

# Kotlin

## An Introduction



# Who's that guy?



## Martin Häusler

✉ [martin.haeusler89@mail.com](mailto:martin.haeusler89@mail.com) / [martin.haeusler@txture.io](mailto:martin.haeusler@txture.io)

[in](#) [Martin Häusler](#)

[@martinHusler](#)

[DEV https://dev.to/martinhaeusler](https://dev.to/martinhaeusler)

PhD University of Innsbruck 2009 - 2018

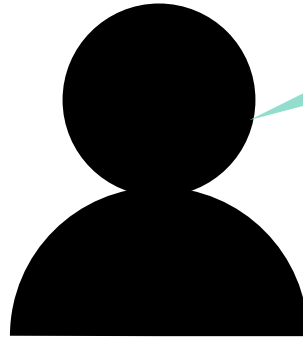
Senior Backend Software Developer &  
Software-Architect at Txture since 2017

My Topics:

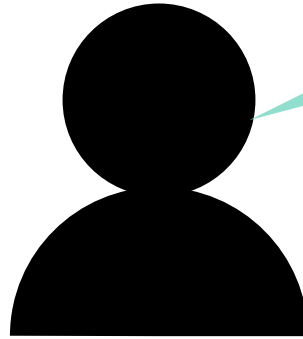
Java, Kotlin, JVM, Databases (Relational, Document, Graph), Data Modelling, Object-Oriented Programming, Functional Programming, Memes & Pop Culture, Gaming, Star Wars, Dungeons & Dragons...

# What is Txture?



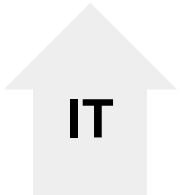


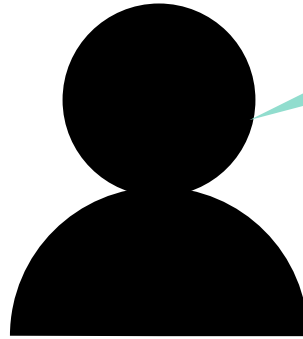
**I want to migrate  
my IT to the Cloud!**



**I want to migrate  
my IT to the Cloud!**

**On-Premise  
and/or Cloud Estate**

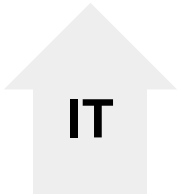




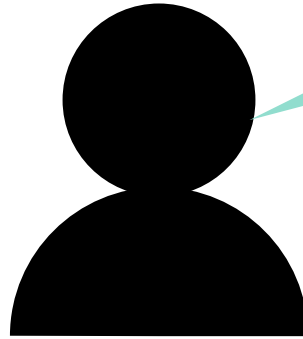
I want to migrate  
my IT to the Cloud!

On-Premise  
and/or Cloud Estate

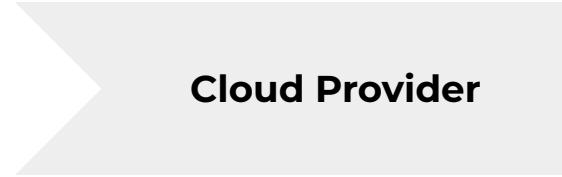
Cloud Provider

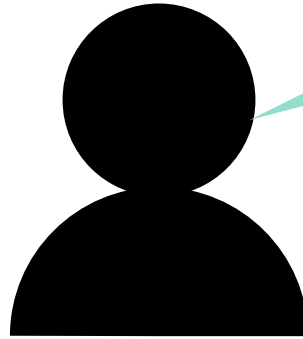


Google Cloud

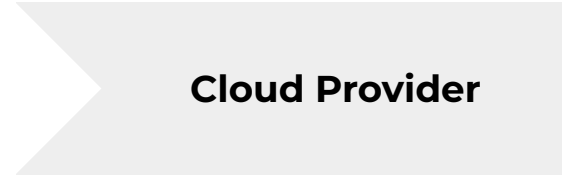
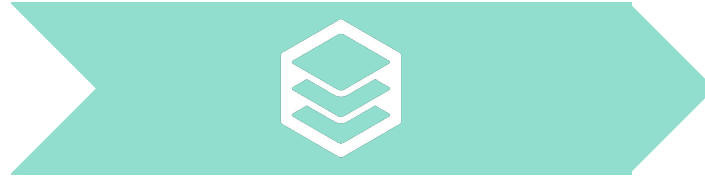
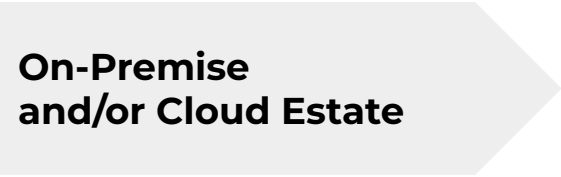


I want to migrate  
my IT to the Cloud!





I want to migrate my IT to the Cloud!





**On-Premise  
and/or Cloud Estate**



**Cloud Provider**



SQL



Excel



Cloud API



Surveys







Google Cloud

**On-Premise  
and/or Cloud Estate**



**Cloud Provider**

-   
SQL
-   
Excel
-   
Cloud API
-   
Surveys





**Import** →

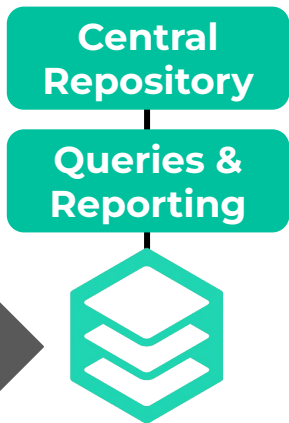


**On-Premise  
and/or Cloud Estate**



**Cloud Provider**





-   
SQL
-   
Excel
-   
Cloud API
-   
Surveys



**On-Premise  
and/or Cloud Estate**

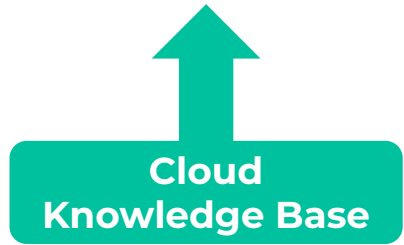


**Cloud Provider**

-   
SQL
-   
Excel
-   
Cloud API
-   
Surveys



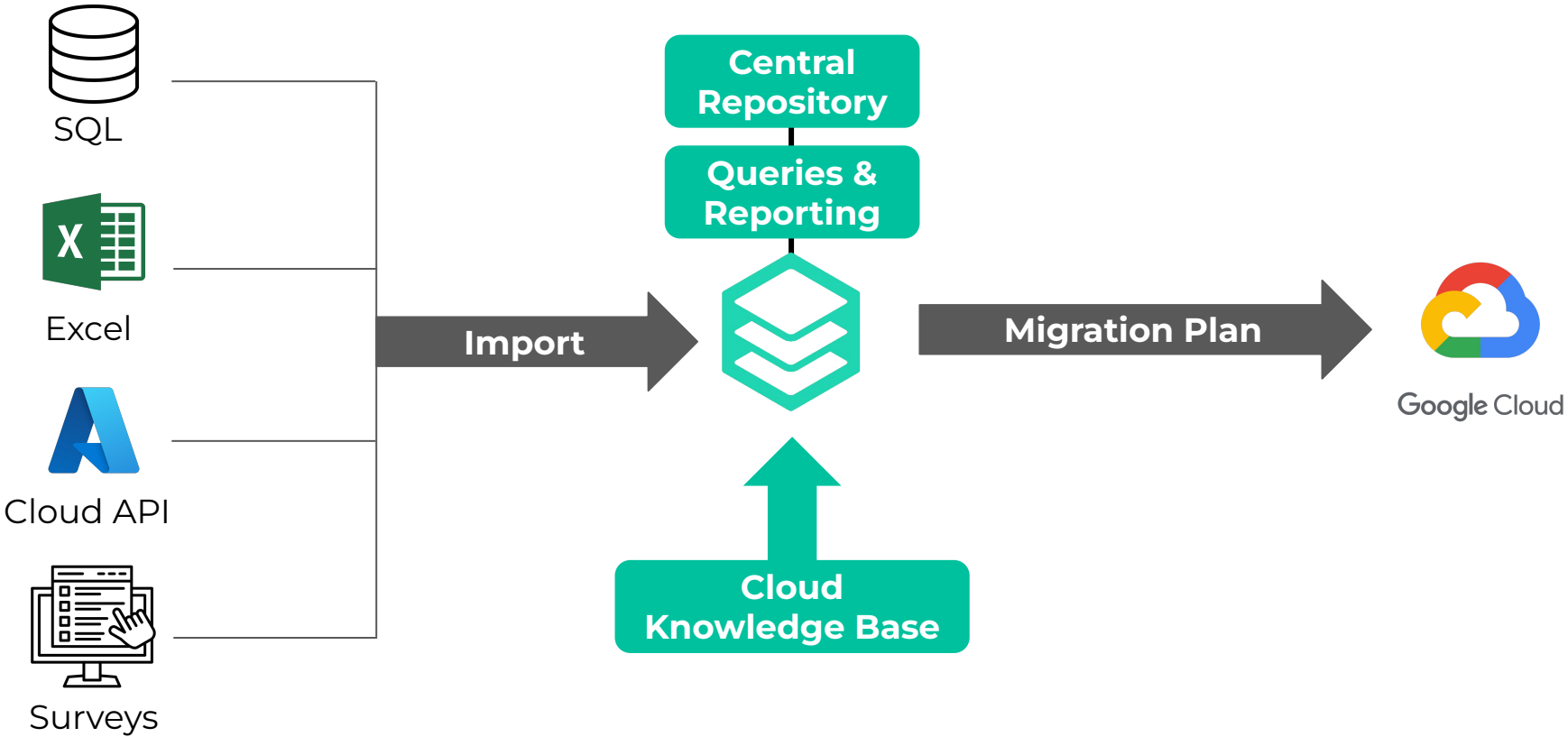
- Central  
Repository**
- Queries &  
Reporting**

