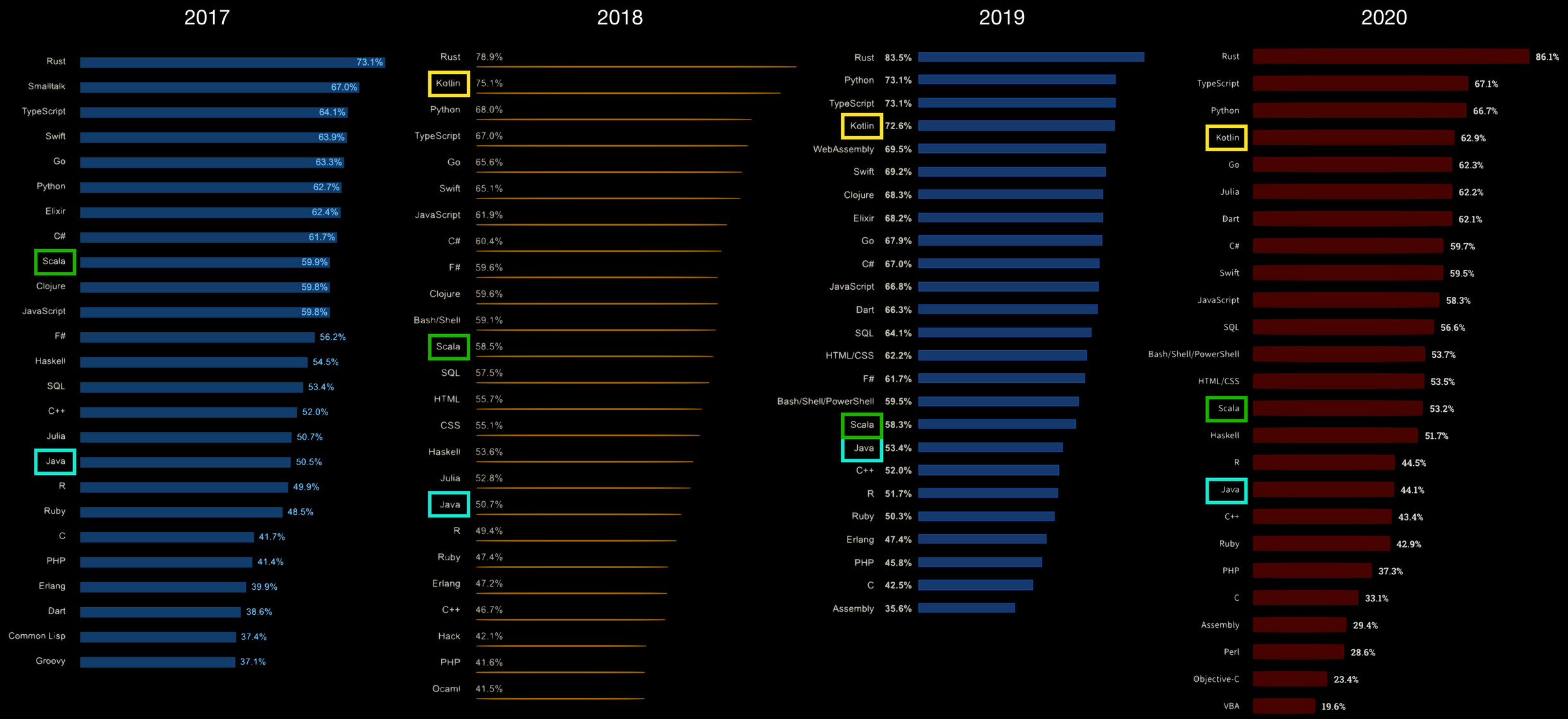
The Past, Present, & Future

Kotlin: What is it?

- Statically typed language
- Targets the JVM (& ...🤫😳)
- 100% interop with Java
- Developed by Jetbrains since 2011
- Official language for Android dev since 2017

Kotlin: Why?

- Solid industry-ready language
- 100% interop with Java
- Expressive + concise
- Null-safe!!
- Good functional paradigm support
- Incredible tooling (looking at you Scala 🧐)



Most Loved Technologies — Stackoverflow Developer Survey [1]

