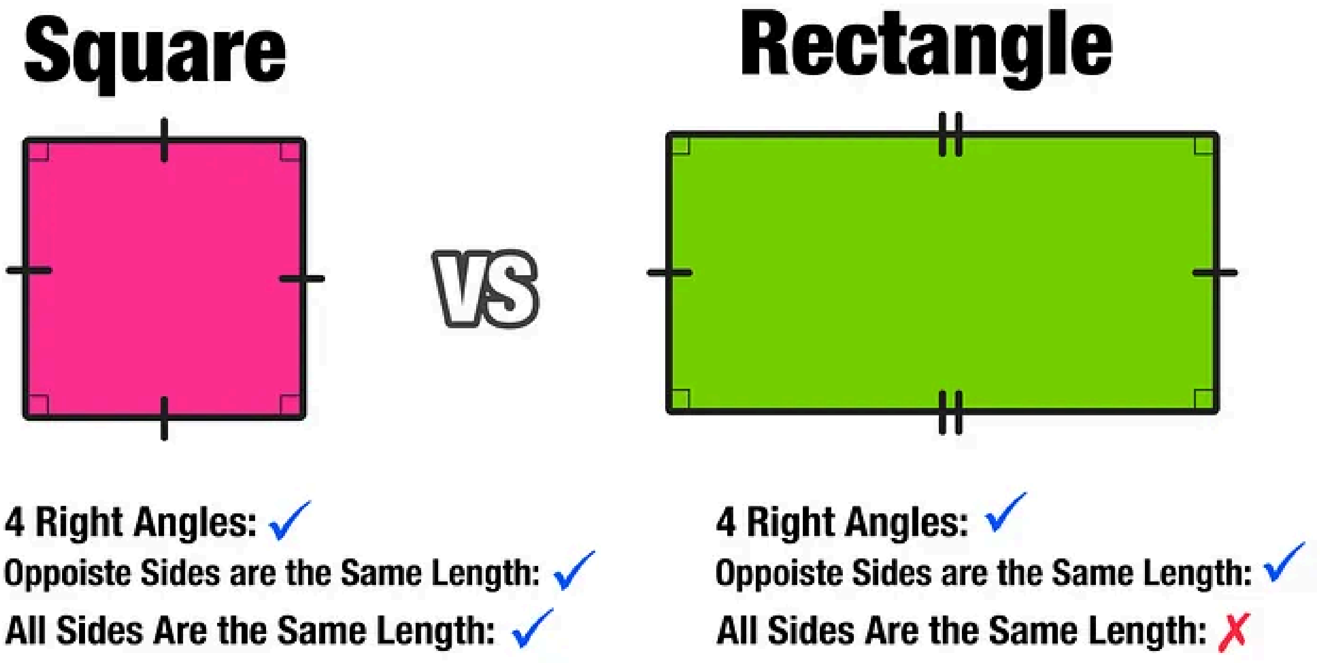
Square, Rectangle and the Liskov Substitution Principle

Alexandre Dutertre · Follow

5 min read · Apr 28, 2023

1 1


Square
VS
Rectangle
4 Right Angles: ✓
Oppoiste Sides are the Same Length: ✓
All Sides Are the Same Length: ✓
4 Right Angles: ✓
Oppoiste Sides are the Same Length: ✓
All Sides Are the Same Length: ✗

When creating classes for shapes, it's easy to imagine a square as just being a rectangle with all sides having the same length, like in geometry. Thus, making the `Square` class inherits from the `Rectangle` class. However, this could bring unexpected behavior from our program, as it violates the **Liskov substitution principle** (LSP). We will see how we can fix that problem.

What is the Liskov substitution principle

The Liskov substitution principle is the third principle of the mnemonic acronym **SOLID**:

- **S**ingle-responsability principle
- **O**pen-closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

It was introduced by Barbara Liskov, during a conference called ***Data abstraction and hierarchy***, in 1987. This principle defines the subtype notion as such:

> *if f(x) is a property provable about objects x of type T, then f(y) should be true for objects y of type S where S is a subtype of T.*

An objet can be replaced by a sub-object without breaking the program, as what holds for T-objects holds for S-objects is what must be understood.

A square breaks that principle since it doesn't have a different `height` and `width`. So not everything true for `Rectangle` is true for `Square`. Let's see it using code and the `Shape`, `Rectangle` and `Square` classes.

