How to make Smoothie !

We want...

- Find the perfect smoothie
- Be rich
- Not the hard way the smart way

Something cool !

The smoothIA

Tell the perfect smoothie for YOU

Here is Melissa

She love smoothie

with **raspberry**

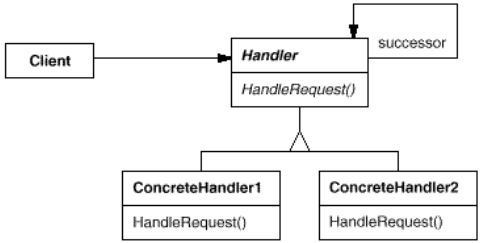
but she is **allergic** to **lemon**


Ready ?
Know the unknow, bring reality now, push the limit

Chain of responsability

Chain :

Avoid coupling the sender of a... blablabla ...object handles it.




BORING

Big title really fun so you are focus

Input aka Sender

A smoothie

People aka recievers

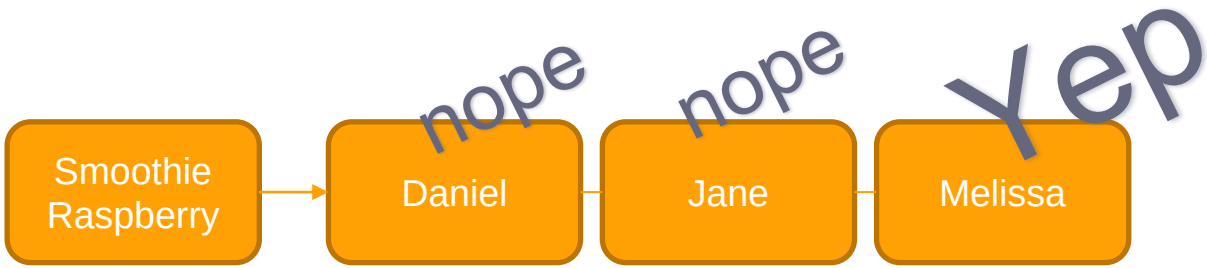
Daniel,

Jane,

Melissa

Output

Does the smoothie has been choosen ?



Another title really catchy, go !

Input aka Sender

A smoothie

People aka recievers

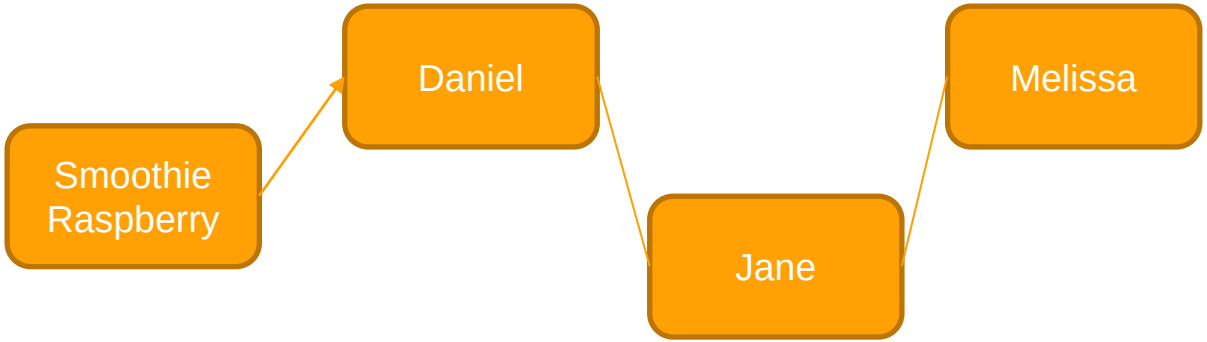
Daniel,

Jane,

Melissa

Output

Does the smoothie has been choosen ?



Another title really catchy, go !