1996 B.K.

Java
JVM

Garbage Collection

Not new

C++


09:01:44
GIFAK.NET
001

Garbage Collection

Less Bugs
Less Performance

Garbage Collection

Less Bugs
Less Performance*

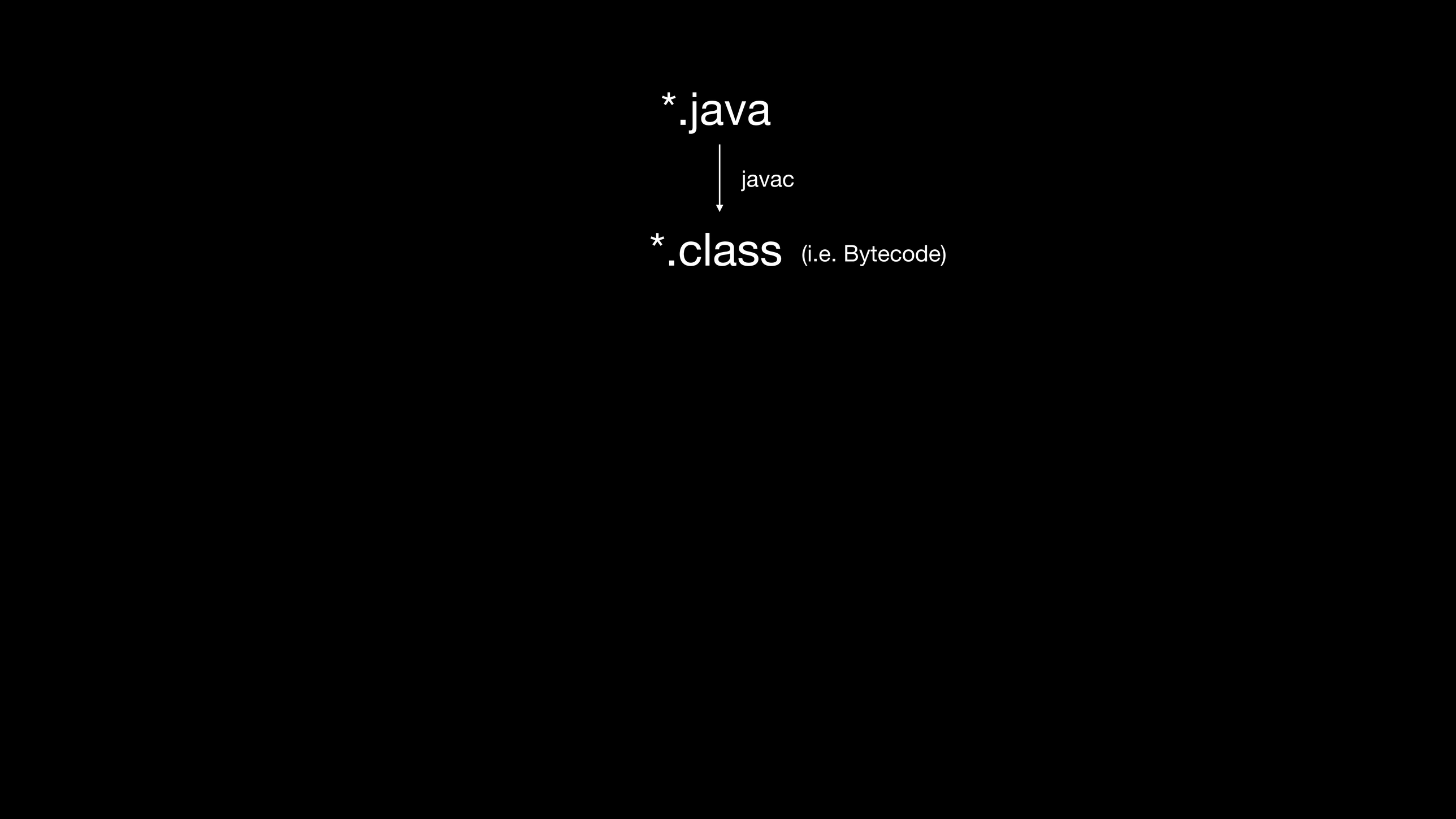
** Especially during the early days*

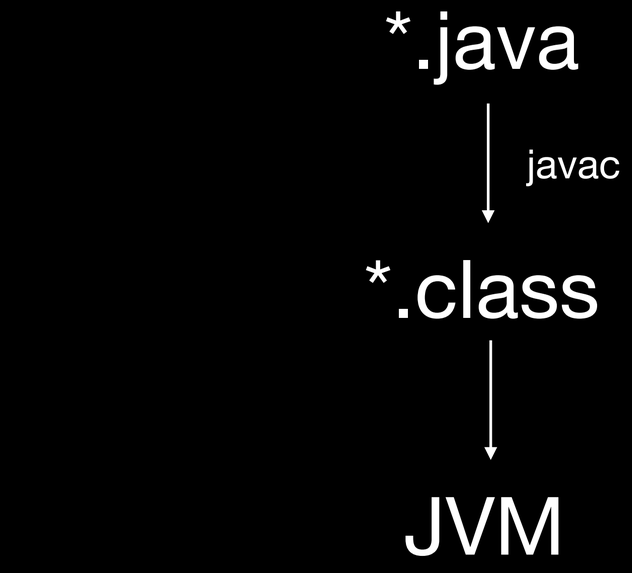
Write Once, Run Everywhere

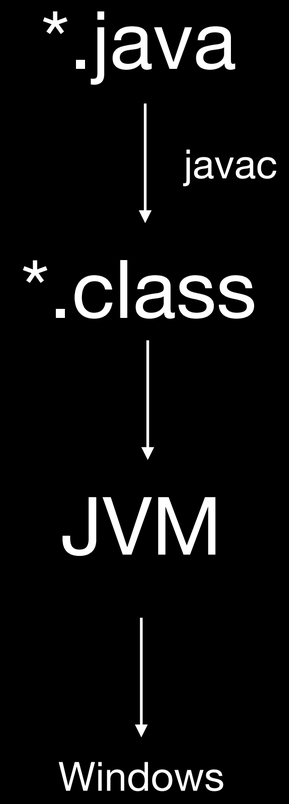
Less Bugs

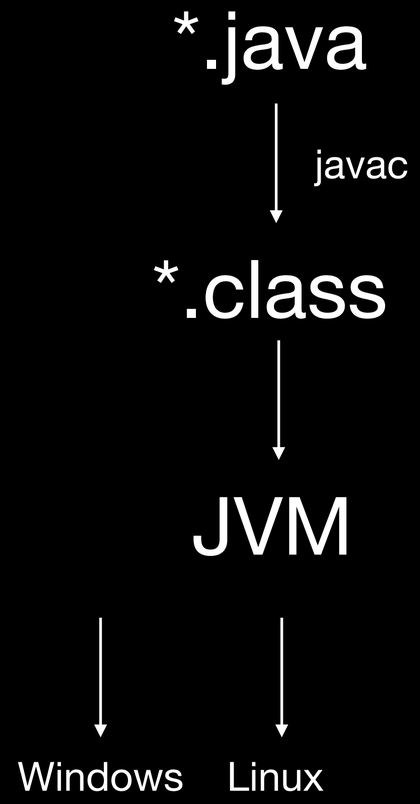
Less Performance*

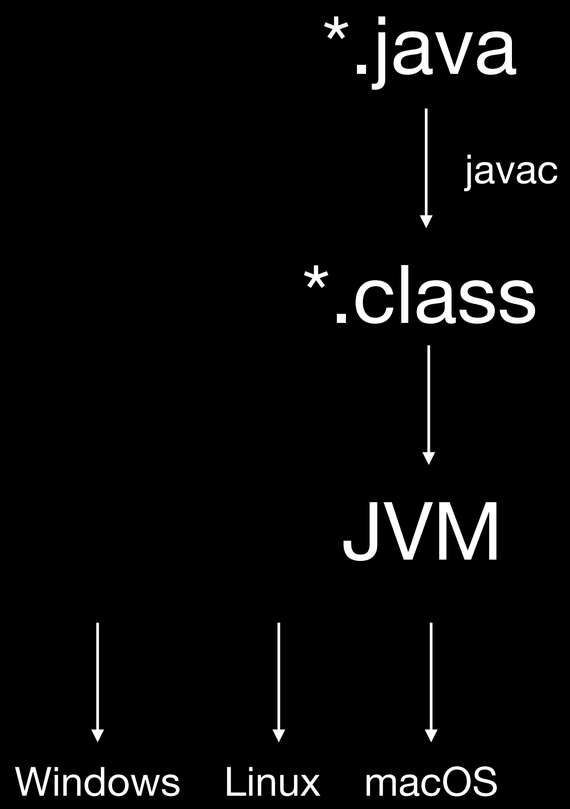
** Especially during the early days*

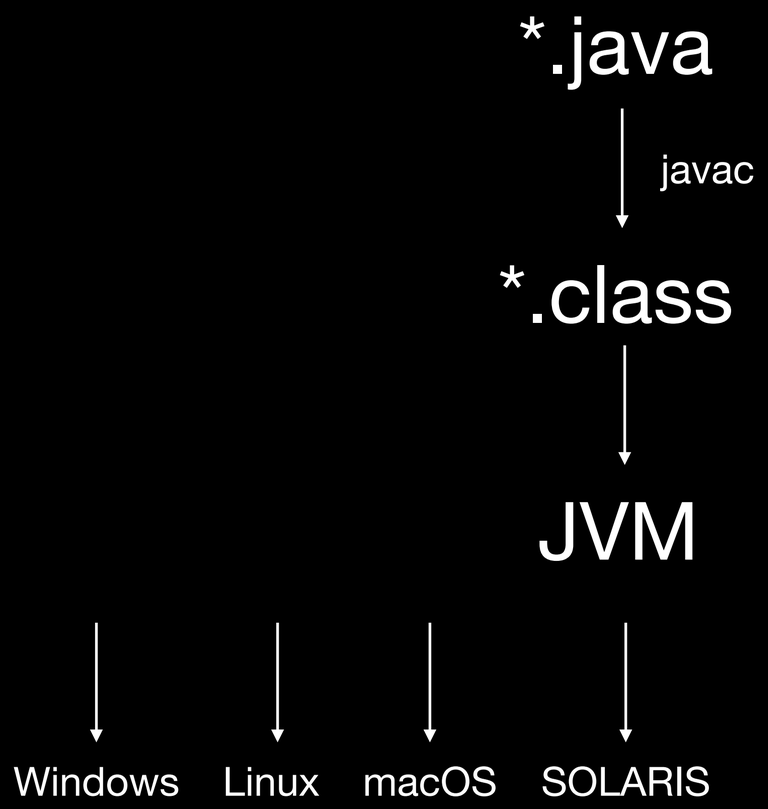

*.java
javac
*.class
(i.e. Bytecode)


