Inheritance

Polymorphism, Subtyping, Reuse

Yegor Bugayenko

Lecture #8 out of 8
80 minutes

The slidedeck was presented by the author in this YouTube Video

Polymorphism

Implementation Inheritance

Chapter #1:

Polymorphism

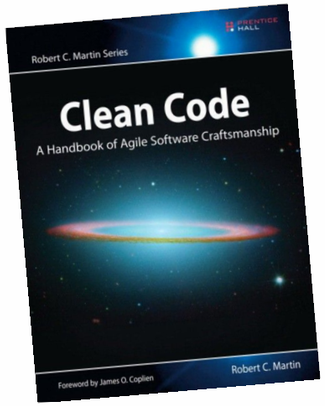
Liskov Substitution Principle (LSP)

"If for each object o_1 of type S there is an object o_2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o_1 is substituted for o_2, then S is a subtype of T."

— Barbara Liskov. Keynote Address — Data Abstraction and Hierarchy, 1987

[LSP SOLID Subtyping Generics Overloading]

SOLID (the “L” part)

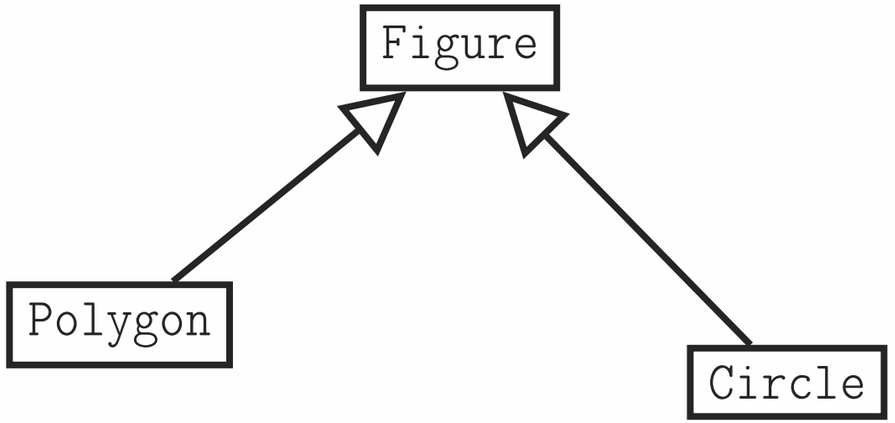


“Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.”

— Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship.* Pearson Education, 2008. doi:10.5555/1388398

Subtyping

```
1  interface Figure
2    float area();
3
4  interface Circle extends Figure
5    float perimeter();
6
7  interface Polygon extends Figure
8    int sides();
9
10 void paint(Figure f)
11   float s = f.area();
12   // ...
```



Circle $\sqsubseteq$ Figure

Circle <: Figure

Parametric Polymorphism (Generics)

```
class StackOfStrings {
  void push(String str) // ...
  String pop() // ...

class StackOfIntegers {
  void push(Integer num) // ...
  Integer pop() // ...

var s1 = new StackOfStrings();
s1.push("Hello, world!");

var s2 = new StackOfIntegers();
s2.push(42);
```

```
class <T> Stack<T> {
  void push(T item) // ...
  T pop() // ...
}

var s1 = new Stack<String>();
s1.push("Hello, world!");

var s2 = new Stack<Integer>();
s2.push(42);
```

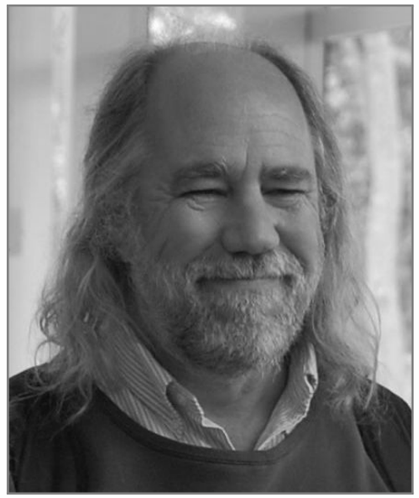
Ad Hoc Polymorphism (Method Overloading)

```
1  class Cart {
2    void add(int pid) // ...
3    void addString(String pid) {
4      this.add(Integer.parseInt(pid));
5    }
6  }
7
8  var c = new Cart();
9  c.add(42);
10 c.addString("17");
11 c.addString("Hello, world!");
```

```
1  class Cart {
2    void add(int pid) // ...
3    void add(String pid) {
4      this.add(Integer.parseInt(pid));
5    }
6  }
7
8  var c = new Cart();
9  c.add(42);
10 c.add("17");
11 c.add("Hello, world!");
```