Input aka Sender

A smoothie

People aka recievers

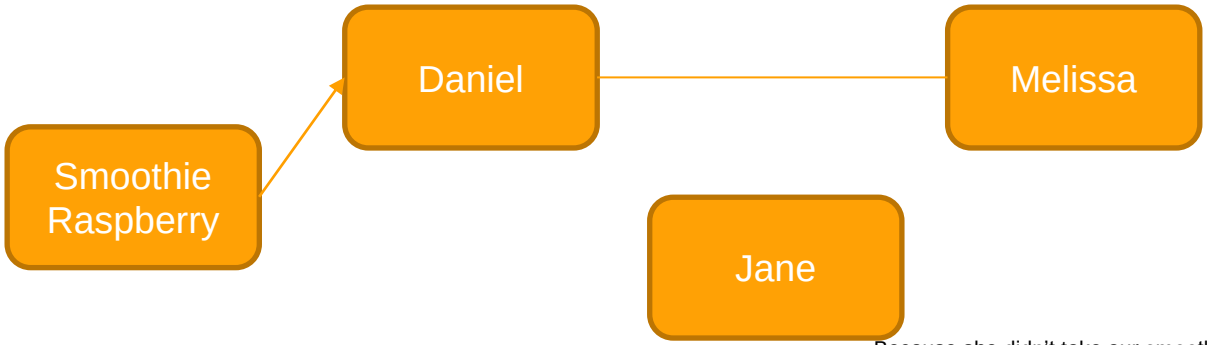
Daniel,

Jane (died),

Melissa

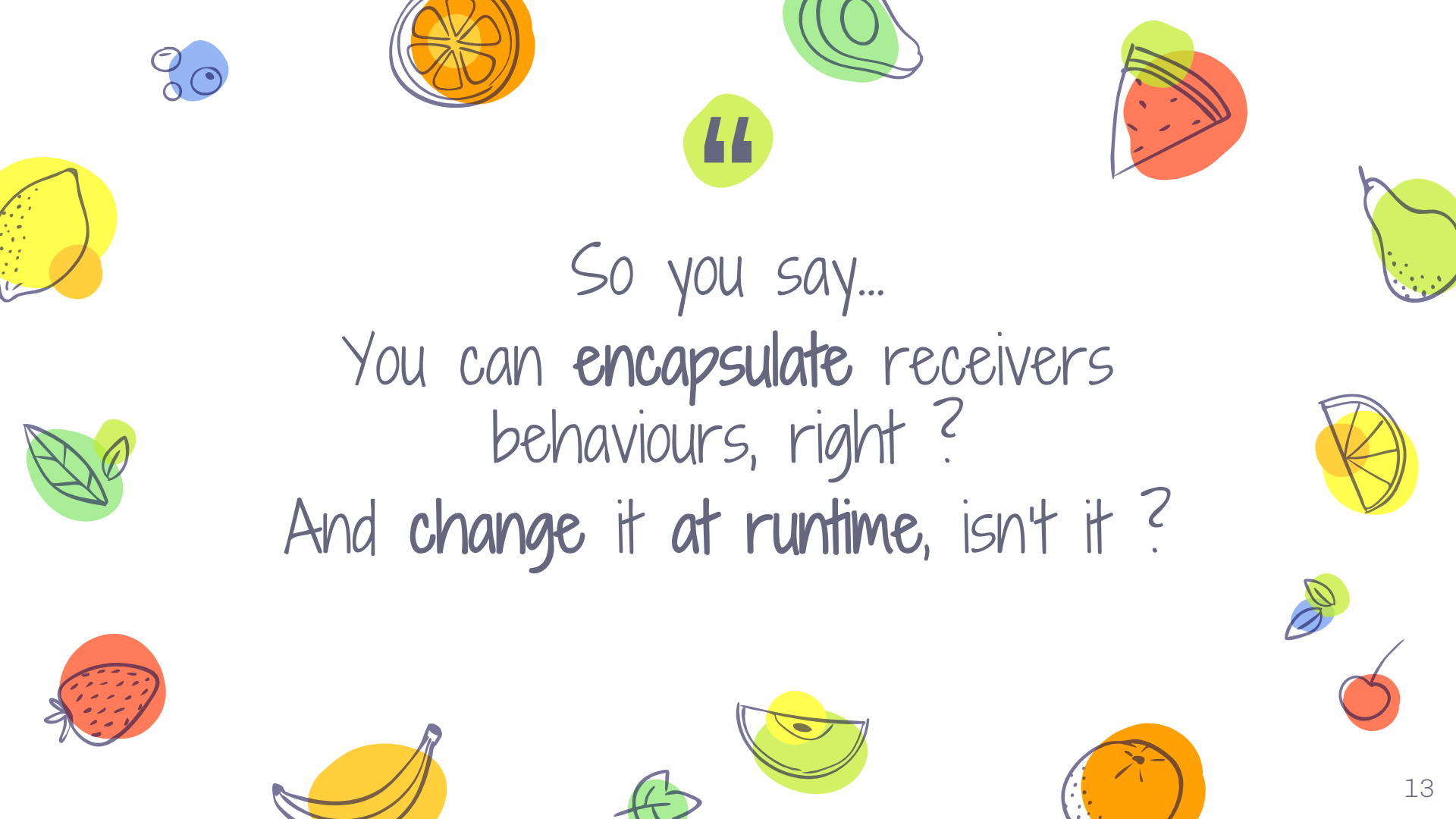
Output

Does the smoothie has been choosen ?