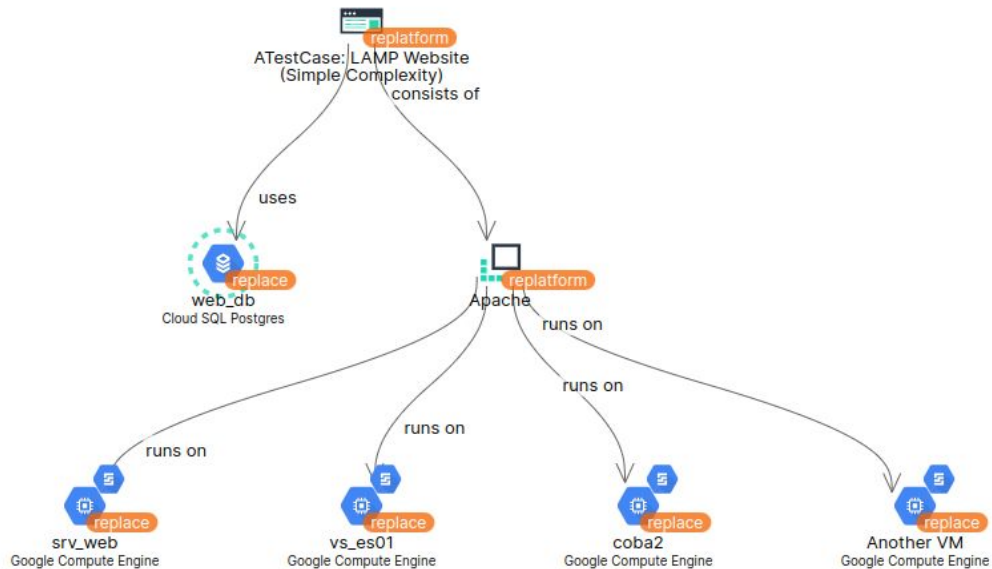


**On-Premise  
and/or Cloud Estate**



**Cloud Provider**






**web\_db**

Action: **Replace** ▾

Migration Strategy: **Replatform** ▾

\$112.18 / mo

**Cloud SQL Postgres**  

Managed Postgres instances

**PaaS**

**db-n1-standard-2**  

**OnDemand** ▾

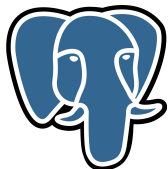
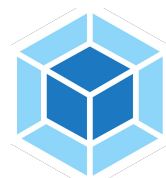
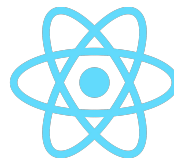
2 vCPU 7.5 GIB RAM Single-AZ

▾ Pricing details

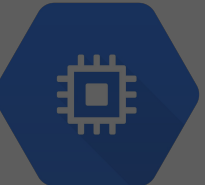
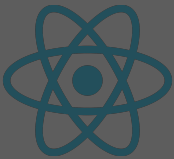
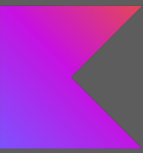
 Google: EU (Finland)

[+ Add Instance](#)

# Our Tech Stack



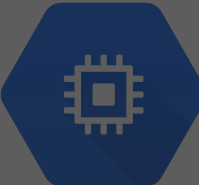
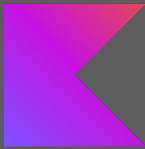
# Our Tech Stack





# Our Tech Stack

Since 2018!



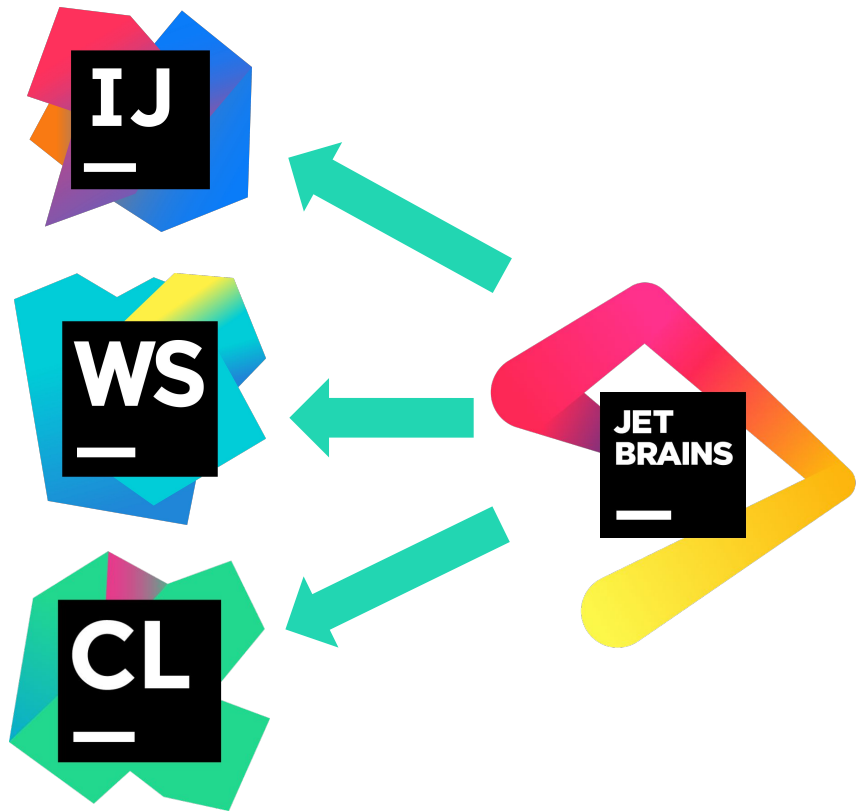
# Let's get started!

