infobip

Untestable code

Infobip Virtual Classroom: Boost your Bip potential

Name of the presenter: Aleksandar Dostic

Date: 08.06.2020

Please mute your mic whenever you are not talking.

You can submit questions during the presentation in the chat thread.

Every presentation will include one Q&A session , during the session feel free to ask questions live or type down the question in the chat thread.

When you need to ask a question live, use the raise hand option and/or send a chat message.

The calls will be recorded, when the recording starts you will see our company policy notification. By clicking on it, you can read more about it.

Last but not least, pay attention in order to learn more and have fun!

Go to **www.menti.com** and use the code **11 68 90**

Mentimeter

QR

Coding

Coding

- **Why?**

Coding

- Why?
- **Solve a problem**

Coding

- Why?
- **Solve a problem or to make money :)**

Coding

- Why?
- Solve a problem or to make money :)
- **Code needs to work as expected**

Coding

- Why?
- Solve a problem or to make money :)
- Code needs to work as expected
- **How?**

Coding

- Why?
- Solve a problem or to make money :)
- Code needs to work as expected
- How?
- **Learn a language**

Coding

- Why?
- Solve a problem or to make money :)
- Code needs to work as expected
- How?
- Learn a language
- **Learn domain**

Coding

- Why?
- Solve a problem or to make money :)
- Code needs to work as expected
- How?
- Learn a language
- Learn domain
- **Test it!**

Testing

Testing

- **Why?**

Testing

- Why?
- **Regression**

Testing

- Why?
- Regression
- **Improve implementation**

Testing

- Why?
- Regression
- Improve implementation
- **It saves time**

Testing

- Why?
- Regression
- Improve implementation
- It saves time
- **Self-updating documentation**

Types of testing

Types of testing

- **Functional testing**

Types of testing

- **Functional testing**

- **Non-functional testing**

Types of testing

- **Functional testing:**
  - **Unit**

- **Non-functional testing**

Types of testing

- **Functional testing:**
  - **Unit**
  - **Integration**

- **Non-functional testing**

Types of testing

- **Functional testing:**
  - **Unit**
  - **Integration**
  - **Regression**

- **Non-functional testing**

Types of testing

- **Functional testing:**
  - **Unit**
  - **Integration**
  - **Regression**
  - **Smoke**
- **Non-functional testing**

Types of testing

- Functional testing:
  - Unit
  - Integration
  - Regression
  - Smoke
- **Non-functional testing:**

Types of testing

- Functional testing:
  - Unit
  - Integration
  - Regression
  - Smoke
- **Non-functional testing:**
  - **Performance**

Types of testing

- Functional testing:
  - Unit
  - Integration
  - Regression
  - Smoke
- **Non-functional testing:**
  - **Performance**
  - **Load**

Types of testing

- Functional testing:
  - Unit
  - Integration
  - Regression
  - Smoke
- **Non-functional testing:**
  - **Performance**
  - **Load**
  - **Security**

How to test?

How to test?

- **There is no secret to writing test**

How to test?

- There is no secret to writing test

```
TheClass aClass = new TheClass(...);
```

How to test?

- There is no secret to writing test

```
TheClass aClass = new TheClass(...);

assertEquals(0, aClass.get(...));
```

How to test?

- There is no secret to writing test

```
TheClass aClass = new TheClass(...);

assertEquals(0, aClass.get(...));
```

- **Only a secret to writing testable code**

Untestable code

Untestable code

- **How to write hard to test code?**

Untestable code

- **How to write hard to test code?**
  - **Global state**

Untestable code

- **How to write hard to test code?**
  - **Global state**
  - **Singletons**

Untestable code

- **How to write hard to test code?**
  - **Global state**
  - **Singletons**
  - **Using new operator everywhere**

Untestable code

- **How to write hard to test code?**
  - **Global state**
  - **Singletons**
  - **Using new operator everywhere**
  - **To many if and switch statements**

Untestable code

- **How to write hard to test code?**
  - **Global state**
  - **Singletons**
  - **Using new operator everywhere**
  - **To many if and switch statements**
  - **Tangled dependencies**

Untestable code

- **How to write hard to test code?**
  - **Global state**
  - **Singletons**
  - **Using new operator everywhere**
  - **To many if and switch statements**
  - **Tangled dependencies**
  - **Nondeterministic**

If-statements are evil

If-statements are evil

- **`goto` to Procedural Programming**

If-statements are evil

- goto to Procedural Programming
- **`if` to Object Oriented Programming**

If-statements are evil

- `goto` to Procedural Programming
- `if` to Object Oriented Programming
- **Most can be replaced by polymorphism**

Why?

Why?

- **Easier to read**

Why?

- Easier to read
- **Easier to maintain**

Why?

