

# Java Programming Spring Boot



Somkiat Puisungnoen

Search

Somkiat | Home

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามช่างนาฏกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata



somkiat.cc

Page Messages Notifications 3 Insights Publishing Tools Settings Help ▾

somkiat.cc  
@somkiat.cc

Home Posts Videos Photos

Liked Following Share ... + Add a Button



**[https://github.com/up1/  
workshop-springboot-2026](https://github.com/up1/workshop-springboot-2026)**



# Topics (1)

- Introduction to Java Programming
- Installation and configuration
- IDE
- Basic of Java Programming
- Mistake from Java programmer
- Write automated tests with JUnit
- SOLID :: How to design testable application



# Topics (2)

- Introduction to Spring Boot
- Create starter project
- Structure of project
- Develop RESTful APIs
- Working with Database with JPA
- Testing with Spring Boot



# Java Programming



# TIOBE Index 2025

Sep 2025	Sep 2024	Change	Programming Language	Ratings	Change
1	1		 Python	25.90%	+5.61%
2	2		 C++	8.80%	-1.94%
3	4		 C	8.65%	-0.24%
4	3		 Java	8.35%	-1.09%
5	5		 C#	6.38%	+0.30%
6	6		 JavaScript	3.22%	-0.70%
7	7		 Visual Basic	2.84%	+0.14%
8	8		 Go	2.32%	-0.03%
9	11		 Delphi/Object Pascal	2.26%	+0.49%
10	27		 Perl	2.03%	+1.33%
11	9		 SQL	1.66%	-0.08%

<https://www.tiobe.com/tiobe-index/>



# Software Requirements

Java Development Kit  
IntelliJ IDEA (community edition)



# Download JDK 17/21/25

## Java 25, Java 21, and earlier versions available now

JDK 25 is the latest *Long-Term Support (LTS)* release of the Java SE Platform.

[Learn about Java](#)

JDK 21 is the previous Long-Term Support (LTS) release of the Java SE Platform.

Earlier JDK versions are available below.

[JDK 25](#)    [JDK 21](#)

### Java SE Development Kit 25 downloads

JDK 25 binaries are free to use in production and free to redistribute, at no cost, under the [Oracle No-Fee Terms and Conditions \(NFTC\)](#).

JDK 25 will receive updates under the NFTC, until September 2028, a year after the release of the next LTS. Subsequent JDK 25 updates will be [available on Oracle Technology Network \(OTN\)](#) and production use beyond the [limited free grants](#) of the OTN license will [require a fee](#).

<https://www.oracle.com/eg/java/technologies/downloads/>



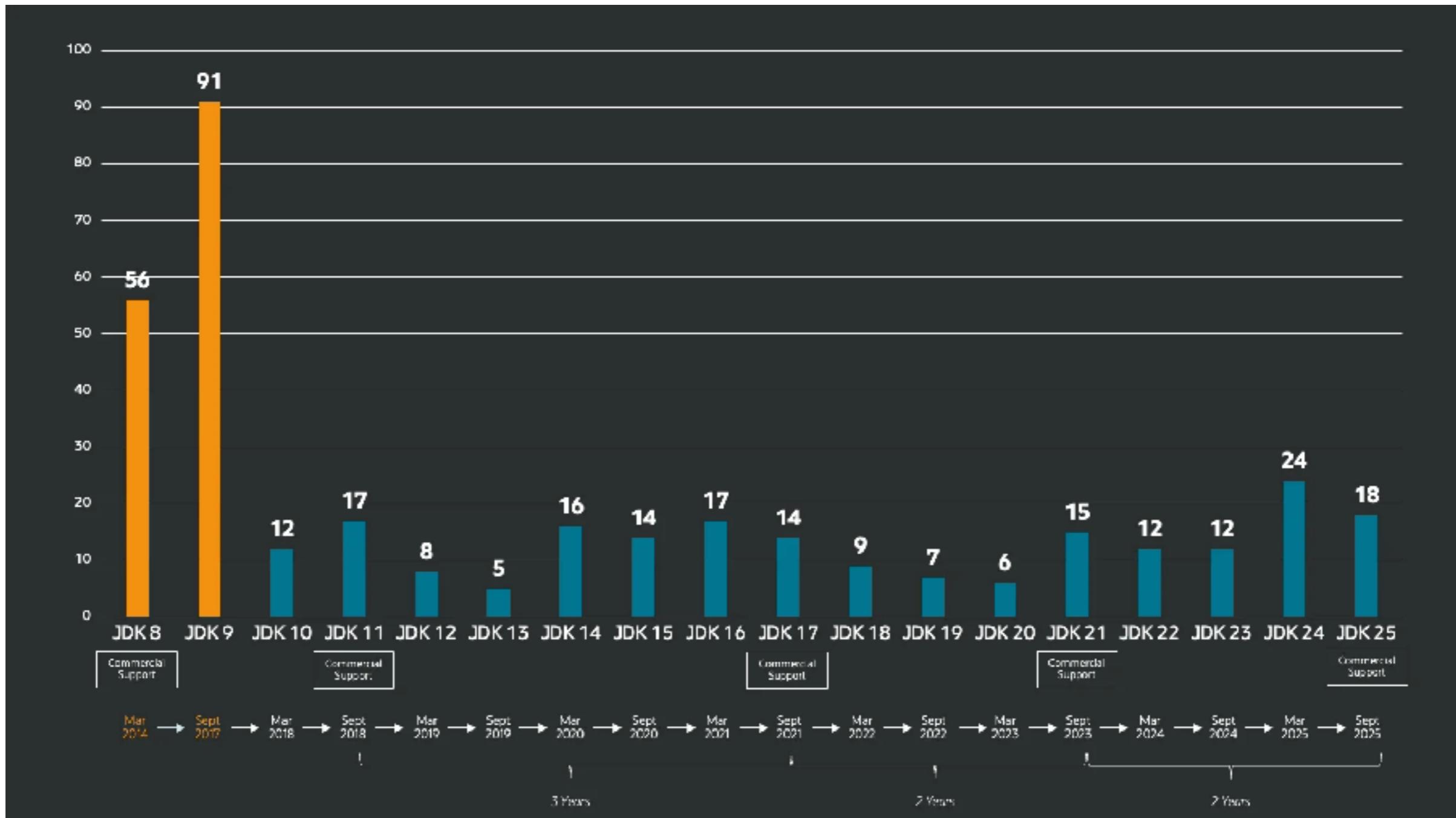
Basic of Java

© 2020 - 2026 Siam Chamnkit Company Limited. All rights reserved.

# Java 25



# History of Java from 8 to 25



<https://www.infoq.com/news/2025/09/java25-released/>



# Config JAVA\_HOME and PATH

```
set JAVA_HOME=<path of JDK>
set PATH=.;%JAVA_HOME%\bin;%PATH%
```



# Testing configuration

```
$javac -version  
$java -version
```



# Download IntelliJ IDEA

We're committed to giving back to our wonderful community, which is why IntelliJ IDEA Community Edition is completely free to use



## IntelliJ IDEA Community Edition

The IDE for pure Java and Kotlin development

Download

.dmg ▾

Free, built on open source

Select an installer for Intel or Apple Silicon

<https://www.jetbrains.com/idea/download/>



# Programming Language



# Programming Language

**Program** is a set of instruction that tell a computer  
We **execute** the instruction lists from program  
Instructions are described using  
**Programming Language**