Chapter #2:

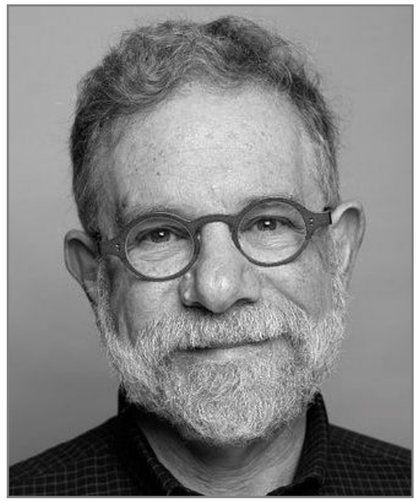
Implementation Inheritance



Grady Booch

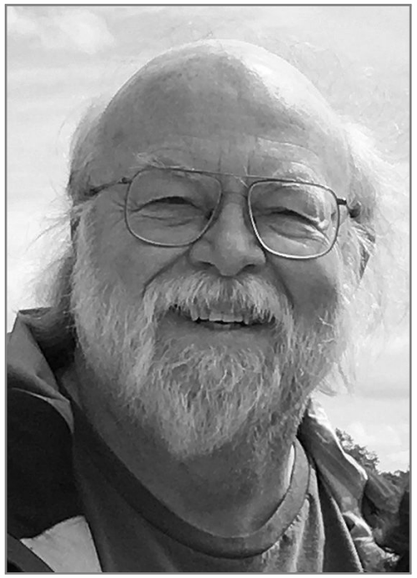
“However, there is tension between the concepts of coupling and inheritance because inheritance introduces significant coupling. On the one hand, weakly coupled classes are desirable; on the other hand, inheritance—which tightly couples superclasses and their subclasses—helps us to exploit the commonality among abstractions.”

— Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Connallen, and Kelli A. Houston. *Object-Oriented Analysis and Design With Applications*. Addison-Wesley, 1994. doi:10.5555/1407387



"The `extends` keyword is evil; maybe not at the Charles Manson level, but bad enough that it should be shunned whenever possible."

— Allen Holub. Why Extends Is Evil. https://jttu.net/holub2003extends, 9 2003. [Online; accessed 12-09-2024]

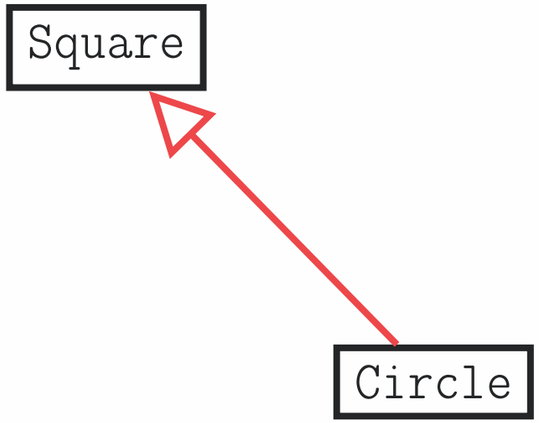


“Someone asked him: “If you could do Java over again, what would you change?” “I’d leave out classes,” he replied.”

— Allen Holub. Why Extends Is Evil. https://jttu.net/holub2003extends, 9 2003. [Online; accessed 12-09-2024]

Code Reuse

```
1  class Square
2    private float width;
3    float area()
4      return width * width;
5
6  class Circle extends Square
7    Circle(float radius)
8      super(radius);
9    @Override float area()
10     return 3.14 * super.area();
```



Here, the `Circle` is not a `Square`. It merely reuses the code that was negligently left open in the `Square`.