*.java
javac
*.class
JVM


*.java
javac
*.class
JVM
Windows


*.java
javac
*.class
JVM
Windows
Linux

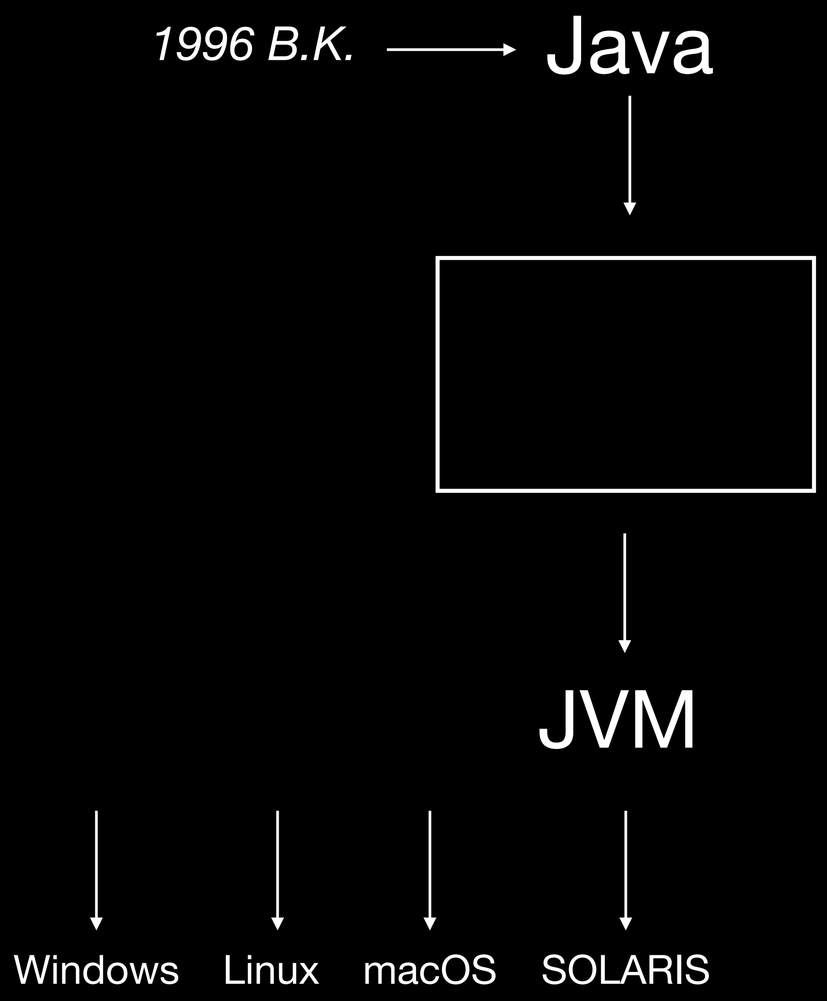

*.java
javac
*.class
JVM
Windows
Linux
macOS

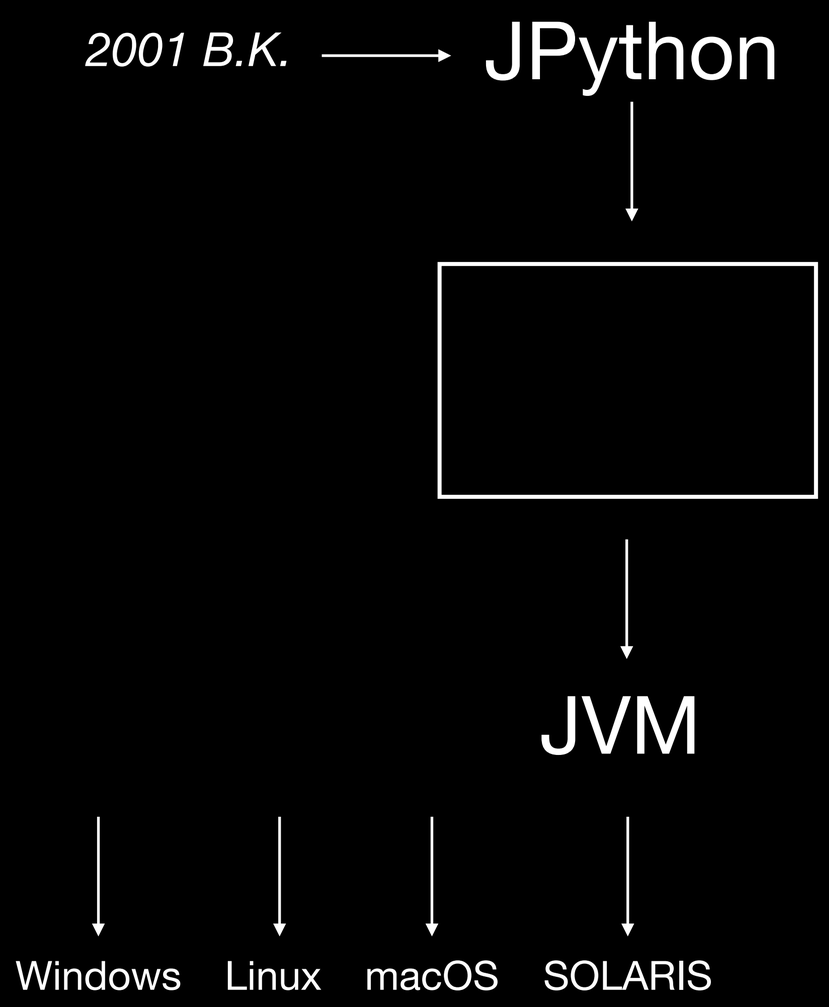

*.java
javac
*.class
JVM
Windows
Linux
macOS
SOLARIS

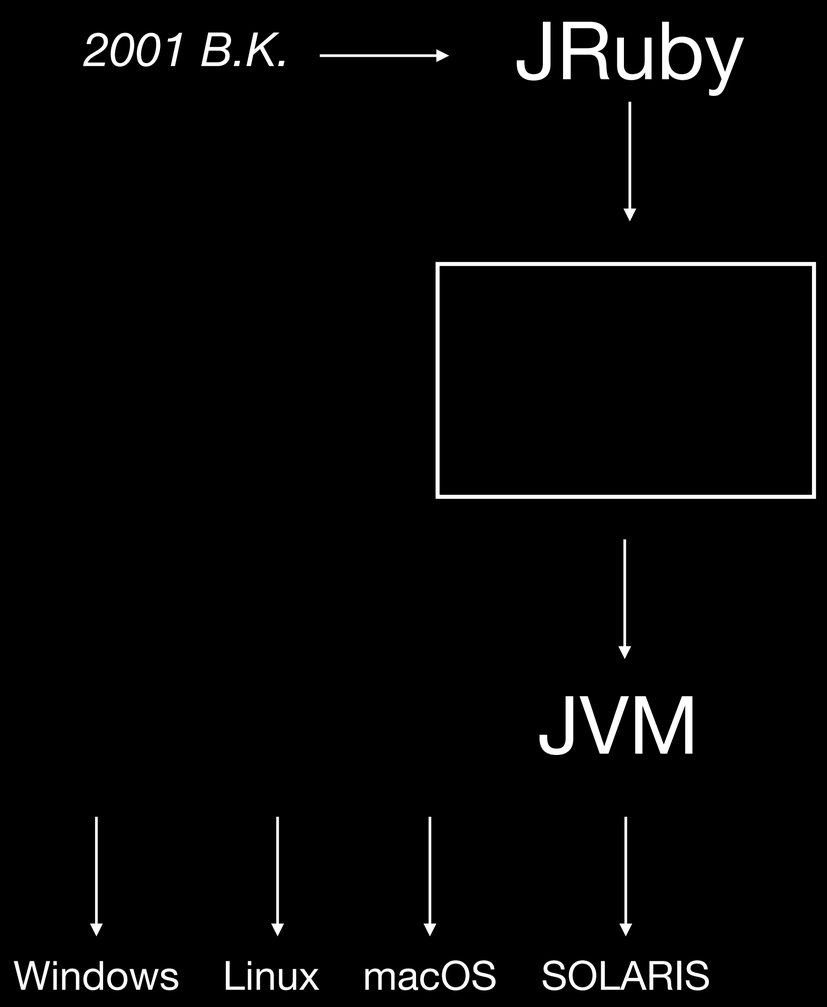
JVM = 

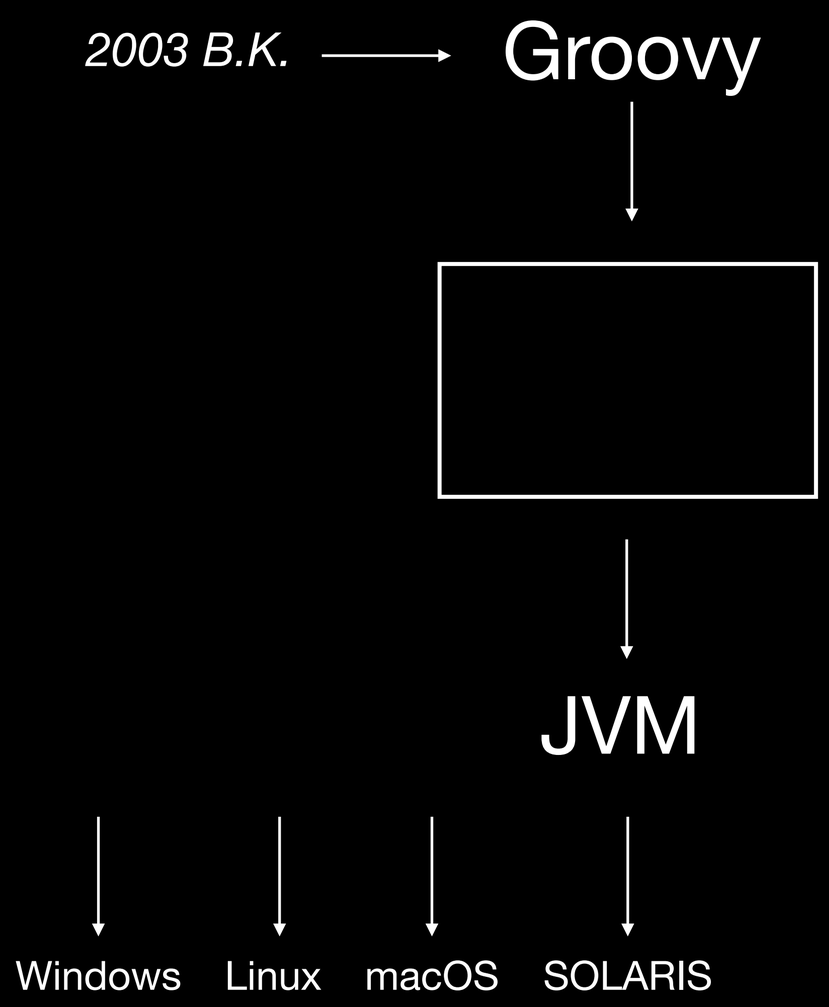
JVM = 

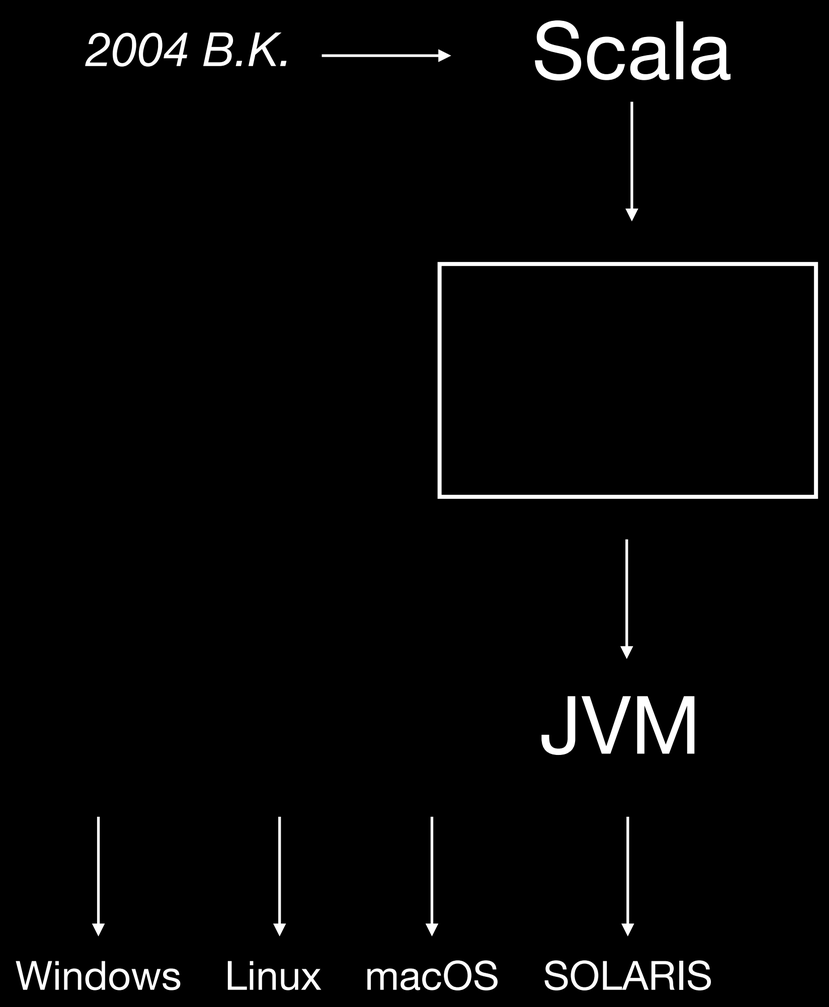
Java = 

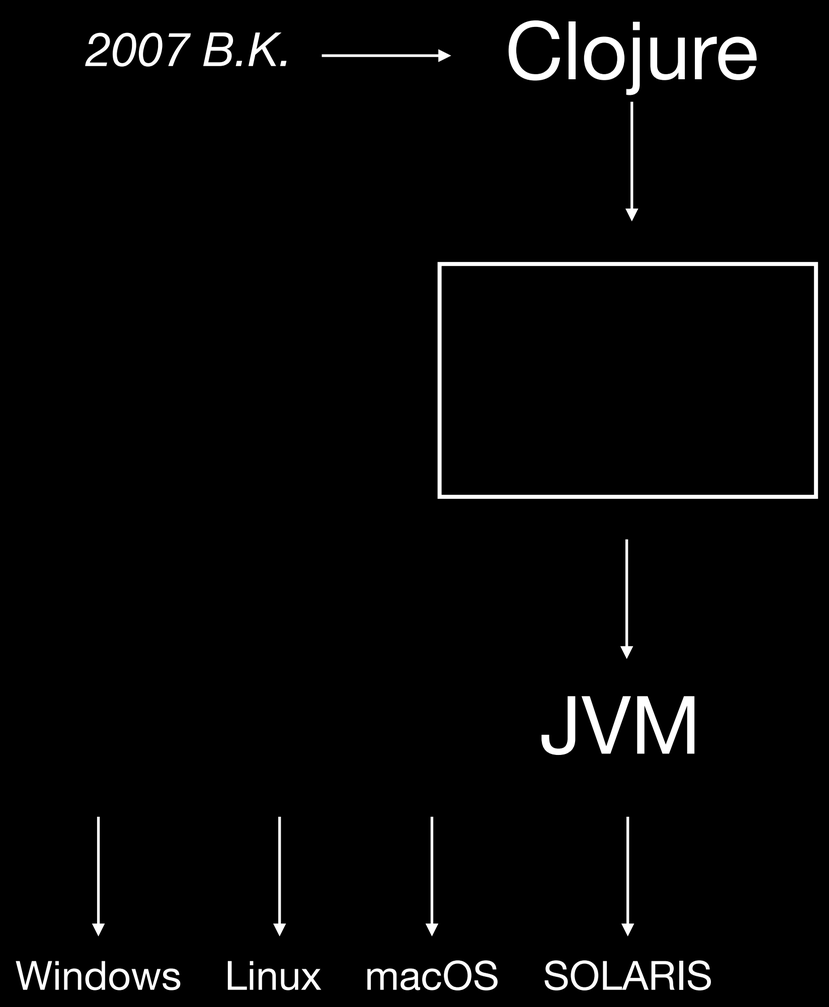

1996 B.K.
Java
JVM
Windows
Linux
macOS
SOLARIS

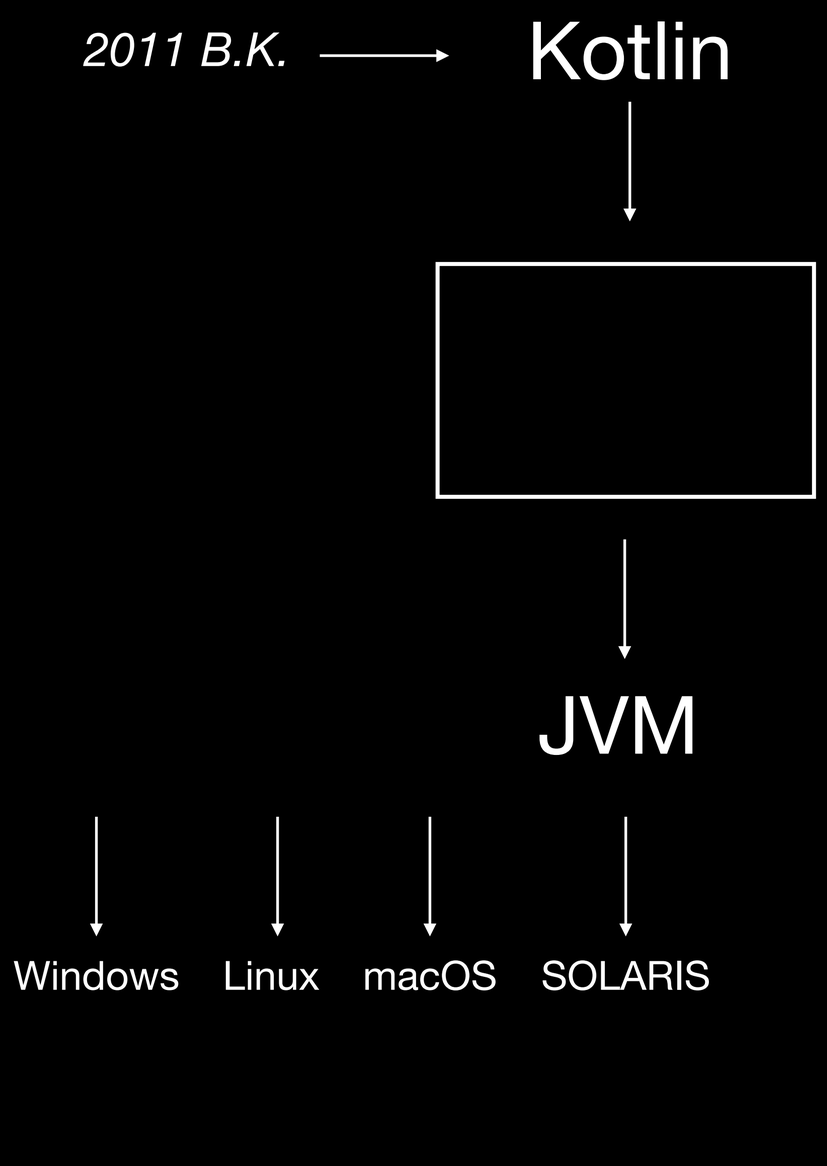

2001 B.K.
JPython
JVM
Windows
Linux
macOS
SOLARIS


2001 B.K.
JRuby
JVM
Windows
Linux
macOS
SOLARIS


2003 B.K.
Groovy
JVM
Windows
Linux
macOS
SOLARIS


2004 B.K.
Scala
JVM
Windows
Linux
macOS
SOLARIS


2007 B.K.
Clojure
JVM
Windows
Linux
macOS
SOLARIS


2011 B.K.
Kotlin
JVM
Windows
Linux
macOS
SOLARIS

0 A.K.

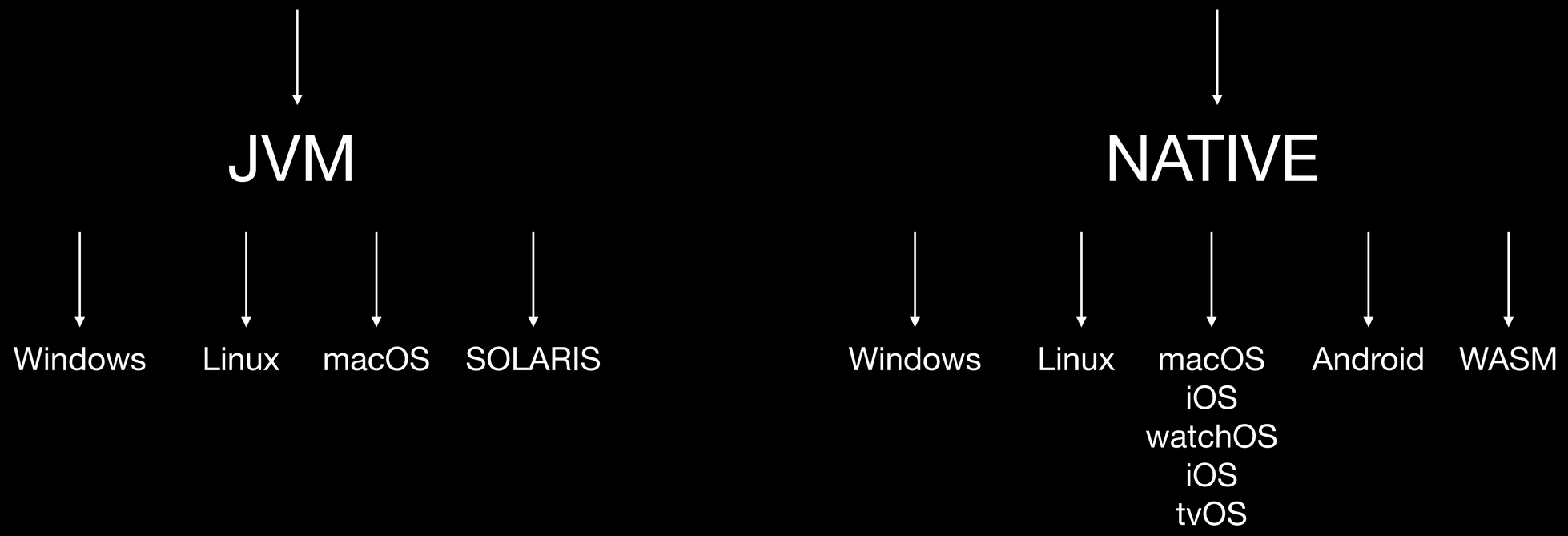
Kotlin is born
But why?

JetBrains

100k lines of Java code
Exposure to tooling
Exposure to languages

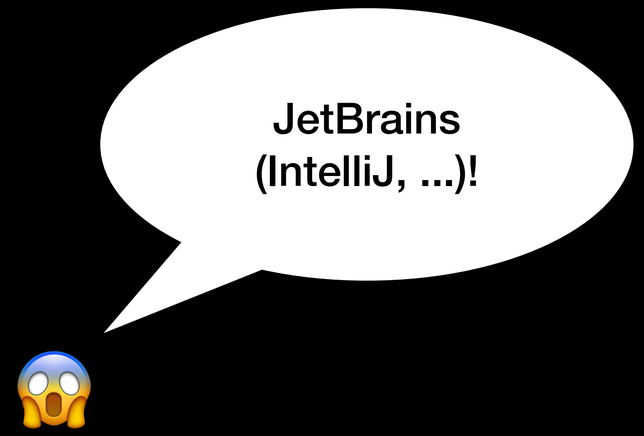
1 language to rule them all?

🤔

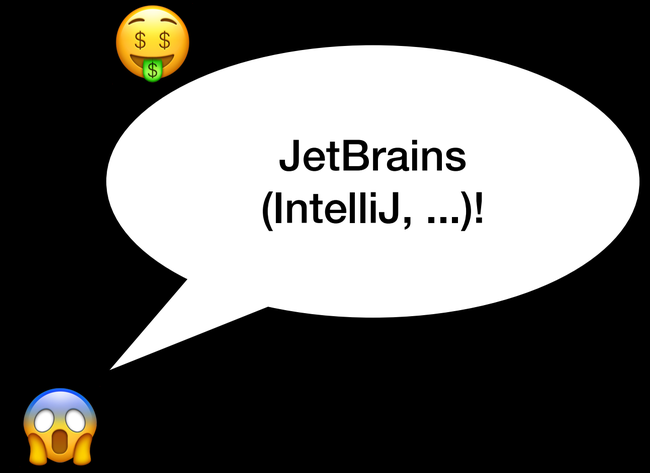


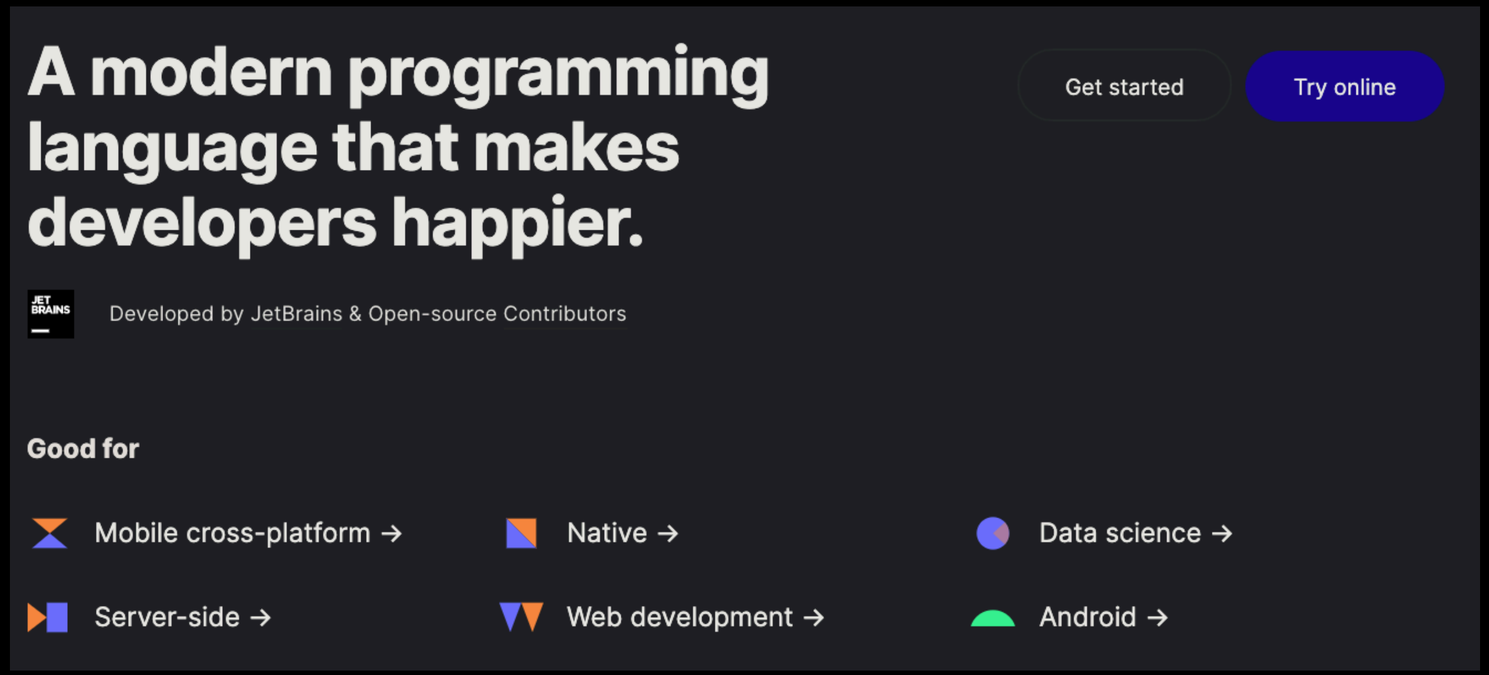
1 language to rule them all?

1 language to rule them all?



1 language to rule them all?





https://kotlinlang.org/

Stories of Kotlin for server-side

Kotlin at Shazam

We talk to Luca Naldini from Shazam about their use of Kotlin on the backend, where they have split a large monolith into a bunch of microservices, some of which are now being written in Kotlin

Listen ↗

Kotlin at Allegro

Allegro is the largest online Polish retailer and in this episode we chat with Rafal and Konrad about their adoption of Kotlin, how they first used it on server-side then mobile

Listen ↗

Greenfield Kotlin at OLX

How do you go about adopting Kotlin for a financial application when you`re asked questions about it`s validity, why it won`t become another {xyz} language or be abandoned?

Listen ↗

Using Kotlin for backend development at Flux

We chat with the folks from Flux around their use of Kotlin on the backend, to develop a service that replaces paper receipts