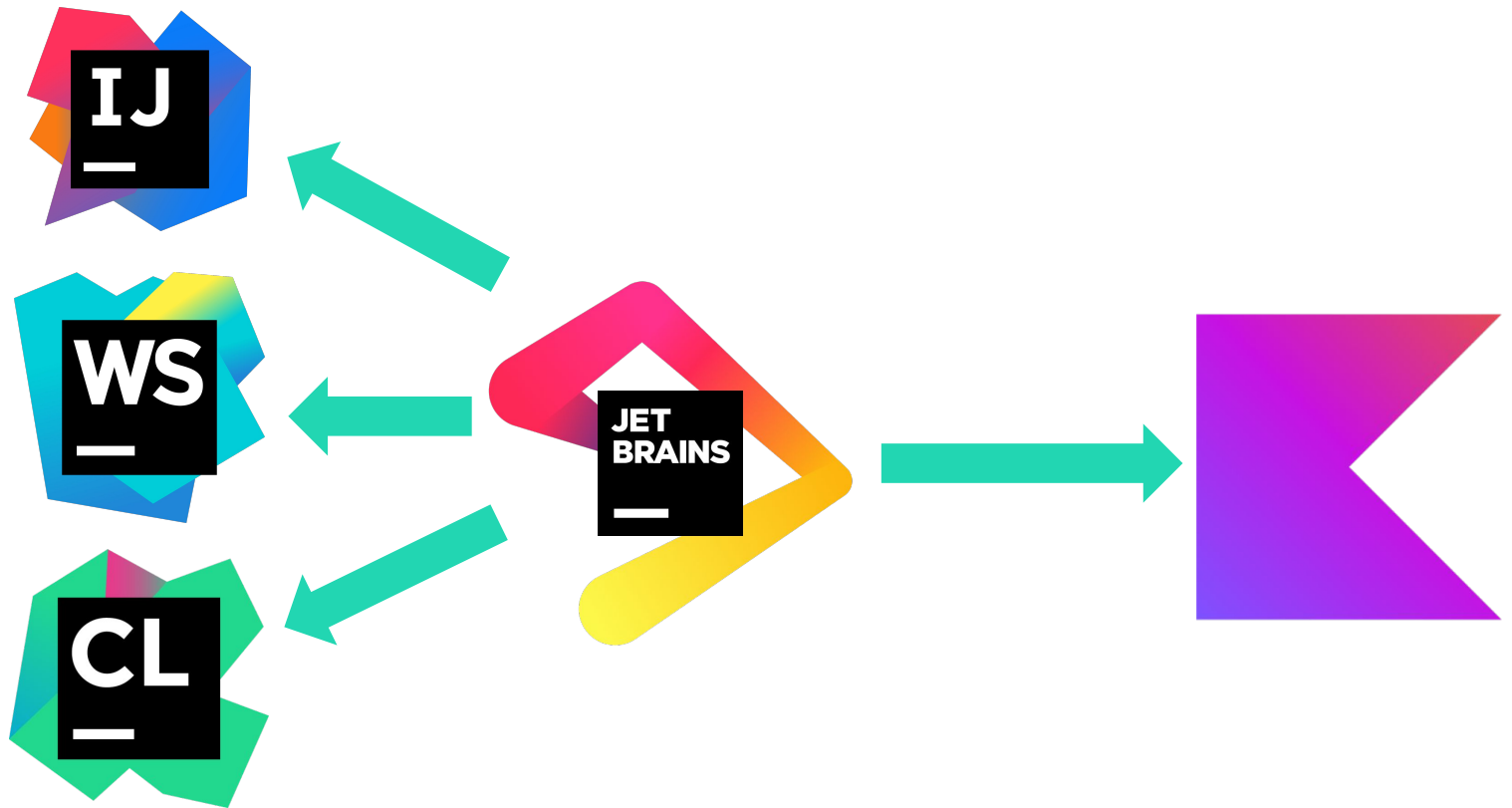




**Kotlin**

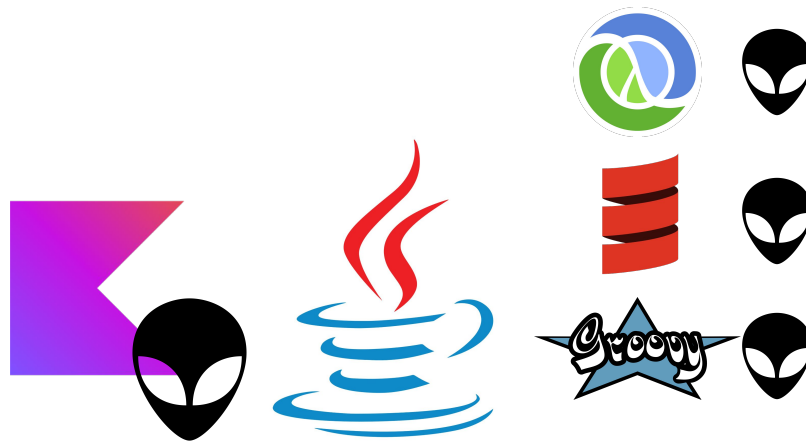








1.0 in 2016

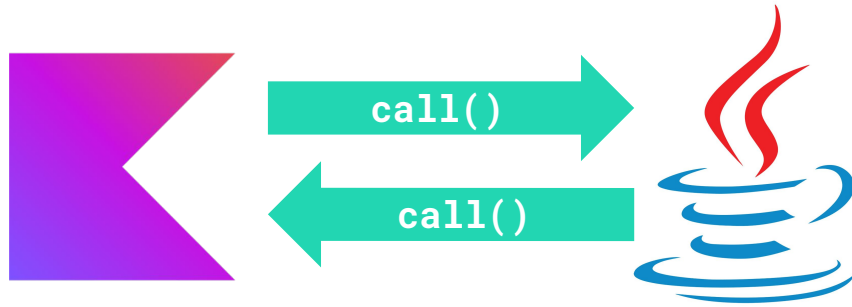


Java Virtual Machine



**JVM Alien**

Runs on the JVM but isn't Java



**Very easy to get started with Kotlin on existing Java Codebase!**

IntelliJ additionally provides (best-effort) automatic conversion of Java files to Kotlin.





**2017: Android app development introduces first-class support for Kotlin**



2019



**2019: Kotlin becomes standard language for Android app development**



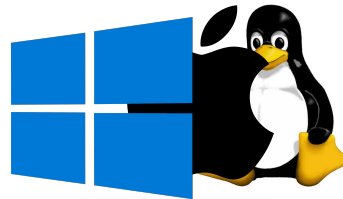
Server



Android



iOS



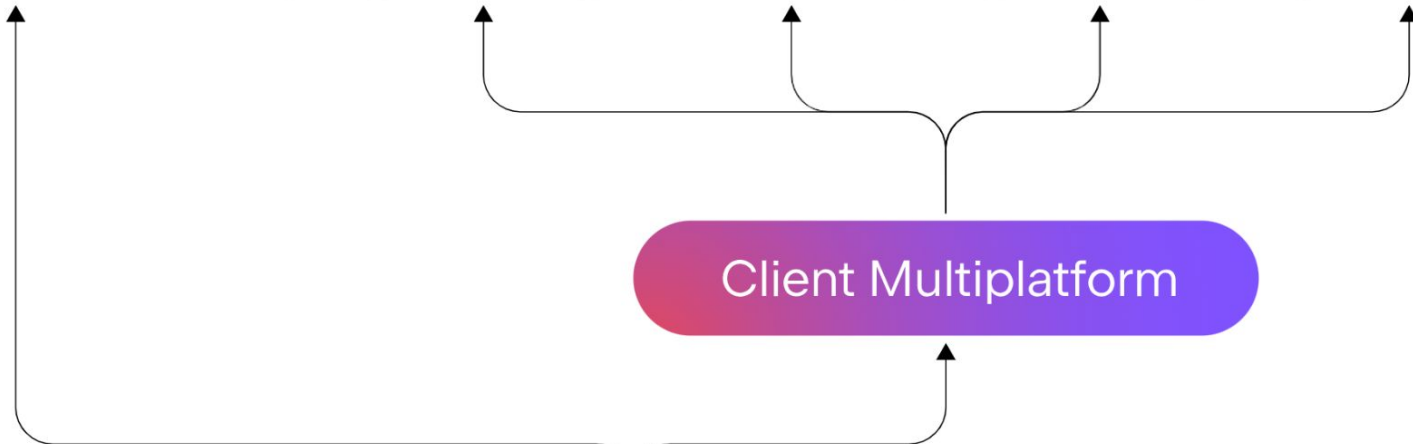
Desktop



Web

Client Multiplatform

Common Multiplatform





2023

**2023: Kotlin becomes standard language for Gradle build scripts**

Compiled

Strongly Typed

Multiplatform



# Kotlin

Garbage Collected

Concurrent

Object Oriented

Functional



# Variables



```
// declare a variable  
int myVar;
```

```
// declare a variable  
var myVar: Int
```



# Variables



```
// declare a variable
```

```
int myVar;
```

```
// declare a readonly variable
```

```
final String name;
```

```
// declare a variable
```

```
var myVar: Int
```

```
// declare a readonly variable
```

```
val name: String
```



# Variables



```
// declare a variable
int myVar;

// declare a readonly variable
final String name;

// declare & assign variable
final String greeting = "Hello World";
```

```
// declare a variable
var myVar: Int

// declare a readonly variable
val name: String

// declare & assign variable
val greeting = "Hello World!"
```





# Variables



```
// declare a variable
int myVar;

// declare a readonly variable
final String name;

// declare & assign variable
final String greeting = "Hello World";

// declare & assign variable
var greeting = "Hello World!"
```

```
// declare a variable
var myVar: Int

// declare a readonly variable
val name: String

// declare & assign variable
val greeting = "Hello World!"

// you can add an explicit type too
val greeting: String = "Hello World!"
```



# Operators



```
// basic operator usage  
int c = 3 + 4;
```

```
// basic operator usage  
val c = 3 + 4
```



# Operators



```
// basic operator usage
int c = 3 + 4;

// equality check
boolean x = "abc".equals("abc");
```

```
// basic operator usage
val c = 3 + 4

// equality check
val x = "abc" == "abc"
```



# Operators



```
// basic operator usage
int c = 3 + 4;

// equality check
boolean x = "abc".equals("abc");

// null-safe equality check
boolean y = Objects.equals(null, "hi!");
```

```
// basic operator usage
val c = 3 + 4

// equality check
val x = "abc" == "abc"

// null-safe equality check
val y = null == "hi!"
```



# Operators



```
// basic operator usage
int c = 3 + 4;

// equality check
boolean x = "abc".equals("abc");

// null-safe equality check
boolean y = Objects.equals(null, "hi!");

// reference identity check
boolean i = me == you;
```

```
// basic operator usage
val c = 3 + 4

// equality check
val x = "abc" == "abc"

// null-safe equality check
val y = null == "hi!"

// reference identity check
val i = me === you
```



# Operators



```
// property access  
student.getName();
```

```
// property access  
student.name
```



# Operators



```
// property access  
student.getName();
```

```
// null-safe property access  
student == null  
    ? null  
    : student.getName();
```

```
// property access  
student.name
```

```
// null-safe property access  
student?.name
```



# Operators



```
// property access  
student.getName();
```

```
// null-safe property access  
student == null  
    ? null  
    : student.getName();
```

```
// default-on-null  
mood == null  
    ? "good"  
    : mood;
```

```
// property access  
student.name
```

```
// null-safe property access  
student?.name
```

```
// default-on-null  
mood ?: "good"
```





# Collections



```
// adding elements  
list.add("apples")
```

```
// adding elements  
list += "apples"
```



# Collections



```
// adding elements  
list.add("apples")  
  
// removing elements  
list.remove("bananas")
```

```
// adding elements  
list += "apples"  
  
// removing elements  
list -= "bananas"
```



# Collections



```
// adding elements
list.add("apples")

// removing elements
list.remove("bananas")

// concatenating lists
List<String> newList = ArrayList<>()
newList.addAll(listA)
newList.addAll(listB)
```

```
// adding elements
list += "apples"

// removing elements
list -= "bananas"

// concatenating lists
val newList = listA + listB
```



# Operators



```
// type check  
something instanceof String
```

```
// type check  
something is String
```



# Operators



```
// type check  
something instanceof String
```

```
// type cast  
(String) something
```

```
// type check  
something is String
```

```
// type cast  
something as String
```



# Operators



```
// type check  
something instanceof String
```

```
// type cast  
(String) something
```

```
// safe cast  
something instanceof String  
? (String) something  
: null
```

```
// type check  
something is String
```

```
// type cast  
something as String
```

```
// safe cast  
something as? String
```



# Flow Typing



```
String studentId = null;
if(person instanceof Student){
    // cast is required here!
    studentId = ((Student)person).getStudentId();
}
```

```
val studentId = if(person is Student){
    // compiler knows that "person" must be
    // of type Student in here!
    person.studentId
} else {
    null
}
```



# Flow Typing



```
String studentId = null;
if(person instanceof Student){
    // cast is required here!
    studentId = ((Student)person).getStudentId();
}

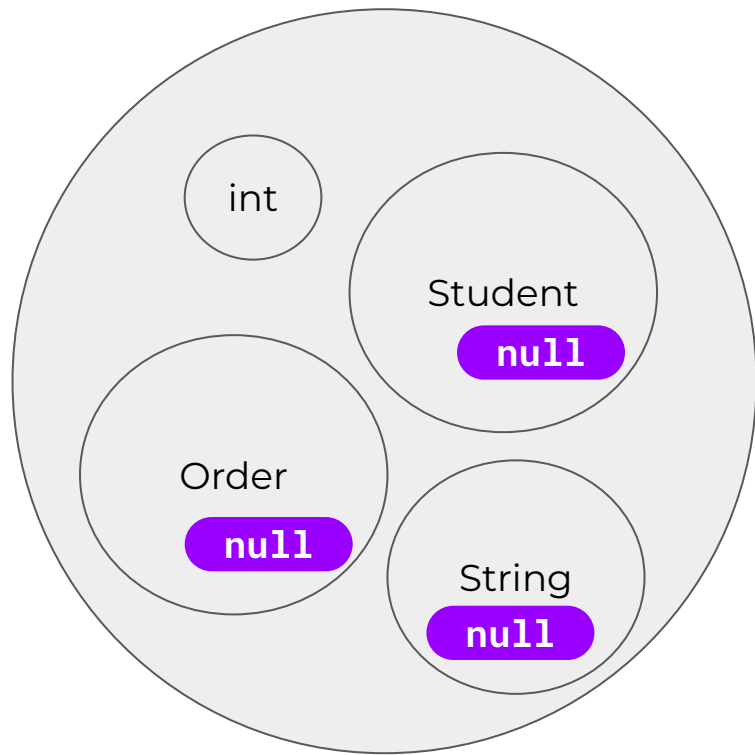
// since Java 11+
String studentId = null;
if(person instanceof Student s){
    studentId = s.getStudentId();
}
```

```
val studentId = if(person is Student){
    // compiler knows that "person" must be
    // of type Student in here!
    person.studentId
} else {
    null
}

// or, shorthand:
val studentId = (person as? Student)?.studentId
```