# Categories of language

High-level Language

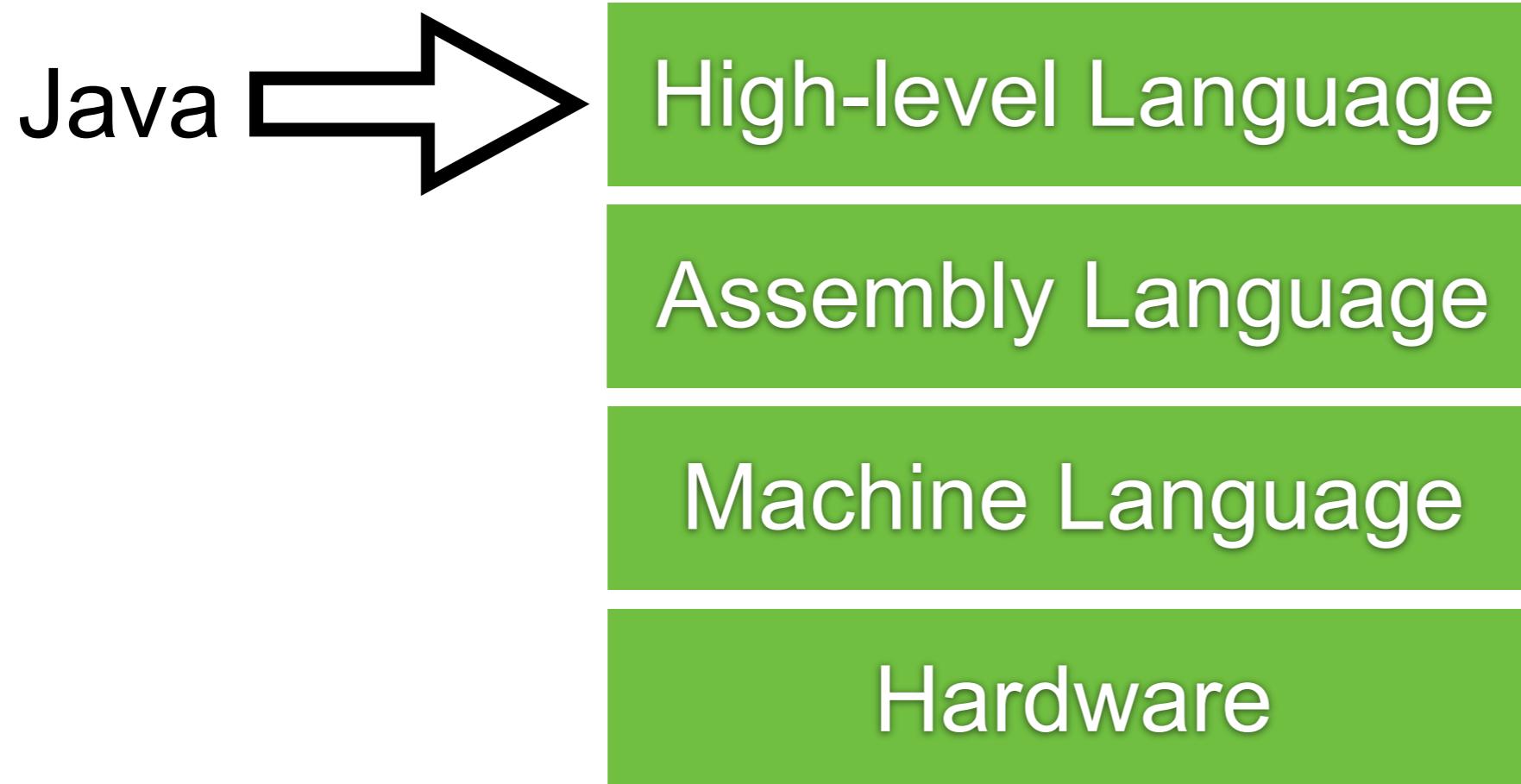
Assembly Language

Machine Language

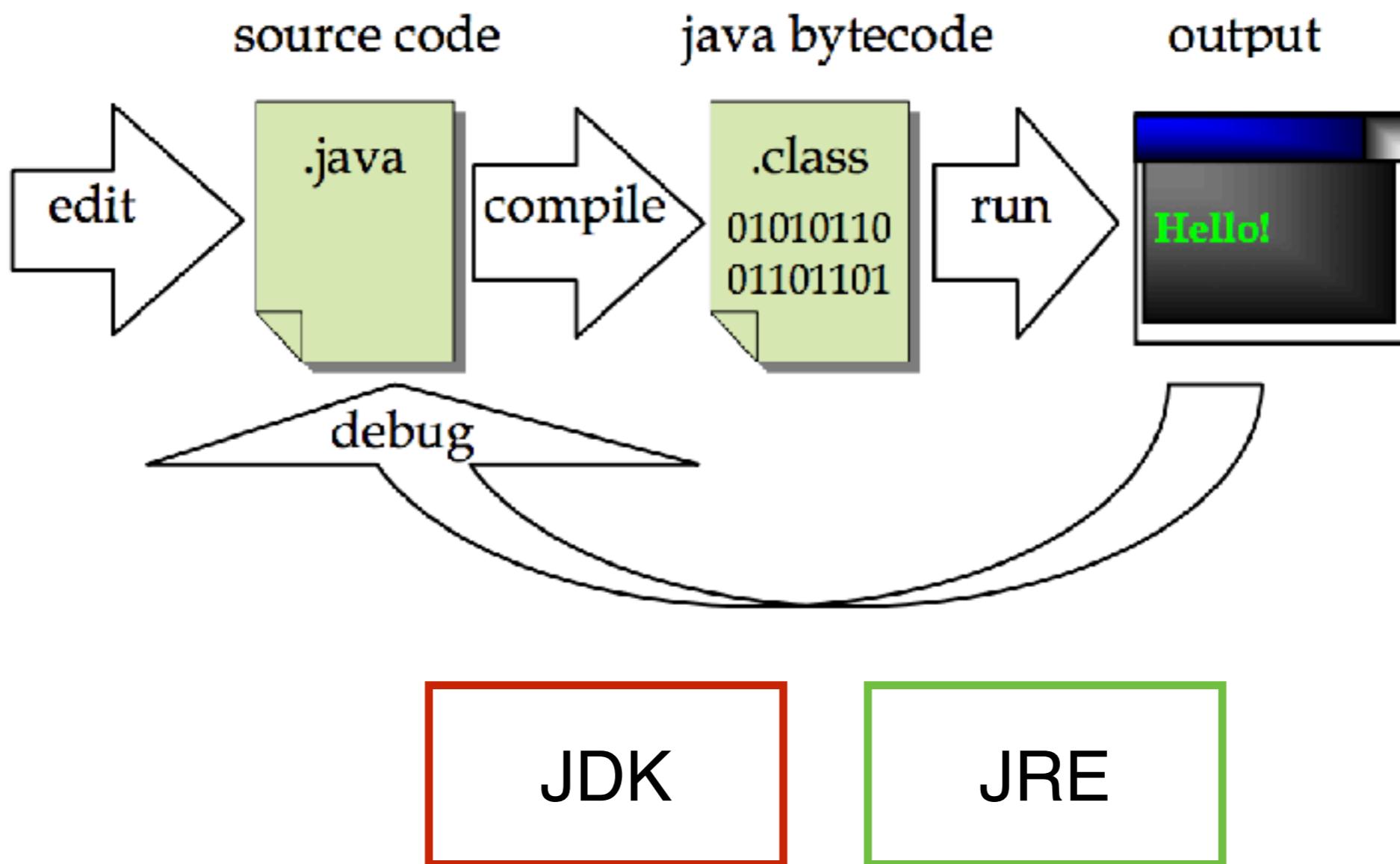
Hardware



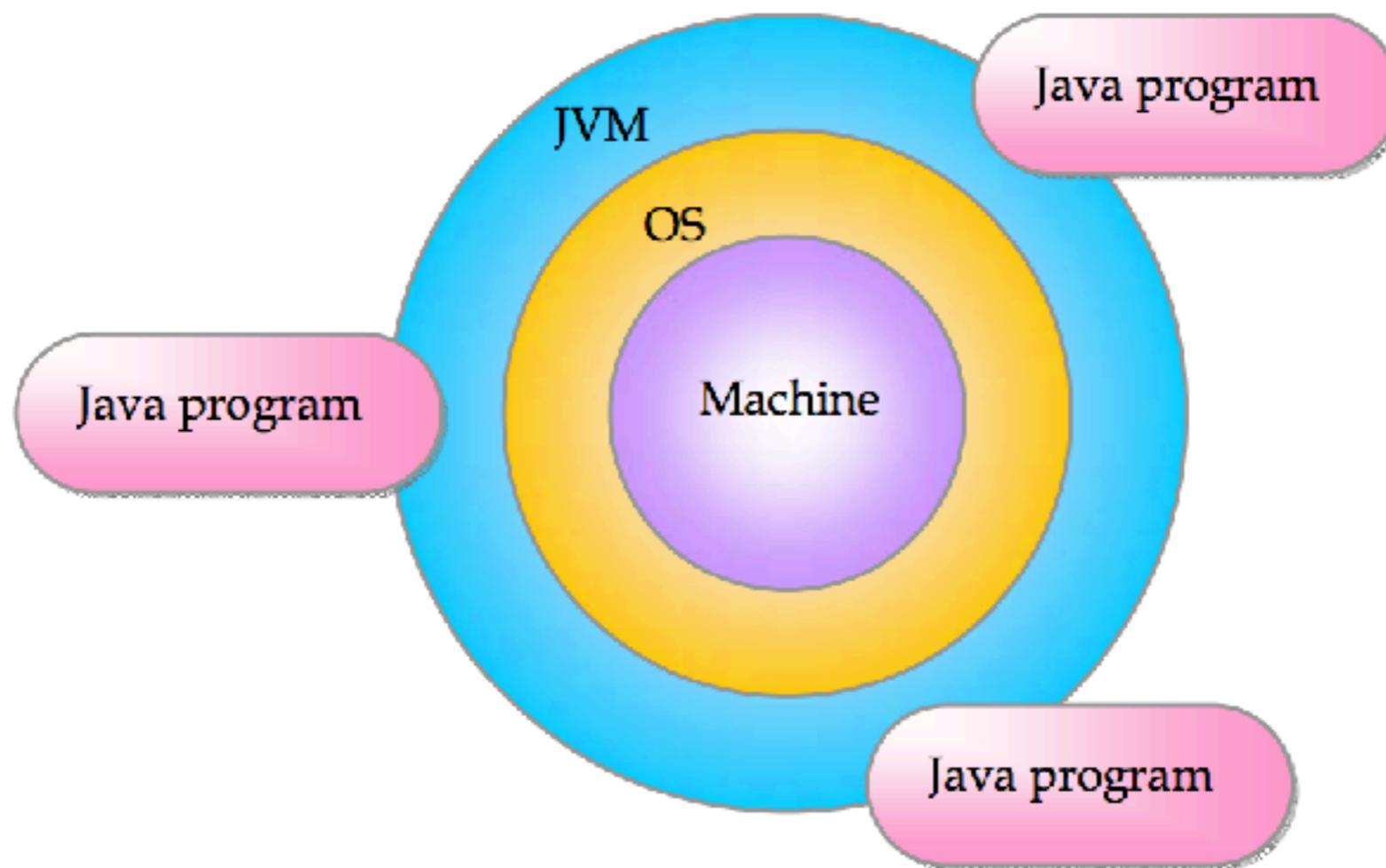
# Categories of language



# Java workflow



# Write once run anywhere ?



# Java in Cloud Native App

Improve performance and resource usages



QUARKUS

GraalVM™



# Introduction to Java



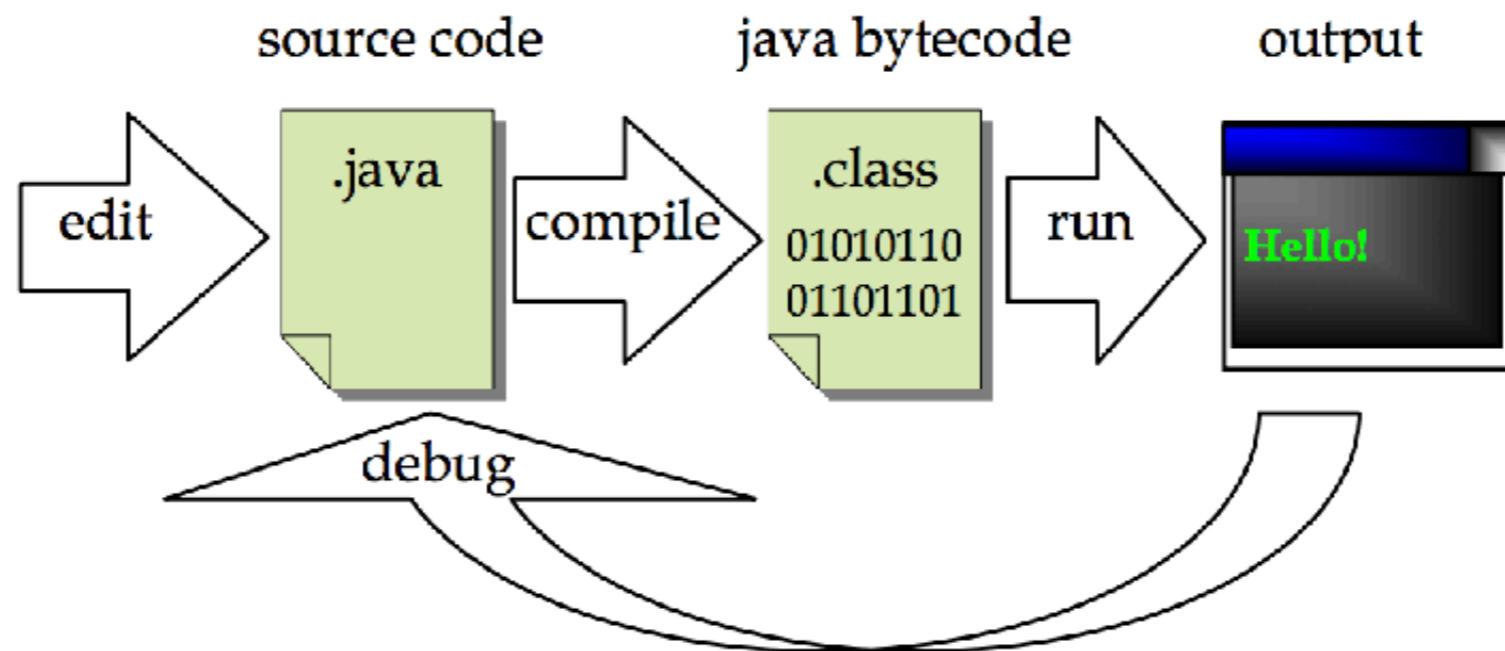
# Structure of Java program

Create file HelloWorld.java

```
1 public class HelloWorld {  
2  
3     // Show message in a console  
4     public static void main(String[] args) {  
5         System.out.println("Hello World");  
6     }  
7 }  
8  
9 }
```



# Compile and Run program



```
$javac HelloWorld.java
```

```
$java HelloWorld
```

*Hello World*



# Keywords

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

[https://docs.oracle.com/javase/tutorial/java/nutsandbolts/\\_keywords.html](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html)



# Comments

Documentation comment

Single line comment (C++ style)

Multi-line comment (C style)



# Comments

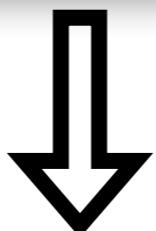
```
1 /**
2  * 1. Documentation comment
3  *
4  * @author somkiat
5 */
6 public class HelloWorld {
7
8     // 2. Single line comment (C++ style)
9 public static void main(String[] args) {
10    /*
11        3. Multi-line comment (C style)
12    */
13    System.out.println("Hello World");
14
15 }
16
17 }
```



# Formatting/ Readability

Whitespace characters in program are ignored during the compilation

```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.println("Hello World");  
4     }  
5 }
```



```
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.println("Hello World");  
4     }  
5 }
```



Any fool can write code that  
a computer can understand.  
Good programmers write  
code that humans can  
understand.

Martin Fowler

---

**Quote**Addicts.com



# Java Code convention

## Oracle (1997)

<https://goo.gl/pfPc1S>

## Google

<https://github.com/google/styleguide>

<https://google.github.io/styleguide/javaguide.html>

## Spring Framework

<https://github.com/spring-projects/spring-framework/wiki/Code-Style>

<https://github.com/47deg/coding-guidelines/tree/master/java/spring>



# Basic of Java



# Basic of Java Part 1

Variables name  
Expressions  
Statements  
Data types  
Operators



# Variables and assign value

Symbolic names of memory location

A variable must have name and data type (static type)

Must be declared before using it

```
int number;  
int counter = 1;  
String name;  
double averageScore = 3.5;
```



# Naming rule for Java identifier

Can't be a Java keyword

Case-sensitive

**Unlimited length** of Unicode letters and digits

Must begin with a letter, underscore(\_) and dollar sign (\$)

**White space** not allowed



# White space ?

Such as spacebar

## isWhitespace

```
public static boolean isWhitespace(char ch)
```

Determines if the specified character is white space according to Java. A character is a Java whitespace if:

- It is a Unicode space character (SPACE\_SEPARATOR, LINE\_SEPARATOR, or PARAGRAPH\_SEPARATOR) ('\u0020' through '\u002F').
- It is '\t', U+0009 HORIZONTAL TABULATION.
- It is '\n', U+000A LINE FEED.
- It is '\u000B', U+000B VERTICAL TABULATION.
- It is '\f', U+000C FORM FEED.
- It is '\r', U+000D CARRIAGE RETURN.
- It is '\u001C', U+001C FILE SEPARATOR.
- It is '\u001D', U+001D GROUP SEPARATOR.
- It is '\u001E', U+001E RECORD SEPARATOR.
- It is '\u001F', U+001F UNIT SEPARATOR.

<https://docs.oracle.com/javase/8/docs/api/java/lang/Character.html#isWhitespace-char->



# Naming convention

Use meaningful name

For compound words use **camelCase**

**Class name** begin with an **Uppercase** letter

**Variable** and **method** name begin with a **Lowercase** letter

For a **constant** use **all uppercase letter** and  
**underscore(\_)** to separate word



# Meaningful name

```
int d = 10;  
int days = 10;  
int daysSinceCreation = 10;  
int daysSinceLastModification = 10;
```



# Expressions

Expression is a value, variable, method or one of their combination that can be evaluated to a value

3.14

a + b

a + c -10

a > 10

Math.sqrt(4)

“Hi, all”

(a + 3 > b) && ( a - 3 < c)



# Statements

Complete sentence that causes some action to occur  
End with semicolon (;

```
int a;  
int b = 10;  
c = a + b;  
boolean isPass = a > b;  
System.out.println("Hello");
```



# Data types (Statically typed)

1. Primitive data types
2. Classes



# 1. Primitive data types

Basic of Java data types have 8 types  
Can't add and change

Value type	Primitive data types
Integer ( -10, 10, 0)	byte, short, int, long
Floating-point (1.0, -1.5, 3.144)	float, double
Character ( 'a', 'ñ')	char
Logical (true, false)	boolean

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

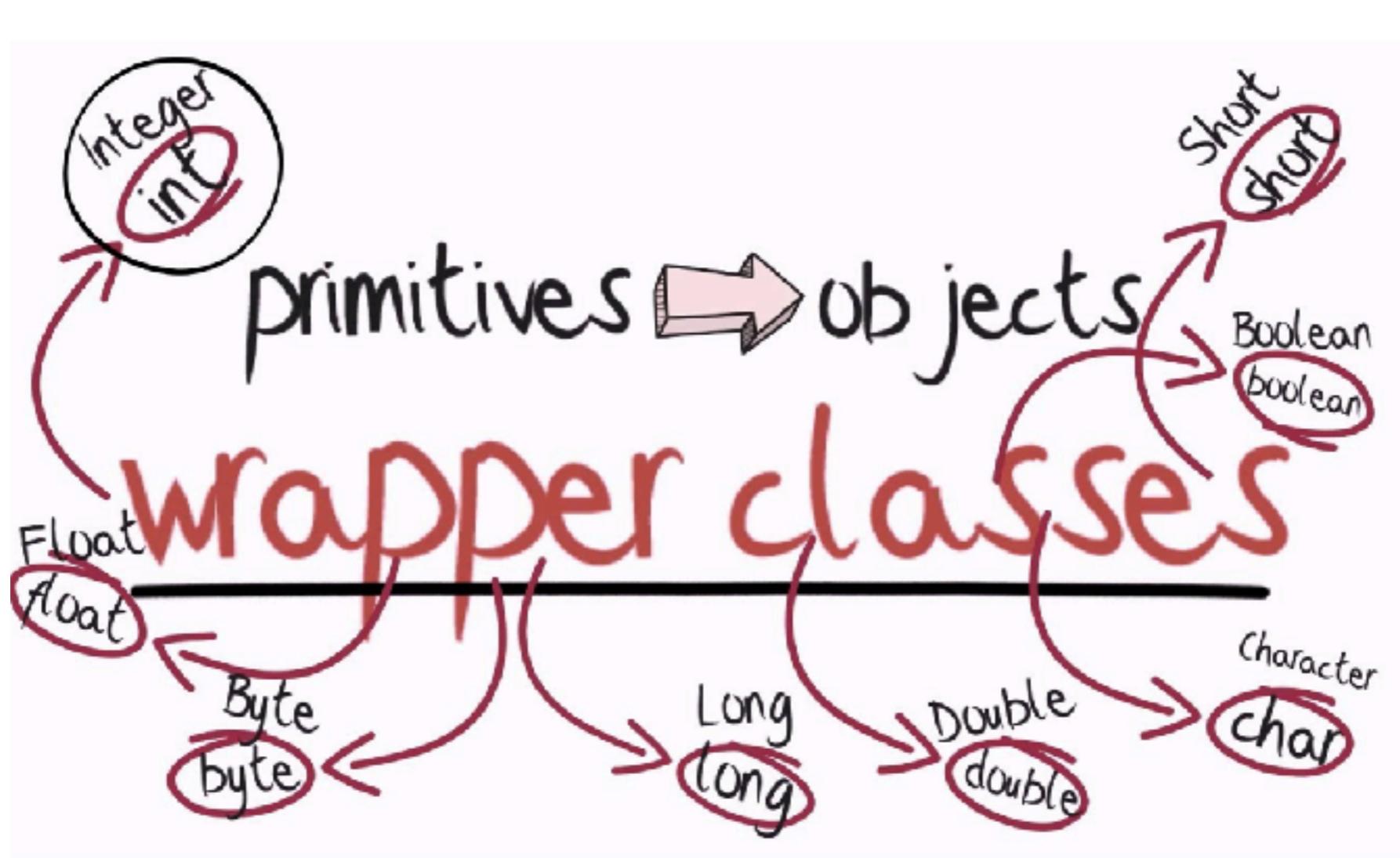


# Primitive data types

Primitive data types		Min/Max value
byte	8-bit signed two complement integer	-128 / 127
short	16-bit signed two complement integer	-32,768 / 32,767
int	32-bit signed two complement integer	-2 <sup>31</sup> / 2 <sup>31</sup> - 1
long	64-bit signed two complement integer	-2 <sup>63</sup> / 2 <sup>63</sup> - 1
float	32-bit IEEE 754 floating point	
double	64-bit IEEE 754 floating point	
boolean		true, false
char	16-bit Unicode character	\u0000 / \uffff



# Wrapper class

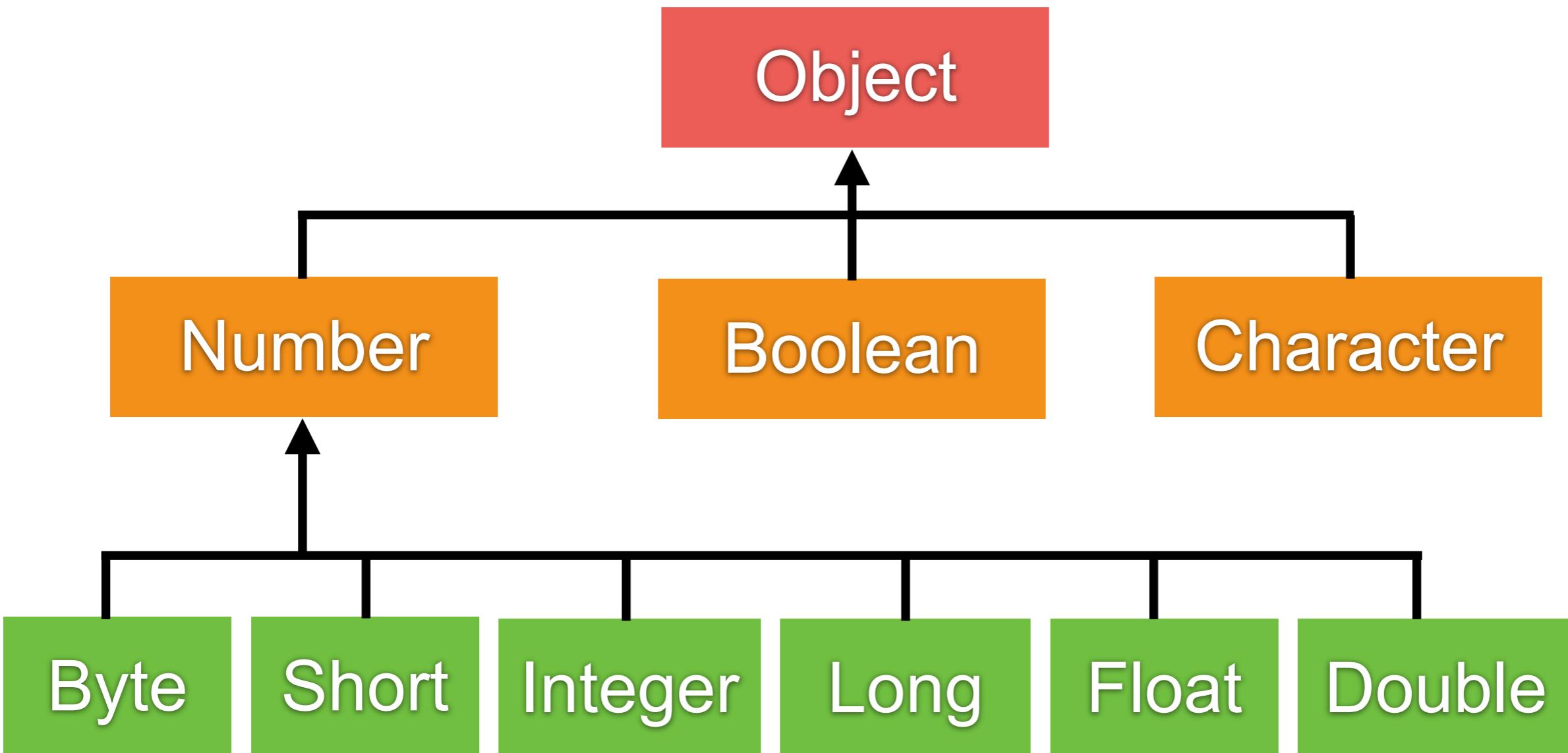


# Wrapper class

Primitive data types	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character



# Wrapper class hierarchy

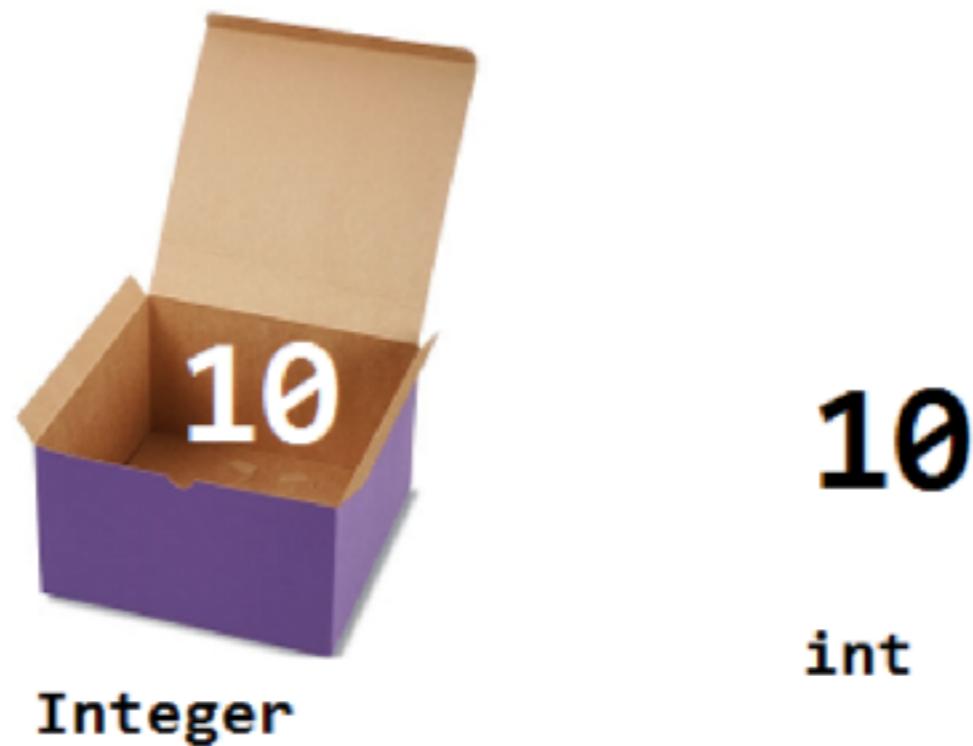


# Min/Max value of primitive type

Primitive data types	Minimum value	Maximum value
byte	Byte.MIN_VALUE	Byte.MAX_VALUE
short	Short.MIN_VALUE	Short.MAX_VALUE
int	Integer.MIN_VALUE	Integer.MAX_VALUE
long	Long.MIN_VALUE	Long.MAX_VALUE
float	Float.MIN_VALUE	Float.MAX_VALUE
double	Double.MIN_VALUE	Double.MAX_VALUE
char	Character.MIN_VALUE	Character.MAX_VALUE



# Working with wrapper class (1)



# Working with wrapper class (2)

```
// Wrapping
int number = 10;
Integer number2 = new Integer(number);
Integer number3 = Integer.valueOf(number);

// Unwrapping
int valueBack2 = number2.intValue();
int valueBack3 = number3.intValue();
System.out.println(valueBack2);
System.out.println(valueBack3);
```



# Wrapper class conversion

## Autoboxing and Autounboxing

```
// Autoboxing
int number = 10;
Integer number2 = number;
```

```
// Autounboxing
int valueBack2 = number2;
System.out.println(valueBack2);
```



# Final variables

Can't change the value of variables (immutable)

Use keyword **final**

```
// Constant variable
private static final int NUMBER_ONE = 1;

private void init() {
    // Final variable
    final int number = 10;
    number = 200; // Can't modify value
}
```