Listen ↗

QLDB at Amazon

Amazon Quantum Ledger Database (QLDB) is, their usage of Kotlin, why they choose Kotlin over Java for this new project, and how their overall experience has been

Listen ↗

Kotless

We chat with Vladislav about Kotless, a Kotlin Serverless Framework, which eliminates the need for a deployment DSL and makes serverless computations easily understandable for anyone familiar with event-based architectures

Listen ↗

Ktor with Ryan Harter

We talk to Ryan about Ktor, the asynchronous framework for connected systems, developed and maintained by JetBrains. We talk about how it can be used for developing server side applications and its differences with some other existing ones

Listen ↗

KotlinFest2019: Future of Jira Software powered by Kotlin

This is a story of Jira Software services built in Kotlin with Spring Boot and Project Reactor, and the success in interoperability with Java

Listen ↗

https://kotlinlang.org/lp/server-side/

INTUIT Blog

Kotlin Development Plan

UNCATEGORIZED

June 21, 2019 / Shelby Cohen / Katie Levy

Context

The original purpose of this paper was to help drive the consideration to introduce Kotlin at Intuit. The current version is a slight variation of the paper featuring me and Katie Levy's opinions and observations and why we think Kotlin has helped increase our developer productivity velocity.

5 Reasons Why N26 is Moving to Kotlin

Pat Kua Follow
May 3, 2018 · 4 min read

Like any modern technology company, we use many tools to brin
banking platform to life. Many of our backend services, for exam
implemented in Java. And our engineering team has tried out bo
and Typescript — but later decided it wasn't for us.

talkingkotlin.com/qldb/

QLDB at Amazon

Recorded March 05, 2020. Published June 30, 2020

We sit down with the folks from the team at AWS that work on QLDB, to discuss what Amazon Quantum Ledger Database (QLDB) is, their usage of Kotlin, why they choose Kotlin over Java for this new project, and how their overall experience has been.