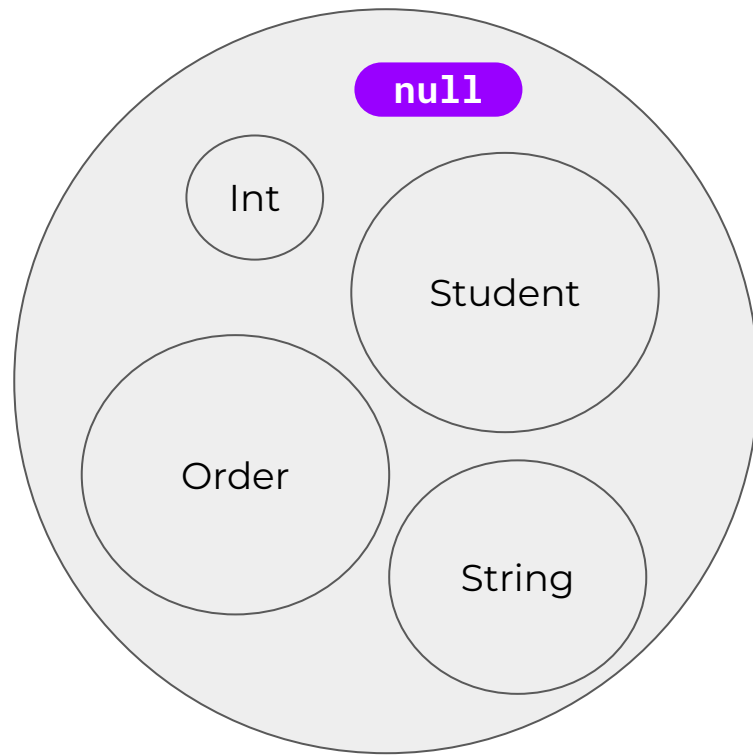


## Java's Type System



Every non-primitive type can be null!

## Kotlin's Type System



Null is a separate type!



```
String x = null;  
int length = x.length();
```



```
String x = null;  
int length = x.length();
```

At compile time:



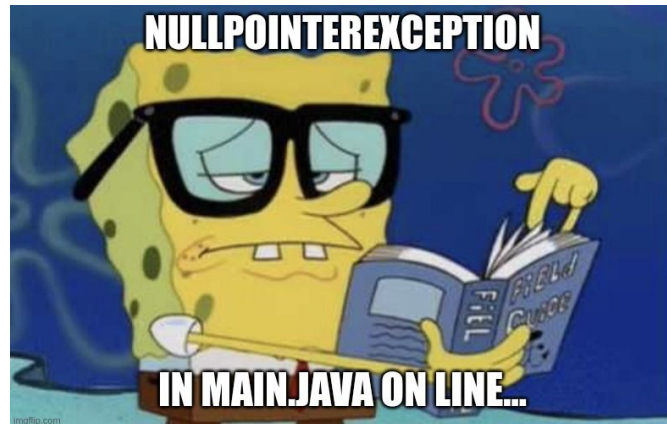


```
String x = null;  
int length = x.length();
```

At compile time:



At runtime:





```
String x = null;  
int length = x.length();
```

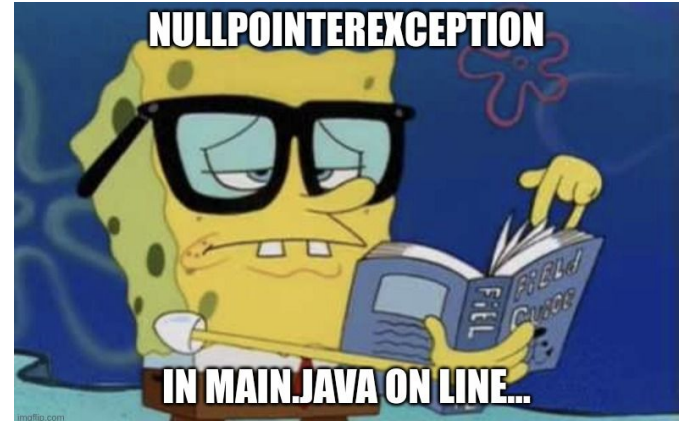
**ARE YOU KIDDING ME**



At compile time:



At runtime:





```
val x: String = null
```



```
val x: String = null
```

At compile time:





```
val x: String = null  
^^^^^^^^^^^^^^^^
```

Compile error: cannot  
assign null to non-nullable  
type String.

At compile time:







At compile time:

```
val x: String? = null  
val length = x.length
```



```
val x: String? = null  
val length = x.length
```

At compile time:





```
val x: String? = null  
val length = x.length
```

^^^^^^

Compile error: cannot  
invoke method length on  
nullable type String?

At compile time:





```
val x: String? = null  
val length = x?.length ?: 0
```

At compile time:



At runtime:



*A program written purely in Kotlin can **never** throw a `NullPointerException` because **the compiler will not permit it.***



# Meanwhile at Oracle...



- Installing
- Contributing
- Sponsoring
- Developers' Guide
- Vulnerabilities
- JDK GA/EA Builds
- Mailing lists
- Wiki - IRC
- Bylaws - Census
- Legal
- Workshop**
- JEP Process**
- Source code**
- Mercurial
- GitHub
- Tools**
- Git
- jtreg harness
- Groups**
- (overview)
- Adoption
- Build
- Client Libraries
- Compatibility & Specification
- Review
- Compiler
- Conformance
- Core Libraries
- Governing Board
- HotSpot
- IDE Tooling & Support
- Internationalization

## JEP 358: Helpful NullPointerExceptions

*Authors* Goetz Lindenmaier, Ralf Schmelter  
*Owner* Goetz Lindenmaier  
*Type* Feature  
*Scope* JDK  
*Status* Closed / Delivered  
*Release* 14  
*Component* hotspot / runtime  
*Discussion* hotspot dash runtime dash dev at openjdk dot java dot net, core dash libs dash dev at openjdk dot java dot net  
*Effort* S  
*Duration* S  
*Reviewed* Alex Buckley, Coleen Phillimore  
*by*  
*Endorsed* Mikael Vidstedt  
*by*  
*Created* 2019/03/15 10:27  
*Updated* 2021/12/22 14:02  
*Issue* 8220715

### Summary

Improve the usability of NullPointerExceptions generated by the JVM by describing precisely which variable was null.

# Meanwhile at Oracle...

## OpenJDK

Installing  
Contributing  
Sponsoring  
Developers' Guide  
Vulnerabilities  
JDK GA/EA Builds  
Mailing lists  
Wiki - IRC  
Bylaws - Census  
Legal  
**Workshop**  
**JEP Process**  
**Source code**  
Mercurial  
GitHub  
**Tools**  
Git  
jreg harness  
**Groups**  
(overview)  
Adoption  
Build  
Client Libraries  
Compatibility &  
Specification  
Review  
Compiler  
Conformance  
Core Libraries  
Governing Board  
HotSpot  
IDE Tooling & Support  
Internationalization

### JEP 358: Helpful NullPointerExceptions

*Authors* Goetz Lindenmaier, Ralf Schmelter  
*Owner* Goetz Lindenmaier  
*Type* Feature  
*Scope* JDK  
*Status* Closed / Delivered  
*Release* 14  
*Component* hotspot / runtime  
*Discussion* hotspot dash runtime dash dev at openjdk dot java dot net, core dash libs dash dev at openjdk dot java dot net  
*Effort* S  
*Duration* S  
*Reviewed* Alex Buckley, Coleen Phillimore  
*by*  
*Endorsed* Mikael Vidstedt  
*by*  
*Created* 2019/03/15 10:27  
*Updated* 2021/12/22 14:02  
*Issue* 8220715

#### Summary

Improve the usability of NullPointerExceptions generated by the JVM by describing precisely which variable was null.



# Meanwhile at Oracle...

## OpenJDK

- Installing
- Contributing
- Sponsoring
- Developers' Guide
- Vulnerabilities
- JDK GA/EA Builds
- Mailing lists
- Wiki - IRC
- Bylaws - Census
- Legal
- Workshop**
- JEP Process**
- Source code**
- Mercurial
- GitHub
- Tools**
- Git
- jtreg harness
- Groups**
- (overview)
- Adoption
- Build
- Client Libraries
- Compatibility & Specification
- Review
- Compiler
- Conformance
- Core Libraries
- Governing Board
- HotSpot
- IDE Tooling & Support
- Internationalization

## JEP 358: Helpful NullPointerExceptions

*Authors* Goetz Lindenmaier, Ralf Schmelter  
*Owner* Goetz Lindenmaier  
*Type* Feature  
*Scope* JDK  
*Status* Closed / Delivered  
*Release* 14  
*Component* hotspot / runtime  
*Discussion* hotspot dash runtime dash dev at openjdk dot java dot net, core dash libs dash dev at openjdk dot java dot net  
*Effort* S  
*Duration* S  
*Reviewed by* Alex Buckley, Coleen Phillimore  
*Endorsed by* Mikael Vidstedt  
*Created* 2019/03/15 10:27  
*Updated* 2021/12/22 14:02  
*Issue* 8220715

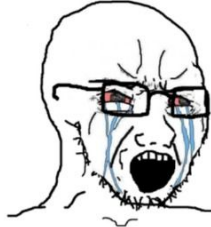
### Summary

Improve the usability of NullPointerExceptions generated describing precisely which variable was null.