Because she didn't take our smoothie…

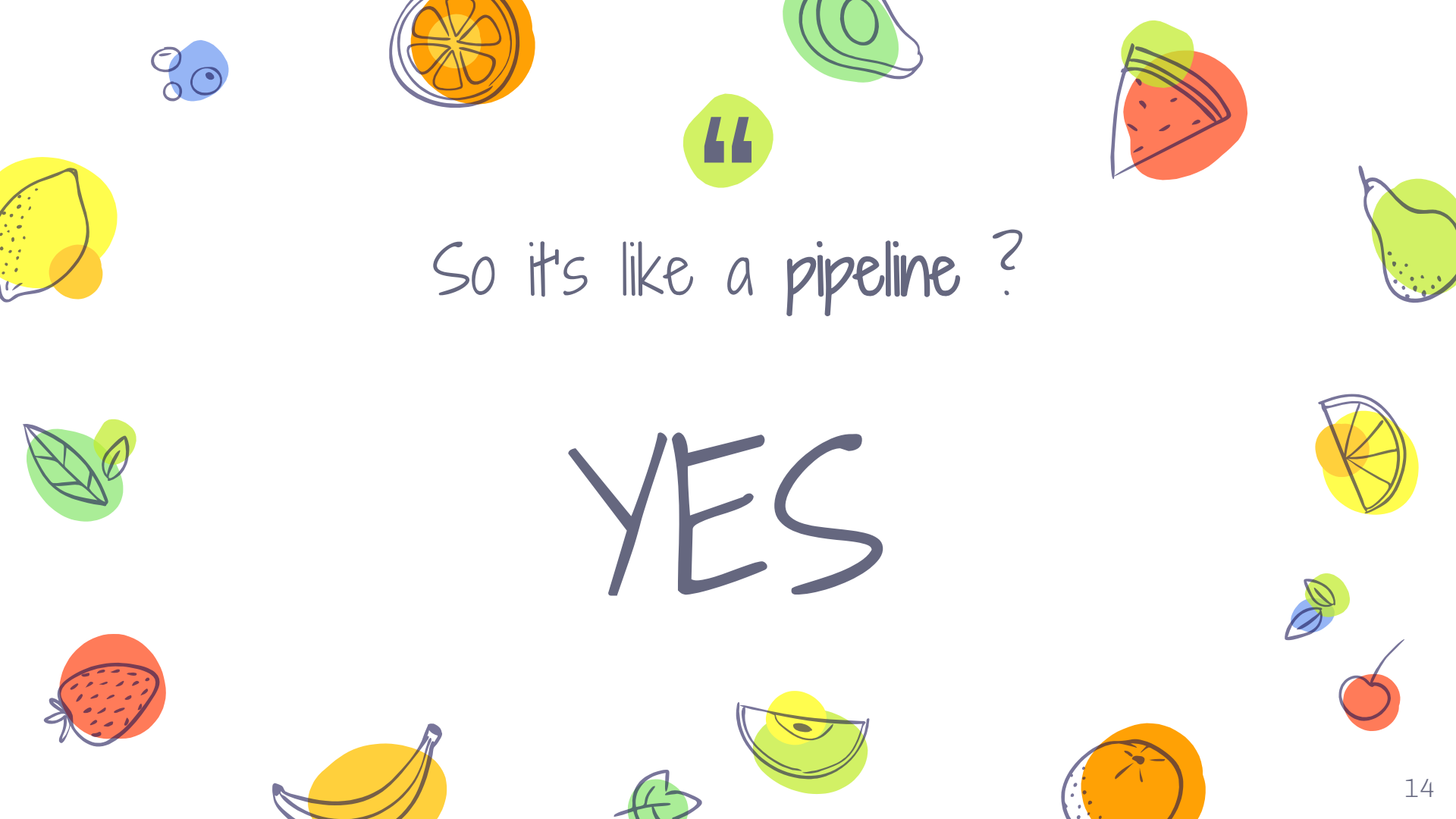
The chain is dynamic



So you say...

You can **encapsulate** receivers behaviours, right ?

And **change** it **at runtime**, isn't it ?


So it's like a pipeline ?
YES


So it like a "dynamic" pipeline ?
YES !


“
Easy I got it !
Can you show me some code ?