# Functions !!

Write small functions

Name them properly

Hide implementation details

Document immediately

Tests



# 2. Classes

Complex data types

Classes are **standard** from Java such as String,  
Integer

We can create new classes as we need



# Working with String

Not primitive data type

Standard class in Java

Represent a sequence of one or more characters

```
String shortName = "S";  
String name = "Somkiat Puisungngoen";
```

<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>



# Working with String operation

```
String name = "Somkiat";
System.out.println(name.concat(" Pui"));
System.out.println(name.substring(1, 2));
System.out.println(name.charAt(0));
System.out.println(name.startsWith("S"));
System.out.println(name.endsWith("t"));
```



# Operators

Operators that require 2 operands are called  
**Binary operator**

Operators that require only 1 operand are called  
**Unary operator**

Parentheses, () are called **Grouping operator**



# Operators

Arithmetic operators

    Unary operators

Equality and Relational operators

    Conditional operators

Bitwise and bit shift operators



# Arithmetic operators

Operator	Description
+	Addition operator ( can use for String concatenation)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Remainder operator



# String with additional operator

```
String firstName = "Somkiat";
String lastname = "Puisungnoen";
int age = 30;
```

```
String result = firstName + lastname + age;
```

```
System.out.println(result);
```



# Formatting result

```
String firstName = "Somkiat";
String lastname = "Puisungnoen";
int age = 30;
```

```
System.out.printf("%s %s %d"
    , firstName, lastname, age);
```



# Formatting result with String

```
String firstName = "Somkiat";
String lastname = "Puisungnoen";
int age = 30;
```

```
String result = String.format("%s %s %d",
    firstName, lastname, age);
```

```
System.out.println(result);
```



# Unary operators

Required only one operand

Operator	Description
+	Unary plus operator
-	Unary minus operator
++	Increment operator
--	Decrement operator
!	Logical complement operator



# Equality and relational operator

Operator	Description
<code>==</code>	Equal to
<code>!=</code>	Not equal to
<code>&gt;</code>	Greater than
<code>&gt;=</code>	Greater than or equal to
<code>&lt;</code>	Less than
<code>&lt;=</code>	Less than or equal to



# Conditional operator

Operator	Description
&&	Conditional-AND
	Conditional-OR
?:	Ternary (shorthand for if-then-else statement)



# Type comparison with instanceof

```
String object1 = new String();
```

```
System.out.println(object1 instanceof String);  
System.out.println(object1 instanceof Object);
```



# Bitwise and bit shift operator

Operator	Description
&	Bitwise AND operator
	Bitwise inclusive OR operator
^	Bitwise exclusive OR operator
~	Unary bitwise complement
>>	Signed left shift
<<	Signed right shift
>>>	Unsigned right shift



# Useful methods from Math (1)

Method	Description
abs()	returns the absolute value of the input value
round()	returns the integer nearest to the input value
ceil()	returns the smallest integer that is bigger than or equal to the input value
floor()	returns the biggest integer that is smaller than or equal to the input value
exp()	returns the exponential of the input value
max()	returns the bigger between the two input values
min()	returns the smaller between the two input values

<https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>



# Useful methods from Math (2)

Method	Description
pow()	returns the value of the first value raised to the power of the second value.
sqrt()	returns the square root of the input value.
sin()	returns the trigonometric sine value of the input value
cos()	returns the trigonometric cosine value of the input value.
tan()	returns the trigonometric tangent value of the input value.

<https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>



# Workshop with Math

The distance,  $d$ , between two points in the three-dimensional space  $(x_1, y_1, z_1)$  and  $(x_2, y_2, z_2)$  can be computed from:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

The following program computes and shows the distance between  $(2, 1, 3)$  and  $(0, 0, 6)$



# Compound assignment

Original	Shorthand
<code>x = x + 1</code>	<code>x += 1</code>
<code>x = x - 1</code>	<code>x -= 1</code>
<code>x = x * 1</code>	<code>x *= 1</code>
<code>x = x / 1</code>	<code>x /= 1</code>



# Caution for complex expression

Original	Shorthand
$x = x / y + 1$	$x /= y + 1$ (Wrong !!)
$x = x / y + 1$	$x /= (y + 1)$



# Operator precedence (1)

No.	Operator	Precedence
1	Grouping	( )
2	Postfix	expr++, expr--
3	Unary/Prefix	++expr, --expr, +expr, -expr, ~, !
4	Multiplicative	*, /, %
5	Addtitive	+, -
6	Shift	>>, <<, >>>
7	Relational	<, >, <=, >=, instance

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html>



# Operator precedence (2)

No.	Operator	Precedence
8	Equality	<code>==, !=</code>
9	Bitwise AND	<code>&amp;</code>
10	Bitwise exclusive OR	<code>^</code>
11	Bitwise inclusive OR	<code> </code>
12	Logical AND	<code>&amp;&amp;</code>
13	Logical OR	<code>  </code>
14	Ternary	<code>?:</code>
15	Assignment	<code>=, +=, &lt;&lt;=, &gt;&gt;= ...</code>



# Increment/Decrement operator

## Prefix

the value inside the variable is incremented (or decremented) and then use to evaluate the expression

## Postfix

the value inside the variable is used to evaluate the expression and then the value is incremented (or decremented)



# Workshop

```
int a, b, c = 0;  
a = c++;  
b = ++a;  
c++;  
b = ++c + a++;  
a = --b + c++;
```

a	b	c
0	0	0

```
System.out.println(a); // ?  
System.out.println(b); // ?  
System.out.println(c); // ?
```



# Basic of Java Part 2



# Basic of Java Part 2

Control flow statements

Conditional statements (if-else/switch case)

For loop

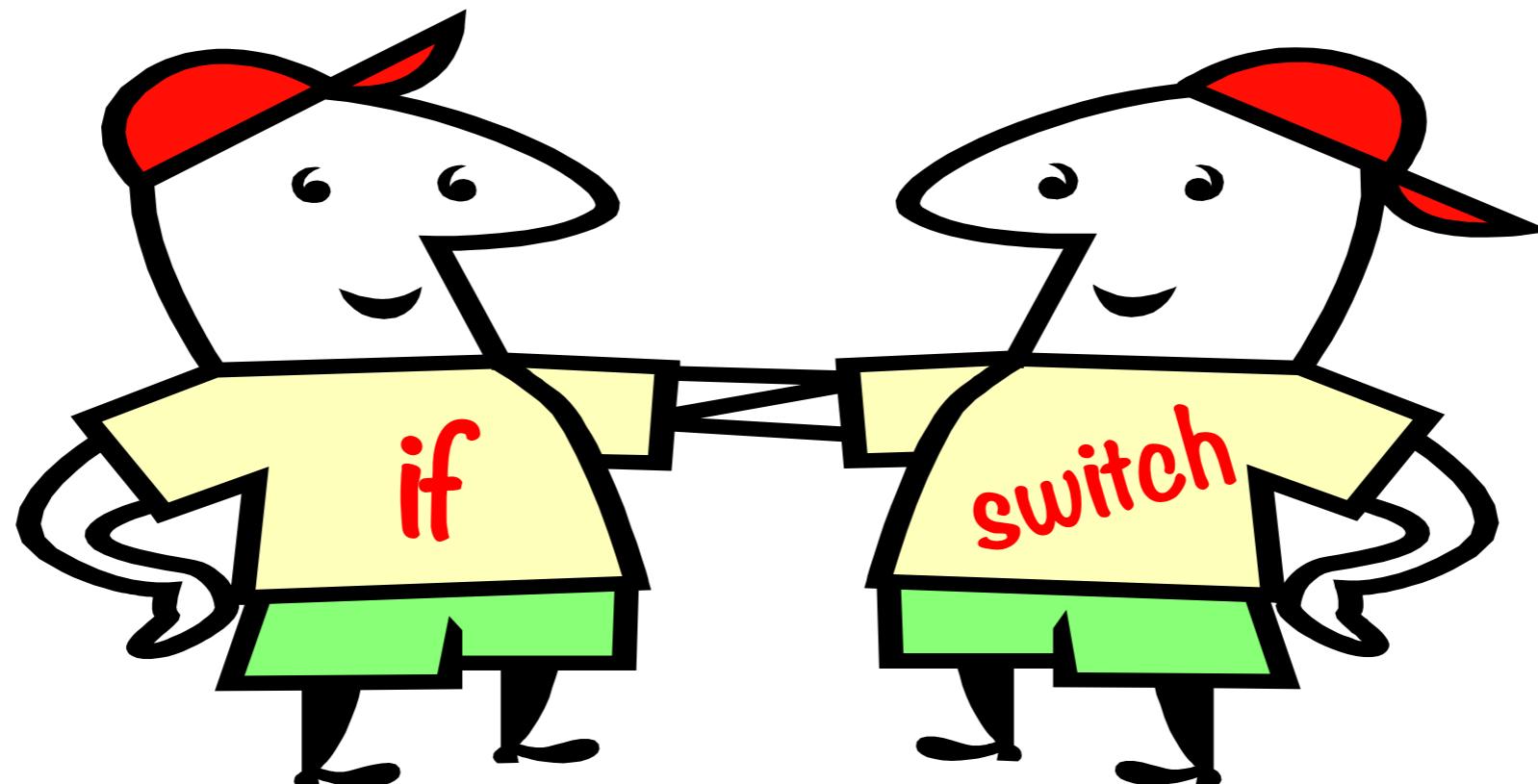
While loop

Do-while loop

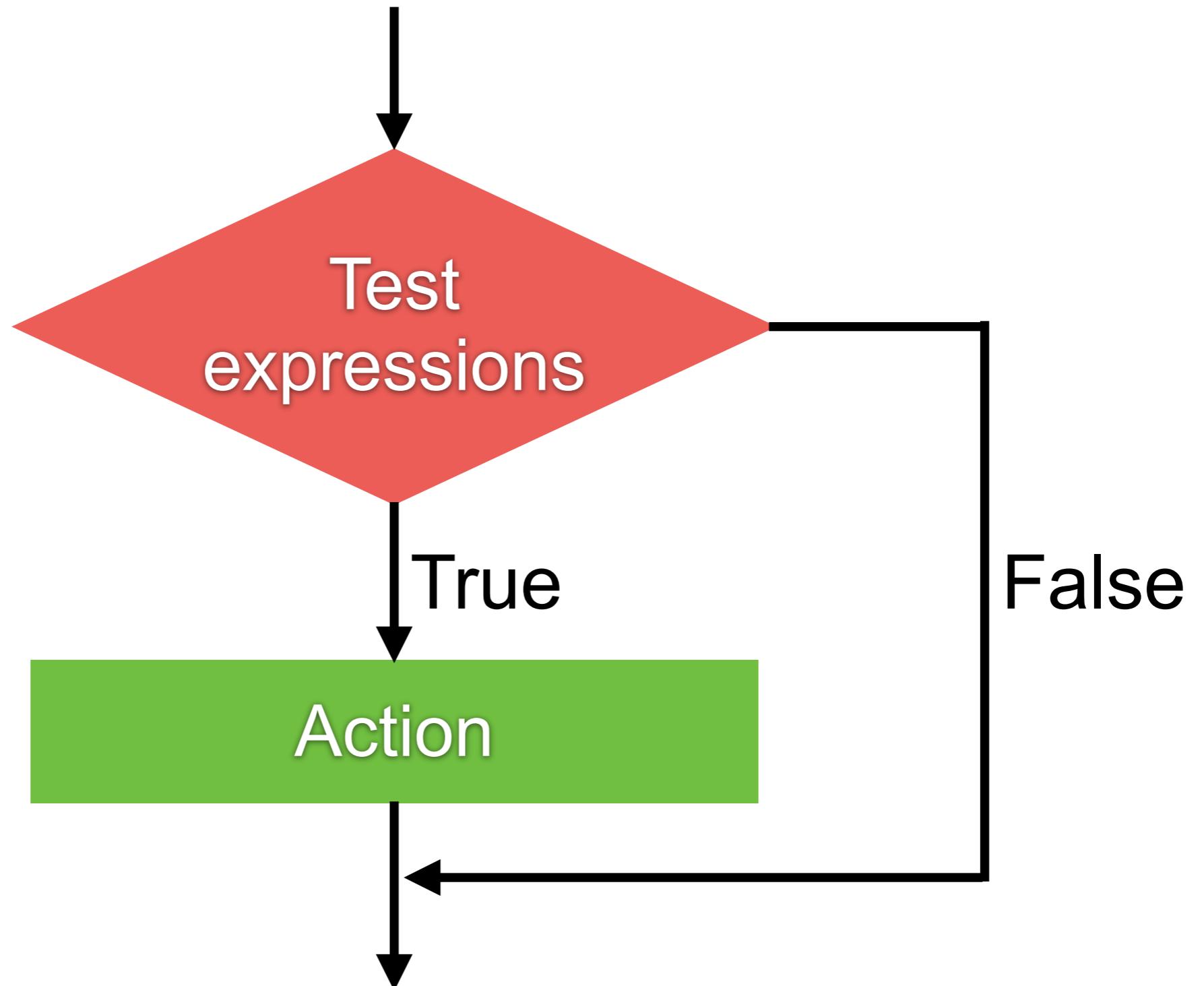


# Conditional statements

if-else if - else  
switch-case



# IF - ELSE IF - ELSE



# IF - ELSE IF - ELSE

```
int score = 50;

if (score >= 80 && score <= 100) {
    System.out.println("Grade A");
} else if (score >= 70 && score < 80) {
    System.out.println("Grade B");
} else if (score >= 60 && score < 70) {
    System.out.println("Grade C");
} else if (score >= 50 && score < 60) {
    System.out.println("Grade D");
} else {
    System.out.println("Grade F");
}
```



# Ternary operator (short if)

```
int score = 50;
```

```
String grade = score > 80 ? "A" : score > 70 ? "B" : "C";
```

```
System.out.println(grade);
```

*Easy to read and understand ?*



# Workshop

# String comparison



# String comparison (1)

```
String name1 = new String("Somkiat");
String name2 = new String("Somkiat");

if(name1 == name2) {
    System.out.println("Equal");
} else {
    System.out.println("Not equal");
}
```

---



# String comparison (2)

```
String name1 = new String("Somkiat");
String name2 = new String("Somkiat");

if(name1.equals(name2)) {
    System.out.println("Equal");
} else {
    System.out.println("Not equal");
}
```



# String Equality



# String comparison (3)

```
String name1 = "Somkiat";
String name2 = "Somkiat";

if(name1 == name2) {
    System.out.println("Equal");
} else {
    System.out.println("Not equal");
}
```



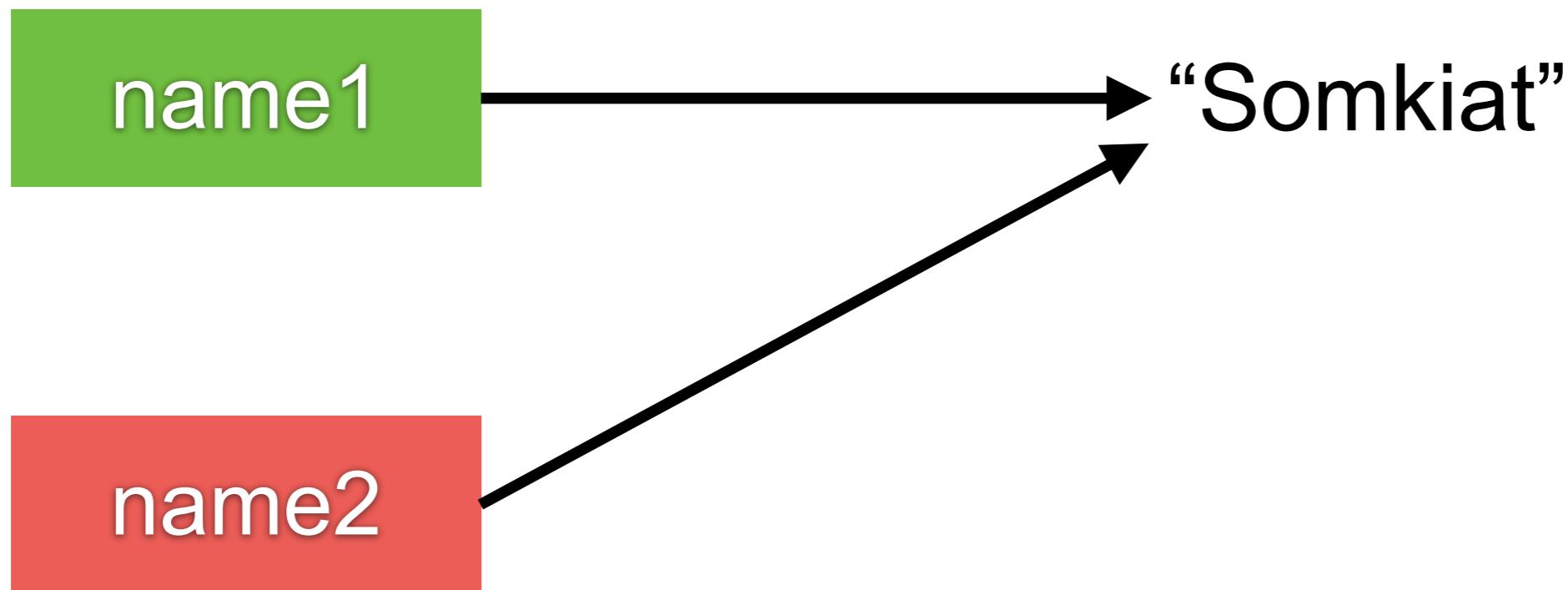
# String comparison (4)

```
String name1 = "Somkiat";
String name2 = "Somkiat";

if(name1.equals(name2)) {
    System.out.println("Equal");
} else {
    System.out.println("Not equal");
}
```