SPRUCE

Why We Write Micro-Services in Kotlin

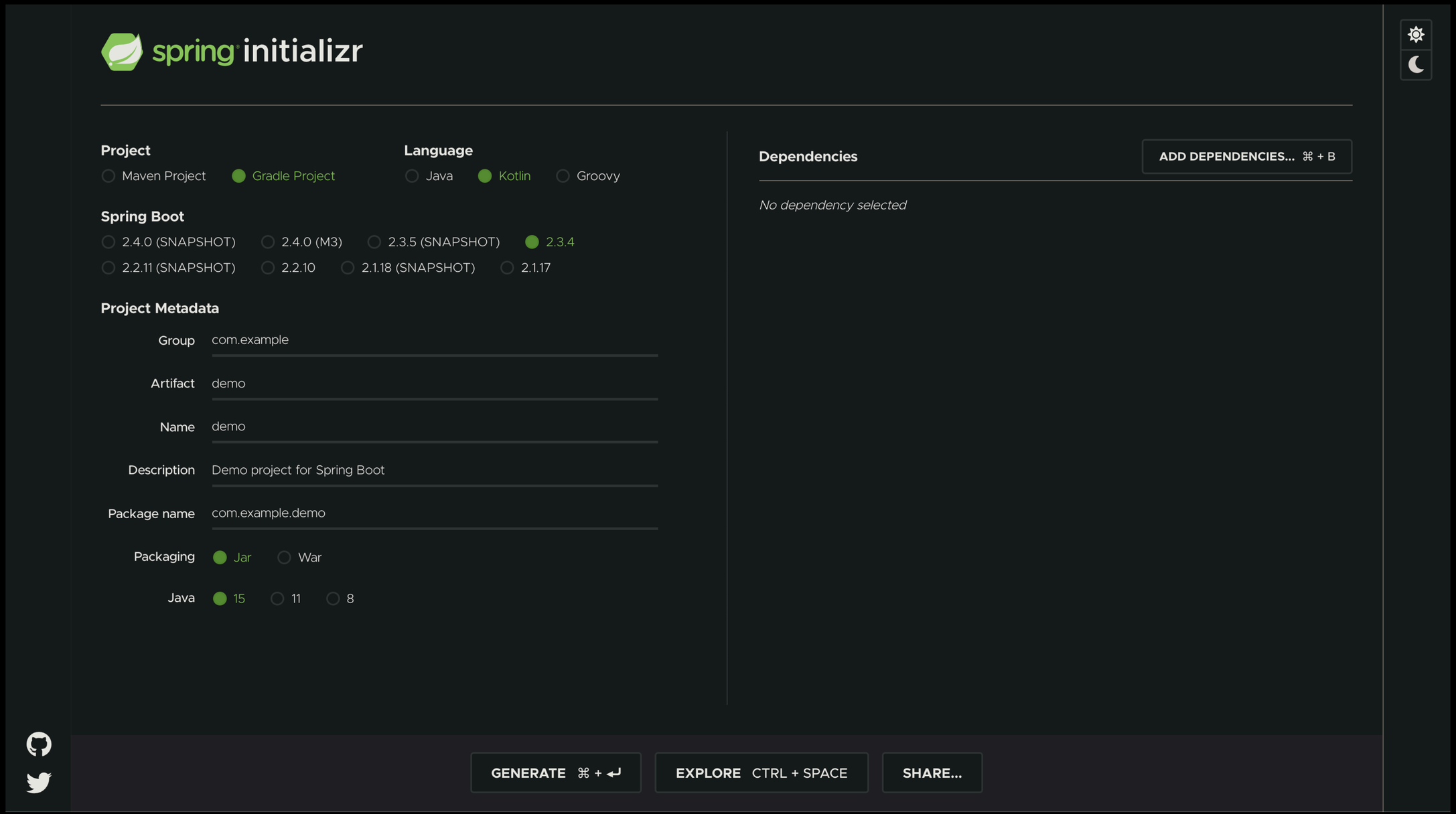
Written by **Alex Zuzin**
March 5, 2019

ING Blog

Introducing Kotlin at ING, a long but rewarding story

julien lengrand-lambert Follow
Dec 3, 2019 · 11 min read

This article describes our journey being the first team to run Kotlin in the backend at ING (Netherlands). Hopefully this article will give you some keys on how to introduce it as well in your own large company, and reduce some of the friction you might encounter.



https://start.spring.io/

Java Kotlin

```kotlin
@GetMapping("/owners/{ownerId}/pets/{petId}")
fun findPet(@PathVariable ownerId: Long, @PathVariable petId: Long): Pet {
    // ...
}
```

You can declare URI variables at the class and method levels, as the following example shows:

Java Kotlin

```kotlin
@Controller
@RequestMapping("/owners/{ownerId}")
class OwnerController {

    @GetMapping("/pets/{petId}")
    fun findPet(@PathVariable ownerId: Long, @PathVariable petId: Long): Pet {
        // ...
    }
}
```

URI variables are automatically converted to the appropriate type, or `TypeMismatchException` is raised. Simple types (`int`, `long`, `Date`, and so on) are supported by default and you can register support for any other data type. See Type Conversion and `DataBinder`.

You can explicitly name URI variables (for example, `@PathVariable("customId")`), but you can leave that detail out if the names are the same and your code is compiled with debugging information or with the `-parameters` compiler flag on Java 8.

The syntax `{varName:regex}` declares a URI variable with a regular expression that has syntax of `{varName:regex}`. For example, given URL `"/spring-web-3.0.5 .jar"`, the following method extracts the name, version, and file extension:

Java Kotlin

```kotlin
@GetMapping("/{name:[a-z-]+}-{version:\\d\\.\\d\\.\\d}{ext:\\.[a-z]+}")
fun handle(@PathVariable name: String, @PathVariable version: String, @PathVariable ext: String) {
    // ...
}
```

URI path patterns can also have embedded `${…}` placeholders that are resolved on startup by using `PropertyPlaceHolderConfigurer` against local, system, environment, and other property sources. You can use this, for example, to parameterize a base URL based on some external configuration.

Spring MVC uses the `PathMatcher` contract and the `AntPathMatcher` implementation from `spring-core` for URI path matching.

```java
public class Person {
}
```

```java
public class Person {
    private final String name;
    private int age;
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```java
@AllArgsConstructor
@Data
class Person {
    private final String name;
    private int age;

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
data class Person(val name: String, var age: Int = 0)
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

```
data:
- copy()
- toString()
- equals()
- hashCode()
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

property declaration
& constructor signature

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

val = immutable

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

var = mutable

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
) {}
```

optional

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

alternative with Java 11+ ?

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```kotlin
record Person(
    String name,
    int age
) {}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```java
record Person(
    String name,
    int age
) {}
```

immutable by default

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public Person(final String name) {
        this.name = name;
        age = 0;
    }

    public Person copy() {
        return new Person(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
            return false;
        Person person = (Person) o;
        return age == person.age &&
            name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

```java
record Person(
    String name,
    int age
) {}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun main() {
    val andrew = Person("Andrew")
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun main() {
    val andrew = Person("Andrew")
}
```

top-level functions!

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun main() {
    val andrew = Person("Andrew")
}
```

no new

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun main() {
    val andrew = Person("Andrew")
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

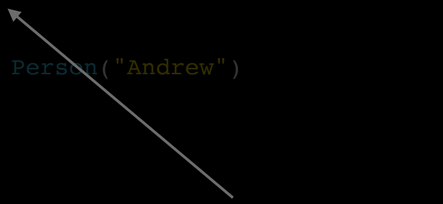
```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun main() {
    val andrew = Person(name = "Andrew", age = 21)
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun main() {
    val andrew = Person("Andrew")
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

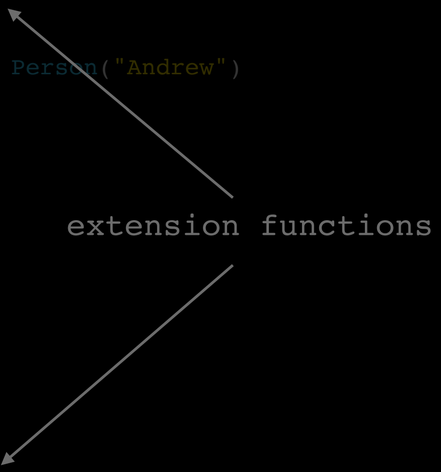
```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun Person.sayHello(): String = "Hello, $name"

fun main() {
    val andrew = Person("Andrew")
}
```



extension functions

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public String sayHello() {
        return MessageFormat.format("Hello, {0}", name);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun Person.sayHello(): String = "Hello, $name"

fun main() {
    val andrew = Person("Andrew")
}
```




```kotlin
fun String.sayHello(): String = "$Hello, $this"
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public String sayHello() {
        return MessageFormat.format("Hello, {0}", name);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun Person.sayHello(): String = "Hello, $name"

fun main() {
    val andrew = Person("Andrew")
}
```

extension functions

```kotlin
fun String.sayHello(): String = "$Hello, $this"
"Andrew".sayHello() // Hello, Andrew
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public String sayHello() {
        return MessageFormat.format("Hello, {0}", name);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun Person.sayHello(): String = "Hello, $name"

fun main() {
    val andrew = Person("Andrew")
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public String sayHello() {
        return MessageFormat.format("Hello, {0}", name);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
) {
    fun sayHello(): String = "Hello, $name"
}

fun Person.sayHello(): String = "Hello, $name"

fun main() {
    val andrew = Person("Andrew")
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public String sayHello() {
        return MessageFormat.format("Hello, {0}", name);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun Person.sayHello(): String = "Hello, $name"

fun main() {
    val andrew = Person("Andrew")
    println(andrew.sayHello()) // Hello, Andrew
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public String sayHello() {
        return MessageFormat.format("Hello, {0}", name);
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
        println(andrew.sayHello()); // Hello, Andrew
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun Person.sayHello(): String = "Hello, $name"

operator fun Person.plus(lifetime: Int): Person {
    age += lifetime
    return this
}

fun main() {
    val andrew = Person("Andrew")
    println(andrew.sayHello()) // Hello, Andrew
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public String sayHello() {
        return MessageFormat.format("Hello, {0}", name);
    }

    public Person plus(final int lifetime) {
        age += lifetime;
        return this;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
        println(andrew.sayHello()); // Hello, Andrew
    }
}
```

```kotlin
data class Person(
    val name: String,
    var age: Int = 0
)

fun Person.sayHello(): String = "Hello, $name"

operator fun Person.plus(lifetime: Int): Person {
    age += lifetime
    return this
}

fun main() {
    val andrew = Person("Andrew")
    println(andrew.sayHello()) // Hello, Andrew
    println(andrew + 20) // Person(name=Andrew, age=20)
}
```

```java
public class Person {
    private final String name;
    private int age;

    public Person(final String name, final int age) {
        this.name = name;
        this.age = age;
    }

    public String sayHello() {
        return MessageFormat.format("Hello, {0}", name);
    }

    public Person plus(final int lifetime) {
        age += lifetime;
        return this;
    }

    @Override
    public String toString() {
        return "Person{" +
            "name='" + name + '\'' +
            ", age=" + age +
            '}';
    }

    public static void main(String[] args) {
        var andrew = new Person("Andrew", 0);
        println(andrew.sayHello()); // Hello, Andrew
        println(andrew.plus(20)); // Person{name='Andrew', age=20}
    }
}
```