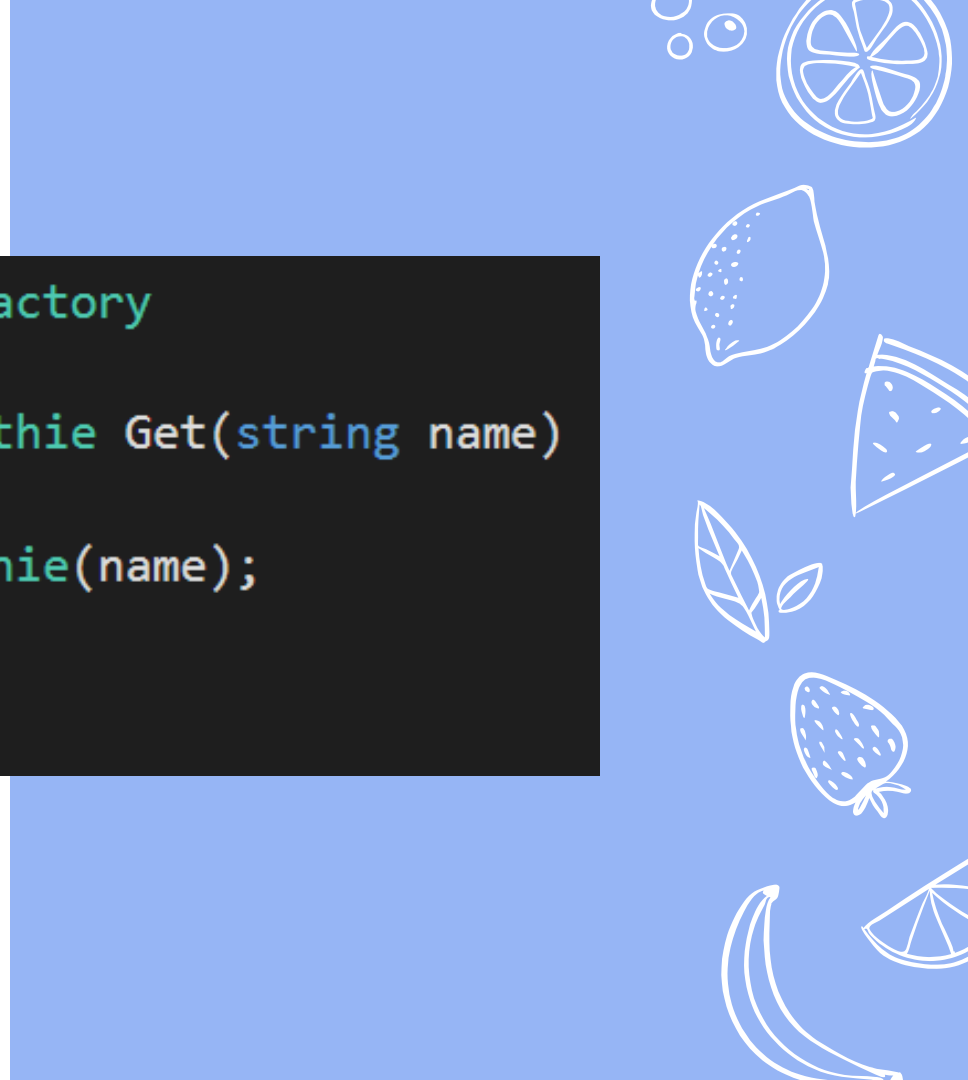
```
[TestMethod]
public void Melissa_Like_Raspberry()
{
    var smoothie = SmoothieFactory.Get("Raspberry");

    var smoothieLover = new ChainOfSmothieLovers()
        .Of("Daniel")
        .Then("Jane")
        .Then("Melissa") //Melissa like Raspberry
        .GetHead();

    bool hasBeenChoosen = smoothieLover.Recieve(smoothie);

    Assert.IsTrue(hasBeenChoosen);
}
```

```
internal class SmoothieFactory
{
    internal static Smoothie Get(string name)
    {
        return new Smoothie(name);
    }
}
```



```
[TestMethod]
public void Melissa_Like_Raspberry()
{
    var smoothie = SmoothieFactory.Get("Raspberry");

    var smoothieLover = new ChainOfSmothieLovers()
        .Of("Daniel")
        .Then("Jane")
        .Then("Melissa") //Melissa like Raspberry
        .GetHead();

    bool hasBeenChoosen = smoothieLover.Recieve(smoothie);

    Assert.IsTrue(hasBeenChoosen);
}
```

```
internal ChainOfSmothieLovers Then(string name)
{
    Tail.SetNextSmothieLover(smoothieLovers.Get(name));
```

```
internal class SmothieLoversFactory
{
    Dictionary<string, SmoothieLover> dict = new Dictionary<string, SmoothieLover>();
    public SmothieLoversFactory()
    {
        dict.Add("Melissa", new Melissa());
        dict.Add("Jane", new Jane());
        dict.Add("Daniel", new Daniel());
    }
    internal SmoothieLover Get(string name)
    {
        return dict[name];
    }
}
```

```
internal ChainOfSmothieLovers Then(string name)
{
    Tail.SetNextSmothieLover(smoothieLovers.Get(name));
```