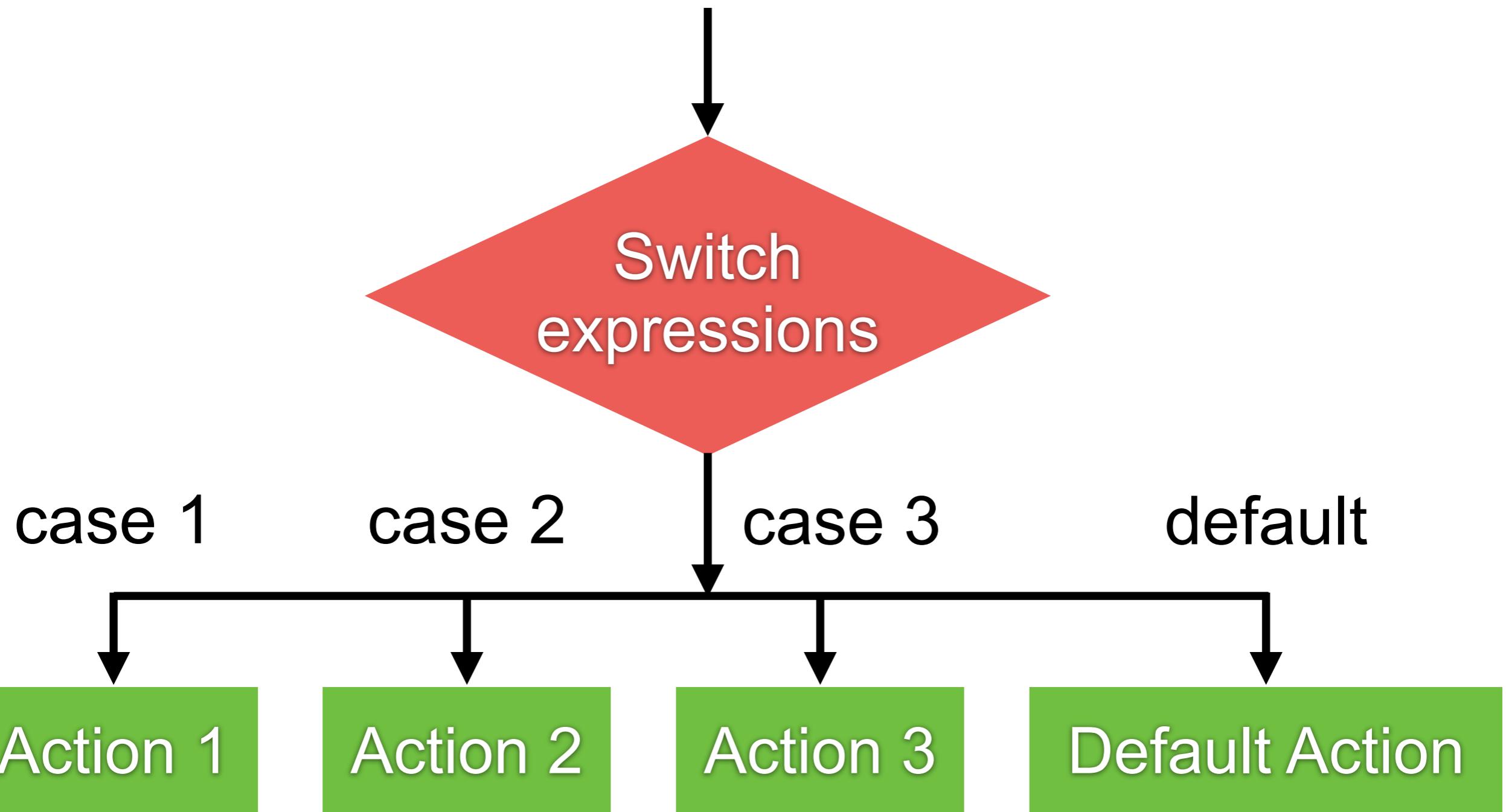
# String Equality



# Switch-Case



# Switch-Case



# Switch-Case

```
int number = 1;

switch (number) {
case 1:
    System.out.println("Case 1");
    break;
case 2:
    System.out.println("Case 1");
    break;

default:
    System.out.println("Default");
}
```



# All about break !!

```
int number = 1;

switch (number) {
case 1:
    System.out.println("Case 1");
case 2:
    System.out.println("Case 1");

default:
    System.out.println("Default");
}
```

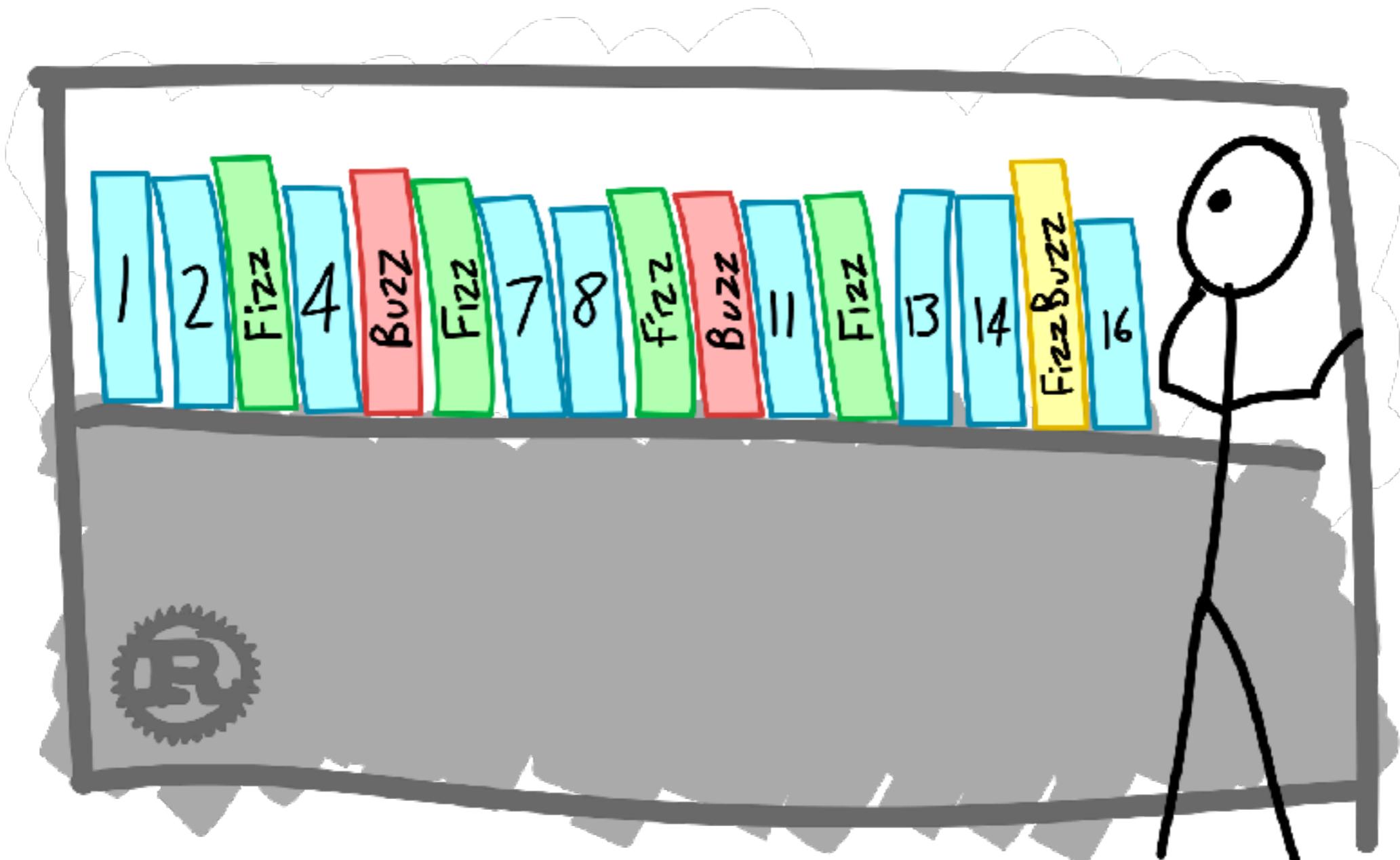


# **Workshop**

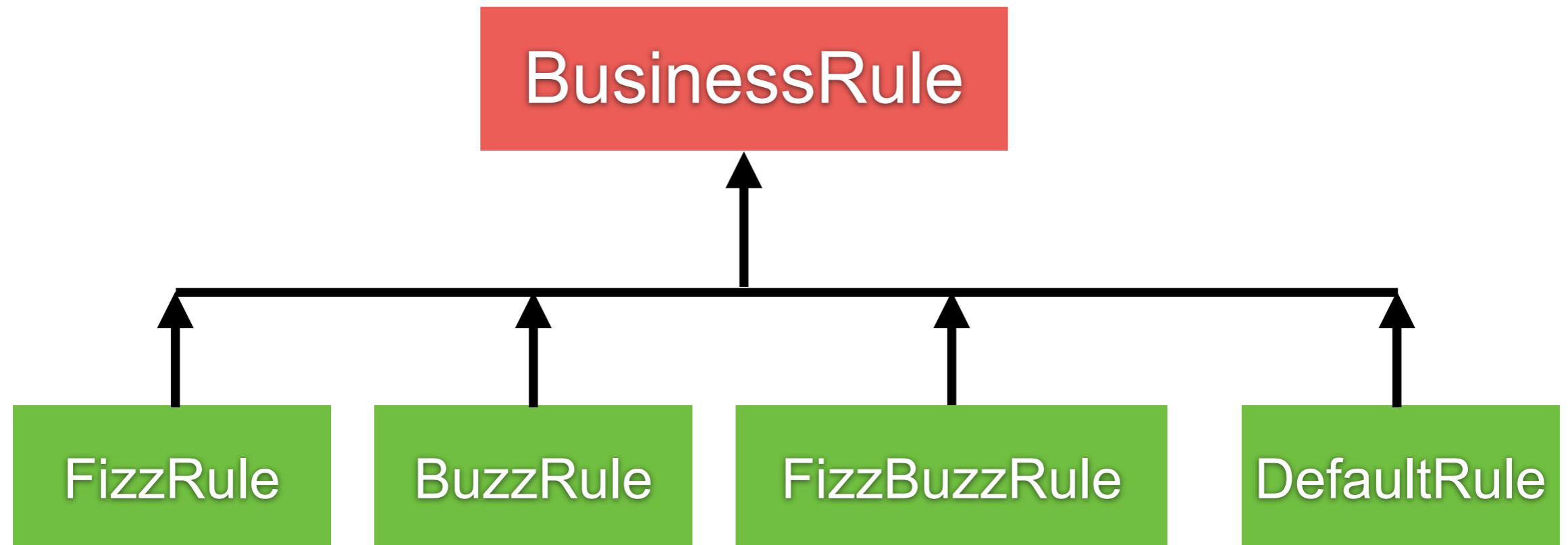
# **Conditional statement**



# FizzBuzz



# FizzBuzz



# Iteration statements

**for** statement

**while** statement

**do-while** statement

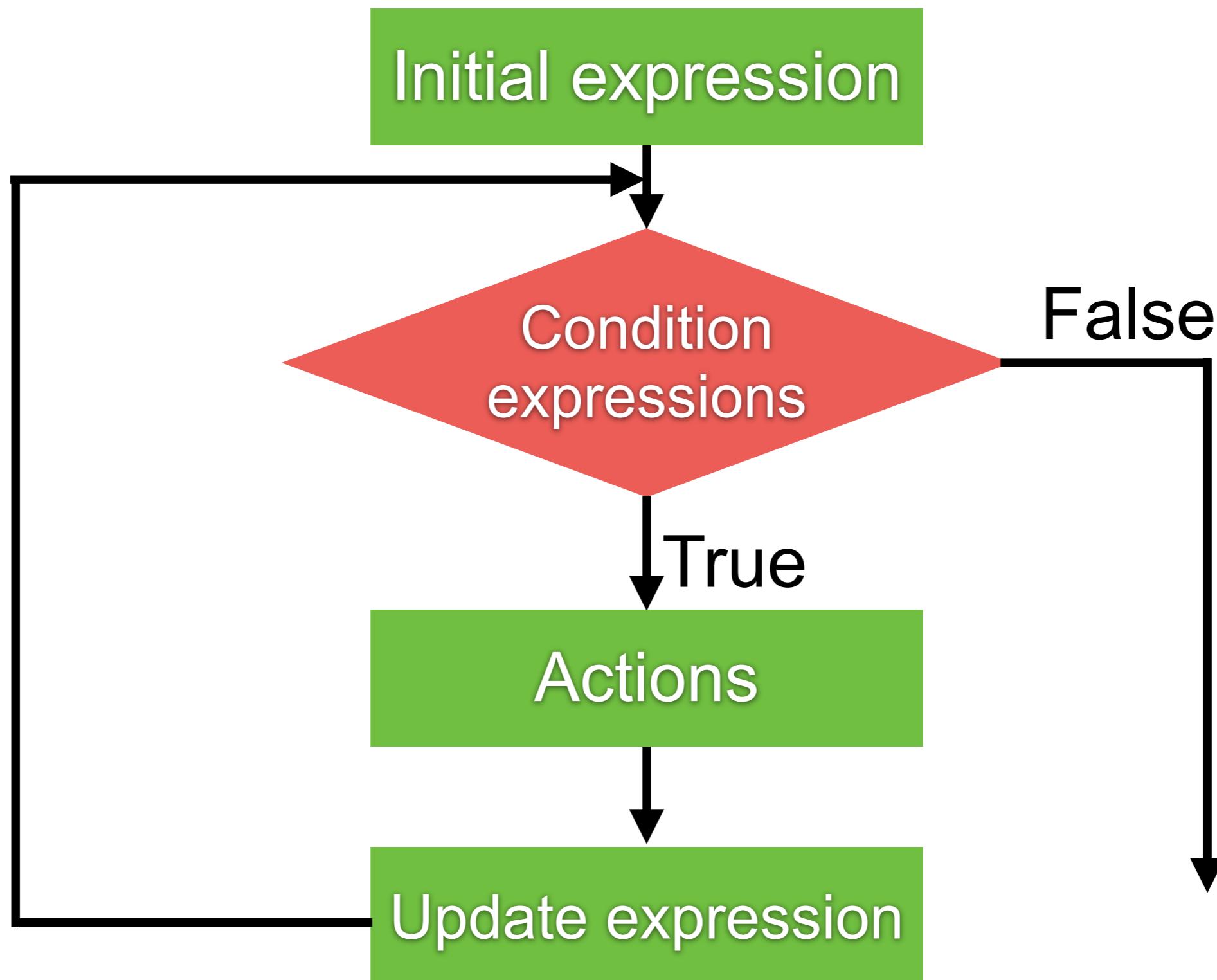


# Why we need it ?

Find the sum of integer from 1 to 10 ?



# For statement



# For statement

In Java, For statement is recommended

```
for(int i=0; i<10; i++) {  
    System.out.println(i);  
}
```

```
String name = "Somkiat";  
for (int i = 0; i < name.length(); i++) {  
    System.out.println(name.charAt(i));  
}
```



# For-each statement

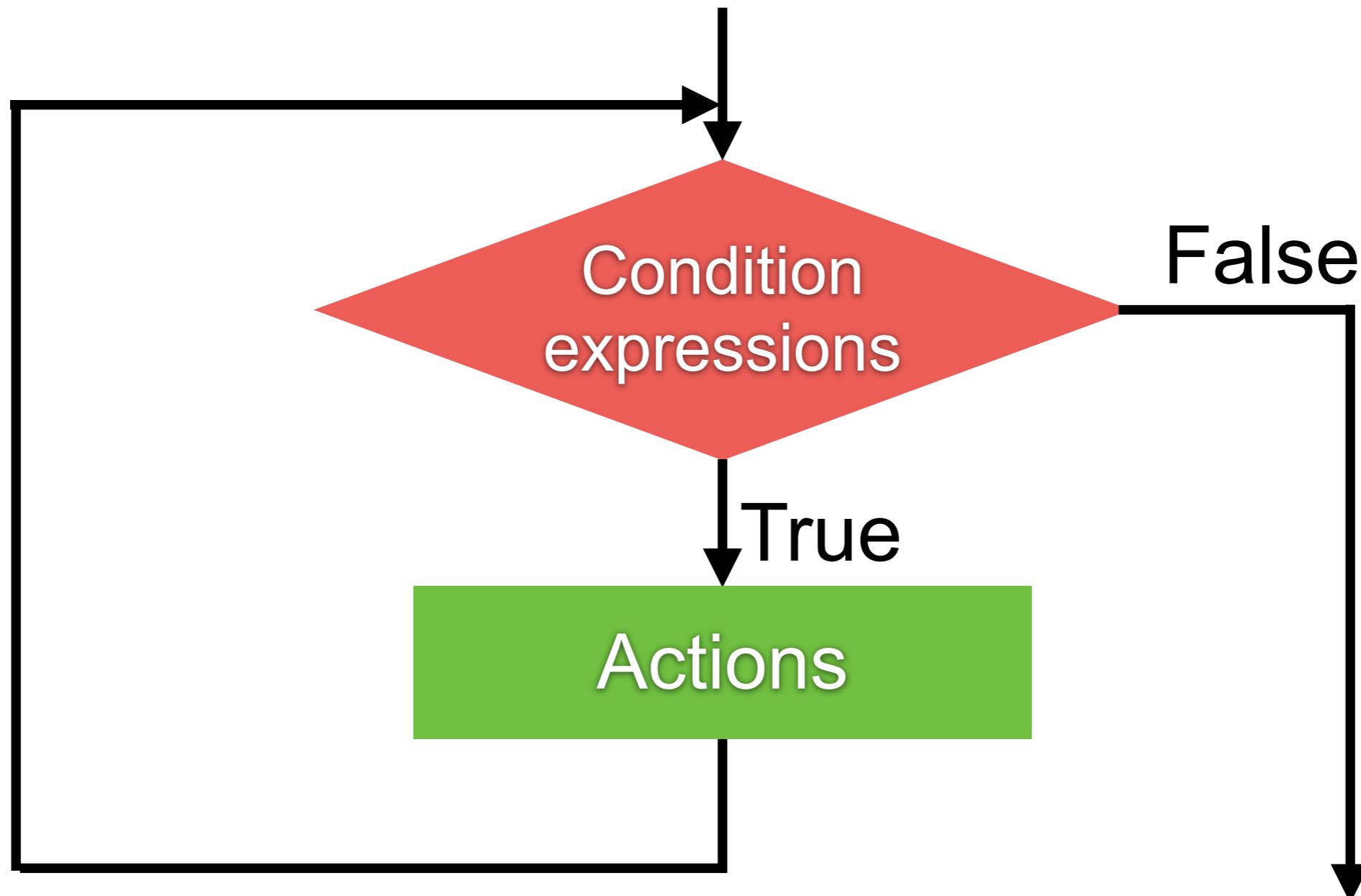
*Array in Java ?*

```
int[] numbers = {1, 2, 3, 4, 5};
```

```
for (int i : numbers) {  
    System.out.println(i);  
}
```