Shape class

```
class Shape
{
    public virtual int Area()
    {
        throw new NotImplementedException("Area() is not implemented");
    }
}
```

Rectangle class

```
class Rectangle : Shape
{
    private int width;
    private int height;

    public int Width
    {
        get => width;
        set
        {
            if (value < 0)
                throw new ArgumentException("Width must be greater than or equal
            width = value;
        }
    }

    public int Height
    {
        get => height;
        set
        {
            if (value < 0)
                throw new ArgumentException("Height must be greater than or equa
            height = value;
        }
    }

    public new int Area() => width * height;
    public override string ToString() => $"[Rectangle] {width} / {height}";
}
```

Square class

```
class Square : Rectangle
{
    private int size;

    public int Size
    {
        get => size;
        set
        {
            if (value < 0)
                throw new ArgumentException("Size must be greater than or equal
            size = value;
            Height = value;
            Width = value;
        }
    }

    public override string ToString() => $"[Square] {size} / {size}";
}
```

If we were to use the setter `Size` , the program would still work as intended. However if we use directly the setters `Width` and `Height` with our square, it would break the program.

```
class Program
{
    static void Main(string[] args)
    {
        Square aSquare = new Square();
        try
        {
            aSquare.Width = 12;
            aSquare.Height = 8;

            Console.WriteLine("aSquare width: {0}", aSquare.Width);
            Console.WriteLine("aSquare height: {0}", aSquare.Height);
            Console.WriteLine("aSquare size: {0}", aSquare.Size);
            Console.WriteLine("aSquare area: {0}", aSquare.Area());
            Console.WriteLine(aSquare.ToString());
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
    }
}
```

Here is the output obtained when running the `dotnet run` command:

```
aSquare width: 12
aSquare height: 8
aSquare size: 0
aSquare area: 96
[Square] 0 / 0
```

We can see that there is a problem when getting the `Size`, using the `Area()` and `ToString()` methods since the field `size` was never assigned. We're using the methods as if we were expecting a rectangle, aside from `aSquare.Size`, but it behaves differently since it's a square.

Overriding the getters and setters isn't an option since it modifies the behavior of the methods from the `Rectangle` class which violates the LSP.

Possible correction

One possible correction to this problem, would be to make the `Shape` class **`abstract`** and make both `Rectangle` and `Square` inherits from the `Shape` class. Here is a possible implementation:

Shape class

```
abstract class Shape
{
    protected int width;
    protected int height;

    public abstract int Area();
}
```

Rectangle class

```
class Rectangle : Shape
{
    public int Width
    {
        get => width;
        set
        {
            if (value < 0)
                throw new ArgumentException("Width must be greater than or equal
            width = value;
        }
    }

    public int Height
    {
        get => height;
        set
        {
            if (value < 0)
                throw new ArgumentException("Height must be greater than or equa
            height = value;
        }
    }

    public override int Area() => width * height;

    public override string ToString() => $"[Rectangle] {width} / {height}";
}
```

Square class

```
class Square : Shape
{
    public int Size
    {
        get => width;
        set
        {
            if (value < 0)
                throw new ArgumentException("Size must be greater than or equal
            width = value;
            height = value;
        }
    }

    public int Width
    {
        get => width;
        set
        {
            if (value < 0)
                throw new ArgumentException("Width must be greater than or equal
            width = value;
            height = value;
        }
    }

    public int Height
    {
        get => height;
        set
        {
            if (value < 0)
                throw new ArgumentException("Height must be greater than or equa
            width = value;
            height = value;
        }
    }

    public override int Area() => width * width;

    public override string ToString() => $"[Square] {width} / {width}";
}
```

If we modify `main.cs` to add output for a rectangle:

```
class Program
{
    static void Main(string[] args)
    {
        Square aSquare = new Square();
        Rectangle aRectangle = new Rectangle();

        try
        {
            aSquare.Width = 12;
            aSquare.Height = 8;
            aRectangle.Width = 12;
            aRectangle.Height = 8;

            Console.WriteLine("aSquare width: {0}", aSquare.Width);
            Console.WriteLine("aSquare height: {0}", aSquare.Height);
            Console.WriteLine("aSquare size: {0}", aSquare.Size);
            Console.WriteLine("aSquare area: {0}", aSquare.Area());
            Console.WriteLine(aSquare.ToString());
            Console.WriteLine("------------------------");
            Console.WriteLine("aRectangle width: {0}", aRectangle.Width);
            Console.WriteLine("aRectangle height: {0}", aRectangle.Height);
            Console.WriteLine("aRectangle area: {0}", aRectangle.Area());
            Console.WriteLine(aRectangle.ToString());
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
        }
    }
```

We obtain the following output.

```
aSquare width: 8
aSquare height: 8
```

```
aRectangle width: 12
aRectangle height: 8
aRectangle area: 96
[Rectangle] 12 / 8
```

The `Shape` class is made `abstract` as by itself it doesn't refer to anything concrete, and `abstract` classes are an important part in this principle. The `Rectangle` and `Square` classes inherits from this class as they are shapes.