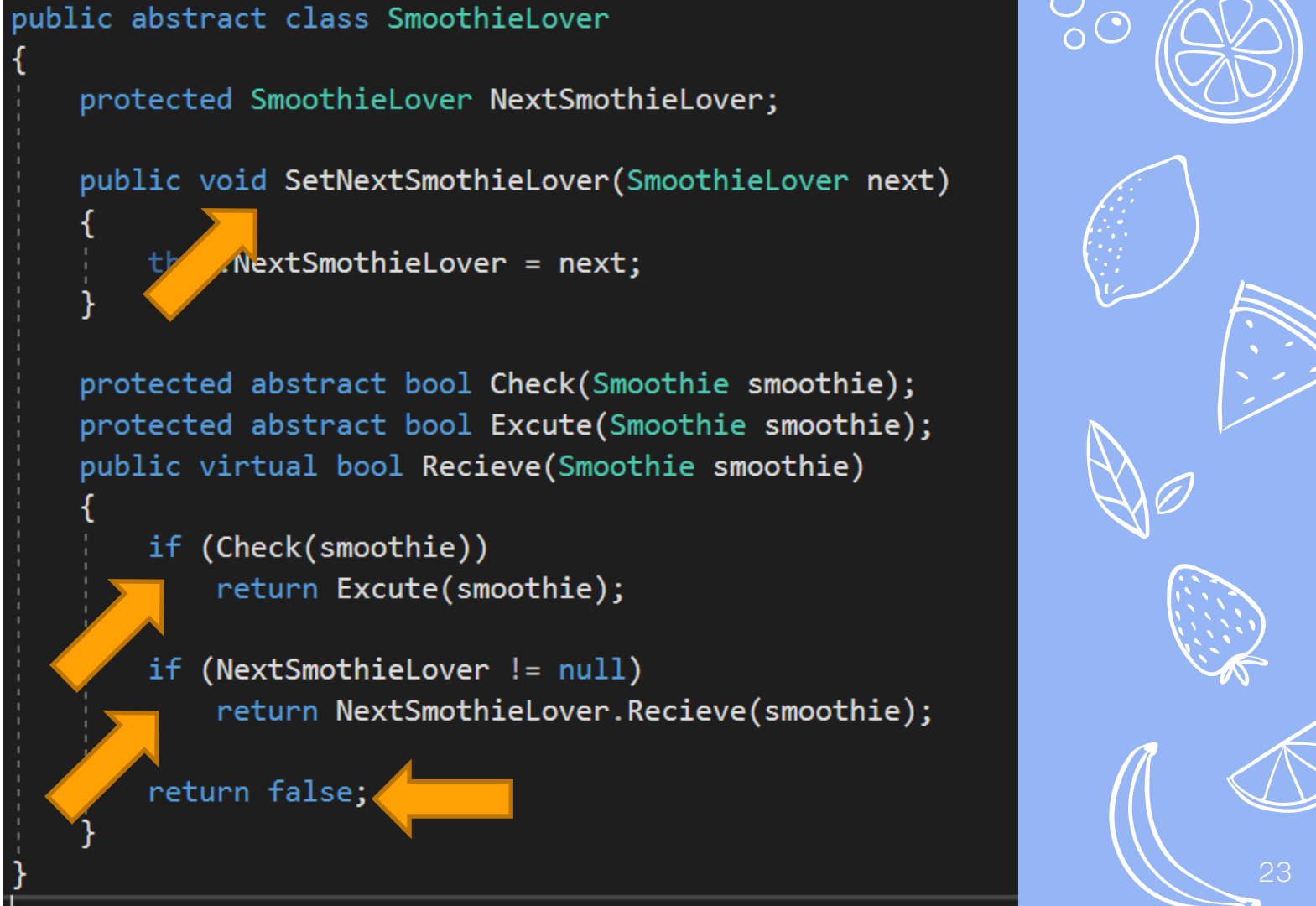
```
public abstract class SmoothieLover
{
    protected SmoothieLover NextSmothieLover;

    public void SetNextSmothieLover(SmoothieLover next)
    {
        th .NextSmothieLover = next;
    }

    protected abstract bool Check(Smoothie smoothie);
    protected abstract bool Excute(Smoothie smoothie);
    public virtual bool Recieve(Smoothie smoothie)
    {
        if (Check(smoothie))
            return Excute(smoothie);

        if (NextSmothieLover != null)
            return NextSmothieLover.Recieve(smoothie);

        return false;
    }
}
```



```
public class Melissa : SmoothieLover
{
    protected override bool Check(Smoothie smoothie)
    {
        return smoothie.Name == "Raspberry";
    }

    protected override bool Excute(Smoothie smoothie)
    {
        Console.WriteLine($"I'm Melissa and this smoothie is mine : {smoothie.Name}");
        return true;
    }
}
```

```
[TestMethod]
public void Melissa_Like_Raspberry()
{
    var smoothie = SmoothieFac

    var smoothieLover = new Ch
        .Of("Daniel")
        .Then("Jane")
        .Then("Melissa") //Mel
        .GetHead();

    bool hasBeenChoosen = smoo

    Assert.IsTrue(hasBeenChoos
}
```


```
bberry

is mine : Raspberry
```

The chain is...

Single responsability

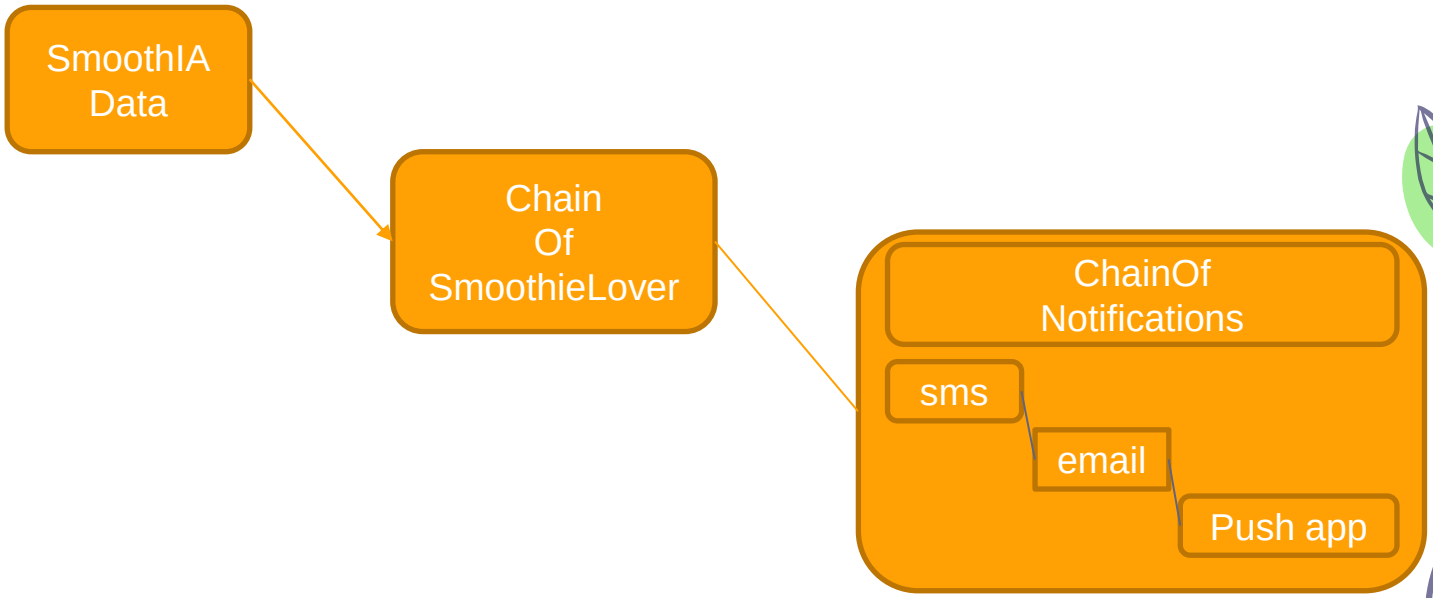
The chain is...