# While statement

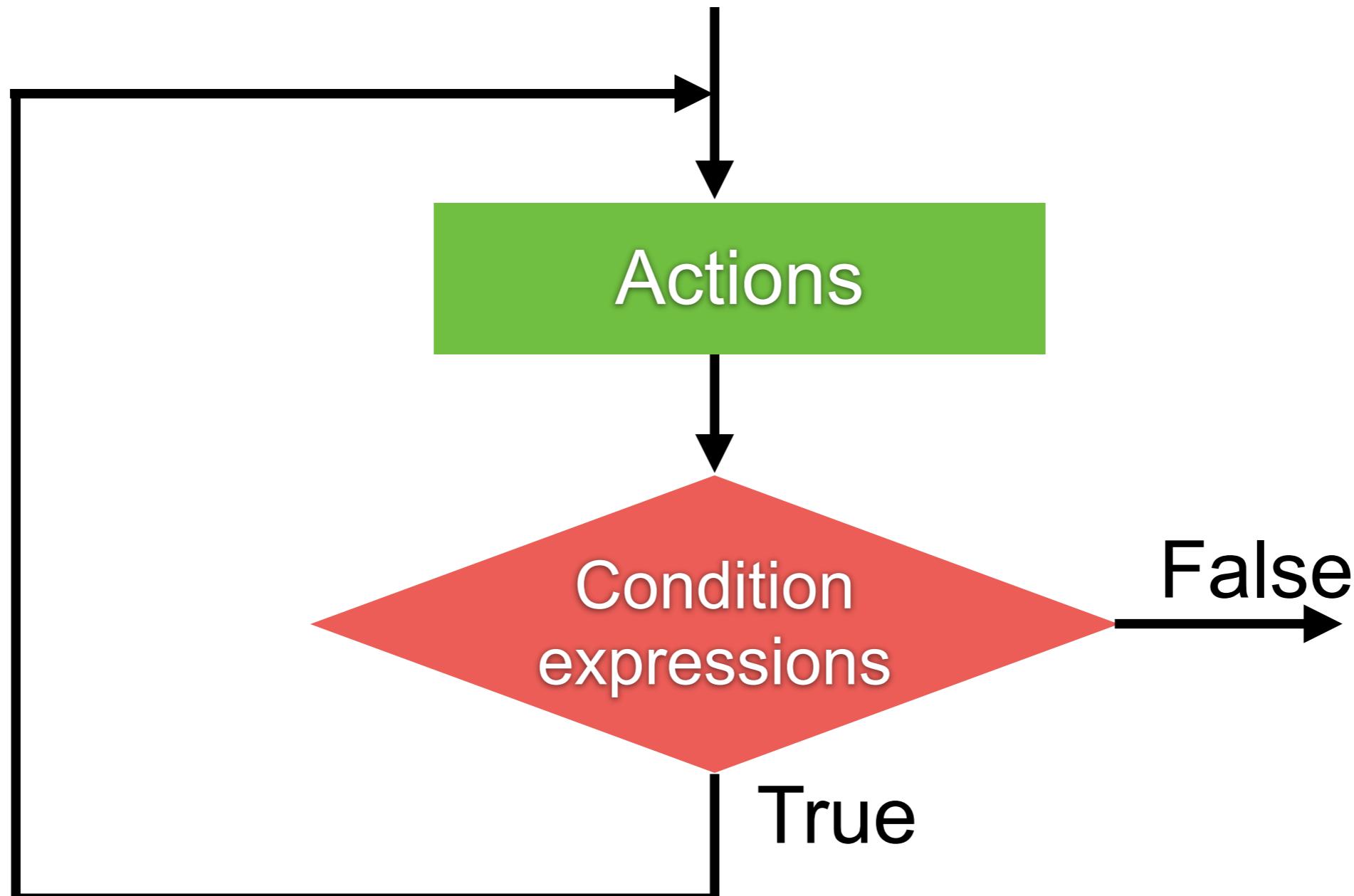


# While statement

```
int i = 0;  
while(i<10) {  
    System.out.println(i);  
    i++;  
}
```



# Do-while statement



# Do-while statement

```
int i = 0;  
do {  
    System.out.println(i);  
    i++;  
} while (i < 10);
```



# Avoid nested if and loop !!



# Avoid nested if and loop !!

```
function register()
{
    if (empty($_POST)) {
        $msg = '';
        if (!$_POST['user_name']) {
            if (!$_POST['user_password_new']) {
                if (!$_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) == $_POST['user_password_repeat']) {
                        if (strlen($_POST['user_password_new']) > 5) {
                            if (strlen($_POST['user_name']) < 6 || strlen($_POST['user_name']) > 24) {
                                if (preg_match('/[a-zA-Z]{2,64}!@/i', $_POST['user_name'])) {
                                    $user = read_user($_POST['user_name']);
                                    if (!check($user['user_name'])) {
                                        if (!$_POST['user_email']) {
                                            if (strlen($_POST['user_email']) < 65) {
                                                if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                    create_user();
                                                    $SESSION['msg'] = 'You are now registered so please login';
                                                    header('Location: ' . SERVER('PHP_SELF'));
                                                    exit();
                                                } else $msg = 'You must provide a valid email address';
                                            } else $msg = 'Email must be less than 64 characters';
                                        } else $msg = 'Email cannot be empty';
                                    } else $msg = 'Username already exists';
                                } else $msg = 'Username must be only a-z, A-Z, 0-9';
                            } else $msg = 'Username must be between 2 and 64 characters';
                        } else $msg = 'Password must be at least 6 characters';
                    } else $msg = 'Passwords do not match';
                } else $msg = 'Empty Password';
            } else $msg = 'Empty Username';
        }
        $SESSION['msg'] = $msg;
    }
    return register_form();
}
```



icompile.eladkarako.com



# Workshop



# Workshop with Prime Factor



# Prime Factor

**Prime numbers** that divide an integer without remainder

Prime numbers are number  $> 1$  that has no divisors

Input	Output
2	2
3	3
4	2, 2
6	2, 3
8	2, 2, 2
10	2, 5
12	2, 2, 3



# Working with Array



# Type of array

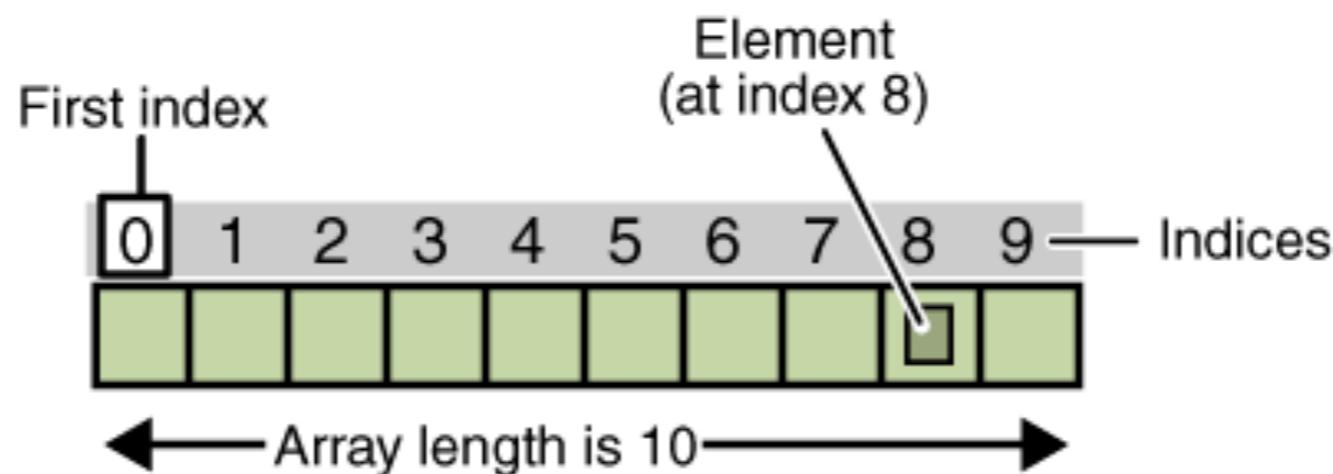
One-dimensional array  
Multi-dimensional array



# One-dimensional array

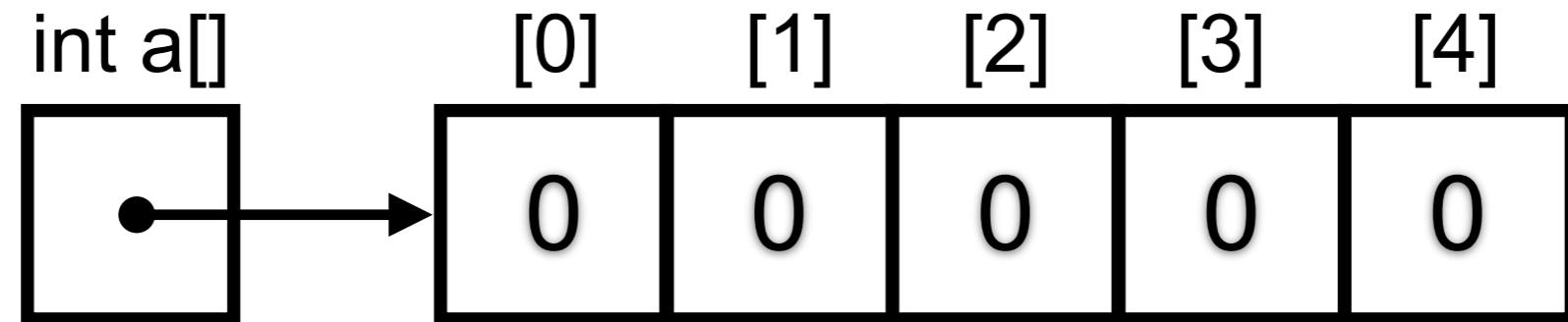
Use to store list of element

*ElementType[] elements;*



# One-dimensional array

```
int[] a = new int[5];
```



# Using and initial array

```
// Declare variables
int[] intergers;
int intergers2[];

// Initial with fixed length
Double[] doubles = new Double[5];

// Initial with value
String[] strings = { "J", "A", "V", "A" };
```



# Default value of number type

Default value = 0

```
int[] numbers = new int[5];  
  
for (int i = 0; i < numbers.length; i++) {  
    System.out.println(numbers[i]); // 0  
}  
  
for (int i : numbers) {  
    System.out.println(i); // 0  
}
```



# Default value of boolean type

Default value = false

```
boolean[] booleans = new boolean[5];  
  
for (boolean b : booleans) {  
    System.out.println(b); // false  
}
```



# Default value of non-primitive

Default value = null

```
Integer[] integers = new Integer[5];  
  
for (Integer integer : integers) {  
    System.out.println(integer);  
}
```



# Access data in array

numbers[index]

index of array start with 0

```
int[] numbers = { 1, 2, 3, 4, 5 };
numbers[0] = 10;
numbers[1] = 20;
numbers[2] = 30;
numbers[3] = 40;
numbers[4] = 50;

for (int i = 0; i < numbers.length; i++) {
    System.out.println(numbers[i]);
}
```



# Workshop with Sorting number in array

1	5	2	10	3	8	3
---	---	---	----	---	---	---



# Multi-dimensional array

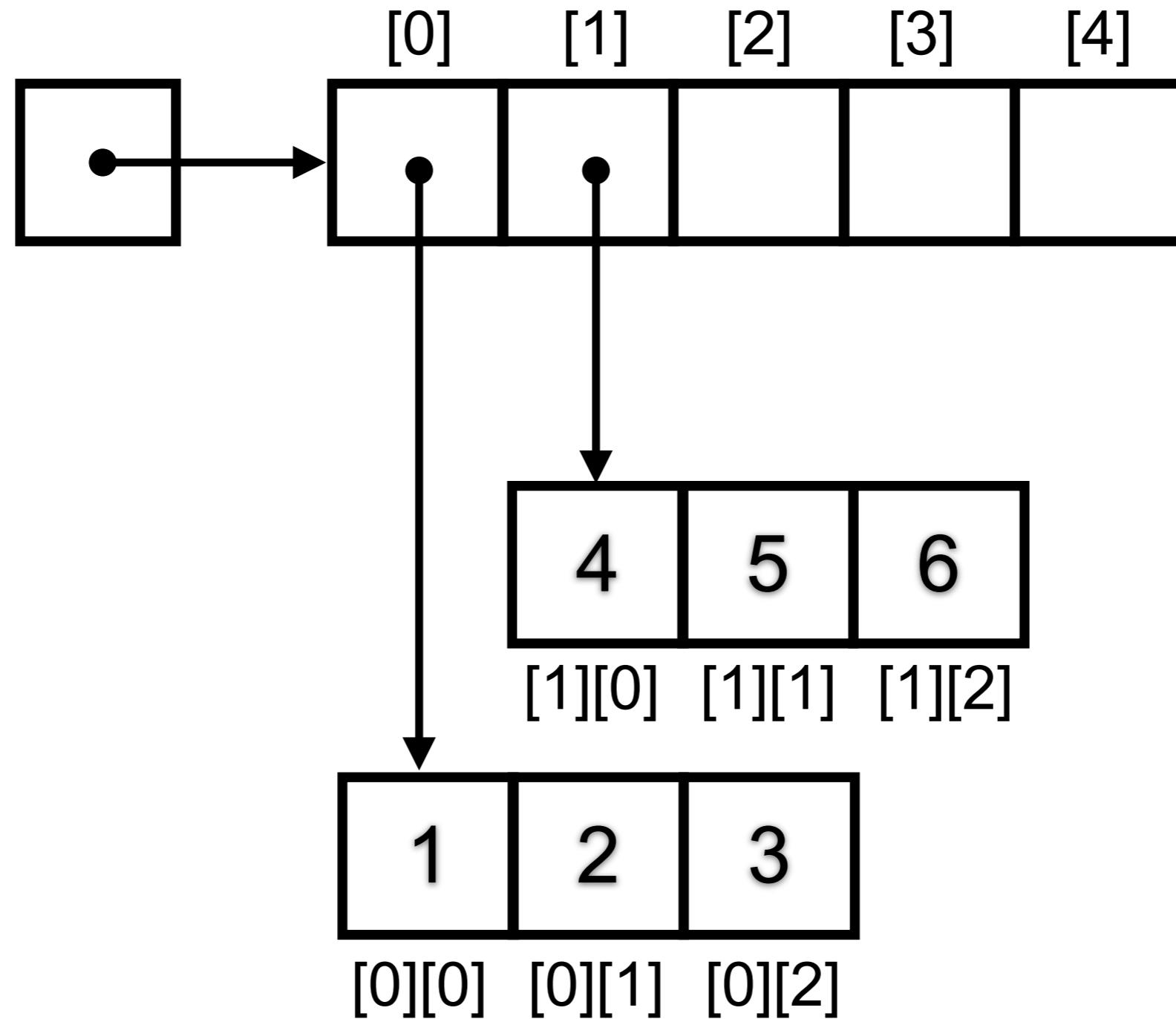
Elements in array can be array

*ElementType[][][] elements;*

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]



# Multi-dimensional array



# Using and initial array

```
int[][] twoDimension = new int[3][3];  
int[][][] threeDimension = new int[2][2][2];
```



# Object-Oriented Programming



# OOP with Java Part 1

Classes and Objects  
Constructors  
Inheritance  
Polymorphism



# Classes and object

## Classes

Non-primitive data type in Java  
Blueprint of object

## Objects

An instance of a class



# Creating new class

Class name

List of properties/attributes (data members)

List of behaviors/methods

Constructor



# Creating a new class

Create file Employee.java

```
public class Employee {  
  
    // Properties/states  
    private int id;  
    private String name;  
    private String address;  
  
    // Behaviors/methods  
    public void setData() {  
  
    }  
  
    public String getData() {  
        return "some data";  
    }  
}
```



# Constructor ?

Special method invoked when objects of the class are created

```
public class Employee {  
    // Properties/states  
    private int id;  
    private String name;  
    private String address;  
  
    // Behaviors/methods  
    public void setData() {  
    }  
  
    public String getData() {  
        return "some data";  
    }  
}
```



# Default constructor

For all classes have default constructor (implicit)

```
public class Employee {  
    // Default constructor  
    public Employee() {  
    }  
}
```



# Overloaded constructor

Constructor can have more than one with different parameters

Many ways to create object



# Overloaded constructor

```
public class Employee {  
    // Default constructor  
    public Employee() {  
    }  
  
    // Overload constructor 1  
    public Employee(int id) {  
        this.id = id;  
    }  
  
    // Overload constructor 2  
    public Employee(int id, String name) {  
        this.id = id;  
        this.name = name;  
    }  
}
```



# Modifier in Java ?

```
public class Employee {  
  
    // Properties/states  
    private int id;  
    private String name;  
    private String address;  
  
    // Behaviors/methods  
    public void setData() {  
  
    }  
}
```



# Modifier in Java

public  
private  
protected  
default



# Modifier in Java

<b>Visibility</b>	<b>Private</b>	<b>Default</b>	<b>Protected</b>	<b>Public</b>
Same class	Yes	Yes	Yes	Yes
Same package	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes



# Create instances from class

```
Employee employee1;  
employee1 = new Employee();  
employee1.setData();
```

```
Employee employee2 = new Employee();  
employee2.setData();
```

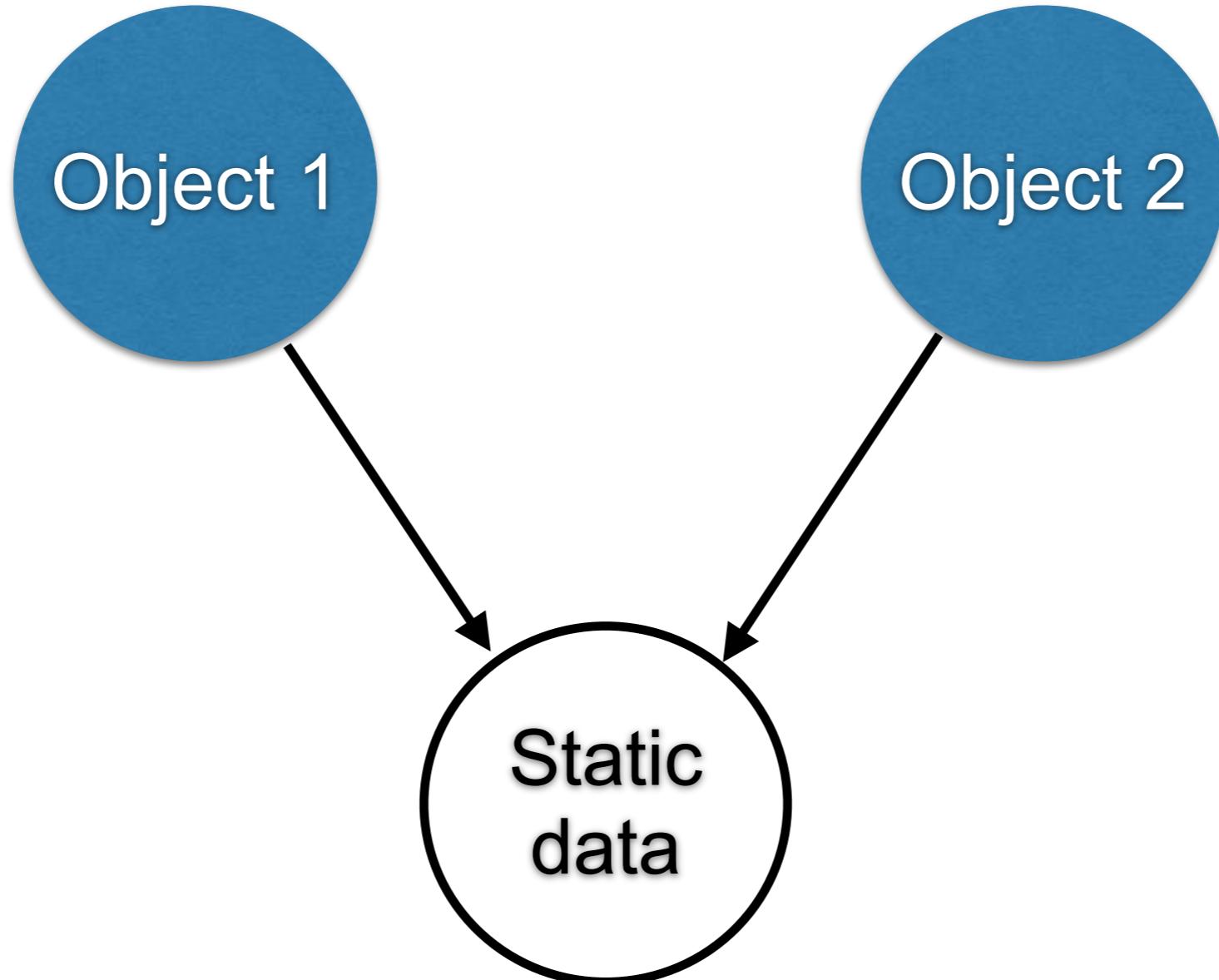


# Static and non-static data member

```
public class Employee {  
    // Non-static data member  
    public int id;  
    public String name;  
  
    // Static data member  
    public static int counter;  
}
```