By separating the `Rectangle` and `Square` classes, we ensure that there are no problems during the code execution as the classes holds their own implementation of the `Area()` method and their own definition for `Width` and `Height`.

Overriding isn't a problem here, because it doesn't modify the expected behavior for `Area()`, which is to return the area. That's why it was made `abstract` in the `Shape` class, to indicate it must be overridden.

Conclusion

- Contrary to geometry, squares aren’t rectangles in Object-Oriented Programming
- Subclasses must respect the Liskov substitution principle
- The LSP dictates that everything true for the class, must be true for the subclass
- The problem can be fixed by making `Square` and `Rectangle` inherits from `Shape` and have their own definitions of the getters and setters and `Area()` without modifying the expected behavior

Sources

Liskov substitution principle - Wikipedia

The Liskov substitution principle (LSP) is a particular definition of a subtyping relation, called strong behavioral…

en.wikipedia.org

Takeaway

External resource: The Liskov Substitution Principle: www.objectmentor.com/resources/articles/lsp.pdf We identified…

stg-tud.github.io

Why would Square inheriting from Rectangle be problematic if we override the SetWidth and SetHeight…

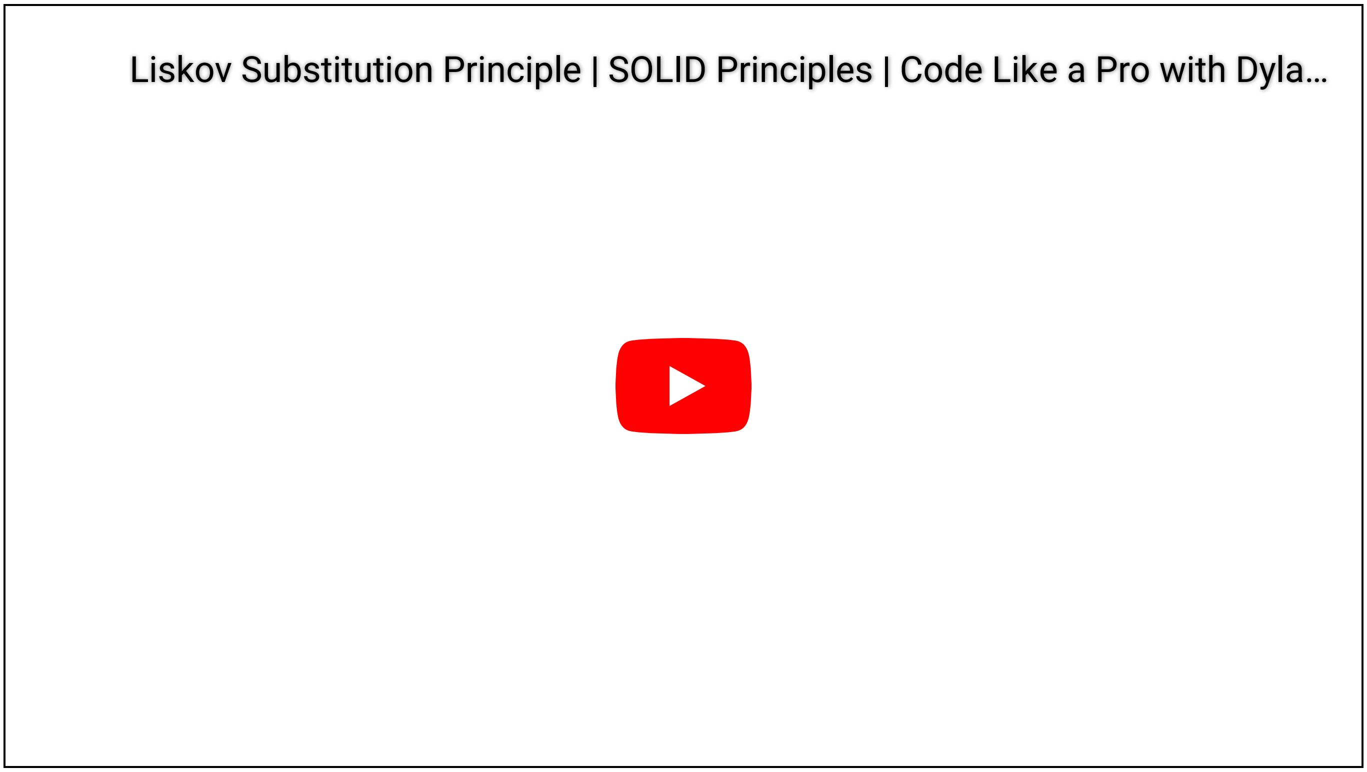
If a Square is a type of Rectangle than why cant a Square inherit from a Rectangle? Or why is it a bad design? First…