Open/Close

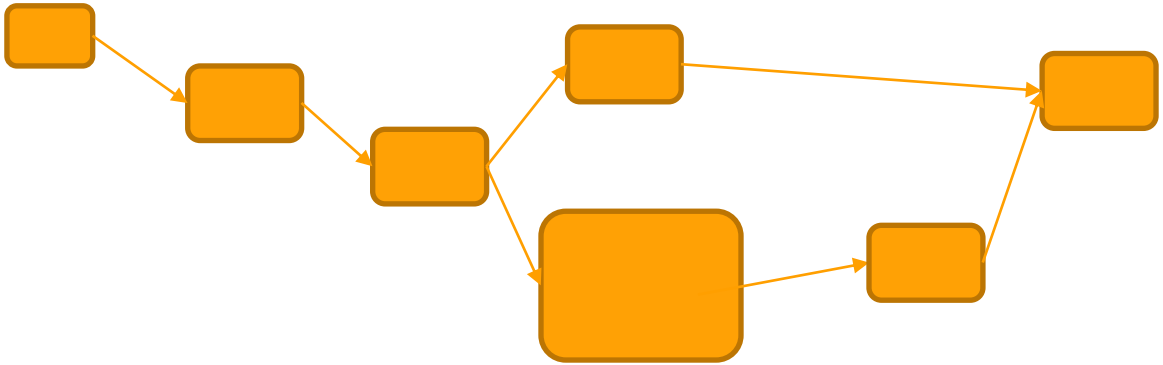
You can...

Chainception

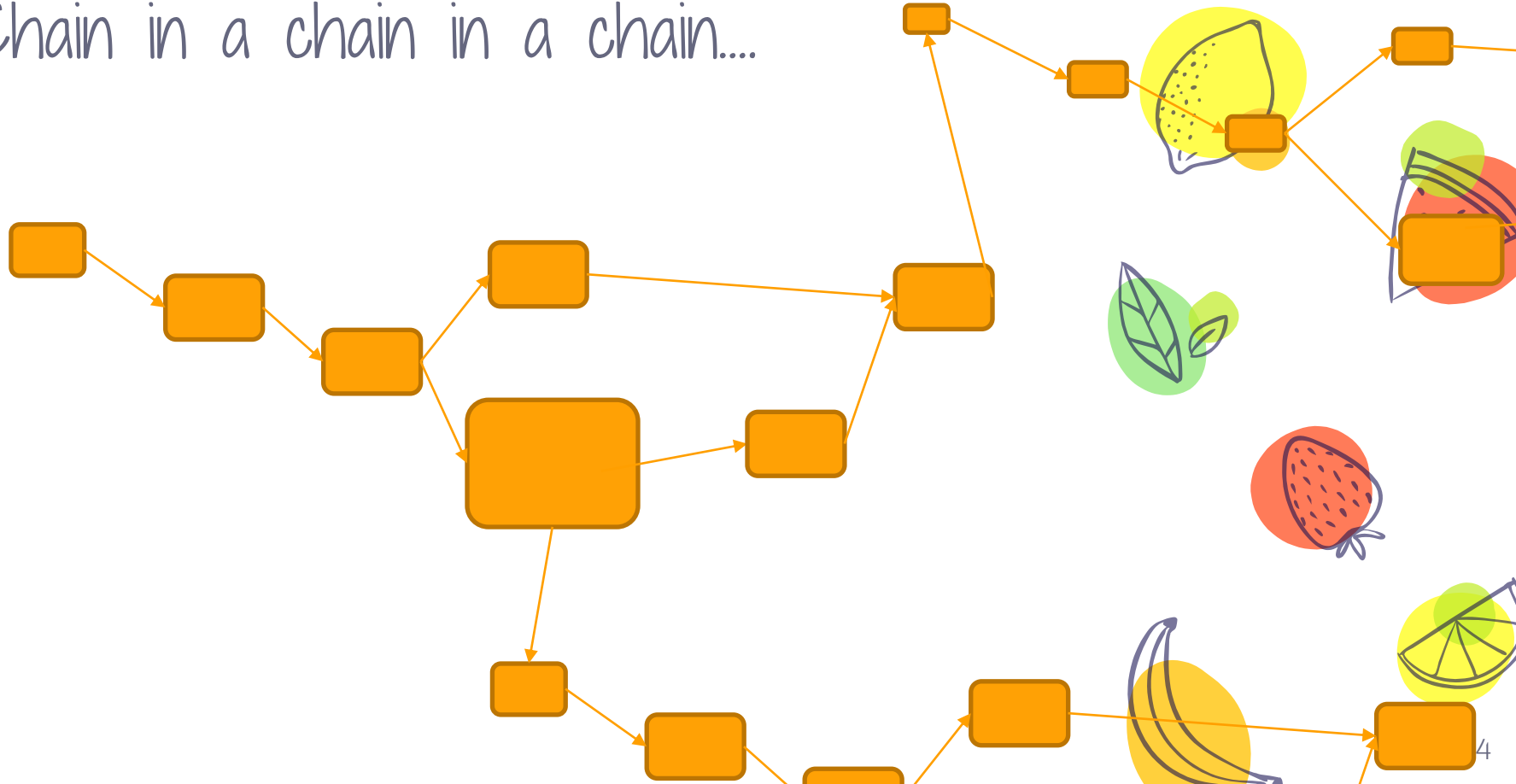
Chain in a chain in a chain....