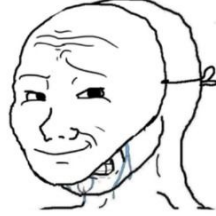


**JAVA DEV**



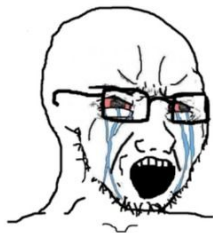
**WHY DO YOU  
THROW NPES AT ME!?**

**ORACLE**



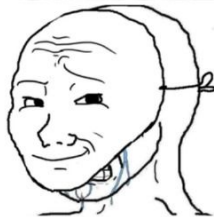
**AT LEAST THEY  
ARE HELPFUL!**

**JAVA DEV**



**WHY DO YOU  
THROW NPES AT ME!?**

**ORACLE**



**AT LEAST THEY  
ARE HELPFUL!**

**KOTLIN DEV 1**



**WHAT'S AN NPE?**

**KOTLIN DEV 2**



**I DON'T KNOW.**



# Data Classes



```
public class Student {  
  
}
```

Step 1: Create the class.



# Data Classes



```
public class Student {  
  
    private String studentId;  
    private String firstName;  
    private String lastName;  
    private Date birthDate;  
  
}
```

Step 2: add the fields



# Data Classes



```
public class Student {  
  
    private String studentId;  
    private String firstName;  
    private String lastName;  
    private Date birthDate;  
  
    public Student(  
        String studentId,  
        String firstName,  
        String lastName,  
        Date birthDate;  
    ) {  
        this.studentId = studentId;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.birthDate = birthDate;  
    }  
}
```

Step 3: add the constructor



# Data Classes



```
public class Student {  
    private String studentId;  
    private String firstName;  
    private String lastName;  
    private Date birthDate;  
  
    public Student(  
        String studentId,  
        String firstName,  
        String lastName,  
        Date birthDate;  
    ) {  
        this.studentId = studentId;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.birthDate = birthDate;  
    }  
  
    public String getStudentId(){  
        return this.studentId;  
    }  
  
    public void setStudentId(String studentId){  
        this.studentId = studentId;  
    }  
  
    public String getFristName(){  
        return this.firstName;  
    }  
  
    public void setFirstName(String firstName){  
        this.firstName = firstName;  
    }  
  
    public String getLastName(){  
        return this.lastName;  
    }  
  
    public void setLastName(String lastName){  
        this.lastName = lastName;  
    }  
  
    public Date getBirthDate(){  
        return this.birthDate;  
    }  
  
    public void setBirthDate(Date birthDate){  
        this.birthDate = birthDate;  
    }  
}
```

Step 4: add the getters and setters



# Data Classes



```
public class Student {  
    private String studentId;  
    private String firstName;  
    private String lastName;  
    private Date birthDate;  
  
    public Student(  
        String studentId,  
        String firstName,  
        String lastName,  
        Date birthDate,  
    ) {  
        this.studentId = studentId;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.birthDate = birthDate;  
    }  
  
    public String getStudentId(){  
        return this.studentId;  
    }  
  
    public void setStudentId(String studentId){  
        this.studentId = studentId;  
    }  
  
    public String getFirstName(){  
        return this.firstName;  
    }  
  
    public void setFirstName(String firstName){  
        this.firstName = firstName;  
    }  
  
    public String getLastName(){  
        return this.lastName;  
    }  
  
    public void setLastName(String lastName){  
        this.lastName = lastName;  
    }  
  
    public Date getBirthDate(){  
        return this.birthDate;  
    }  
  
    public void setBirthDate(Date birthDate){  
        this.birthDate = birthDate;  
    }  
}
```

```
    public int hashCode(){  
        return this.studentId.hashCode() +  
            31 * this.firstName.hashCode() +  
            31 * this.lastName.hashCode() +  
            31 * this.birthDate.hashCode();  
    }  
  
    public boolean equals(Object other){  
        if(other == this){  
            return true;  
        }  
        if(other instanceof Student == false){  
            return false;  
        }  
        Student s = (Student)other;  
        if(!Object.equals(this.studentId, s.studentId)){  
            return false;  
        }  
        if(!Object.equals(this.firstName, s.firstName)){  
            return false;  
        }  
        if(!Object.equals(this.lastName, s.lastName)){  
            return false;  
        }  
        if(!Object.equals(this.birthDate, s.birthDate)){  
            return false;  
        }  
        return true;  
    }  
}
```

Step 5: add hashCode() and equals()



# Data Classes



```
public class Student {
    private String studentId;
    private String firstName;
    private String lastName;
    private Date birthDate;

    public Student(
        String studentId,
        String firstName,
        String lastName,
        Date birthDate) {
        this.studentId = studentId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.birthDate = birthDate;
    }

    public String getStudentId(){
        return this.studentId;
    }

    public void setStudentId(String studentId){
        this.studentId = studentId;
    }

    public String getFristName(){
        return this.firstName;
    }

    public void setFristName(String firstName){
        this.firstName = firstName;
    }

    public String getLastName(){
        return this.lastName;
    }

    public void setLastName(String lastName){
        this.lastName = lastName;
    }

    public Date getBirthDate(){
        return this.birthDate;
    }

    public void setBirthDate(Date birthDate){
        this.birthDate = birthDate;
    }
}
```

```
    public int hashCode(){
        return this.studentId.hashCode() +
            31 * this.firstName.hashCode() +
            31 * this.lastName.hashCode() +
            31 * this.birthDate.hashCode();
    }

    public boolean equals(Object other){
        if(other == this){
            return true;
        }
        if(!other instanceof Student || false){
            return false;
        }
        Student s = (Student)other;
        if(!Object.equals(this.studentId, s.studentId)){
            return false;
        }
        if(!Object.equals(this.firstName, s.firstName)){
            return false;
        }
        if(!Object.equals(this.lastName, s.lastName)){
            return false;
        }
        if(!Object.equals(this.birthDate, s.birthDate)){
            return false;
        }
        return true;
    }

    public String toString(){
        return "Student{id = " + this.studentId +
            ", firstName = " + this.firstName +
            ", lastName = " + this.lastName +
            ", birthDate = " + this.birthDate +
            " }";
    }
}
```

Step 6: add toString()





# Data Classes



```
public class Student {  
    private String studentId;  
    private String firstName;  
    private String lastName;  
    private Date birthDate;  
  
    public Student(  
        String studentId,  
        String firstName,  
        String lastName,  
        Date birthDate,  
    ) {  
        this.studentId = studentId;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.birthDate = birthDate;  
    }  
  
    public String getStudentId(){  
        return this.studentId;  
    }  
  
    public void setStudentId(String studentId){  
        this.studentId = studentId;  
    }  
  
    public String getFirstName(){  
        return this.firstName;  
    }  
  
    public void setFirstName(String firstName){  
        this.firstName = firstName;  
    }  
  
    public String getLastName(){  
        return this.lastName;  
    }  
  
    public void setLastName(String lastName){  
        this.lastName = lastName;  
    }  
  
    public Date getBirthDate(){  
        return this.birthDate;  
    }  
  
    public void setBirthDate(Date birthDate){  
        this.birthDate = birthDate;  
    }  
}
```

```
    public int hashCode(){  
        return this.studentId.hashCode() +  
            31 * this.firstName.hashCode() +  
            31 * this.lastName.hashCode() +  
            31 * this.birthDate.hashCode();  
    }  
  
    public boolean equals(Object other){  
        if(other == this){  
            return true;  
        }  
        if(!other instanceof Student == false){  
            return false;  
        }  
        Student s = (Student)other;  
        if(!Object.equals(this.studentId, s.studentId)){  
            return false;  
        }  
        if(!Object.equals(this.firstName, s.firstName)){  
            return false;  
        }  
        if(!Object.equals(this.lastName, s.lastName)){  
            return false;  
        }  
        if(!Object.equals(this.birthDate, s.birthDate)){  
            return false;  
        }  
        return true;  
    }  
  
    public String toString(){  
        return "Student{id = " + this.studentId +  
            ", firstName = " + this.firstName +  
            ", lastName = " + this.lastName +  
            ", birthDate = " + this.birthDate +  
            " }";  
    }  
}
```

```
class Student(  
)
```

Step 1: Create the class.



# Data Classes



```
public class Student {
    private String studentId;
    private String firstName;
    private String lastName;
    private Date birthDate;

    public Student(
        String studentId,
        String firstName,
        String lastName,
        Date birthDate) {
        this.studentId = studentId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.birthDate = birthDate;
    }

    public String getStudentId(){
        return this.studentId;
    }

    public void setStudentId(String studentId){
        this.studentId = studentId;
    }

    public String getFirstName(){
        return this.firstName;
    }

    public void setFirstName(String firstName){
        this.firstName = firstName;
    }

    public String getLastName(){
        return this.lastName;
    }

    public void setLastName(String lastName){
        this.lastName = lastName;
    }

    public Date getBirthDate(){
        return this.birthDate;
    }

    public void setBirthDate(Date birthDate){
        this.birthDate = birthDate;
    }
}
```

```
public int hashCode(){
    return this.studentId.hashCode() +
        31 * this.firstName.hashCode() +
        31 * this.lastName.hashCode() +
        31 * this.birthDate.hashCode();
}

public boolean equals(Object other){
    if(other == this){
        return true;
    }
    if(other instanceof Student == false){
        return false;
    }
    Student s = (Student)other;
    if(!Object.equals(this.studentId, s.studentId)){
        return false;
    }
    if(!Object.equals(this.firstName, s.firstName)){
        return false;
    }
    if(!Object.equals(this.lastName, s.lastName)){
        return false;
    }
    if(!Object.equals(this.birthDate, s.birthDate)){
        return false;
    }
    return true;
}

public String toString(){
    return "Student{id = " + this.studentId +
        ", firstName = " + this.firstName +
        ", lastName = " + this.lastName +
        ", birthDate = " + this.birthDate +
        "}";
}
```

```
class Student(
    var studentId: String,
    var firstName: String,
    var lastName: String,
    var birthDate: Date,
)
```