# Static data member



# Be careful to use static !!

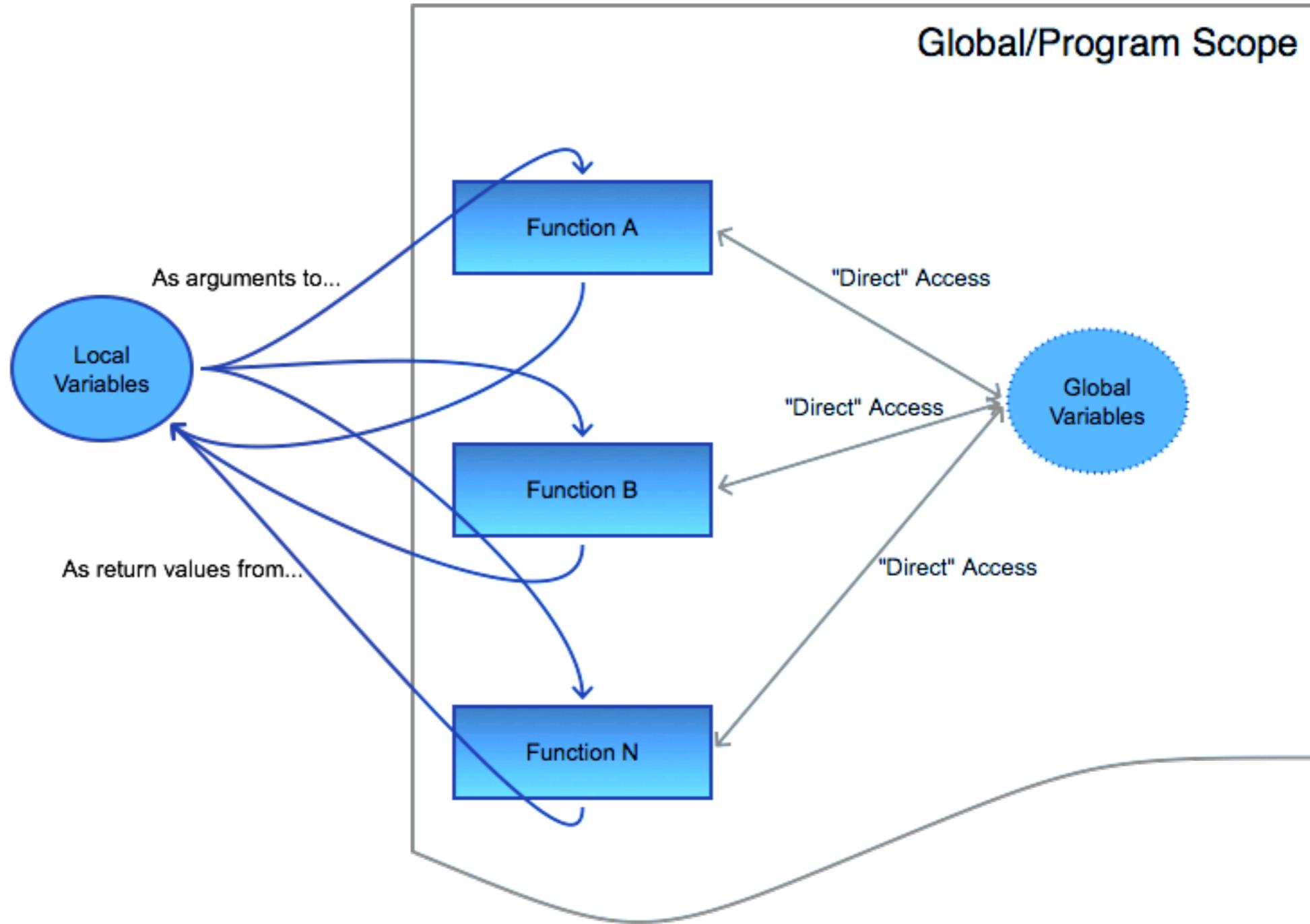
```
Employee employee1 = new Employee();
employee1.counter++; // Warning
Employee.counter++;
```

```
Employee employee2 = new Employee();
employee2.counter++; // Warning
Employee.counter++;
```

```
System.out.println(employee1.counter); // 2 !!
System.out.println(employee2.counter); // 2 !!
```



# Global states are bad !!



# Methods of class

Describe behaviors of object of the class  
Static and Non-static method (instance)



# Methods

## **Non-static method (instance)**

Invoked by other class via the **instance name**

Using dot (.) operator

Default of all methods

## **Static method**

Create method with **static** keyword

Invoked by other class via the **class name**



# Methods of class

```
(public|private|protected) (static) (return type)  
methodName( list of arguments) {  
  
    // Body of method  
  
}
```



# Examples

```
public void setId(int id) {  
    this.id = id;  
}
```

```
public int getId() {  
    return id;  
}
```



# Suggestion in OOP

Data members in class are defined with **private** to prevent users of class accessing directly

Access data members via **public method**



# Tips for class

How to improve the result ?

```
Employee employee = new Employee();  
System.out.println(employee);
```

*Employee@33909752*



# Tips for class

Try to **overrides** `toString()` method

```
@Override  
public String toString() {  
    return "Employee [id=" + id +  
        ", name=" + name + "]";  
}
```



# Thinking before create class



# Single Responsibility Principle



## Single Responsibility Principle

Just because you *can* doesn't mean you *should*.



# **Workshop OOP with Harry Potter**

<https://codingdojo.org/kata/Potter/>



# Design

Book

Order

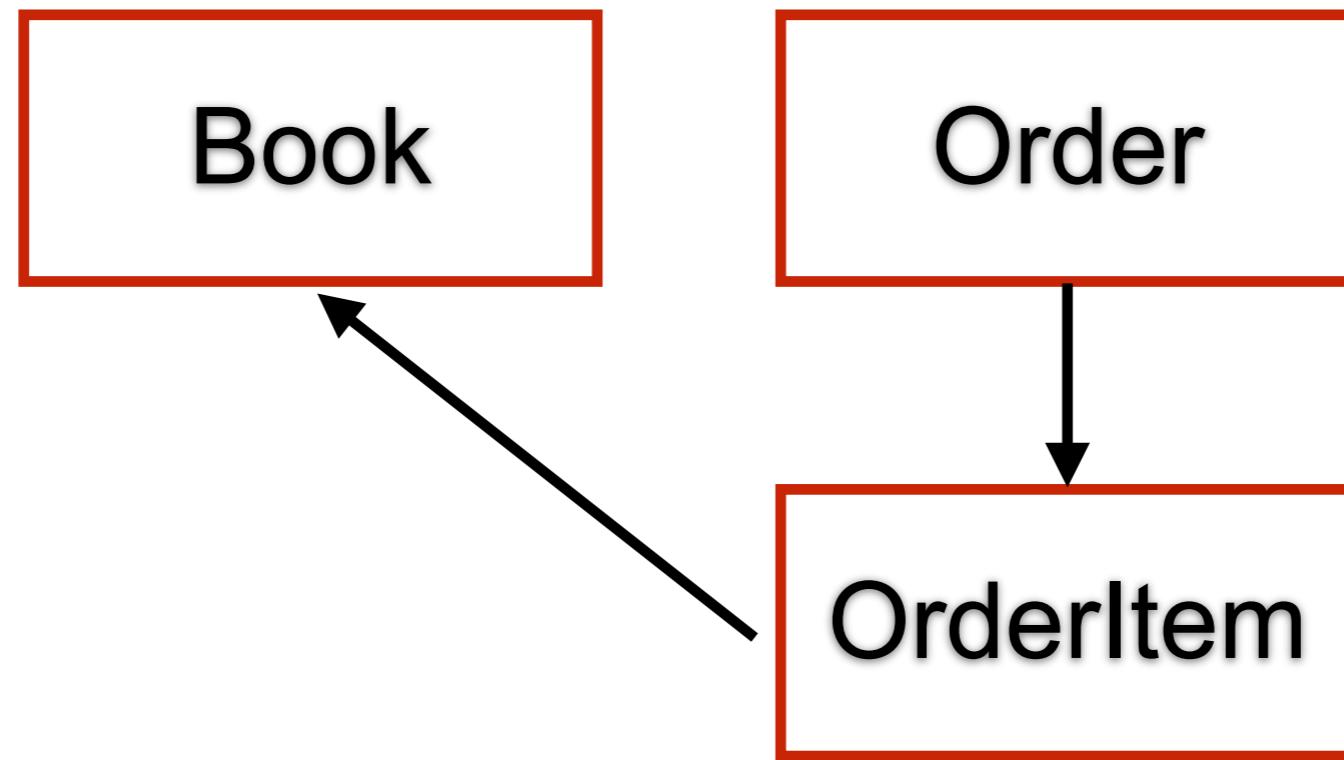
OrderItem

PriceCalculator

DiscountCalculator



# Design

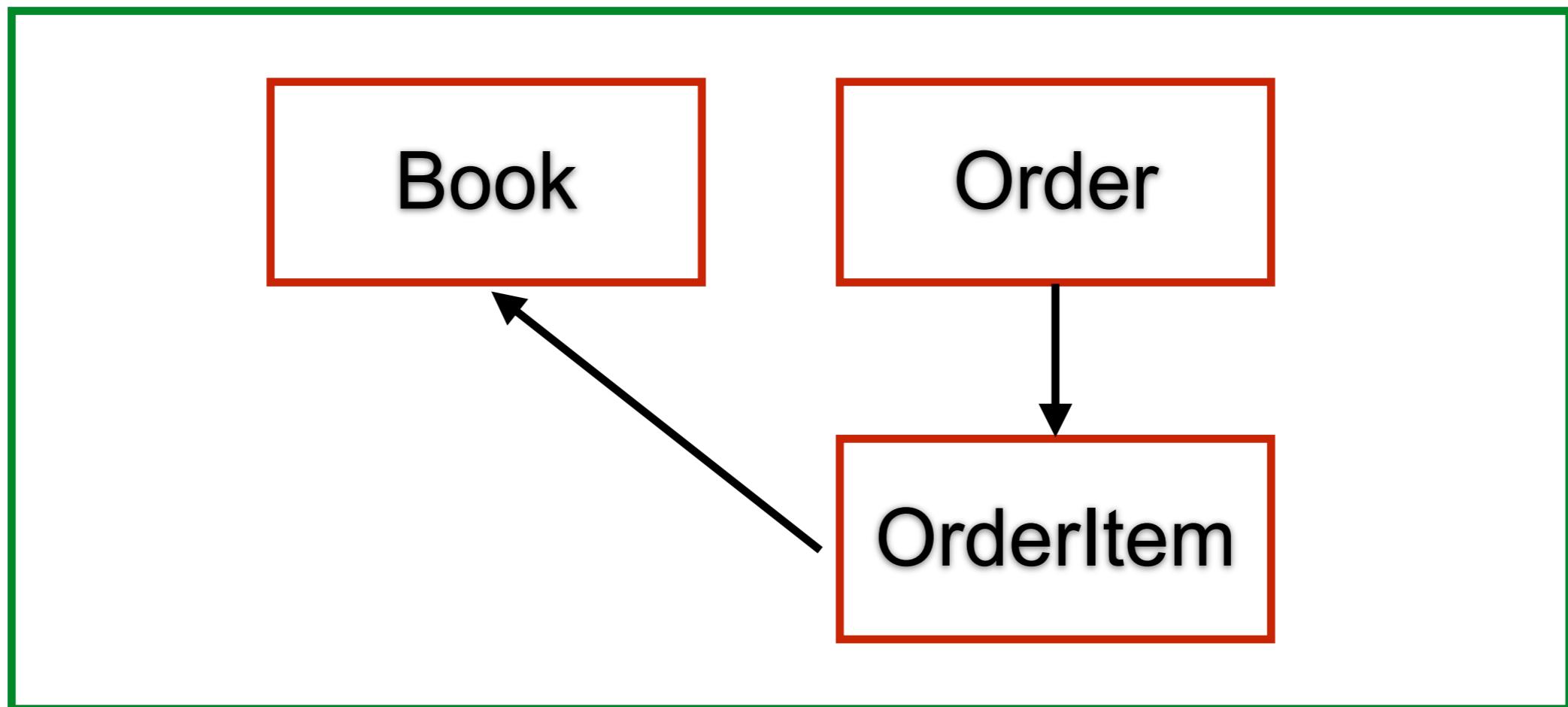


PriceCalculator

DiscountCalculator



# Design

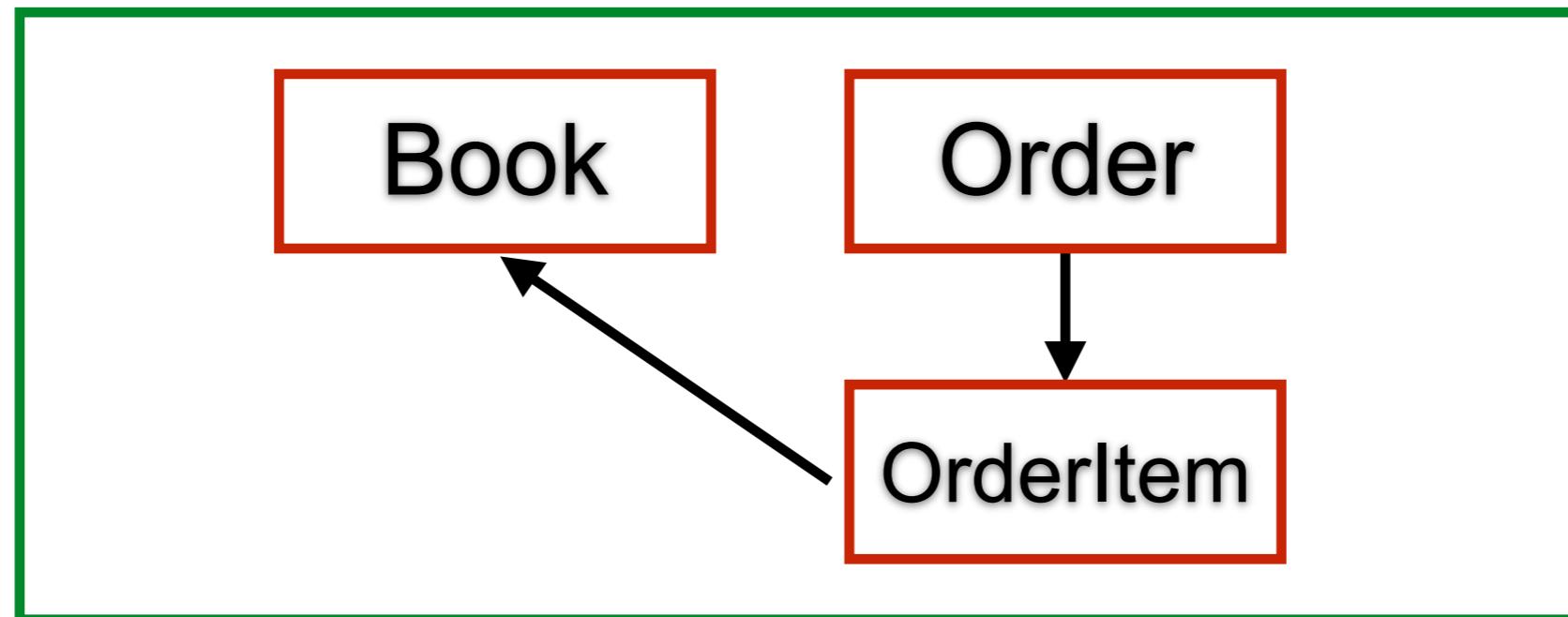


PriceCalculator

DiscountCalculator



# Design



PriceCalculator

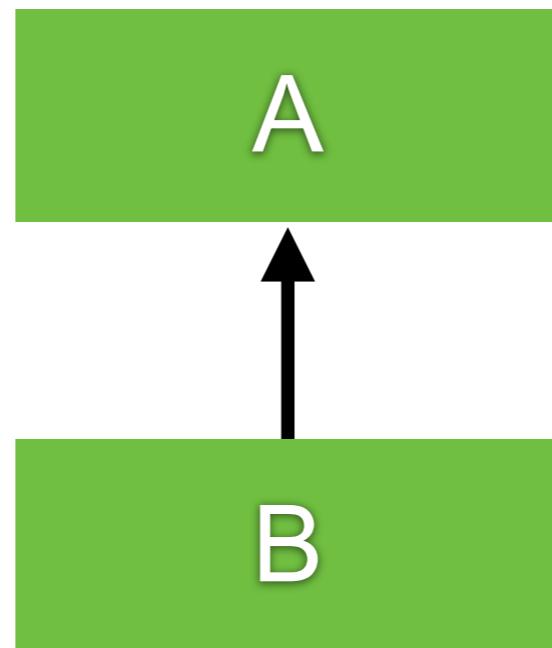
DiscountCalculator



# Inheritance

Ability to derive a new class from existing class

superclass / base class



subclass / derived class

*Additional properties and behaviors*



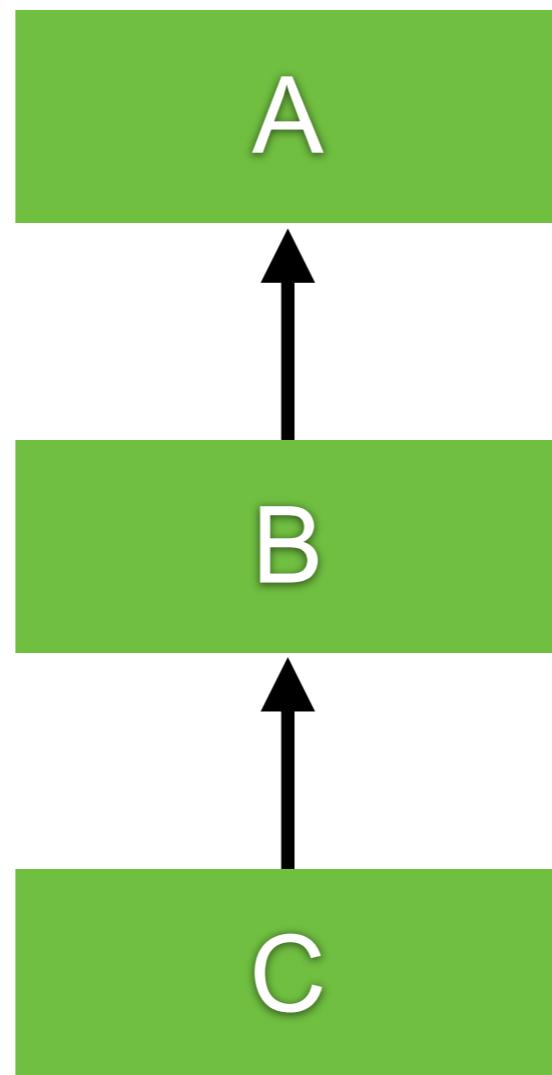
# How to create ?