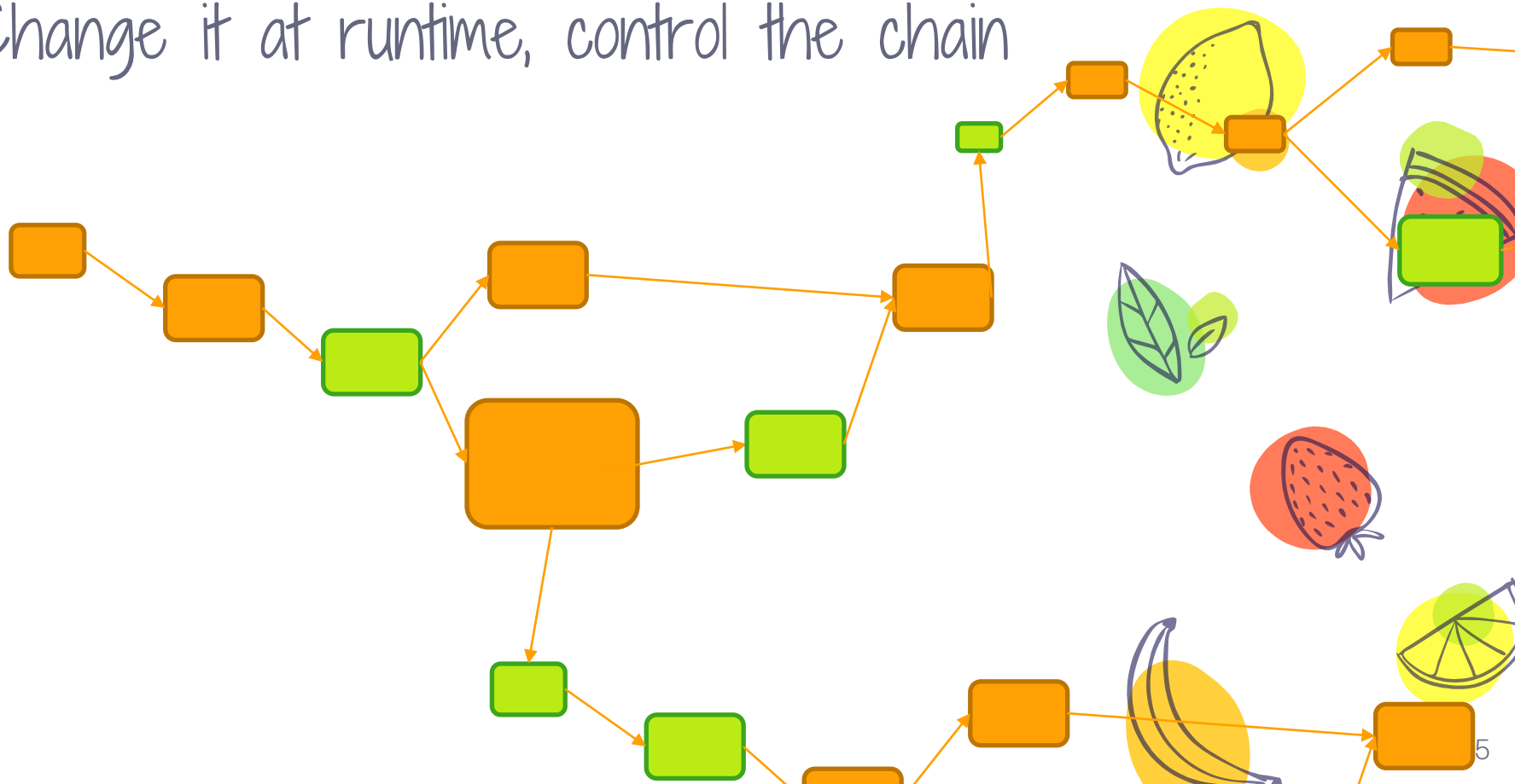
Chain in a chain in a chain....