Step 2: Add the fields



# Data Classes



```
public class Student {  
    private String studentId;  
    private String firstName;  
    private String lastName;  
    private Date birthDate;  
  
    public Student(  
        String studentId,  
        String firstName,  
        String lastName,  
        Date birthDate)  
    {  
        this.studentId = studentId;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.birthDate = birthDate;  
    }  
  
    public String getStudentId(){  
        return this.studentId;  
    }  
  
    public void setStudentId(String studentId){  
        this.studentId = studentId;  
    }  
  
    public String getFirstName(){  
        return this.firstName;  
    }  
  
    public void setFirstName(String firstName){  
        this.firstName = firstName;  
    }  
  
    public String getLastName(){  
        return this.lastName;  
    }  
  
    public void setLastName(String lastName){  
        this.lastName = lastName;  
    }  
  
    public Date getBirthDate(){  
        return this.birthDate;  
    }  
  
    public void setBirthDate(Date birthDate){  
        this.birthDate = birthDate;  
    }  
}
```

```
    public int hashCode(){  
        return this.studentId.hashCode() +  
            31 * this.firstName.hashCode() +  
            31 * this.lastName.hashCode() +  
            31 * this.birthDate.hashCode();  
    }  
  
    public boolean equals(Object other){  
        if(other == this){  
            return true;  
        }  
        if(!other instanceof Student == false){  
            return false;  
        }  
        Student s = (Student)other;  
        if(!Object.equals(this.studentId, s.studentId)){  
            return false;  
        }  
        if(!Object.equals(this.firstName, s.firstName)){  
            return false;  
        }  
        if(!Object.equals(this.lastName, s.lastName)){  
            return false;  
        }  
        if(!Object.equals(this.birthDate, s.birthDate)){  
            return false;  
        }  
        return true;  
    }  
  
    public String toString(){  
        return "Student{id = " + this.studentId +  
            ", firstName = " + this.firstName +  
            ", lastName = " + this.lastName +  
            ", birthDate = " + this.birthDate +  
            " }";  
    }  
}
```

```
class Student(  
    var studentId: String,  
    var firstName: String,  
    var lastName: String,  
    var birthDate: Date,  
)
```

Step 3: Add the constructor



# Data Classes



```
public class Student {  
    private String studentId;  
    private String firstName;  
    private String lastName;  
    private Date birthDate;  
  
    public Student(  
        String studentId,  
        String firstName,  
        String lastName,  
        Date birthDate)  
    {  
        this.studentId = studentId;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.birthDate = birthDate;  
    }  
  
    public String getStudentId(){  
        return this.studentId;  
    }  
  
    public void setStudentId(String studentId){  
        this.studentId = studentId;  
    }  
  
    public String getFirstName(){  
        return this.firstName;  
    }  
  
    public void setFirstName(String firstName){  
        this.firstName = firstName;  
    }  
  
    public String getLastName(){  
        return this.lastName;  
    }  
  
    public void setLastName(String lastName){  
        this.lastName = lastName;  
    }  
  
    public Date getBirthDate(){  
        return this.birthDate;  
    }  
  
    public void setBirthDate(Date birthDate){  
        this.birthDate = birthDate;  
    }  
}
```

```
    public int hashCode(){  
        return this.studentId.hashCode() +  
            31 * this.firstName.hashCode() +  
            31 * this.lastName.hashCode() +  
            31 * this.birthDate.hashCode();  
    }  
  
    public boolean equals(Object other){  
        if(other == this){  
            return true;  
        }  
        if(!other instanceof Student == false){  
            return false;  
        }  
        Student s = (Student)other;  
        if(!Object.equals(this.studentId, s.studentId)){  
            return false;  
        }  
        if(!Object.equals(this.firstName, s.firstName)){  
            return false;  
        }  
        if(!Object.equals(this.lastName, s.lastName)){  
            return false;  
        }  
        if(!Object.equals(this.birthDate, s.birthDate)){  
            return false;  
        }  
        return true;  
    }  
  
    public String toString(){  
        return "Student{id = " + this.studentId +  
            ", firstName = " + this.firstName +  
            ", lastName = " + this.lastName +  
            ", birthDate = " + this.birthDate +  
            " }";  
    }  
}
```

```
class Student(  
    var studentId: String,  
    var firstName: String,  
    var lastName: String,  
    var birthDate: Date,  
)
```

Step 4: Add the getters & setters



# Data Classes



```
public class Student {  
    private String studentId;  
    private String firstName;  
    private String lastName;  
    private Date birthDate;  
  
    public Student(  
        String studentId,  
        String firstName,  
        String lastName,  
        Date birthDate)  
    {  
        this.studentId = studentId;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.birthDate = birthDate;  
    }  
  
    public String getStudentId(){  
        return this.studentId;  
    }  
  
    public void setStudentId(String studentId){  
        this.studentId = studentId;  
    }  
  
    public String getFirstName(){  
        return this.firstName;  
    }  
  
    public void setFirstName(String firstName){  
        this.firstName = firstName;  
    }  
  
    public String getLastName(){  
        return this.lastName;  
    }  
  
    public void setLastName(String lastName){  
        this.lastName = lastName;  
    }  
  
    public Date getBirthDate(){  
        return this.birthDate;  
    }  
  
    public void setBirthDate(Date birthDate){  
        this.birthDate = birthDate;  
    }  
}
```

```
    public int hashCode(){  
        return this.studentId.hashCode() +  
            31 * this.firstName.hashCode() +  
            31 * this.lastName.hashCode() +  
            31 * this.birthDate.hashCode();  
    }  
  
    public boolean equals(Object other){  
        if(other == this){  
            return true;  
        }  
        if(other instanceof Student == false){  
            return false;  
        }  
        Student s = (Student)other;  
        if(!Object.equals(this.studentId, s.studentId)){  
            return false;  
        }  
        if(!Object.equals(this.firstName, s.firstName)){  
            return false;  
        }  
        if(!Object.equals(this.lastName, s.lastName)){  
            return false;  
        }  
        if(!Object.equals(this.birthDate, s.birthDate)){  
            return false;  
        }  
        return true;  
    }  
  
    public String toString(){  
        return "Student{id = " + this.studentId +  
            ", firstName = " + this.firstName +  
            ", lastName = " + this.lastName +  
            ", birthDate = " + this.birthDate +  
            " }";  
    }  
}
```

```
data class Student(  
    var studentId: String,  
    var firstName: String,  
    var lastName: String,  
    var birthDate: Date,  
)
```

Step 5: Add hashCode() and equals()



# Data Classes



```
public class Student {
    private String studentId;
    private String firstName;
    private String lastName;
    private Date birthDate;

    public Student(
        String studentId,
        String firstName,
        String lastName,
        Date birthDate) {
        this.studentId = studentId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.birthDate = birthDate;
    }

    public String getStudentId(){
        return this.studentId;
    }

    public void setStudentId(String studentId){
        this.studentId = studentId;
    }

    public String getFirstName(){
        return this.firstName;
    }

    public void setFirstName(String firstName){
        this.firstName = firstName;
    }

    public String getLastName(){
        return this.lastName;
    }

    public void setLastName(String lastName){
        this.lastName = lastName;
    }

    public Date getBirthDate(){
        return this.birthDate;
    }

    public void setBirthDate(Date birthDate){
        this.birthDate = birthDate;
    }
}
```

```
    public int hashCode(){
        return this.studentId.hashCode() +
            31 * this.firstName.hashCode() +
            31 * this.lastName.hashCode() +
            31 * this.birthDate.hashCode();
    }

    public boolean equals(Object other) {
        if (other == this) {
            return true;
        }
        if (other instanceof Student == false) {
            return false;
        }
        Student s = (Student) other;
        if (!Object.equals(this.studentId, s.studentId)) {
            return false;
        }
        if (!Object.equals(this.firstName, s.firstName)) {
            return false;
        }
        if (!Object.equals(this.lastName, s.lastName)) {
            return false;
        }
        if (!Object.equals(this.birthDate, s.birthDate)) {
            return false;
        }
        return true;
    }

    public String toString(){
        return "Student{id = " + this.studentId +
            ", firstName = " + this.firstName +
            ", lastName = " + this.lastName +
            ", birthDate = " + this.birthDate +
            "}";
    }
}
```

```
data class Student(
    var studentId: String,
    var firstName: String,
    var lastName: String,
    var birthDate: Date,
)
```

Step 6: Add toString()



# Data Classes



```
public class Student {
    private String studentId;
    private String firstName;
    private String lastName;
    private Date birthDate;

    public Student(
        String studentId,
        String firstName,
        String lastName,
        Date birthDate) {
        this.studentId = studentId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.birthDate = birthDate;
    }

    public String getStudentId(){
        return this.studentId;
    }

    public void setStudentId(String studentId){
        this.studentId = studentId;
    }

    public String getFirstName(){
        return this.firstName;
    }

    public void setFirstName(String firstName){
        this.firstName = firstName;
    }

    public String getLastName(){
        return this.lastName;
    }

    public void setLastName(String lastName){
        this.lastName = lastName;
    }

    public Date getBirthDate(){
        return this.birthDate;
    }

    public void setBirthDate(Date birthDate){
        this.birthDate = birthDate;
    }
}
```

```
    public int hashCode(){
        return this.studentId.hashCode() +
            31 * this.firstName.hashCode() +
            31 * this.lastName.hashCode() +
            31 * this.birthDate.hashCode();
    }

    public boolean equals(Object other) {
        if(other == this){
            return true;
        }
        if(other instanceof Student == false){
            return false;
        }
        Student s = (Student)other;
        if(!Object.equals(this.studentId, s.studentId)){
            return false;
        }
        if(!Object.equals(this.firstName, s.firstName)){
            return false;
        }
        if(!Object.equals(this.lastName, s.lastName)){
            return false;
        }
        if(!Object.equals(this.birthDate, s.birthDate)){
            return false;
        }
        return true;
    }

    public String toString(){
        return "Student{id = " + this.studentId +
            ", firstName = " + this.firstName +
            ", lastName = " + this.lastName +
            ", birthDate = " + this.birthDate +
            " }";
    }
}
```

```
data class Student(
    var studentId: String,
    var firstName: String,
    var lastName: String,
    var birthDate: Date,
)
```

