable software but the long coded methods makes it unreadable and enhances the complexity of the methods"

Zeba Khanam, 2018
 Barriers to Refactoring: Issues and Solutions, International
 Journal on Future Revolution in Computer Science &
 Communication Engineering

Thus, we don't know anymore what exactly is object-oriented programming, and whether it helps us write better code :(

You can find more quotes in this blog post of mine: What's Wrong With Object-Oriented Programming?

Chapter #2:

Original Intent



"The contemporary mainstream understanding of objects (which is not behavioral) is but a pale shadow of the original idea and anti-ethical to the original intent"

David West,Object Thinking, 2004

You may enjoy watching our conversation with Dr. West: part I and part II.

A system is a composition of objects that are abstractions, which hide data and expose behavior*

^{*} This is how I understand the original intent.

1) What is an "abstraction"?



• Color: red

• Weight: 120g

• Price: \$0.99



```
var file = {
  path: '/tmp/data.txt',
  read: function() { ... },
  write: function(txt) { ... }
}
```

We deal with an abstraction as if it was a real thing, but eliminating unnecessary details. We do file.read() instead of "open file handler for data.txt, read byte by byte, store in byte buffer, wait for the end of file, return the result."

How many abstractions are needed?

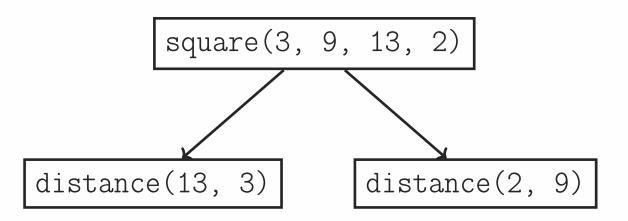
```
int square(x1, y1, x2, y2) {
  int w = x2 - x1;
  if (w < 0) { w = w * -1; }
  int h = y2 - y1;
  if (h < 0) { h = h * -1; }
  return w * h;
}</pre>
```

```
int distance(left, right) {
  int d = right - left;
  if (d < 0) { d = d * -1; }
  return d;
}
int square(x1, y1, x2, y2) {
  return distance(x2, x1)
     * distance(y2, y1);
}</pre>
```

There are two abstractions at the right snippet ("square" and "distance"), while only one abstraction at the left one (just "square").

Levels of abstraction

```
int distance(left, right) {
  int d = right - left;
  if (d < 0) { d = d * -1; }
  return d;
}
int square(x1, y1, x2, y2) {
  return distance(x2, x1)
  * distance(y2, y1);
}</pre>
```



Higher level abstractions must not know and/or rely on semantics of lower level abstractions.

2) What is "data hiding"?

```
f = new File("/tmp/data.txt");
f = new File("/tmp/data.txt");
// The data escapes the object! :(
p = f.getPath();
f = new File("/tmp/data.txt");
// The boolean data escapes too :)
done = f.delete();
FileUtils.deleteFile(p);
assert(done);
```

Obviously, some data must escape your objects.

3) What is "behavior exposing"?

This is so called "anemic" object:

```
var user = {
  login: 'jeff',
  password: 'swordfish',
  age: 32
}
function print(u) {
  console.log(`Hello, ${u.login},
  you are ${u.age} today!`);
}
print(user);
```

This object is "alive":

```
var user = {
  login: 'jeff',
  password: 'swordfish',
  age: 32,
  print: function() {
    console.log(`Hello, ${this.login},
    you are ${this.age} today!`);
  }
}
user.print();
```

An object as a function

```
int distance(left, right) {
  int d = right - left;
  if (d < 0) { d = d * -1; }
  return d; }
  int square(x1, y1, x2, y2) {
  return distance(x2, x1)
     * distance(y2, y1); }</pre>
```

```
class Distance {
  private int r; private int l;
  Distance(l, r) { l = l; r = r; }
  int value() {
    int d = right - left;
    if (d < 0) { d = d * -1; }
    return d; } }
  int square(x1, y1, x2, y2) {
    return new Distance(x2, x1).value()
    * new Distance(y2, y1).value(); } }</pre>
```

The Java object Distance on the right snippet is semantically equivalent to the C function distance() on the left one.

Identity, State, Behavior

```
class Circle {
                                                  1 // Identity:
    private float radius;
                                                  _{2} c1 = new Circle(42.0);
    Circle(float r) {
                                                  |c2| = \text{new Circle}(42.0);
      radius = r; }
                                                  _{4} c1 != c2;
    void getRadius() {
      return radius; }
                                                  6 // State:
                                                  _{7} c1 = new Circle(42.0);
    void setRadius(float r) {
                                                  | c2 = \text{new Circle}(42.0);
      radius = r; }
                                                  g c1.getRadius() == c2.getRadius();
    float square() {
      return 3.14 * radius * radius; }
                                                  10
                                                 11 // Behavior:
11 }
                                                 |c1| = \text{new Circle}(42.0);
                                                 c2 = \text{new Circle}(-42.0);
                                                 14 c1.square() == c2.square();
```

State vs. Behavior

```
class Circle {
   private float r;
  void setR(float r) { this.r = r; }
   float getR() { return this.r; }
 class FigureUtils {
    static float calcuateSquare(Circle c) {
      return 3.14 * c.getR() * c.getR();
10
11 Circle c = new Circle();
12 c.setR(42.0);
float s = FigureUtils.square(c);
```

```
class Circle {
  private float r;
  Circle(float r) { this.r = r; }
  float square() {
    return 3.14 * this.r * this.r;
  }
  Circle c = new Circle(42.0);
  float s = c.square();
```

How to decide what is <u>state</u> and what is behavior?