- Easier to read
- Easier to maintain
- **Easier to test**

Polymorphism

Polymorphism

- **Object behavior depends on state**

Polymorphism

- Object behavior depends on state
- **Same condition on multiple places**

When to use if

When to use if

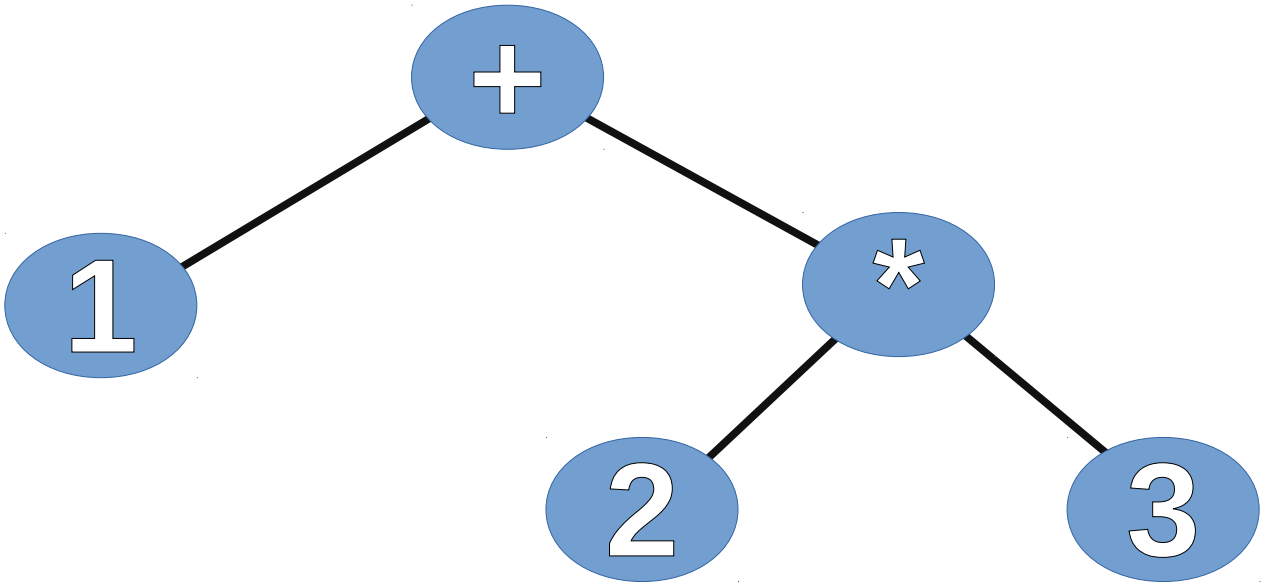
- **Comparison: >, <, ==, !=**

When to use if

- Comparison: >, <, ==, !=
- **But now we focus on avoiding if**

Model: 1 + 2 * 3

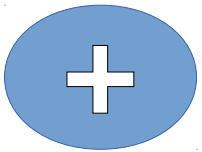
Model: 1 + 2 * 3



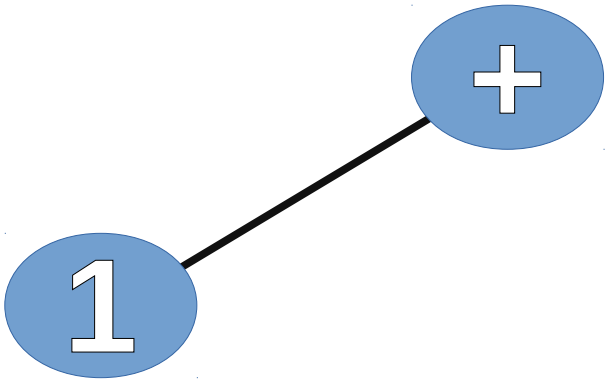
evaluate()

```
class Node {
}
```