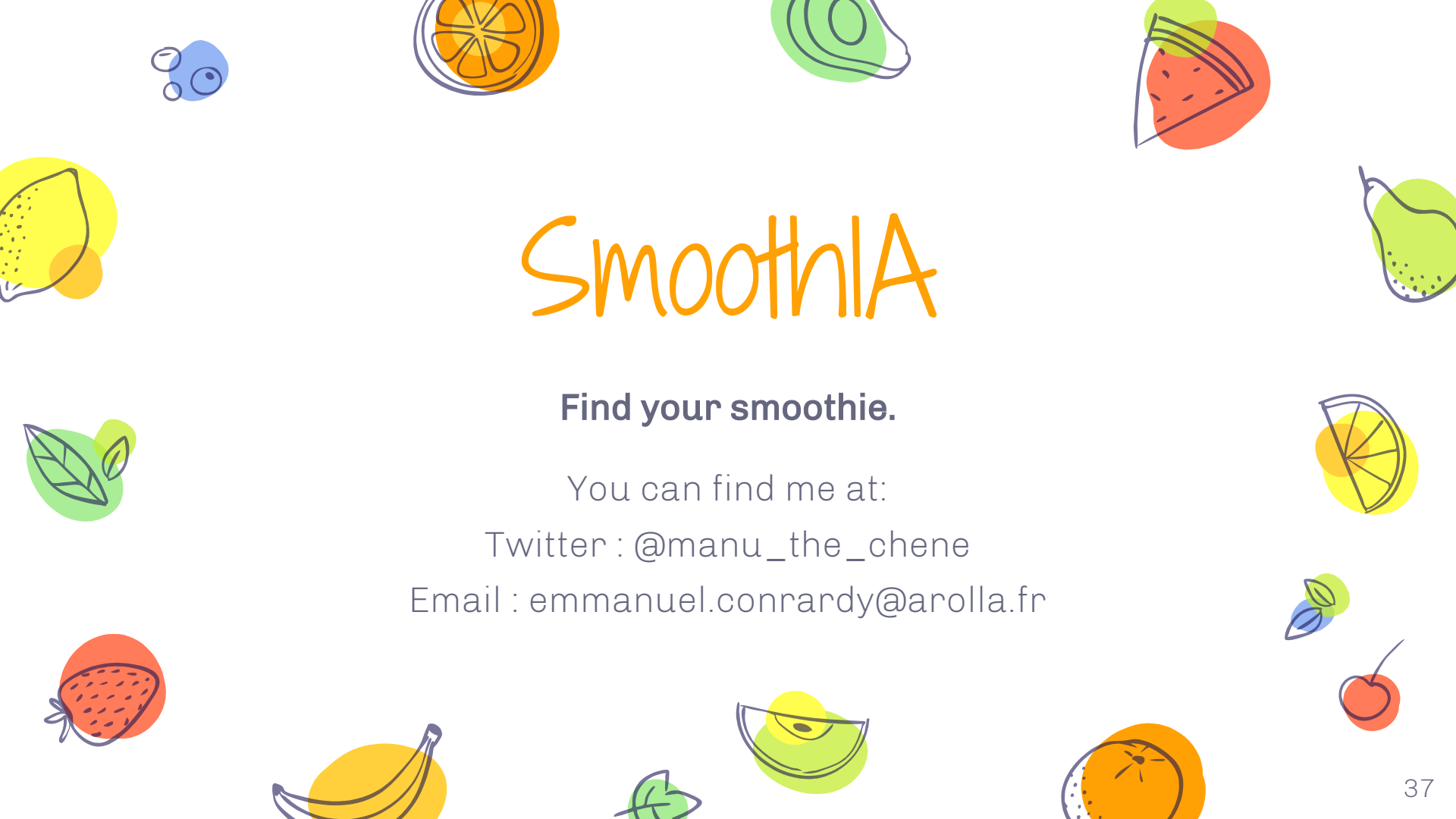
Chain in a chain in a chain....



Change it at runtime, control the chain




What is the perfect smoothie for you ?



SmoothIA

Find your smoothie.

You can find me at:

Twitter : @manu_the_chene

Email : emmanuel.conrardy@arolla.fr

Credits

Special thanks to all the people who made smoothIA:

- Presentation template by SlidesCarnival
- Photographs by Unsplash

Links

My best smoothie :

https://www.750g.com/smoothie-aux-framboises-r52312.htm

Boring but not so bad :

http://codenuclear.com/chain-of-responsibility-design-pattern-in-java/

Response to the main title:

https://www.wikihow.com/Make-a-Smoothie

How to make Smoothie !