Extends from base class only one class

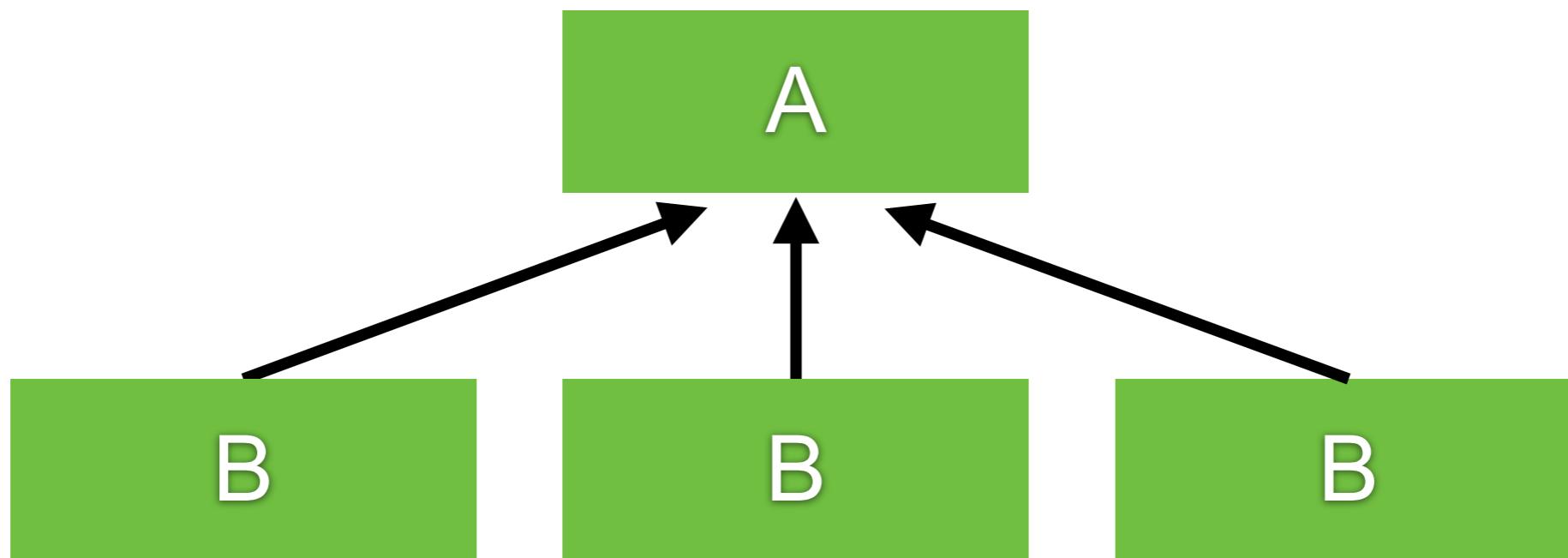
```
class BaseClass {  
}  
  
class SubClass extends BaseClass {  
}  
}
```



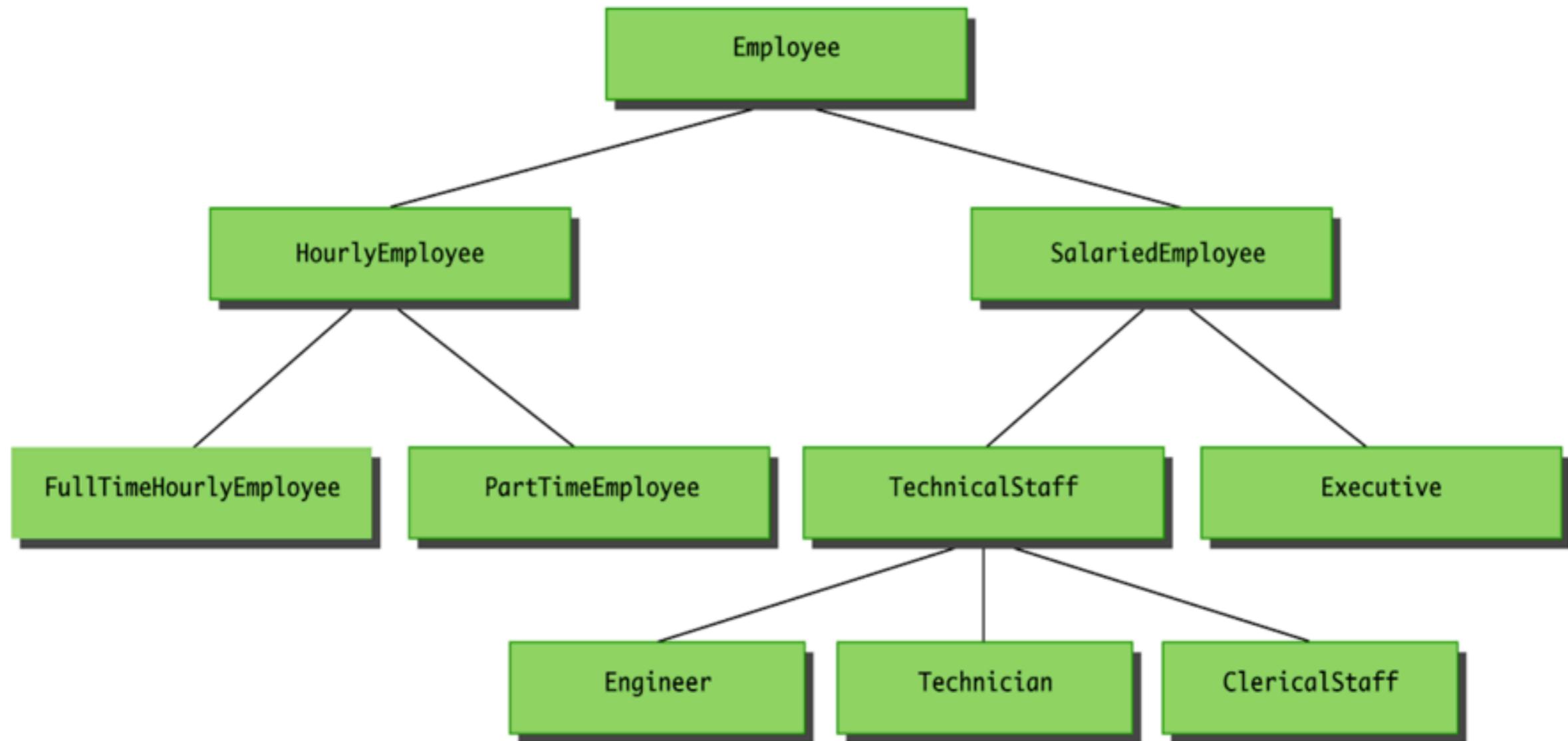
# Multilevel Inheritance



# Hierarchical Inheritance



# Hierarchical Inheritance



# More example with Employee

```
class Employee {  
  
    private int id;  
    protected String name;  
  
    protected void showName() {  
        System.out.println("Employee showName()");  
    }  
  
}
```



# Create Manager class

Use super() to call superclass/base class

```
class Manager extends Employee {  
  
    @Override  
    protected void showName() {  
        super.showName();  
        System.out.println("Manager showName()");  
    }  
  
}
```

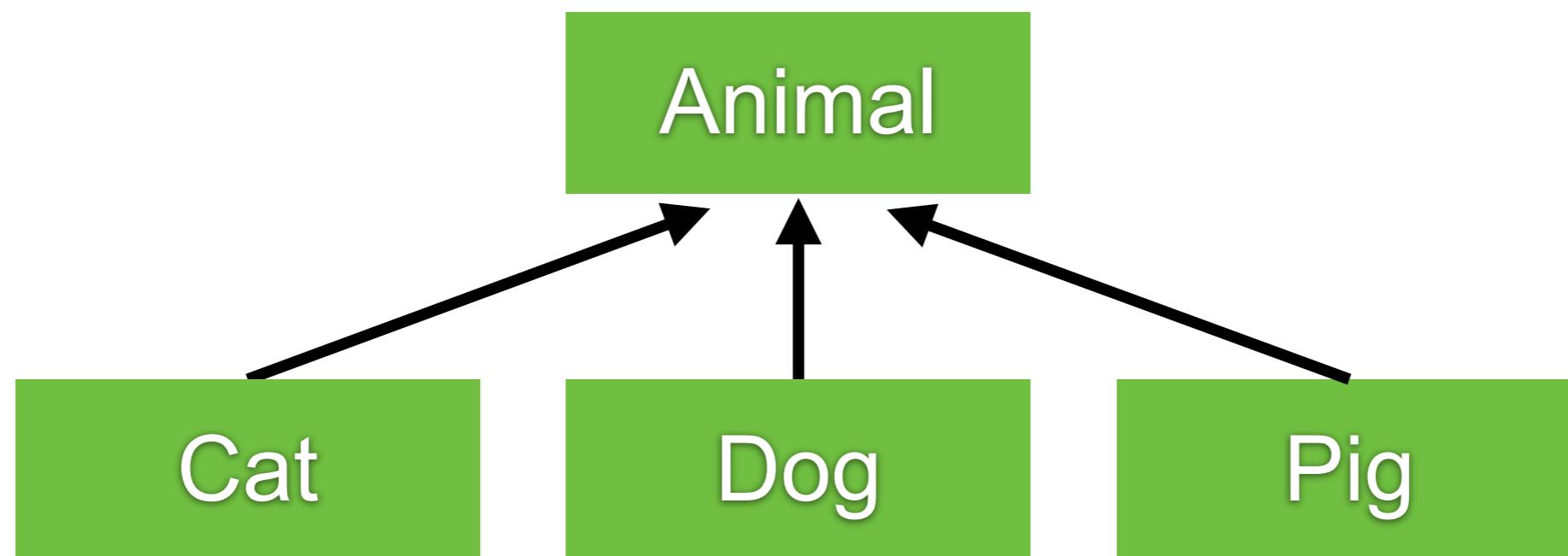


# Visibility

Visibility	Private	Default	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes



# Animal Hierarchical



# Animal class

```
class Animal {  
    public void speak(int i) {  
        System.out.println("Animal "+ i);  
    }  
}
```



# Subclass class

```
class Cat extends Animal {  
    @Override  
    public void speak(int i) {  
        System.out.println("Cat "+ i);  
    }  
}  
  
class Dog extends Animal {  
    @Override  
    public void speak(int i) {  
        System.out.println("Dog "+ i);  
    }  
}  
  
class Pig extends Animal {  
    @Override  
    public void speak(int i) {  
        System.out.println("Pig "+ i);  
    }  
}
```



# Animal in the Zoo

```
public class Zoo {  
  
    public static void main(String[] args) {  
        Cat cat = new Cat();  
        Dog dog = new Dog();  
        Pig pig = new Pig();  
  
        cat.speak(1);  
        dog.speak(2);  
        pig.speak(3);  
    }  
  
}
```



# Animal in the Zoo

```
public class Zoo {  
    public static void main(String[] args) {  
        Cat cat = new Cat();  
        Dog dog = new Dog();  
        Pig pig = new Pig();  
  
        cat.speak(1);  
        dog.speak(2);  
        pig.speak(3);  
    }  
}
```

Problem with Zoo ?



# Problem with Zoo

Contain type-specific in each type

What happen when add new type of Animal ?

We need the better code !!!



# Solution

## Called Polymorphism

```
public class Zoo {  
    public static void main(String[] args) {  
        Cat cat = new Cat();  
        Dog dog = new Dog();  
        Pig pig = new Pig();  
  
        speak(cat, 1);  
        speak(dog, 2);  
        speak(pig, 3);  
    }  
  
    public static void speak(Animal animal, int i) {  
        animal.speak(i);  
    }  
}
```



# OOP with Java Part 2

Abstract classes  
Interfaces



# Abstract classes

Placeholder for a method that fully defined in a subclass

Class have abstract modifier

Method can have abstract modifier

Abstract method can't be private

Abstract method can't have body

*“Class that contains an abstract method is called Abstract class”*



# Classes

*“Class that contains an abstract method is called  
Abstract class”*

*“Class that not contains an abstract method is called  
Concrete class”*



# Abstract classes

```
abstract class Employee {  
    private int id;  
  
    public abstract double getSalary();  
  
    public boolean sameSalary(Employee other) {  
        return this.getSalary() == other.getSalary();  
    }  
}
```



# Pitfall of abstract class

You can't create instances of an abstract class  
Abstract class can only use to derive more classes



# Purpose of abstract class

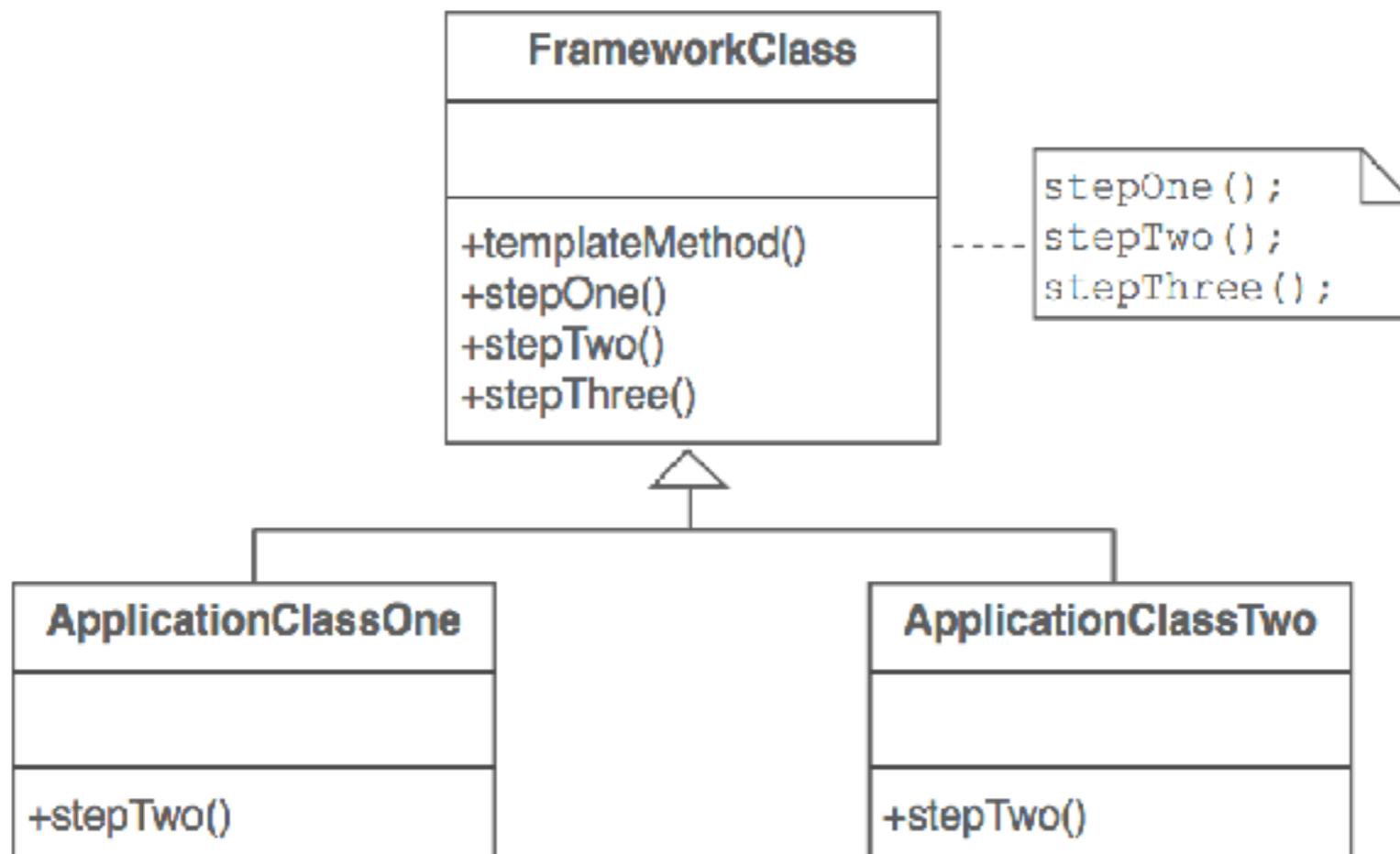
Create a function as base class  
But can be extended by sub class

```
abstract class Report {  
    public abstract void createHeader();  
    public abstract void createBody();  
    public abstract void createFooter();  
  
    public void generate() {  
        createHeader();  
        createBody();  
        createFooter();  
    }  
}
```



# Purpose of abstract class

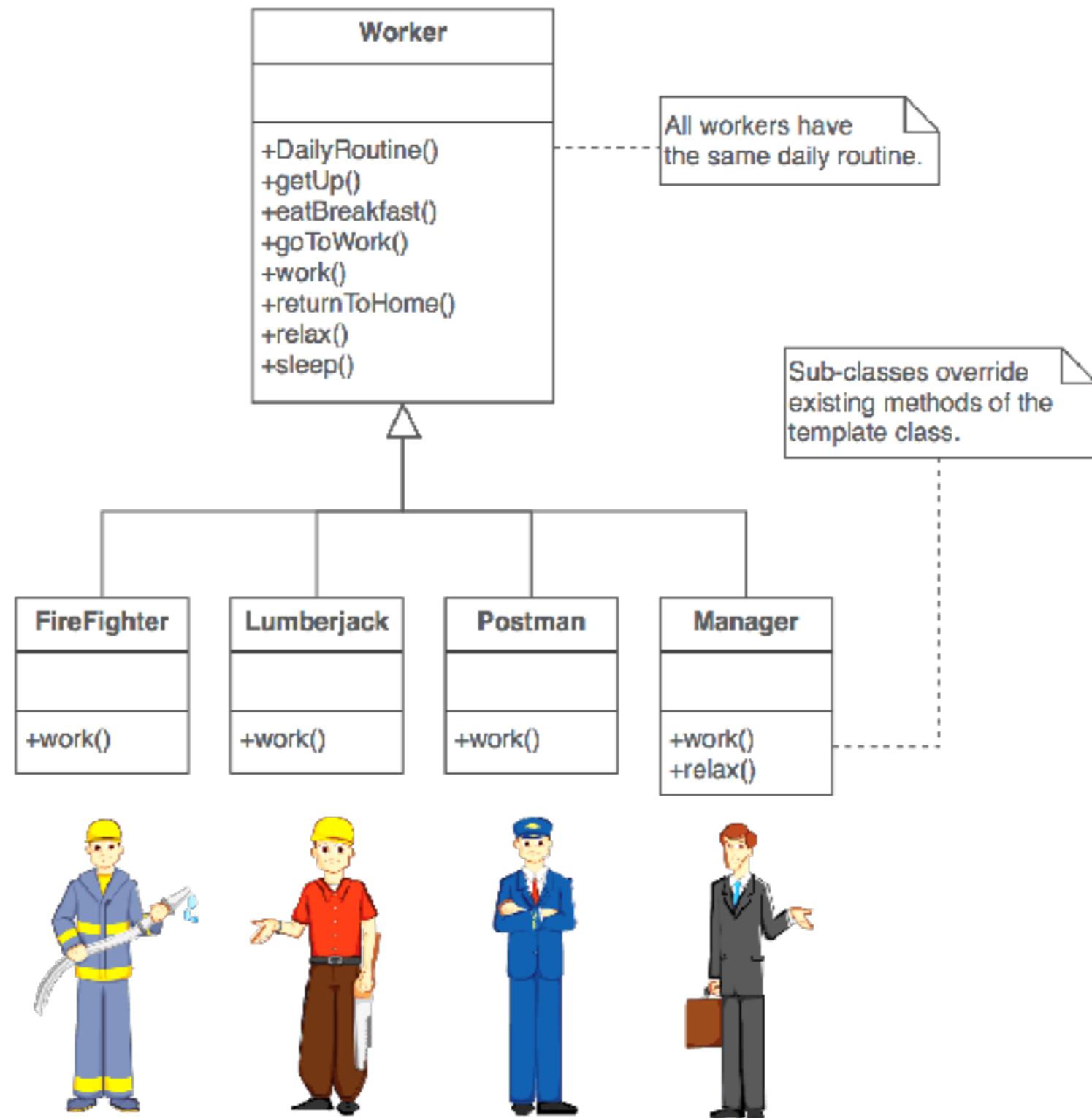
Template method design pattern



[https://sourcemaking.com/design\\_patterns/template\\_method](https://sourcemaking.com/design_patterns/template_method)