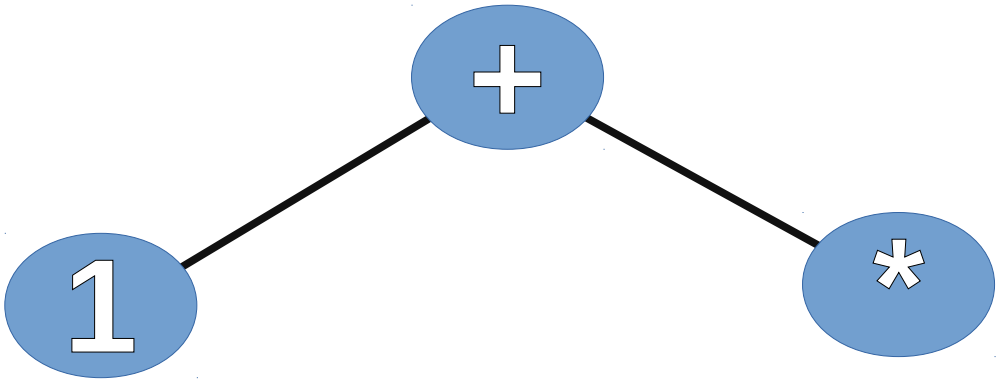
Model: 1 + 2 * 3



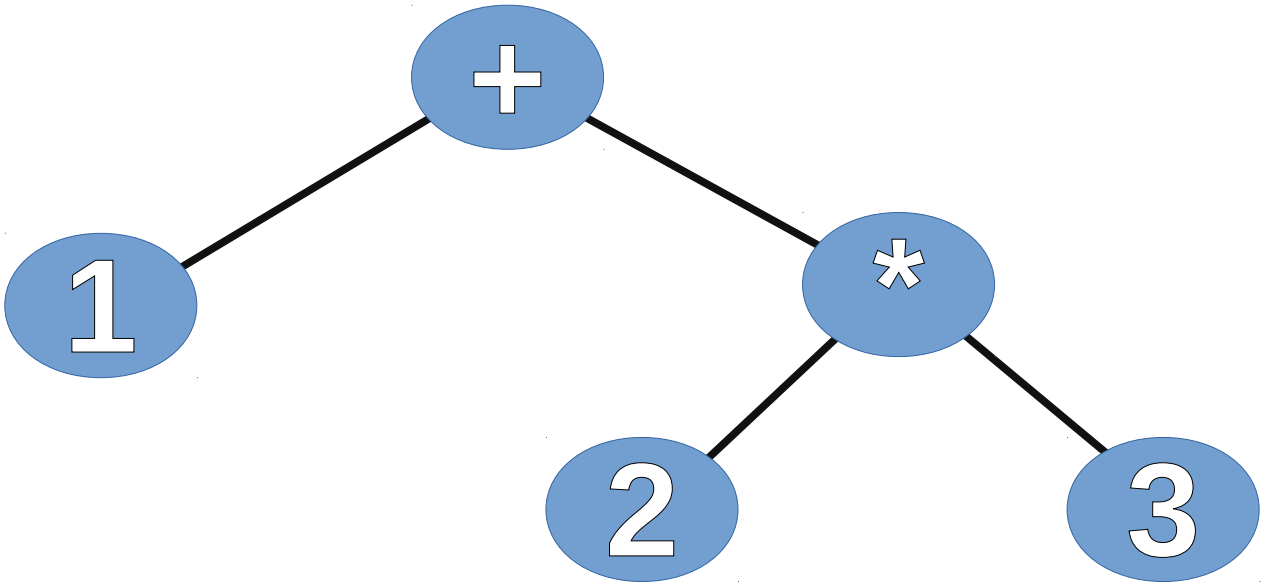
Model: 1 + 2 * 3



Model: 1 + 2 * 3



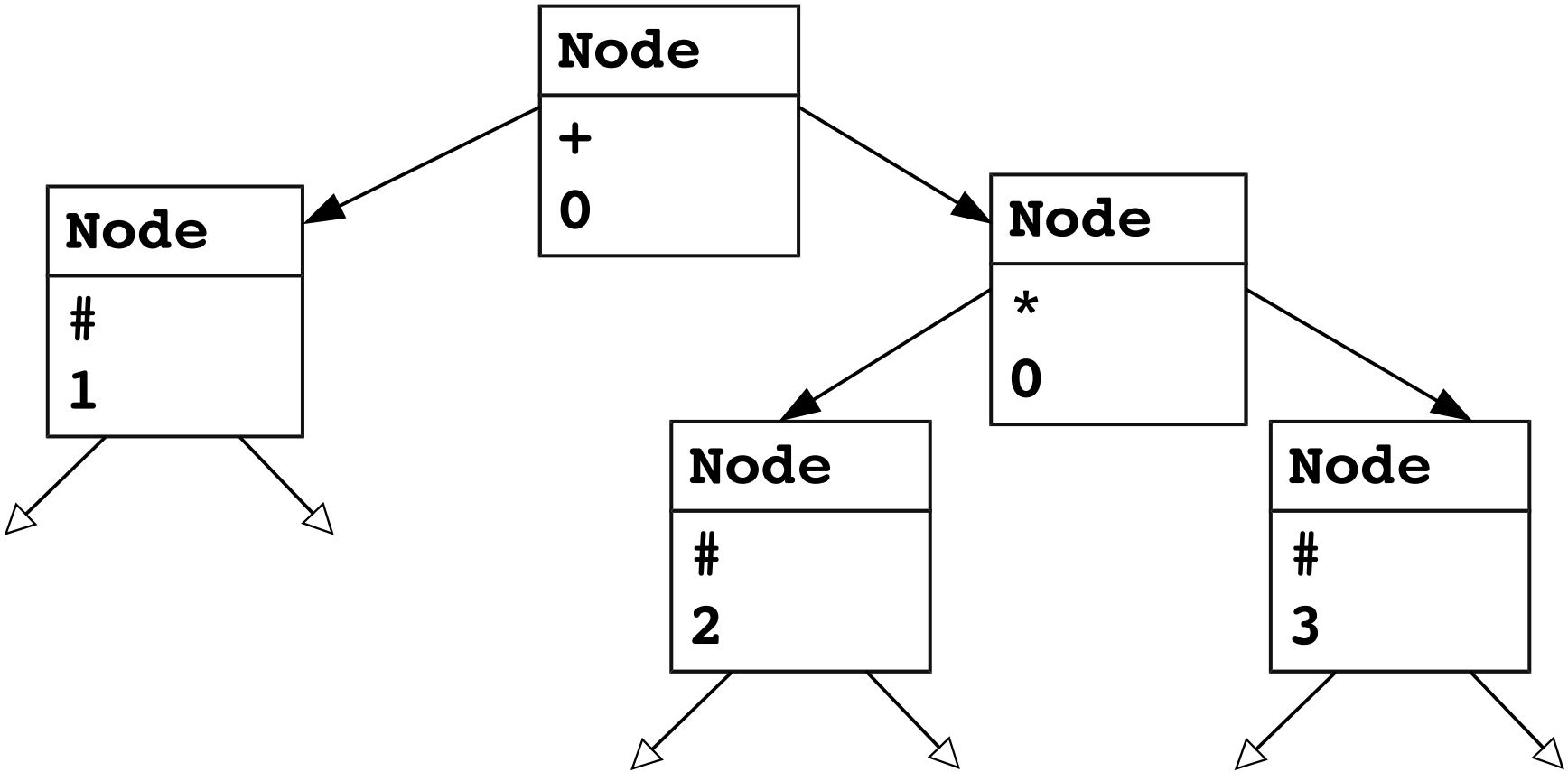
Model: 1 + 2 * 3



Node

Node
operation:char value:int left:Node right:Node
evaluate():int

Object graph



Analyzing

	#	+	*
function	✓	✓	✓
value	✓		
left		✓	✓
right		✓	✓

Analyzing: value

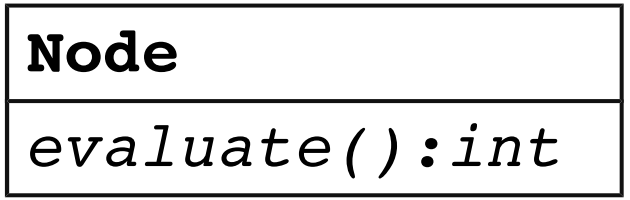
	#	+	*
function	✓	✓	✓
value	✓		
left		✓	✓
right		✓	✓

Analyzing: operation

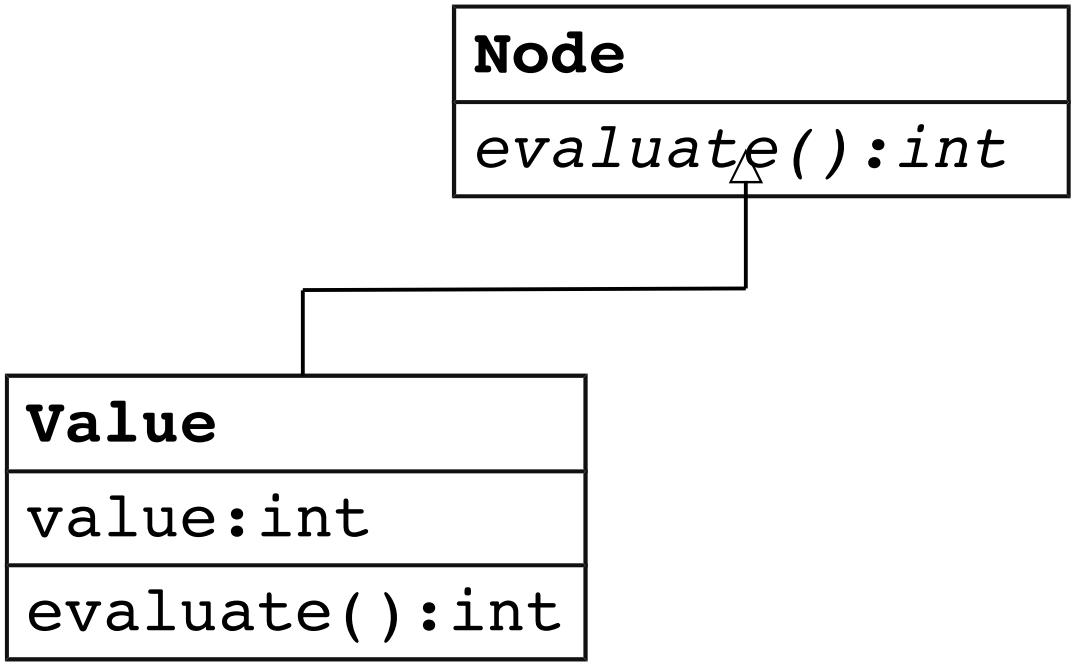
	#	+	*
function	✓	✓	✓
value	✓		
left		✓	✓
right		✓	✓