```java
public interface Expr {
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2)); // 2
```

```kotlin
sealed class Expr {
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
}
```


sealed class = super-powered enums

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
        }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
        }
    }
}
```

when = switch + pattern matching

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
        }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
        }
    }
}
```

'when' expression must be exhaustive, add necessary 'is Div' branch or 'else' branch instead

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            else -> number
        }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> TODO()
        }
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr: Expr = Expr.Div(4)
println(expr.evaluate(2)) // 2
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

alternative with Java 11+ ?

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

```java
public interface Expr {
    record Add(int value) implements Expr {}
    record Sub(int value) implements Expr {}
    record Mult(int value) implements Expr {}
    record Div(int value) implements Expr {}

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

```java
public interface Expr {
    record Add(int value) implements Expr {}
    record Sub(int value) implements Expr {}
    record Mult(int value) implements Expr {}
    record Div(int value) implements Expr {}

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

```java
public interface Expr {
    record Add(int value) implements Expr {}
    record Sub(int value) implements Expr {}
    record Mult(int value) implements Expr {}
    record Div(int value) implements Expr {}

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

```java
public interface Expr {
    record Add(int value) implements Expr {}
    record Sub(int value) implements Expr {}
    record Mult(int value) implements Expr {}
    record Div(int value) implements Expr {}

    default int evaluate(int number) {
        if (this instanceof Add add) {
            return add.value + number;
        } else if (this instanceof Sub sub) {
            return sub.value - number;
        } else if (this instanceof Mult mult) {
            return mult.value * number;
        } else if (this instanceof Div div) {
            return div.value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

```java
public interface Expr {
    record Add(int value) implements Expr {}
    record Sub(int value) implements Expr {}
    record Mult(int value) implements Expr {}
    record Div(int value) implements Expr {}

    default int evaluate(int number) {
        if (this instanceof Add add) {
            return add.value + number;
        } else if (this instanceof Sub sub) {
            return sub.value - number;
        } else if (this instanceof Mult mult) {
            return mult.value * number;
        } else if (this instanceof Div div) {
            return div.value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

```java
public interface Expr {
    record Add(int value) implements Expr {}
    record Sub(int value) implements Expr {}
    record Mult(int value) implements Expr {}
    record Div(int value) implements Expr {}

    default int evaluate(int number) {
        if (this instanceof Add add) {
            return add.value + number;
        } else if (this instanceof Sub sub) {
            return sub.value - number;
        } else if (this instanceof Mult mult) {
            return mult.value * number;
        } else if (this instanceof Div div) {
            return div.value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

add = ((Add) this)

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

```java
public interface Expr {
    record Add(int value) implements Expr {}
    record Sub(int value) implements Expr {}
    record Mult(int value) implements Expr {}
    record Div(int value) implements Expr {}

    default int evaluate(int number) {
        if (this instanceof Add add) {
            return add.value + number;
        } else if (this instanceof Sub sub) {
            return sub.value - number;
        } else if (this instanceof Mult mult) {
            return mult.value * number;
        } else if (this instanceof Div div) {
            return div.value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

```java
public interface Expr {
    record Add(int value) implements Expr {}
    record Sub(int value) implements Expr {}
    record Mult(int value) implements Expr {}
    record Div(int value) implements Expr {}

    default int evaluate(int number) {
        if (this instanceof Add add) {
            return add.value + number;
        } else if (this instanceof Sub sub) {
            return sub.value - number;
        } else if (this instanceof Mult mult) {
            return mult.value * number;
        } else if (this instanceof Div div) {
            return div.value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

```java
public interface Expr {
    record Add(int value) implements Expr {}
    record Sub(int value) implements Expr {}
    record Mult(int value) implements Expr {}
    record Div(int value) implements Expr {}

    default int evaluate(int number) {
        return switch (this) {
            case Add(int value) -> value + number;
            case Sub(int value) -> value - number;
            case Mult(int value) -> value * number;
            case Div(int value) -> value / number;
        };
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

16-18

```kotlin
sealed class Expr {
    class Add(val value: Int) : Expr()
    class Sub(val value: Int) : Expr()
    class Mult(val value: Int) : Expr()
    class Div(val value: Int) : Expr()

    fun evaluate(number: Int): Int {
        return when (this) {
            is Add -> value + number
            is Sub -> value - number
            is Mult -> value * number
            is Div -> value / number
        }
    }
}

val expr = Expr.Div(4)
println(expr.evaluate(2))
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class AuthState {
    data class LoggedIn(val token: UUID) : AuthState()
    object LoggedOut : AuthState()
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class AuthState {
    data class LoggedIn(val token: UUID) : AuthState()
    object LoggedOut : AuthState()
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class AuthState {
    data class LoggedIn(val token: UUID) : AuthState()
    object LoggedOut : AuthState()
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class AuthState {
    data class LoggedIn(val token: UUID) : AuthState()
    object LoggedOut : AuthState()
}

data class User(
    val name: String,
    var authState: AuthState = AuthState.LoggedIn(UUID.randomUUID())
)
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class AuthState {
    data class LoggedIn(val token: UUID) : AuthState()
    object LoggedOut : AuthState()
}

data class User(
    val name: String,
    var authState: AuthState = AuthState.LoggedIn(UUID.randomUUID())
)

fun User.logOut() {
    authState = AuthState.LoggedOut
}
```

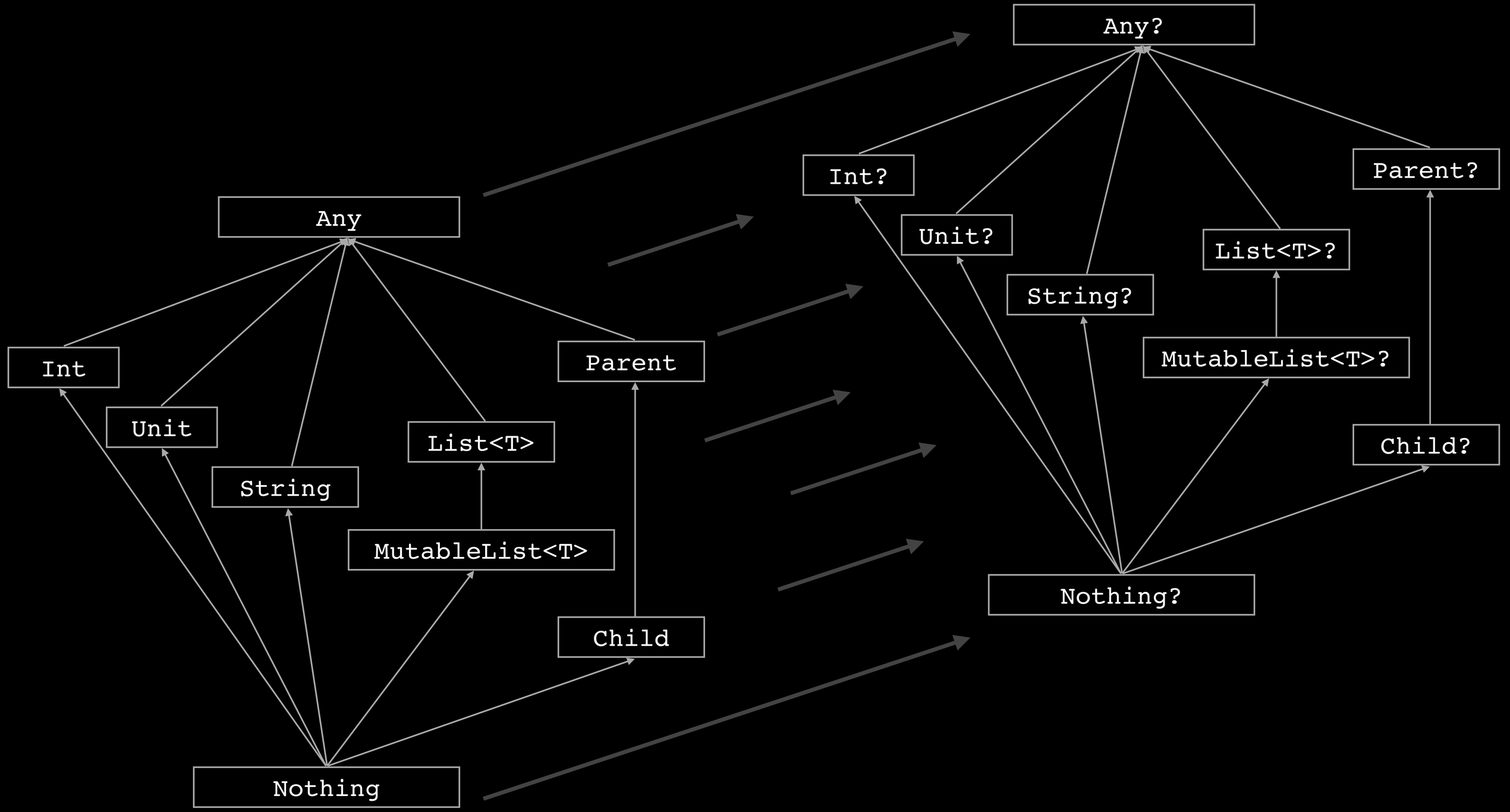
```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

```kotlin
sealed class AuthState {
    data class LoggedIn(val token: UUID) : AuthState()
    object LoggedOut : AuthState()
}

data class User(
    val name: String,
    var authState: AuthState = AuthState.LoggedIn(UUID.randomUUID())
)

fun User.logOut() {
    authState = AuthState.LoggedOut
}

fun redirectUserBasedOnAuthState(user: User) {
    when (user.authState) {
        is AuthState.LoggedIn ->
            println("Already logged in, redirect to profile page")
        is AuthState.LoggedOut ->
            println("Need to log in, redirect to login page")
    }
}
```

```java
public interface Expr {
    class Add implements Expr {
        private final int value;
        public Add(int value) { this.value = value; }
    }

    class Sub implements Expr {
        private final int value;
        public Sub(int value) { this.value = value; }
    }

    class Mult implements Expr {
        private final int value;
        public Mult(int value) { this.value = value; }
    }

    class Div implements Expr {
        private final int value;
        public Div(int value) { this.value = value; }
    }

    default int evaluate(int number) {
        if (this instanceof Add) {
            return ((Add) this).value + number;
        } else if (this instanceof Sub) {
            return ((Sub) this).value - number;
        } else if (this instanceof Mult) {
            return ((Mult) this).value * number;
        } else if (this instanceof Div) {
            return ((Div) this).value / number;
        } else {
            throw new IllegalStateException(":(");
        }
    }
}

Expr expr = new Div(4);
println(expr.evaluate(2));
```

> *I call it my billion-dollar mistake. It was the invention of the null reference in 1965... I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement.*
> *— Tony Hoare*

To be, or not to be *null*


Any
Int
Unit
String
List<T>
Parent
MutableList<T>
Child
Nothing
Any?
Int?
Unit?
String?
List<T>?
Parent?
MutableList<T>?
Child?
Nothing?