**THAT ONE  
JAVA DEV, ALWAYS**



**BUT WE HAVE LOMBOK!!**





# Data Classes



```
import lombok.Data;

@Data
public class Student {

    private String studentId;
    private String firstName;
    private String lastName;
    private Date birthDate;

}
```

```
data class Student(
    var studentId: String,
    var firstName: String,
    var lastName: String,
    var birthDate: Date,
)
```



# Data Classes



```
import lombok.Data;

@Data
public class Student {

    private String studentId;
    private String firstName;
    private String lastName;
    private Date birthDate;

}
```

Lombok is an **annotation processor**.  
It hooks into the Java build process and generates new bytecode based on the annotations. **Your IDE and tooling needs to support this explicitly!**

```
data class Student(
    var studentId: String,
    var firstName: String,
    var lastName: String,
    var birthDate: Date,
)
```



# Data Classes



```
import lombok.Data;

@Data
public class Student {

    private String studentId;
    private String firstName;
    private String lastName;
    private Date birthDate;

}
```

Lombok is an **annotation processor**. It hooks into the Java build process and generates new bytecode based on the annotations. **Your IDE and tooling needs to support this explicitly!**

```
data class Student(
    var studentId: String,
    var firstName: String,
    var lastName: String,
    var birthDate: Date,
)
```

**These are language features.** Every Kotlin IDE and tool knows and supports them out of the box.



# Data Classes



```
import lombok.Data;  
  
@Data  
public class Student {  
  
    private String studentId;  
    private String firstName;  
    private String lastName;  
    private Date birthDate;  
  
}
```

Lombok is an **annotation processor**. It hooks into the Java build process and generates new bytecode based on the annotations. **Your IDE and tooling needs to support this explicitly!**





# Lambdas



```
// Lambdas before Java 8
btn.addClickListener(new ClickHandler{

    @Override
    public void handleClick(e: ClickEvent){
        System.out.println("Clicked!");
    }

});
```



# Lambdas



```
// Lambdas before Java 8
btn.addClickListener(new ClickHandler{

    @Override
    public void handleClick(e: ClickEvent){
        System.out.println("Clicked!");
    }

});
```

```
// Lambdas after Java 8
btn.addClickListener(e -> {
    System.out.println("Clicked!");
});
```



# Lambdas



```
// Lambdas before Java 8
btn.addClickHandler(new ClickHandler{

    @Override
    public void handleClick(e: ClickEvent){
        System.out.println("Clicked!");
    }

});
```

```
// Lambdas after Java 8
btn.addClickHandler(e -> {
    System.out.println("Clicked!");
});
```

```
// Lambda (long form)
btn.addClickHandler({ e ->
    println("Clicked!")
});
```



# Lambdas



```
// Lambdas before Java 8
btn.addClickHandler(new ClickHandler{

    @Override
    public void handleClick(e: ClickEvent){
        System.out.println("Clicked!");
    }

});
```

```
// Lambdas after Java 8
btn.addClickHandler(e -> {
    System.out.println("Clicked!");
});
```

```
// Lambda (long form)
btn.addClickHandler({ e ->
    println("Clicked!")
})
```

```
// Lambda (implicit argument)
btn.addClickHandler({
    println("Clicked!")
})
```





# Lambdas



```
// Lambdas before Java 8
btn.addClickHandler(new ClickHandler{

    @Override
    public void handleClick(e: ClickEvent){
        System.out.println("Clicked!");
    }

});
```

```
// Lambdas after Java 8
btn.addClickHandler(e -> {
    System.out.println("Clicked!");
});
```

```
// Lambda (long form)
btn.addClickHandler({ e ->
    println("Clicked!")
})
```

```
// Lambda (implicit argument)
btn.addClickHandler({
    println("Clicked!")
})
```

```
// Lambda (no braces)
btn.addClickHandler {
    println("Clicked!")
}
```

# Fun with Kotlin Magic



```
// let's define the "forEach" method!
fun <T> forEach(collection: Collection<T>, action: (T) -> Unit) {
    for(element in collection){
        action(element)
    }
}

// ... and call it!
val collection = listOf("banana", "apples")

forEach(collection, { element -> println(element) } )
```

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> forEach(collection: Collection<T>, action: (T) -> Unit) {  
    for(element in collection){  
        action(element)  
    }  
}  
  
// ... and call it!  
val collection = listOf("banana", "apples")  
  
forEach(collection, { element -> println(element) } )
```

**This works, but it's rather... verbose.**  
Kotliners don't like this. It's the type of  
Kotlin a Java developer would write.

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> forEach(collection: Collection<T>, action: (T) -> Unit) {  
    for(element in collection){  
        action(element)  
    }  
}  
  
// ... and call it!  
val collection = listOf("banana", "apples")  
  
forEach(collection, { element -> println(element) } )
```

**This works, but it's rather... verbose.**  
Kotliners don't like this. It's the type of Kotlin a Java developer would write.

**Let's try to make it better!**

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> forEach(collection: Collection<T>, action: (T) -> Unit) {  
    for(element in collection){  
        action(element)  
    }  
}  
  
// ... and call it!  
val collection = listOf("banana", "apples")  
  
forEach(collection, { element -> println(element) } )
```

**If a lambda has a single argument,  
we can address the argument  
using "it".**

# Fun with Kotlin Magic



```
// let's define the "forEach" method!
fun <T> forEach(collection: Collection<T>, action: (T) -> Unit) {
    for(element in collection){
        action(element)
    }
}

// ... and call it!
val collection = listOf("banana", "apples")

forEach(collection, { println(it) } )
```

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> forEach(collection: Collection<T>, action: (T) -> Unit) {  
    for(element in collection){  
        action(element)  
    }  
}  
  
// ... and call it!  
val collection = listOf("banana", "apples")  
  
forEach(collection, { println(it) } )
```

**If the last argument of a method call is a lambda, we can move it out of the parentheses.**

# Fun with Kotlin Magic



```
// let's define the "forEach" method!
fun <T> forEach(collection: Collection<T>, action: (T) -> Unit) {
    for(element in collection){
        action(element)
    }
}

// ... and call it!
val collection = listOf("banana", "apples")

forEach(collection) { println(it) }
```



# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> forEach(collection: Collection<T>, action: (T) -> Unit) {  
    for(element in collection){  
        action(element)  
    }  
}  
  
// ... and call it!  
val collection = listOf("banana", "apples")  
  
forEach(collection) {  
    println(it)  
}
```

**That's nice, but can we go further?**

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> forEach(collection: Collection<T> action: (T) -> Unit) {  
    for(element in collection){  
        action(element)  
    }  
}
```

This collection is actually “the thing we operate on”. In Kotlin speak, we call this a **“receiver”**.

```
// ... and call it!  
val collection = listOf("banana", "apples")  
  
forEach(collection) {  
    println(it)  
}
```

**That's nice, but can we go further?**

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> forEach(collection: Collection<T> action: (T) -> Unit) {  
    for(element in collection){  
        action(element)  
    }  
}
```

This collection is actually “the thing we operate on”. In Kotlin speak, we call this a **“receiver”**.

```
// ... and call it!  
val collection = listOf("banana", "apples")  
  
forEach(collection) {  
    println(it)  
}
```

Kotlin allows us to demarcate that...

**That's nice, but can we go further?**

# Fun with Kotlin Magic



```
// let's define the "forEach" method!
fun <T> Collection<T>.forEach(action: (T) -> Unit) {
    for(element in this){
        action(element)
    }
}

// ... and call it!
val collection = listOf("banana", "apples")

collection.forEach() {
    println(it)
}
```

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> Collection<T>.forEach(action: (T) -> Unit) {  
    for(element in this) {  
        action(element)  
    }  
}  
  
// ... and call it!  
val collection = listOf("banana", "apples")  
  
collection.forEach() {  
    println(it)  
}
```

The collection is now the **receiver** of the method. So inside the body, **“this”** refers to the collection. This is called an Extension Method and can be defined anywhere, i.e. it doesn't need to be inside the Collection class to work.

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> Collection<T>.forEach(action: (T) -> Unit) {  
    for(element in this) {  
        action(element)  
    }  
}
```

The collection is now the **receiver** of the method. So inside the body, **"this"** refers to the collection. This is called an Extension Method and can be defined anywhere, i.e. it doesn't need to be inside the Collection class to work.

```
// ... and call it!  
val collection = listOf("banana", "apples")
```

```
collection.forEach() {  
    println(it)  
}
```

An extension method can be called **like a regular method** on the receiver! This is great for discoverability via Code Completion. No need to remember the name of the CollectionUtilXY class anymore!

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> Collection<T>.forEach(action: (T) -> Unit) {  
    for(element in this){  
        action(element)  
    }  
}  
  
// ... and call it!  
val collection = listOf("banana", "apples")  
  
collection.forEach() {  
    println(it)  
}
```

Neat! ... but we're not quite done.

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
fun <T> Collection<T>.forEach(action: (T) -> Unit) {  
    for(element in this){  
        action(element)  
    }  
}  
  
// ... and call it!  
val collection = listOf("banana", "apples")  
  
collection.forEach() {  
    println(it)  
}
```

If a method call contains only a single lambda as argument, the parentheses can be **dropped**.

Neat! ... but we're not quite done.



# Fun with Kotlin Magic



```
// let's define the "forEach" method!
fun <T> Collection<T>.forEach(action: (T) -> Unit) {
    for(element in this){
        action(element)
    }
}

// ... and call it!
val collection = listOf("banana", "apples")

collection.forEach {
    println(it)
}
```

# Fun with Kotlin Magic



```
// let's define the "forEach" method!
fun <T> Collection<T>.forEach(action: (T) -> Unit) {
    for(element in this){
        action(element)
    }
}

// ... and call it!
val collection = listOf("banana", "apples")

collection.forEach {
    println(it)
}
```

But wait! We will call the action as a method every time! That's overhead!

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
inline fun <T> Collection<T>.forEach(action: (T) -> Unit) {  
    for(element in this){  
        action(element)  
    }  
}  
  
// ... and call it!  
val collection = listOf("banana", "apples")  
  
collection.forEach {  
    println(it)  
}
```

But wait! We will call the action as a method every time! That's overhead!