Abstracting

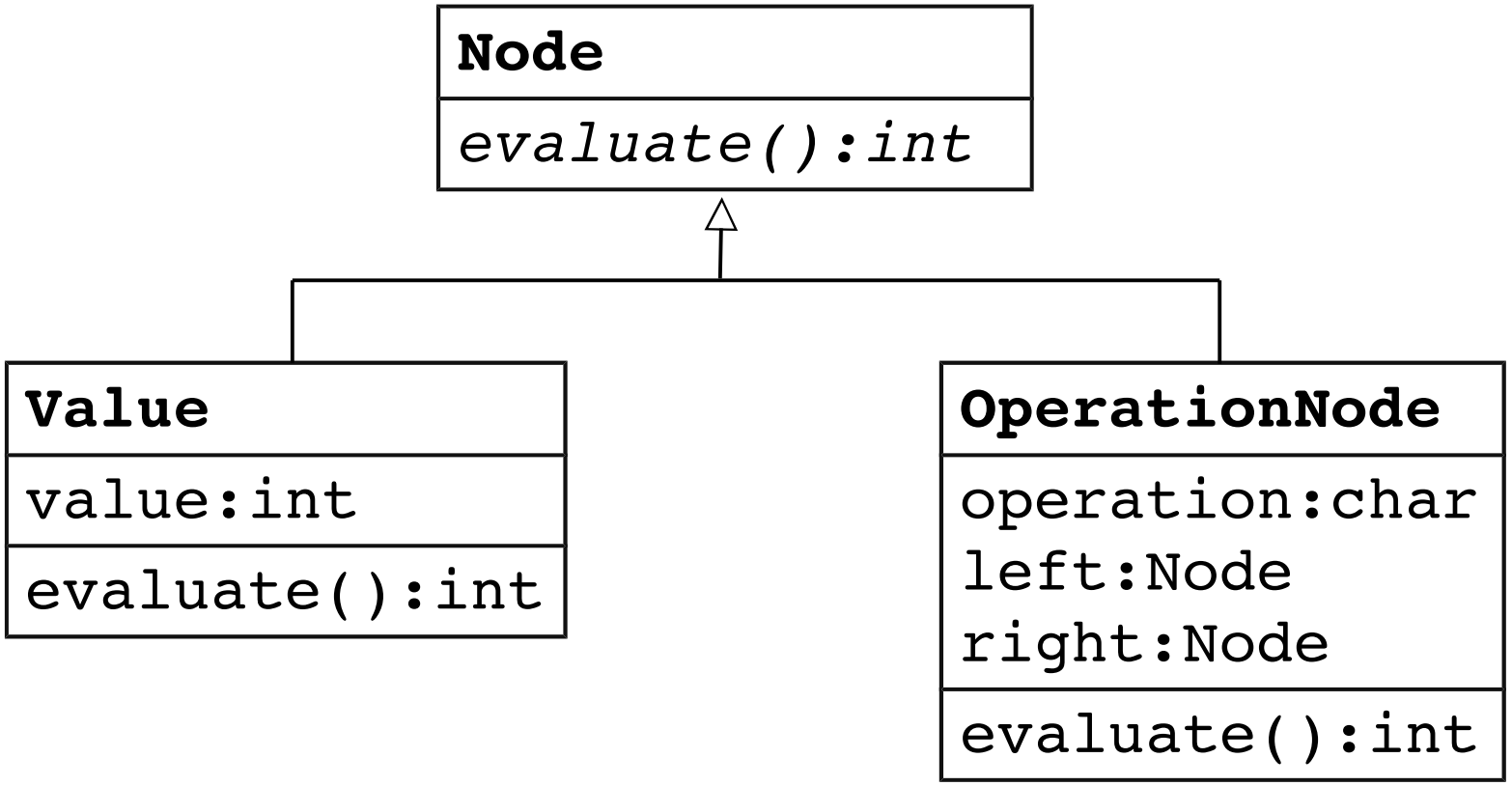
Abstracting



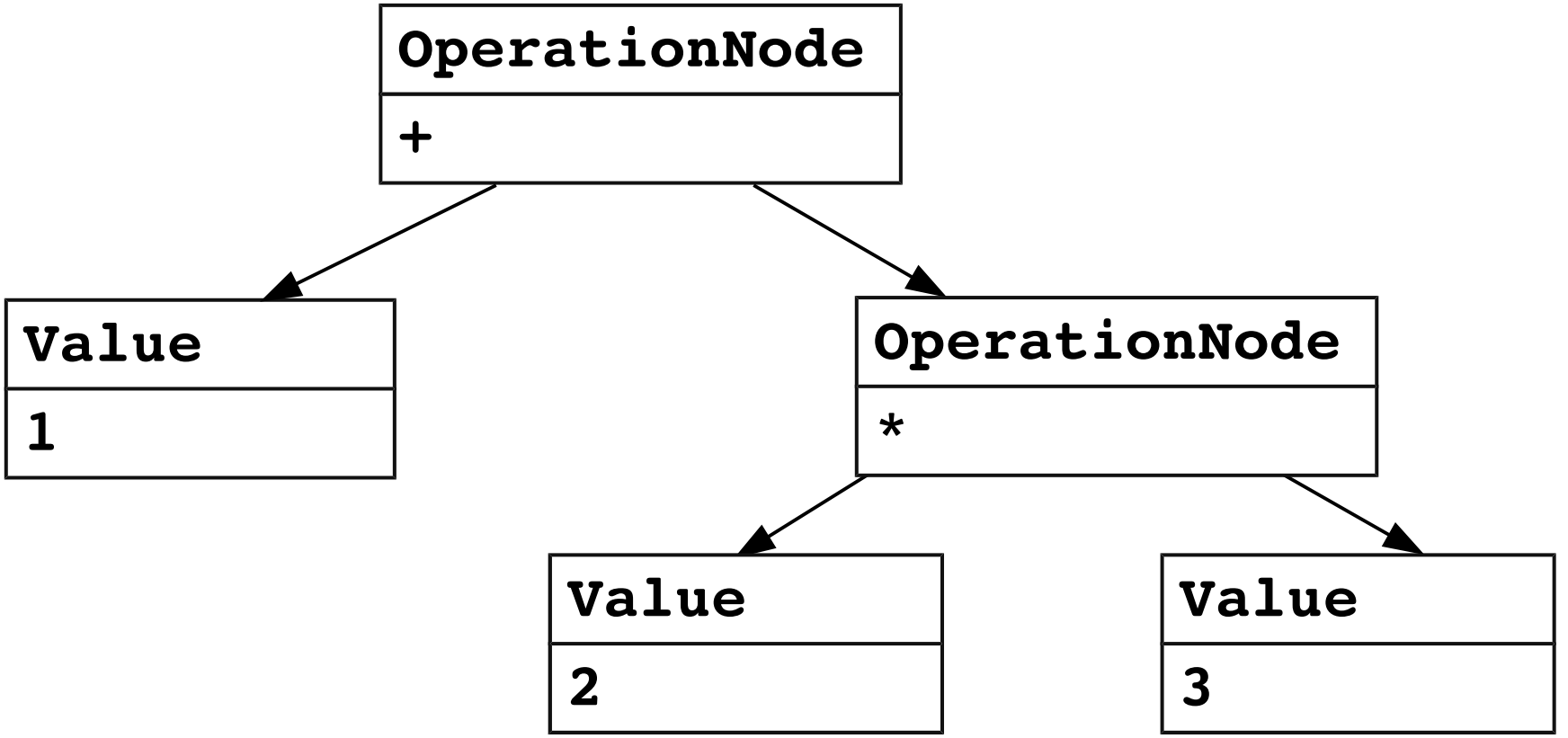
Abstracting



Abstracting



Operations



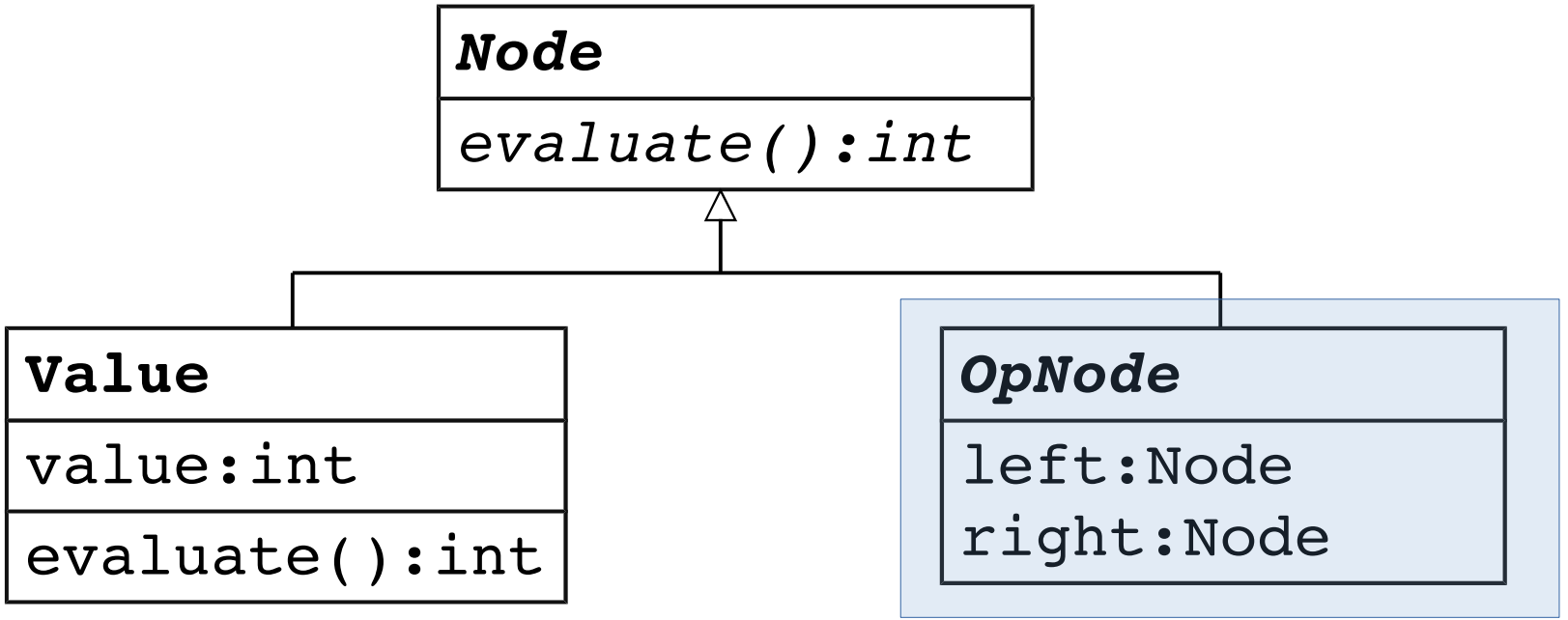
Analyzing: operation

	#	+	*
function	✓	✓	✓
value	✓		
left		✓	✓
right		✓	✓