softwareengineering.stackexchange.com

LSP - Is Square A Rectangle? - Code Coach

Today we'll discover that in programming you sometimes end up with surprising conclusions. We're continuing our journey...

codecoach.co.nz


Liskov Substitution Principle | SOLID Principles | Code Like a Pro with Dyla...

Liskov Substitution · Oop Concepts · C Sharp Programming · Development

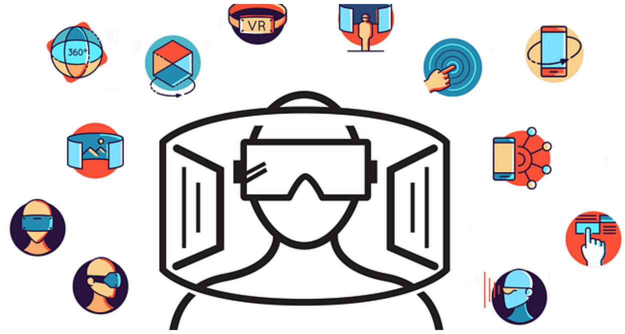
Written by Alexandre Dutertre

Follow

5 Followers

Student of Holberton School in Laval, France (Cohort #17)

More from Alexandre Dutertre

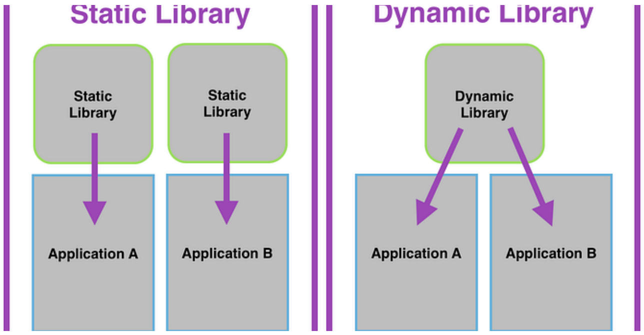


Alexandre Dutertre

The ethical challenges of AR/VR

While not as recent as most people think, the first VR machine was created in 1956,...

Jul 26, 2023 50

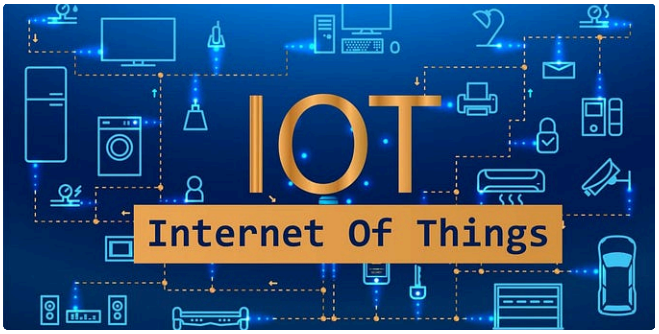


Alexandre Dutertre

The differences between a static library and a dynamic library

After explaining briefly the differences in my article about the GCC compilation and talkin...

May 12, 2022 2



Alexandre Dutertre

What are those Things on the Internet? (IoT)

Since the early 90s, new technologies keep appearing every year, bringing with them…

Aug 16, 2022

Alexandre Dutertre

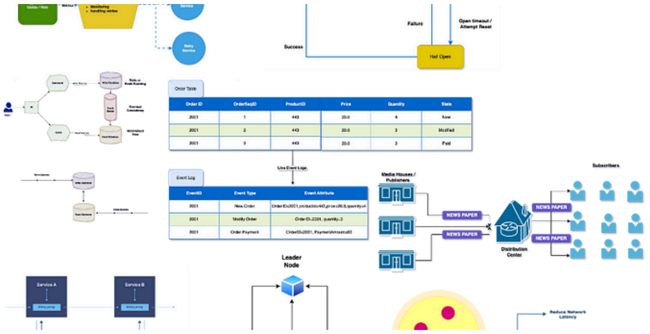
Recursion. Recursio. Recursi. Recurs… What is recursion?