> Inheriting means “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.

Composition over Inheritance

Implementation Inheritance:

```
1  class Square
2    private float width;
3    float area()
4      return width * width;
5
6  class Circle extends Square
7    Circle(float radius)
8      super(radius);
9    @Override float area()
10     return 3.14 * super.area();
```

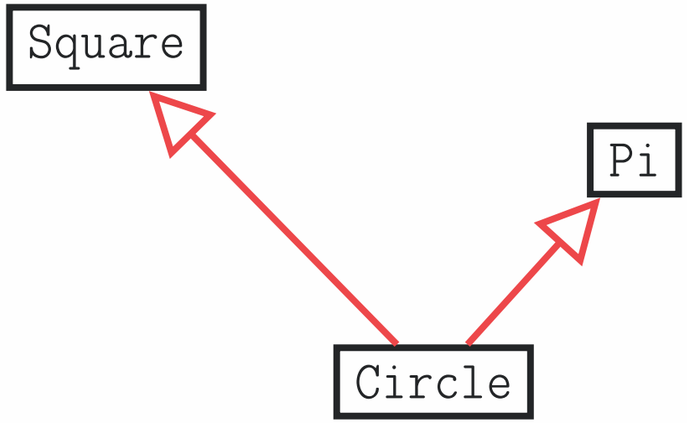
Composition:

```
1  final class Square
2    private float width;
3    float area()
4      return width * width;
5
6  final class Circle
7    private Square s;
8    Circle(float radius)
9      this.s = new Square(radius);
10   float area()
11     return 3.14 * s.area();
```

> All classes, without exceptions, should be either `final` or `abstract`

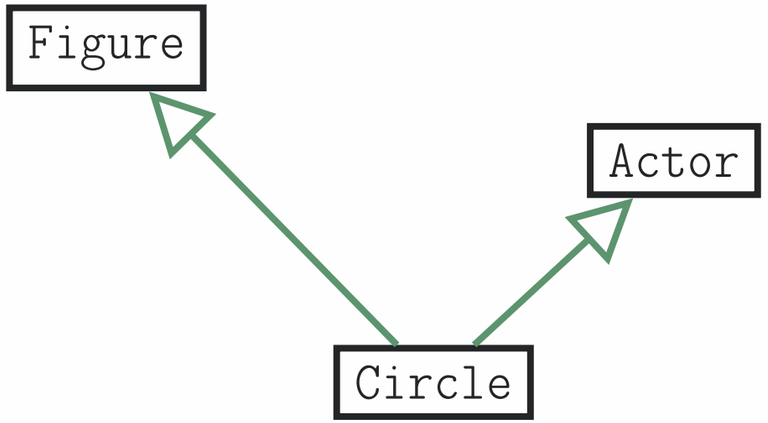
Multiple Inheritance

```
1  class Pi
2    float value()
3      return 3.1415926;
4
5  class Square
6    private float width;
7    float area()
8      return width * width;
9
10 class Circle extends Square, Pi
11   Circle(float r): Square(r), Pi() {}
12   virtual float area()
13     return Pi.value() * Square.area();
```



Multiple Super Types

```
1  interface Actor
2    void move(int dx, int dy);
3
4  interface Figure
5    float area();
6
7  class Circle implements Figure, Actor
8    Circle(float r)
9    @Override float area()
10     // ...
11   @Override void move(int dx, int dy)
12     // ...
```



Bibliography

Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Connallen, and Kelli A. Houston. *Object-Oriented Analysis and Design With Applications*. Addison-Wesley, 1994. doi:10.5555/1407387.

Allen Holub. Why Extends Is Evil. `https://jttu.net/holub2003extends`, 9 2003. [Online; accessed 12-09-2024].

Barbara Liskov. Keynote Address — Data Abstraction and Hierarchy, 1987.

Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2008. doi:10.5555/1388398.