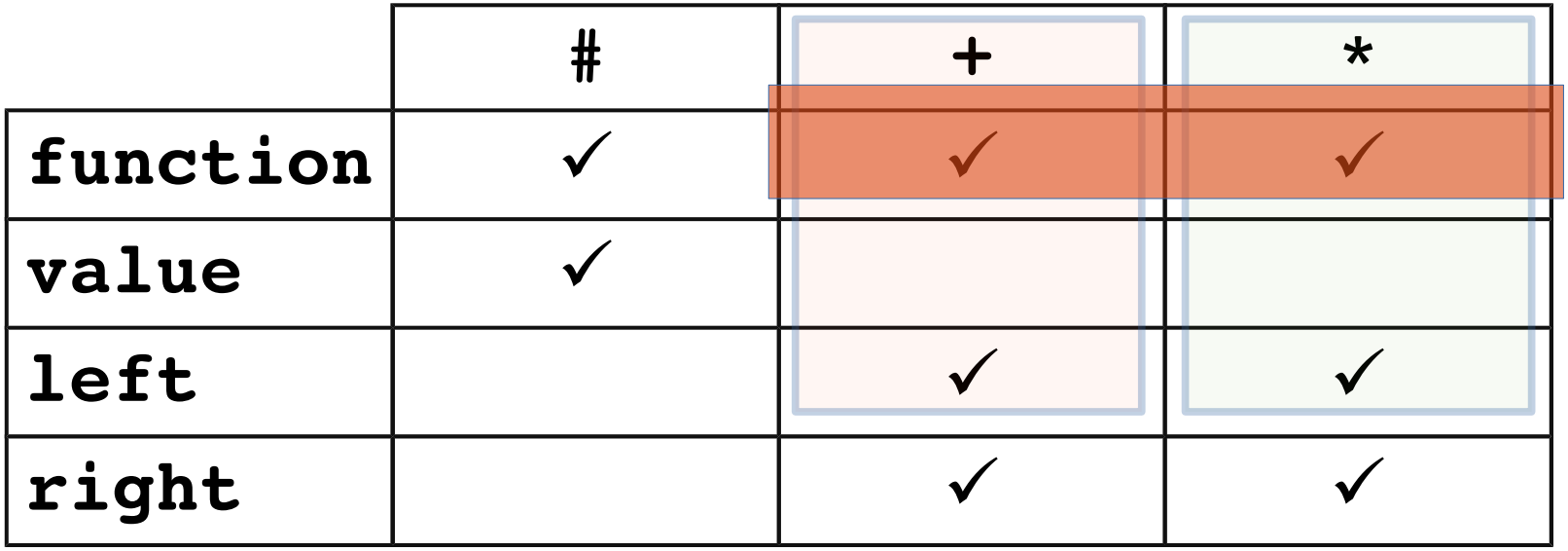
Analyzing: operation

	#	+	*
function	✓	✓	✓
value	✓		
left		✓	✓
right		✓	✓

Abstracting

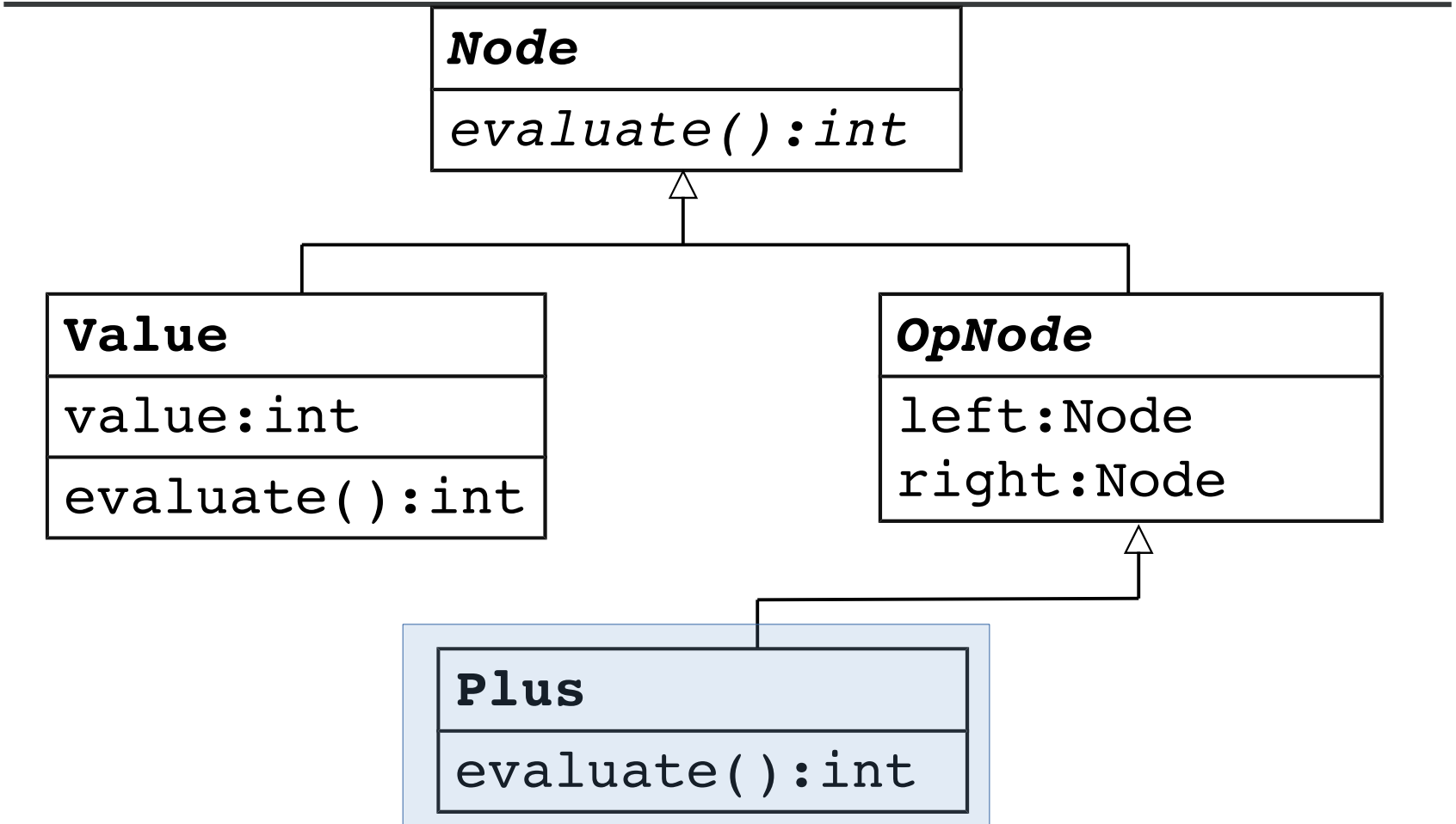


Analyzing: operation

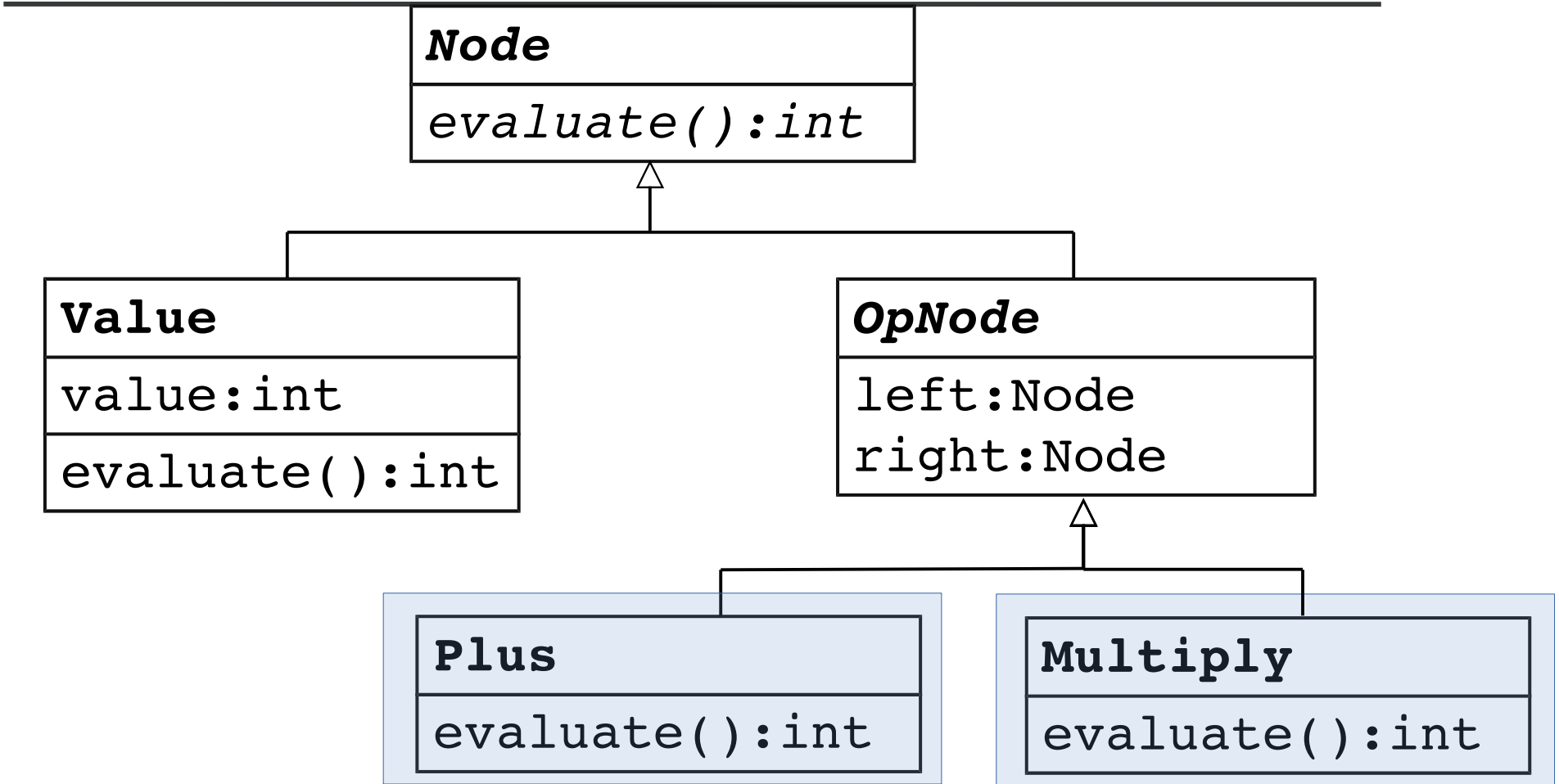


	#	+	*
function	✓	✓	✓