```java
String str = "";
str = null;
```

```kotlin
var str = ""
str = null
```

```java
String str = "";
str = null;
```

```kotlin
var str = ""
str = null
```

Null can not be a value of a non-null type String

```java
String str = "";
str = null;
```

```kotlin
var str: String = ""
str = null
```

Null can not be a value of a non-null type String

```java
String str = "";
str = null;
```

```kotlin
var str: String? = ""
str = null
```

```java
String str = "";
str = null;
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = level1.level2.level3.value; // NPE!!!
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = level1.level2.level3.value; // NPE!!!
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value: String? = level1.level2?.level3?.value
```


?. = safe call OU nullable chaining

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = level1.level2.level3.value; // NPE!!!
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = level1.level2.level3.value; // NPE!!!
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = null;
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            println("💩");
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));

if (level1.level2 == null
    || level1.level2.level3 == null
    || level1.level2.level3.value == null) {
    throw new IllegalStateException(":(");
}
String value = level1.level2.level3.value;
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value
```

```java
@AllArgsConstructor
class Level1 {
    Optional<Level2> level2;
}

@AllArgsConstructor
class Level2 {
    Optional<Level3> level3;
}

@AllArgsConstructor
class Level3 {
    Optional<String> value;
}

Optional<Level1> level1Maybe = Optional.of(
    new Level1(Optional.of(
        new Level2(Optional.empty()))));
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value
```

```java
@AllArgsConstructor
class Level1 {
    Optional<Level2> level2;
}

@AllArgsConstructor
class Level2 {
    Optional<Level3> level3;
}

@AllArgsConstructor
class Level3 {
    Optional<String> value;
}

Optional<Level1> level1Maybe = Optional.of(
    new Level1(Optional.of(
        new Level2(Optional.empty()))));

level1Maybe.ifPresent(level1 -> {
    level1.level2.ifPresent(level2 -> {
        level2.level3.ifPresent(level3 -> {
            level3.value.ifPresent(value -> {
                // UMMMMMMMMM
                ...
            });
        });
    });
});
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value
```

```java
@AllArgsConstructor
class Level1 {
    Optional<Level2> level2;
}

@AllArgsConstructor
class Level2 {
    Optional<Level3> level3;
}

@AllArgsConstructor
class Level3 {
    Optional<String> value;
}

Optional<Level1> level1Maybe = Optional.of(
    new Level1(Optional.of(
        new Level2(Optional.empty()))));

level1Maybe
    .flatMap(level1 -> level1.level2)
    .flatMap(level2 -> level2.level3)
    .flatMap(level3 -> level3.value)
    .ifPresent(value -> {
        // UMMMMMMMMM
        ...
    });
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value
```

```java
@AllArgsConstructor
class Level1 {
    Optional<Level2> level2;
}

@AllArgsConstructor
class Level2 {
    Optional<Level3> level3;
}

@AllArgsConstructor
class Level3 {
    Optional<String> value;
}

Optional<Level1> level1Maybe = Optional.of(
    new Level1(Optional.of(
        new Level2(Optional.empty()))));

Optional<String> valueMaybe = level1Maybe
    .flatMap(level1 -> level1.level2)
    .flatMap(level2 -> level2.level3)
    .flatMap(level3 -> level3.value);

...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value
```

```java
@AllArgsConstructor
class Level1 {
    Optional<Level2> level2;
}

@AllArgsConstructor
class Level2 {
    Optional<Level3> level3;
}

@AllArgsConstructor
class Level3 {
    Optional<String> value;
}

Optional<Level1> level1Maybe = Optional.of(
    new Level1(Optional.of(
        new Level2(Optional.empty()))));

String value;
if (level1Maybe.isPresent()
    && level1Maybe.get().level2.isPresent()
    && level1Maybe.get().level2.get().level3.isPresent()
    && level1Maybe.get().level2.get().level3.get().value.isPresent()
) {
    value = level1Maybe.get().level2.get().level3.get().value.get();
}
```

YIKES!

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value ?: "default"
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value ?: "default"
```

Elvis operator

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let({
    // lambda body...
})
```

let ~ Optional.map

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let({
    // lambda body...
})
```

let ~ Optional.map

{ ... } = lambda!

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let({
    // lambda body...
})
```

({ lambda }) brackets optional

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let {
    // lambda body...
}
```

({ lambda }) brackets optional

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let {
    // lambda body...
}
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let {
    it // = level3.value
}
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let { level3Value ->
}
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let {
    // Some complicated logic...
}
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let {
    // Some complicated logic...
    something(it)
}
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let {
    // Some complicated logic...
    something(it)
}
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

Nulls to indicate the presence or absence of value

```kotlin
class Level1(val level2: Level2?)
class Level2(val level3: Level3?)
class Level3(val value: String?)


val level1 = Level1(Level2(null))
val value = level1.level2?.level3?.value?.let {
    // Some complicated logic...
    something(it)
}
```

```java
@AllArgsConstructor
class Level1 {
    Level2 level2;
}

@AllArgsConstructor
class Level2 {
    Level3 level3;
}

@AllArgsConstructor
class Level3 {
    String value;
}

Level1 level1 = new Level1(new Level2(null));
String value = "default";
if (level1.level2 != null) {
    if (level1.level2.level3 != null) {
        if (level1.level2.level3.value != null) {
            value = level1.level2.level3.value;
        }
    }
}
...
```

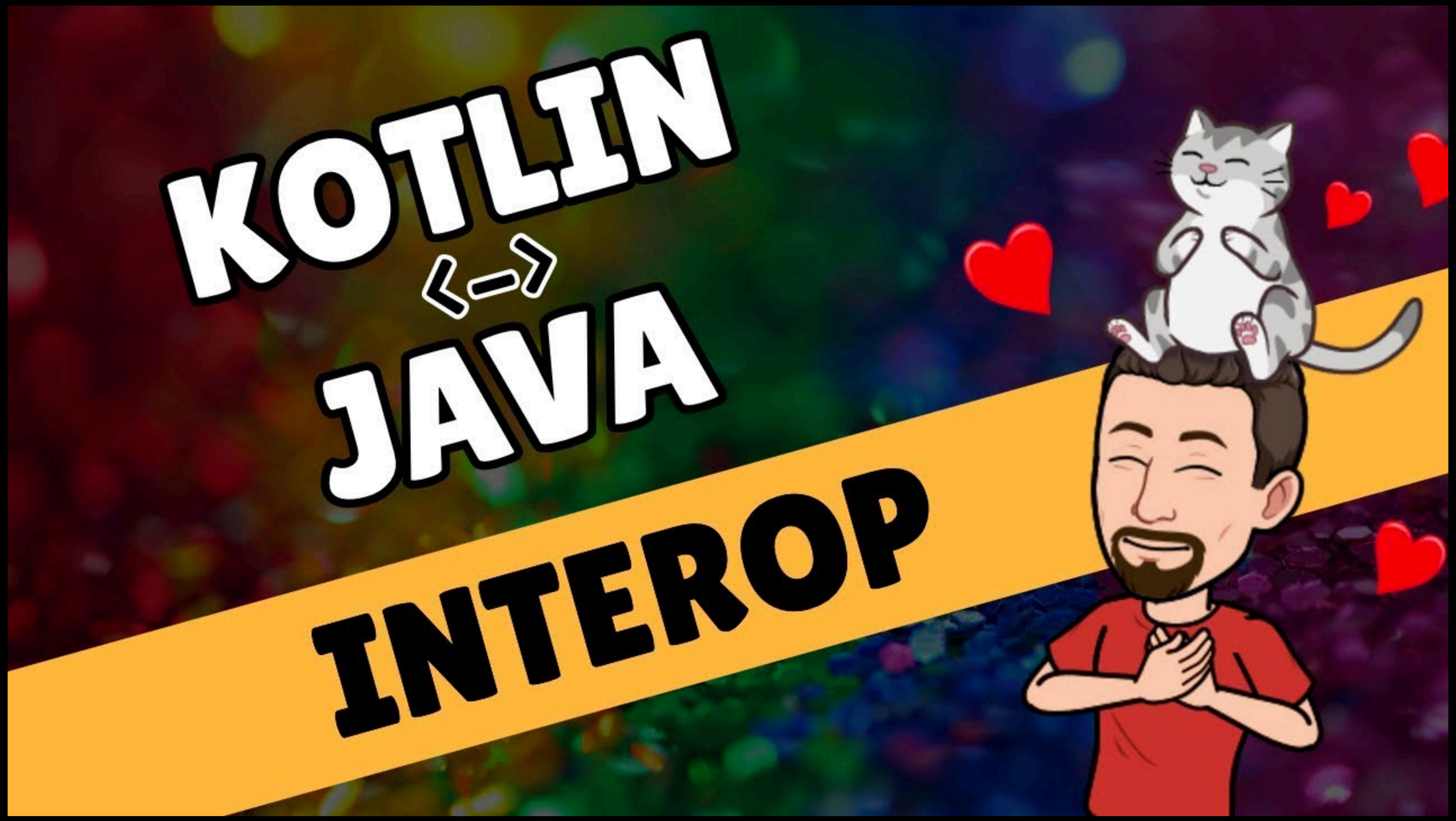
Nulls to indicate the presence or absence of value

No boxing overhead (e.g. Optional)

"{ "name": "Andrew" }"

```kotlin
data class Person(val name: String)


fun <T : Any> String.deserialize(clazz: KClass<T>): T? {
    val regex = "\\{\\s*\"(\\w+)\"\\s*:\\s*\"(\\w+)\"\\s*}".toRegex()
    return regex.find(this)?.destructured?.let { (_, value) ->
        @Suppress("UNCHECKED_CAST")
        clazz.java.constructors[0]?.newInstance(value) as T
    }
}

val person = """{ "name": "Andrew" }""".deserialize(Person::class)
println(person) // Person(name=Andrew)
```

```java
@AllArgsConstructor
@ToString
static class Person {
    final String name;
}

interface StringUtils {
    @SuppressWarnings("unchecked")
    static <T> Optional<T> deserialize(
        String input,
        Class<T> clazz
    ) {
        var regex = new Regex("\\{\\s*\"(\\w+)\"\\s*:\\s*\"(\\w+)\"\\s*}");
        var matches = regex.find(input, 0);
        if (matches != null) {
            var key = matches.getGroupValues().get(0);
            var value = matches.getGroupValues().get(1);
            return Optional.of((T) clazz.getConstructors()[0]
                .newInstance(value));
        }
        return Optional.empty();
    }
}

Optional<Person> deserialized = StringUtils.deserialize(
    "{ \"name\": \"Andrew\" }",
    Person.class);
println(deserialized); // Optional[Person(name=Andrew)]
```



Java Interoperability

```java
public interface JavaInterop {
    static String russianRoulette() {
        return null;
    }
}
```

```kotlin
val maybe = JavaInterop.russianRoulette().length
```

```java
public interface JavaInterop {
    static String russianRoulette() {
        return null;
    }
}
```

```kotlin
val maybe = JavaInterop.russianRoulette().length
```

```java
public interface JavaInterop {
    static String russianRoulette() {
        return null;
    }
}
```

2 choix :

1) assume nullable
2) assume non-null

```kotlin
// NPE!!!
val maybe = JavaInterop.russianRoulette().length
```

```java
public interface JavaInterop {
    static String russianRoulette() {
        return null;
    }
}
```