[Abstraction Rectangle Levels Rectangle Rectangle Function State FigureUtils Composition]

4) What is "composition"?

```
canvas = new Canvas();
canvas = new Canvas();
canvas.addCircle(new Circle(42));
canvas.draw();
canvas.draw();
canvas = new Canvas();
circle = new Circle(42);
circle.drawOn(canvas);
```

What is composition? What is the "right" composition?

Chapter #3:

Object Thinking vs. Algorithms

[While Buffer Loop Loop Composition]

While-Do loop

```
buffer = []
while true
    c = STDIN.readchar
break if c == "\n"
if buffer.length > 3
    STDOUT.puts buffer.join
    buffer = []
end
buffer << c
end</pre>
```

```
$ echo 'Hello, world!' | ruby a.rb
Hell
o, world!' | ruby a.rb
ruby
orld
```

[While Buffer Loop Loop Composition]

Buffer abstraction

```
buffer = []
while true
    c = STDIN.readchar
    break if c == "\n"
    if buffer.length > 3
        STDOUT.puts buffer.join
    buffer = []
    end
    buffer << c
end</pre>
```

```
1 class Buffer
    def initialize; @data = []; end
    def push(c)
      if @data.length > 3
        STDOUT.puts @data.join
        @data = []
      end
      @data << c
    end
10 end
11 buffer = Buffer.new
12 while true
   c = STDIN.readchar
   break if c == "\n"
    buffer.push c
16 end
```

[While Buffer Loop Loop Composition]

Loop abstraction

```
1 class Buffer
    def initialize; @data = []; end
    def push(c)
      if @data.length > 3
        STDOUT.puts @data.join
        @data = []
      end
      @data << c
    end
10 end
11 buffer = Buffer.new
12 while true
    c = STDIN.readchar
   break if c == "\n"
    buffer.push c
16 end
```

```
1 class Buffer
    # the same
3 end
  class Pull
    def initialize(b); @buf = b; end
    def again
      c = STDIN.readchar
     return false if c == "\n"
      @buf.push c
      true
10
11
    end
12 end
13 buffer = Buffer.new
14 pull = Pull.new(buffer)
15 while pull.again; end
```

[While Buffer Loop Composition]

Loop abstraction

```
1 class Buffer
   # the same
3 end
4 class Pull
    def initialize(b); @buf = b; end
    def again
      c = STDIN.readchar
   return false if c == "\n"
      @buf.push c
   true
10
    end
11
12 end
13 buffer = Buffer.new
14 pull = Pull.new(buffer)
15 while pull.again; end
```

```
class Buffer
    # the same
end
class Pull
    # the same
end
class Pulls
    def initialize(p); @pull = p; end
def fetch
    while @pull.again; end
end
Pulls.new(Pull.new(Buffer.new)).fetch
```

[While Buffer Loop Loop Composition]

Object composition

```
1 class Buffer
    def initialize; @data = []; end
    def push(c)
      if @data.length > 3
         STDOUT.puts @data.join
        @data = []
      end
      @data << c</pre>
    end
  end
10
11
12 class Pull
    def initialize(b); @buf = b; end
    def again
14
      c = STDIN.readchar
15
      return false if c == "\n"
16
      @buf.push c
17
```

```
true
18
    end
19
20 end
21
22 class Pulls
    def initialize(p); @pull = p; end
    def fetch
24
      while @pull.again; end
25
26
    end
27 end
28
29 Pulls.new(
    Pull.new(
30
       Buffer.new
31
32
33 ).fetch
```

Chapter #4:

Enemies of Object Thinking

[List]

What makes us think as algorithms

Global scope (static methods)

Anemic objects (getters)

Mutability (setters)

Workers ("-er" suffix)

NULL references

Type casting (reflection)

Inheritance

Chapter #5:

How to Pass the Exam?

Make a software project, which...

- ... is larger than 5,000 lines of functional code,
- ... compiles and works,
- ... doesn't have static methods,
- ... has no getters or public attributes,
- ... doesn't use NULL references,
- ... has only immutable objects,
- ... doesn't use inheritance.

Otherwise, just attend 75% of all lectures and you will get your "C".

Chapter #6:

Read and Watch

Read and watch:

David West, Object Thinking, 2004

Yegor Bugayenko, Elegant Objects, 2016

Read my 80+ blog posts about OOP, here

Watch my 15+ lectures about OOP, on YouTube

"Object Thinking" meetup, watch on YouTube.