value	✓		
left		✓	✓
right		✓	✓

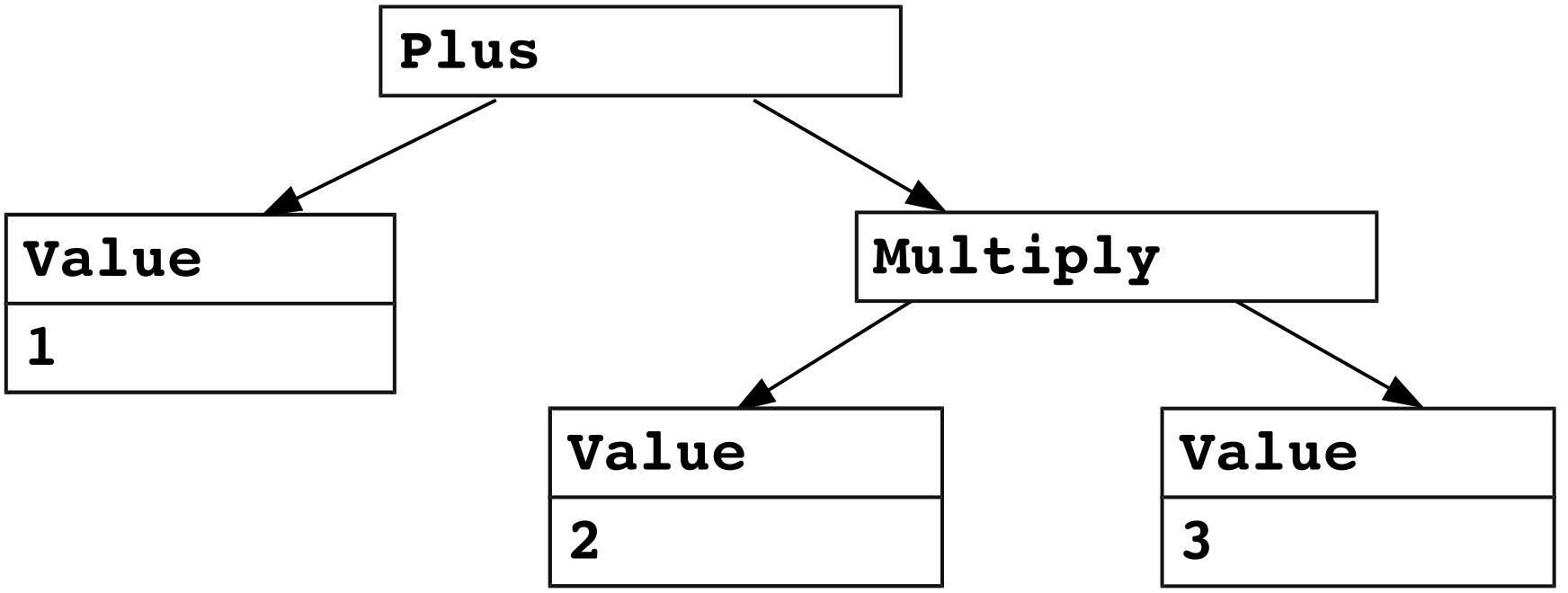
Abstracting



Abstracting



Operations



Further exploration

- **Implement:**

Further exploration

- **Implement:**
  - **exponentiation**

Further exploration

- **Implement:**
  - **exponentiation**
  - **factorial**

Further exploration

- **Implement:**
  - **exponentiation**
  - **factorial**
  - **logarithm**

Further exploration

- **Implement:**
  - **exponentiation**
  - **factorial**
  - **logarithm**
  - **trigonometry**

Where is the switch?