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
inline fun <T> Collection<T>.forEach(action: (T) -> Unit) {  
    for(element in this){  
        action(element)  
    }  
}
```

By marking the method as “inline” we tell the kotlin compiler to copy the lambda body in place of the method call. Zero overhead!

```
// ... and call it!  
val collection = listOf("banana", "apples")  
  
collection.forEach {  
    println(it)  
}
```

But wait! We will call the action as a method every time! That's overhead!

# Fun with Kotlin Magic



```
// let's define the "forEach" method!  
inline fun <T> Collection<T>.forEach(action: (T) -> Unit) {  
    for(element in this){  
        action(element)  
    }  
}  
  
// ... and call it!  
val collection = listOf("banana", "apples")  
  
collection.forEach {  
    println(it)  
}
```

Does this look like a **Code Block** to you? That's no coincidence. Kotlin uses this to build Domain Specific Languages (DSLs).

# A basic HTML Builder that fits on a slide



```
class Html {
    var head: Head = Head()

    constructor(builder: Html.() -> Unit) {
        this.builder()
    }

    fun head(builder: Head.() -> Unit) {
        head.builder()
    }
}

class Head(
    var title: String = "",
    var style: String = ""
)
```

```
val htmlDocument = Html {
    head {
        title = "Hello World"
        style = ".h1{ color: red; }"
    }
}
```

# A basic HTML Builder that fits on a slide



```
class Html {  
    var head: Head = Head()  
}
```

```
class Head(  
    var title: String = "",  
    var style: String = ""  
)
```

```
val htmlDocument = Html(  
    Head(  
        title = "Hello World",  
        style = ".h1{ color: red; }"  
    )  
)
```

# A basic HTML Builder that fits on a slide



```
class Html {  
    var head: Head = Head()  
}
```

```
class Head(  
    var title: String = "",  
    var style: String = ""  
)
```

**Named Method  
Call Parameters**

```
val htmlDocument = Html(  
    Head(  
        title = "Hello World",  
        style = ".h1{ color: red; }"  
    )  
)
```





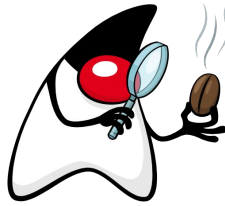
**Did you forget about me?**



ANTLR



Apache



KRYO

JACKSON  
JSON



okio

GUAVA™

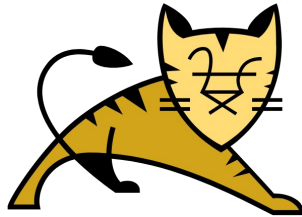


JPA  
Java Persistence API  
HIBERNATE

JUnit 5



Jdbc



Gremlin  
 $G = (V, E)$



ANTLR



Apache



Java has great libraries.  
Tons of them.



okio

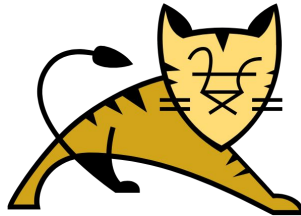


JUnit 5



JPA

Java Persistence API



Gremlin  
 $G = (V, E)$



ANTLR



Apache



Java has great libraries.  
Tons of them.



okio



JPA

Java Persistence API

JUnit 5



They ALL work in Kotlin too!



Gremlin  
 $G = (V, E)$



# Java-Kotlin-Interop



```
public class Student {  
  
    private String name;  
    private List<Course> courses;  
  
    public String getName(){  
        return this.name;  
    }  
  
    public void setName(String name){  
        this.name = name;  
    }  
  
    public List<Course> getCourses(){  
        return this.courses;  
    }  
  
}
```

```
val student = Student()  
student.name = "John"  
student.courses += Course(  
    "Programming in Kotlin"  
)
```



# Java-Kotlin-Interrop



Create objects from Java classes in Kotlin!

```
public class Student {  
    private String name;  
    private List<Course> courses;  
  
    public String getName(){  
        return this.name;  
    }  
  
    public void setName(String name){  
        this.name = name;  
    }  
  
    public List<Course> getCourses(){  
        return this.courses;  
    }  
}
```

```
val student = Student()  
student.name = "John"  
student.courses += Course(  
    "Programming in Kotlin"  
)
```

Call Java  
getters/setters  
with  
Property syntax!



# Java-Kotlin-Interop



```
Course course = new Course("Programming in Java");  
course.getName(); // "Programming in Java"  
course.setLocation("Seminar Room 3");
```

```
@JvmOverloads  
class Course(  
    var name: String,  
    var location: String? = null,  
)
```



# Java-Kotlin-Interop



Create objects from Kotlin classes in Java!

```
Course course = new Course("Programming in Java");  
course.getName(); // "Programming in Java"  
course.setLocation("Seminar Room 3");
```

Call Kotlin properties with  
Java getters/setters syntax!

```
@JvmOverloads  
class Course(  
    var name: String,  
    var location: String? = null,  
)
```



**Full Backend Feature:  
Saving and listing Students with Spring via JSON+REST with SQL persistence**

**Full Backend Feature:  
Saving and listing Students with Spring via JSON+REST with SQL persistence**

**ON A SINGLE SLIDE**



# Java-Kotlin-Interop



```
@Entity
class Student(
    @Id
    val id: UUID,
    @Column
    val name: String,
    @Column
    val email: String,
    @Column
    val birthDate: Date,
)
```



# Java-Kotlin-Interop



```
@Entity
class Student(
    @Id
    val id: UUID,
    @Column
    val name: String,
    @Column
    val email: String,
    @Column
    val birthDate: Date,
)

interface StudentRepo: JpaRepository<Student, UUID>
```



# Java-Kotlin-Interop



```
@Entity
class Student(
    @Id
    val id: UUID,
    @Column
    val name: String,
    @Column
    val email: String,
    @Column
    val birthDate: Date,
)

interface StudentRepo: JpaRepository<Student, UUID>

@Service
class StudentService(
    private val repo: StudentRepo
) {

    @Transactional(readOnly = false)
    fun saveStudent(student: Student): Student {
        return this.repo.save(student)
    }

    @Transactional(readOnly = true)
    fun getAllStudents(): List<Student>{
        return this.repo.findAll()
    }
}
```



# Java-Kotlin-Interop



```
@Entity
class Student(
    @Id
    val id: UUID,
    @Column
    val name: String,
    @Column
    val email: String,
    @Column
    val birthDate: Date,
)

interface StudentRepo: JpaRepository<Student, UUID>

@Service
class StudentService(
    private val repo: StudentRepo
) {

    @Transactional(readOnly = false)
    fun saveStudent(student: Student): Student {
        return this.repo.save(student)
    }

    @Transactional(readOnly = true)
    fun getAllStudents(): List<Student>{
        return this.repo.findAll()
    }
}

@RestController
class StudentController(
    private val service: StudentService
) {

    @GetMapping("/api/students")
    fun getAllStudents(): List<Student> {
        return this.service.getAllStudents()
    }

    @PostMapping("/api/students")
    fun saveStudent(@RequestBody student: Student): Student {
        return this.service.saveStudent(student)
    }
}
```



# Java-Kotlin-Interop



```
@Entity
class Student(
    @Id
    val id: UUID,
    @Column
    val name: String,
    @Column
    val email: String,
    @Column
    val birthDate: Date,
)

interface StudentRepo: JpaRepository<Student, UUID>

@Service
class StudentService(
    private val repo: StudentRepo
) {

    @Transactional(readOnly = false)
    fun saveStudent(student: Student): Student {
        return this.repo.save(student)
    }

    @Transactional(readOnly = true)
    fun getAllStudents(): List<Student>{
        return this.repo.findAll()
    }
}
```

## Used Java libraries:



```
@RestController
class StudentController(
    private val service: StudentService
) {

    @GetMapping("/api/students")
    fun getAllStudents(): List<Student> {
        return this.service.getAllStudents()
    }

    @PostMapping("/api/students")
    fun saveStudent(@RequestBody student: Student): Student {
        return this.service.saveStudent(student)
    }
}
```



# Java-Kotlin-Interop



```
@Entity
class Student(
    @Id
    val id: UUID,
    @Column
    val name: String,
    @Column
    val email: String,
    @Column
    val birthDate: Date,
)
```

```
interface StudentRepo: JpaRepository<Student, UUID>
```

```
@Service
class StudentService(
    private val repo: StudentRepo
) {
    @Transactional(readOnly = false)
    fun saveStudent(student: Student): Student {
        return this.repo.save(student)
    }
    @Transactional(readOnly = true)
    fun getAllStudents(): List<Student>{
        return this.repo.findAll()
    }
}
```

## Used Java libraries:



All in 100% Kotlin.

Yes, this works. Txture is doing this since 2018.

```
@RestController
class StudentController(
    private val service: StudentService
) {
    @GetMapping("/api/students")
    fun getAllStudents(): List<Student> {
        return this.service.getAllStudents()
    }
    @PostMapping("/api/students")
    fun saveStudent(@RequestBody student: Student): Student {
        return this.service.saveStudent(student)
    }
}
```





# Summary



- **The Java Virtual Machine is pretty damn cool**
  - Great platform, tools and libraries
  - Future looks very bright! (Valhalla, Loom, Panama, Lilliput, Babylon ...)
- **Java - the language itself - is still sluggish in 2024**
  - Still extremely verbose in spite of all efforts
  - Not null-safe, and likely never will be because of backwards (in)compatibility
- **Kotlin offers a modern, safe and highly productive alternative**
  - Low cost of entry: Integrates well into existing Java projects
  - Effortless bidirectional interoperability with Java
  - Can use all existing Java libraries (plus new Kotlin libraries!)
  - Null-Safety by default, Extension Methods, DSLs...
  - Powerful type system with flow typing and type inference
- **Get the best of both worlds - run Kotlin code on your JVM today!**