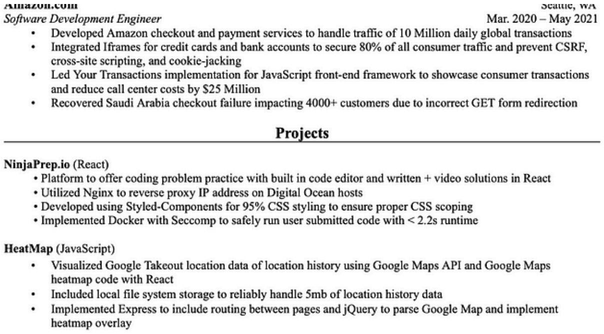
Today, let's talk about a concept as interesting as intriguing: recursion. So, what's recursion?

Jun 27, 2022 2

See all from Alexandre Dutertre

Recommended from Medium





Amazon.com Seattle, WA

Software Development Engineer Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)

- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)

- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

Anil Gudigar in Javarevisited

Most-Used Distributed System Design Patterns

Distributed system design patterns provide architects and developers with proven...

Jun 20 1.1K 8

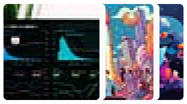
Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

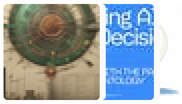
Jun 1 18.8K 312

Lists



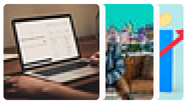
Growth Marketing

11 stories · 209 saves



data science and AI

40 stories · 225 saves



MODERN MARKETING

177 stories · 808 saves

Medium's Huge List of Publications Accepting...

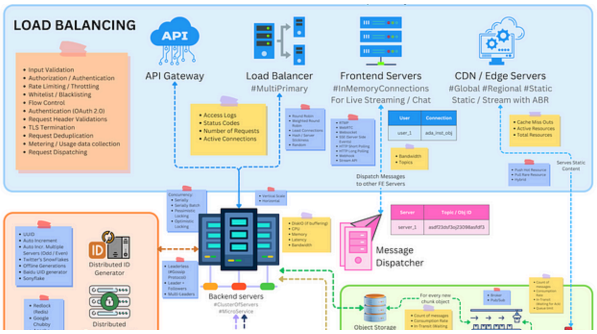
334 stories · 3383 saves

Axel Casas, PhD Candid... in Python in Plain Engli...

Stop Doing Tutorials. Learn Programming Like This

Learn programming faster and better

Jul 13 3.2K 48

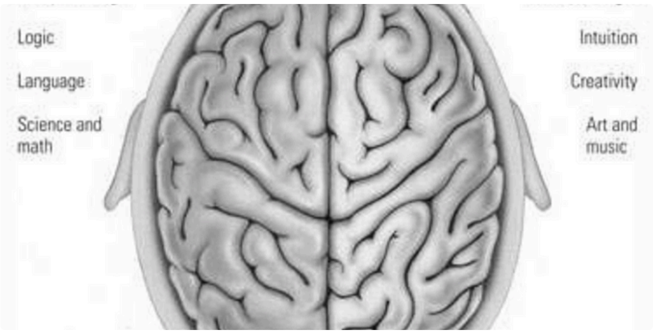


Love Shar... in ByteByteGo System Design Allian...

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However,...

Sep 17, 2023 8.6K 60

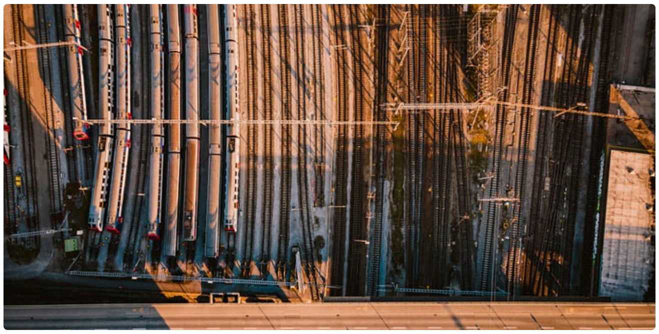


Sufyan Maan, M.Eng

What Happens When You Start Reading Every Day

Think before you speak. Read before you think.—Fran Lebowitz

Mar 12 31K 753



Andrew Zuo

Async Await Is The Worst Thing To Happen To Programming

I recently saw this meme about async and await.

Jun 22 3K 172

See more recommendations