Where is the switch?

```
public class NodeFactory {
    public Node parse(String expression){
        Node root;
        ...
        switch (currentChunk) {
            case '[0-9]*':
                ... new Value(value);
            case '+':
                ... new Plus(left, right);
            case '-':
                ... new Minus(left, right);
            case '*':
                ... new Multiply(left, right);
            case '/':
                ... new Division(left, right);
            default:
                throw new IllegalArgumentException();
        }
        ...
        return root;
    }
}
```

Where is the switch?

```
public class NodeFactory {
    public Node parse(String expression){
        Node root;
        ...
        switch (currentChunk) {
            case '[0-9]*':
                ... new Value(value); // new Node('#', currentChunk, null, null)
            case '+':
                ... new Plus(left, right);
            case '-':
                ... new Minus(left, right);
            case '*':
                ... new Multiply(left, right);
            case '/':
                ... new Division(left, right);
            default:
                throw new IllegalArgumentException();
        }
        ...
        return root;
    }
}
```

Where is the switch?

```
public class NodeFactory {
    public Node parse(String expression){
        Node root;
        ...
        switch (currentChunk) {
            case '[0-9]*':
                ... new Value(value); // new Node('#', currentChunk, null, null)
            case '+':
                ... new Plus(left, right); // new Node('+', currentChunk, left, right)
            case '-':
                ... new Minus(left, right);
            case '*':
                ... new Multiply(left, right);
            case '/':
                ... new Division(left, right);
            default:
                throw new IllegalArgumentException();
        }
        ...
        return root;
    }
}
```

Summary

Summary

- **Polymorphic solution is often better:**

Summary

- **Polymorphic solution is often better:**
  - **new behavior can be added without original source**

Summary

- **Polymorphic solution is often better:**
  - **new behavior can be added without original source**
  - **each concern in separate file**

Summary

- **Polymorphic solution is often better:**
  - **new behavior can be added without original source**
  - **each concern in separate file**
  - **easier to understand**

Summary

- **Polymorphic solution is often better:**
  - new behavior can be added without original source
  - each concern in separate file
  - **easier to understand / test**

Summary

Summary

- **Prefer polymorphism over conditional**

Summary

- Prefer polymorphism over conditional
- `switch` often means polymorphism

Polymorphism

Polymorphism

- **Different behavior depending on type/flag**

Polymorphism

- Different behavior depending on type/flag
- **Move each in subclass**

Polymorphism

- Different behavior depending on type/flag
- Move each in subclass
- **Make the original method abstract**

Benefits

Benefits

- **Conditional in one place**

Benefits

- Conditional in one place
- **No duplication**

Benefits

- Conditional in one place
- No duplication
- **SRP**

Benefits

- Conditional in one place
- No duplication
- SRP
- **Say NO to Global state**

Benefits

Benefits

- **Common code in one place**

Benefits

- Common code in one place
- **Testing independently and in parallel easily**

Benefits

- Common code in one place
- Testing independently and in parallel easily
- **Subclasses makes it clear what is different**

When to go polymorphic

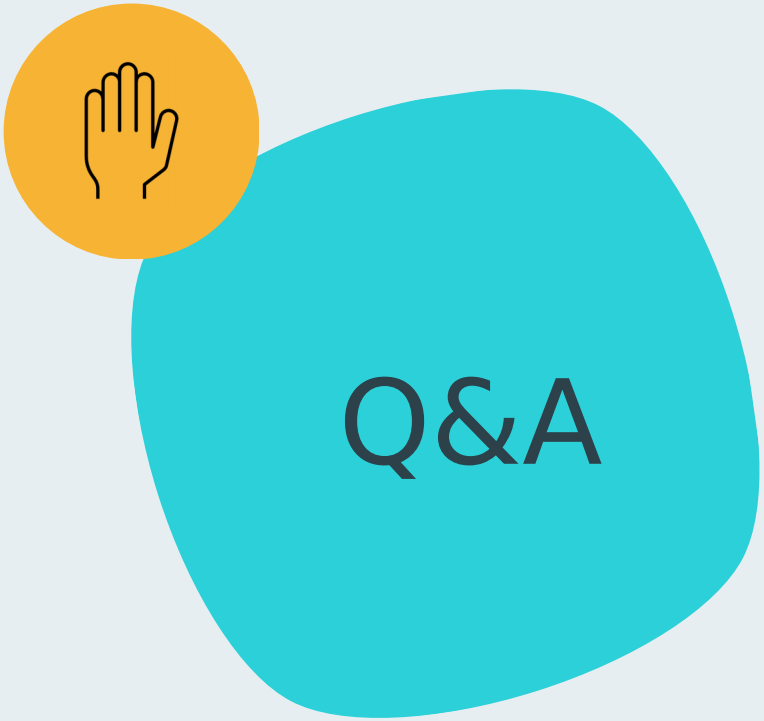
When to go polymorphic

- **Behavior changes based on state**

When to go polymorphic

- Behavior changes based on state
- **Parallel conditional in multiple places**

We are the **HUMBLE ENGINEERS** *led by our philosophy of* **LEARNING BY DOING** *and fuelled by our* **PASSION FOR TECHNOLOGY.** *We value* **CREATIVITY, PERSISTENCE** *and* **INNOVATION. INTEGRITY** *and living* **MEANINGFUL LIVES** *are the* **FOUNDATIONS** *of* **ALL OUR VALUES**


Q&A

https://github.com/adostic/virtual-classroom-untestable

Thank you!

infobip