```kotlin
// OK
val maybe = JavaInterop.russianRoulette()?.length
```

```java
public interface JavaInterop {
    static String russianRoulette() {
        return null;
    }
}
```

Map<Integer, List<Data>>

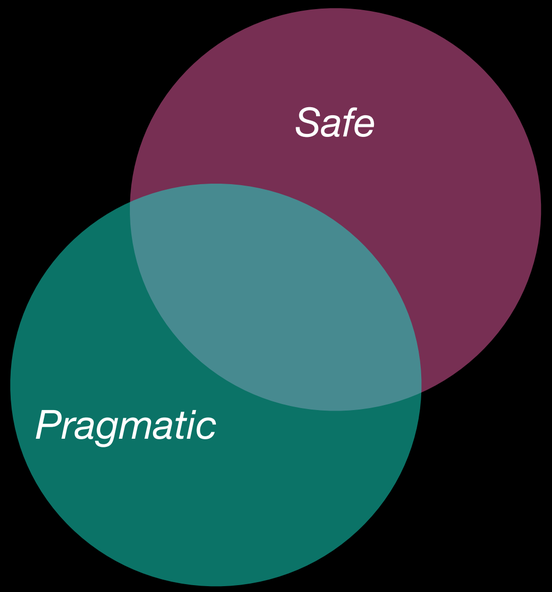
Map<Integer?, List<Data?>?>?

- Check map if null
- Check value (list) if null
- Use index to get item
- Check item if null
- Check if data's fields are null

Map<Integer?, List<Data?>?>?

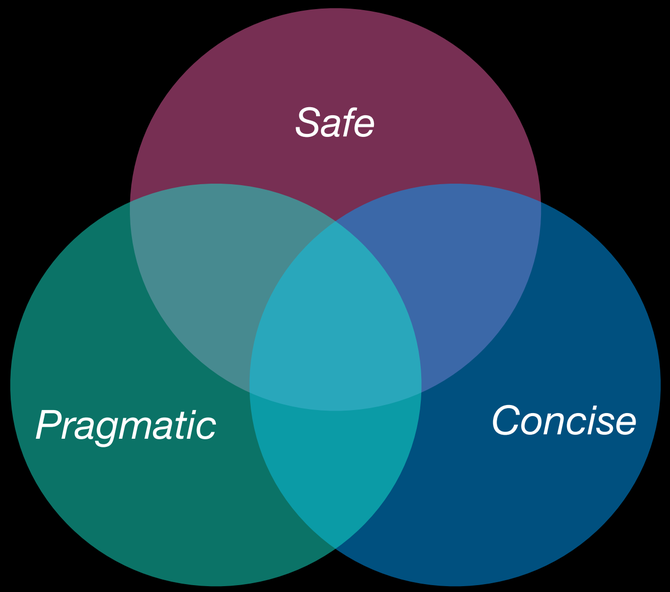
- Check map if null
- Check value (list) if null
- Use index to get item
- Check item if null
- Check if data's fields are null

Map<Integer?, List<Data?>?>?



- Check map if null
- Check value (list) if null
- Use index to get item
- Check item if null
- Check if data's fields are null

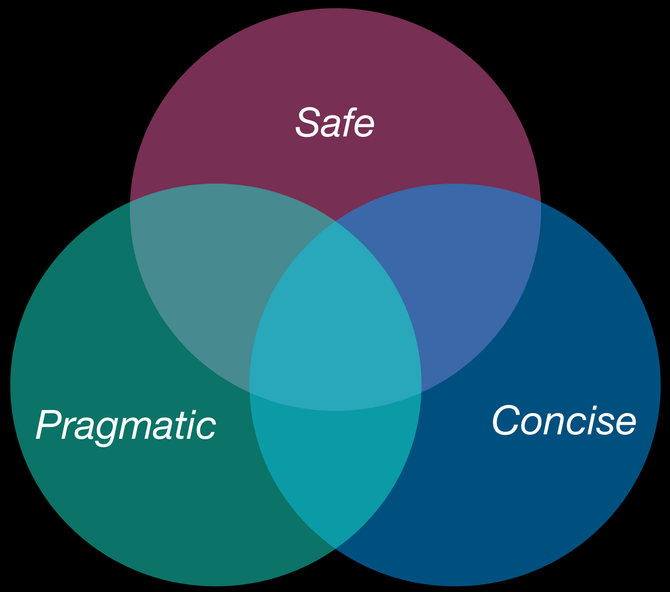
Map<Integer?, List<Data?>?>?



- Check map if null
- Check value (list) if null
- Use index to get item
- Check item if null
- Check if data's fields are null

✅ Safe

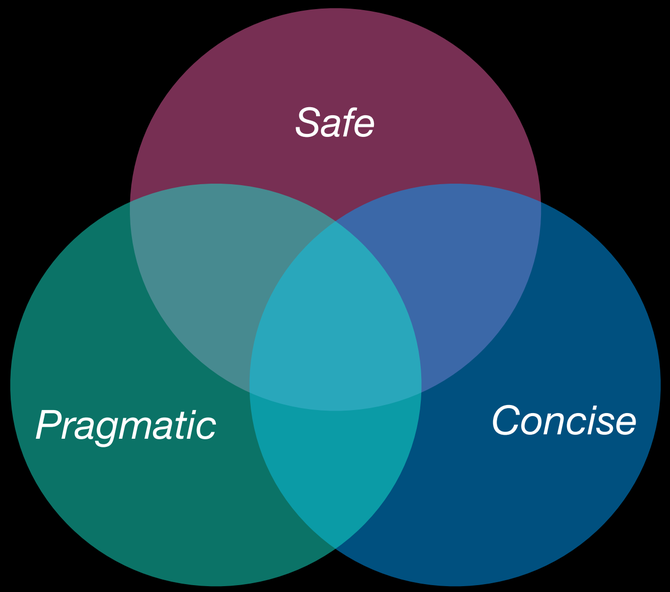
Map<Integer?, List<Data?>?>?



- Check map if null
- Check value (list) if null
- Use index to get item
- Check item if null
- Check if data's fields are null

✅ Safe
👎 Concise

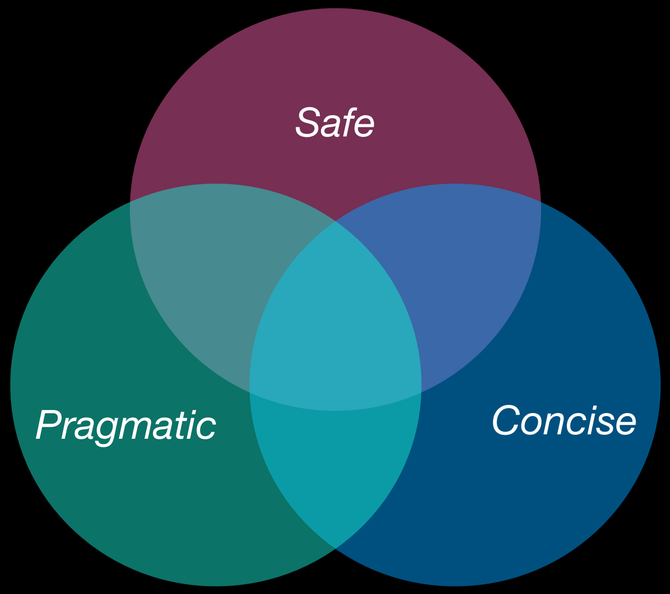
Map<Integer?, List<Data?>?>?



- Check map if null
- Check value (list) if null
- Use index to get item
- Check item if null
- Check if data's fields are null

✅ Safe
👎 Concise
👎 Pragmatic

Map<Integer?, List<Data?>?>?



- Check map if null
- Check value (list) if null
- Use index to get item
- Check item if null
- Check if data's fields are null

```kotlin
// NPE!!!
val maybe = JavaInterop.russianRoulette().length
```

```java
public interface JavaInterop {
    static String russianRoulette() {
        return null;
    }
}
```

possible solution to enforce null safety?

```kotlin
// NPE!!!
val maybe = JavaInterop.russianRoulette().length
```

```java
public interface JavaInterop {
    @Nullable
    static String russianRoulette() {
        return null;
    }
}
```

```kotlin
// Compile error!!!
val maybe = JavaInterop.russianRoulette().length
                                         ...
```

Only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type String?

```java
public interface JavaInterop {
    @Nullable
    static String russianRoulette() {
        return null;
    }
}
```

```kotlin
// SAFE!!!
val maybe = JavaInterop.russianRoulette()?.length
```

```java
public interface JavaInterop {
    @Nullable
    static String russianRoulette() {
        return null;
    }
}
```

```kotlin
// SAFE!!!
val maybe = JavaInterop.russianRoulette()?.length
```

```java
public interface JavaInterop {
    @Nonnull
    static String russianRoulette() {
        return "";
    }
}
```

```kotlin
// Compile warning!!!
val maybe = JavaInterop.russianRoulette()?.length
                                          ...
```

Unnecessary safe call on a non-null receiver of type String!

```java
public interface JavaInterop {
    @Nonnull
    static String russianRoulette() {
        return "";
    }
}
```

```kotlin
val user = User(null, null, 0)
val user = User("1", "Andrew", 0)
user.name = null
```

Type mismatch
Required: String
Found: Nothing

```java
public class User {
    @Nonnull private String id;
    @Nonnull private String name;
    private int age;

    public User(@Nonnull String id, @Nonnull String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    @Nonnull
    public String getName() {
        return name;
    }

    public void setName(@Nonnull String name) {
        this.name = name;
    }
}
```

```kotlin
val user = User(null, null, 0)
val user = User("1", "Andrew", 0)
user.name = null
```

Type mismatch
Required: String
Found: Nothing

```java
public class User {
    @Nonnull private String id;
    @Nonnull private String name;
    private int age;

    public User(@Nonnull String id, @Nonnull String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    @Nonnull
    public String getName() {
        return name;
    }

    public void setName(@Nonnull String name) {
        this.name = name;
    }
}
```

```kotlin
val user = User(null, null, 0)
val user = User("1", "Andrew", 0)
user.name = null
```

Type mismatch
Required: String
Found: Nothing

```java
@EverythingIsNonnullByDefault
public class User {
    private String id;
    private String name;
    private int age;

    public User(String id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```kotlin
val user = User(null, null, 0)
val user = User("1", "Andrew", 0)
user.name = null
```

Type mismatch
Required: String
Found: Nothing

```java
@EverythingIsNonnullByDefault
public class User {
    private String id;
    private String name;
    private int age;

    public User(String id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```java
@Nonnull
@TypeQualifierDefault({
    ElementType.PARAMETER,
    ElementType.CONSTRUCTOR,
    ElementType.FIELD,
    ElementType.LOCAL_VARIABLE,
    ElementType.METHOD})
@Retention(RetentionPolicy.SOURCE)
public @interface EverythingIsNonnullByDefault {
}
```

```kotlin
val user = User(null, null, 0)
val user = User("1", "Andrew", 0)
user.name = null
```

Type mismatch
Required: String
Found: Nothing

```java
@EverythingIsNonnullByDefault
public class User {
    private String id;
    private String name;
    private int age;

    public User(String id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

`user.name = null` is equivalent to `user.setName(null)`

```kotlin
val user = User(null, null, 0)
val user = User("1", "Andrew", 0)
user.name = null
```

OK

Type mismatch
Required: String
Found: Nothing

```java
@EverythingIsNonnullByDefault
public class User {
    private String id;
    @Nullable private String name;
    private int age;

    public User(String id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(@Nullable String name) {
        this.name = name;
    }
}
```

MERCI !

References

[1] https://insights.stackoverflow.com/survey/2020#most-loved-dreaded-and-wanted

[2] https://tjpalmer.github.io/languish/

[3] http://pypl.github.io/PYPL.html

[4] https://blog.jetbrains.com/kotlin/2020/08/the-state-of-kotlin-support-in-spring/