# Template method design pattern



# Interfaces

Like an extreme abstract class

All of methods in an interface are abstract by default

Not a class

No instance variables

```
interface Animal {  
    void eat();  
    void sleep();  
}
```



# Using interface from class (1)

```
class Cat implements Animal {
```

```
    @Override  
    public void eat() {  
    }
```

```
    @Override  
    public void sleep() {  
    }
```

```
}
```



# Using interface from class (2)

```
class Cat implements Animal {  
  
    @Override  
    public void eat() {  
    }  
  
    @Override  
    public void sleep() {  
    }  
}
```

*Need to implements all method from interface*



# Multiple implements interfaces

Class can be implement multiple interfaces

```
interface Animal {  
    void eat();  
    void sleep();  
}
```

```
interface Animal2018 {  
    void die();  
}
```



# Multiple implements interfaces

```
class BlackPanther implements Animal, Animal2018 {  
  
    @Override  
    public void die() {  
    }  
  
    @Override  
    public void eat() {  
    }  
  
    @Override  
    public void sleep() {  
    }  
  
}
```



# Extending an interface

A new interface can add method definitions to existing interface by **extending** the old

```
interface Animal {  
    void eat();  
    void sleep();  
}
```

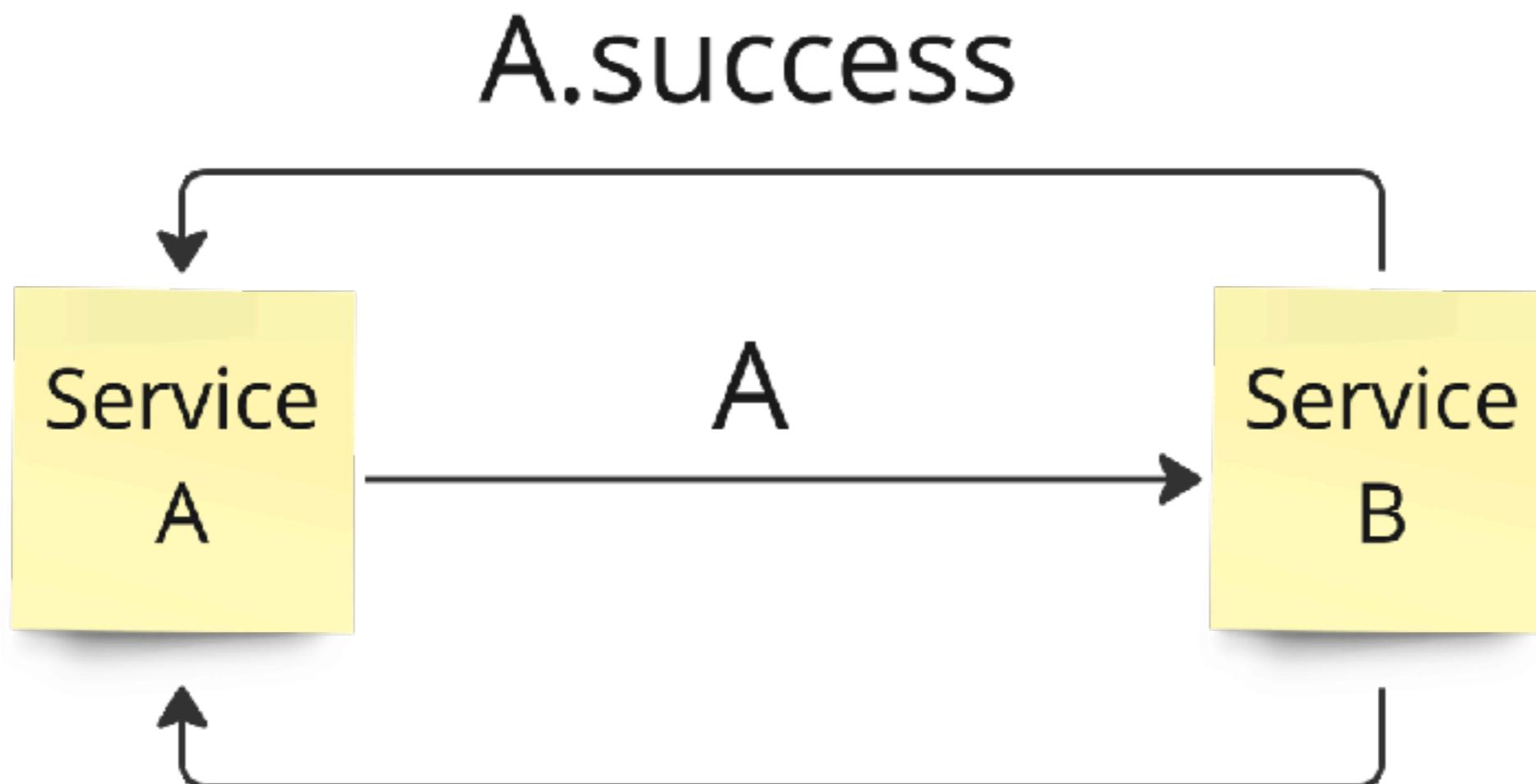
```
interface Animal2018 extends Animal {  
    void die();  
}
```



# Callback with Interface



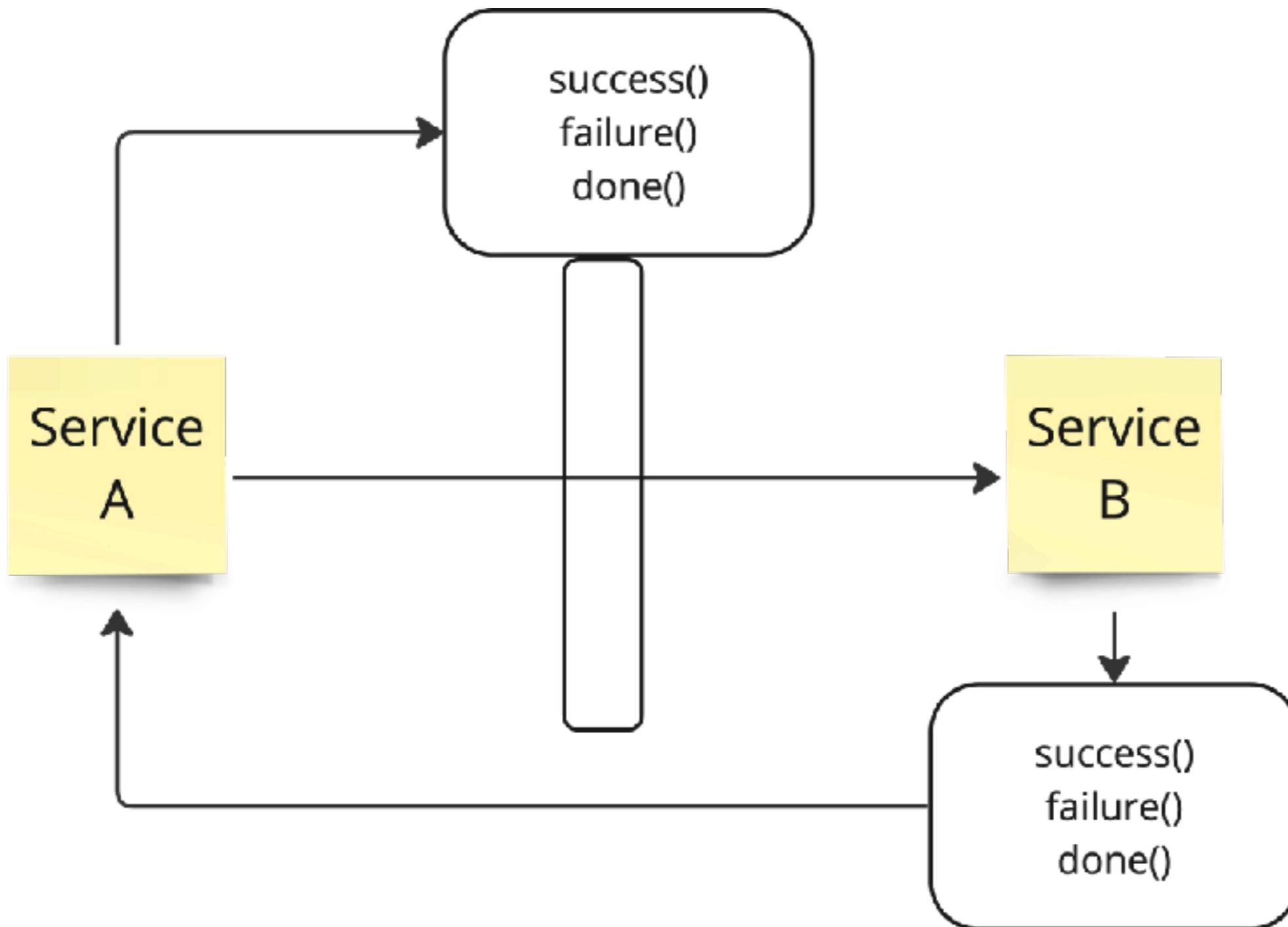
# Call back with class



A.failure



# Call back with interface

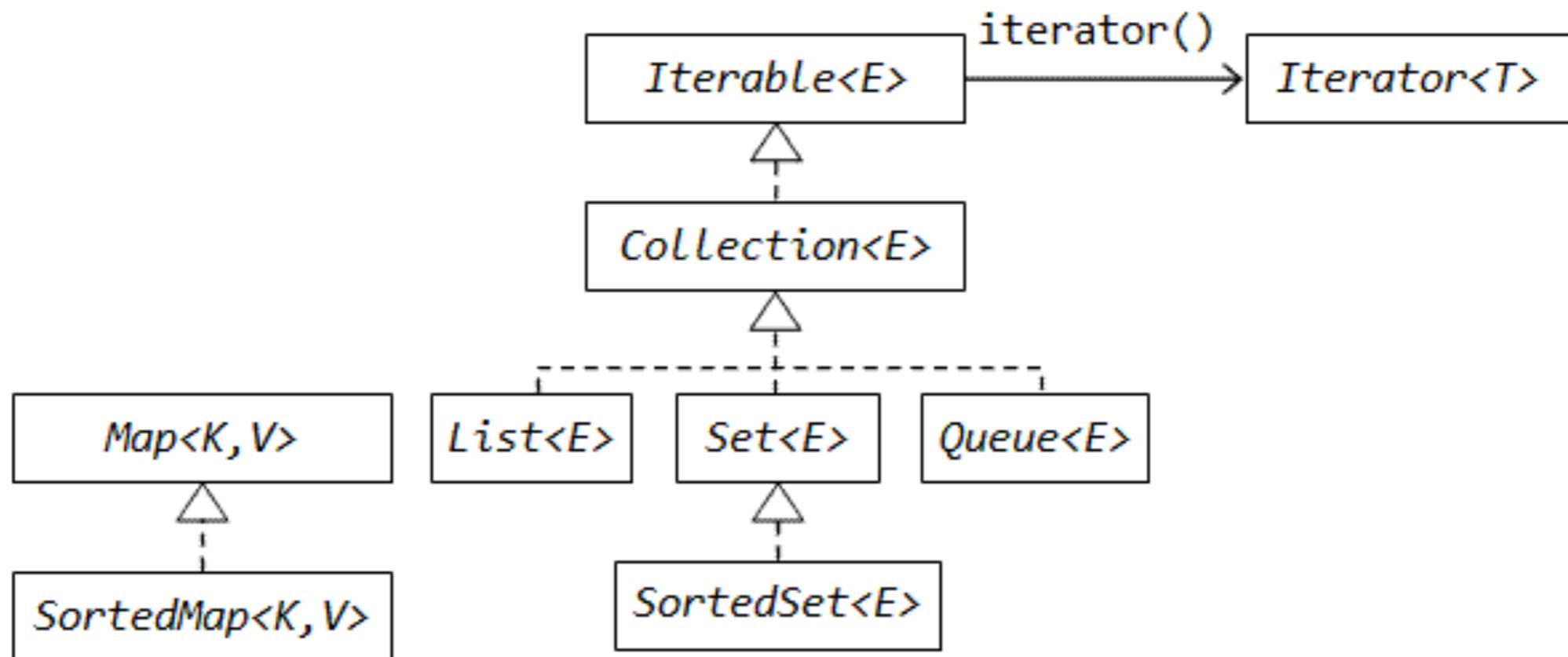


# Java Collection Framework

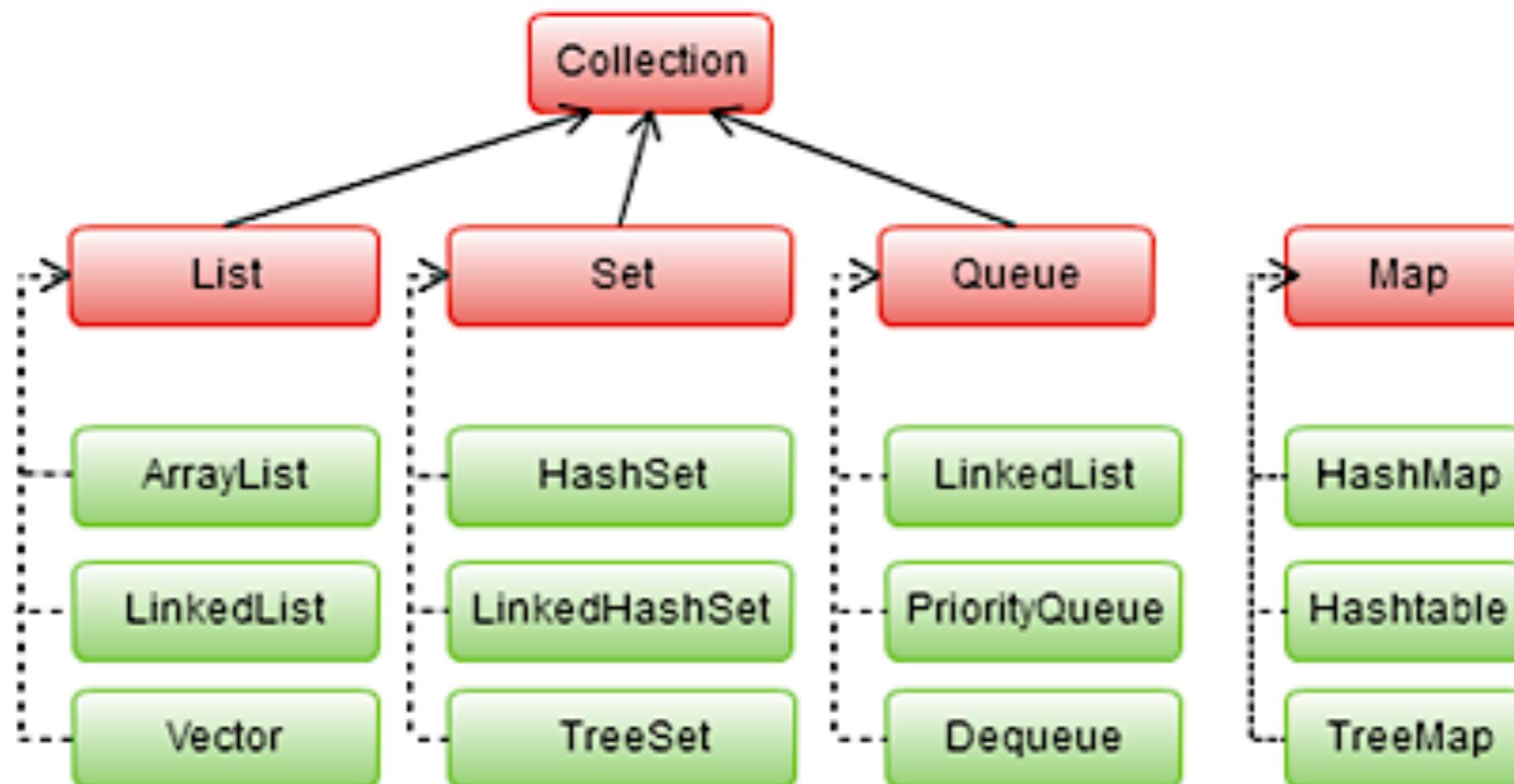
<https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html>



# Java Collection Framework



# Java Collection Framework



## Notable Java collections libraries

### Fastutil

<http://fastutil.dl.unimi.it/>

Fast & compact type-specific collections for Java  
Great default choice for collections of primitive types, like int or long. Also handles big collections with more than  $2^{31}$  elements well.

### Guava

<https://github.com/google/guava>

Google Core Libraries for Java 6+  
Perhaps the default collection library for Java projects. Contains a magnitude of convenient methods for creating collection, like fluent builders, as well as advanced collection types.

### Eclipse Collections

<https://www.eclipse.org/collections/>

Features you want with the collections you need  
Previously known as gs-collections, this library includes almost any collection you might need: primitive type collections, multimap, bidirectional maps and so on.

### JCTools

<https://github.com/JCTools/JCTools>

Java Concurrency Tools for the JVM.  
If you work on high throughput concurrent applications and need a way to increase your performance, check out JCTools.

## What can your collection do for you?

Collection class	Thread-safe alternative	Your data				Operations on your collections					
		Individual elements	Key-value pairs	Duplicate element support	Primitive support	FIFO	Sorted	LIFO	Perform 'contains' check	By key	By value
HashMap	ConcurrentHashMap	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗
HashBiMap (Guava)	Maps.synchronizedBiMap (new HashBiMap())	✗	✓	✗	✗	✗	✗	✗	✓	✓	✓
ArrayListMultimap (Guava)	Maps.synchronizedMultiMap (new ArrayListMultimap())	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗
LinkedHashMap	Collections.synchronizedMap (new LinkedHashMap())	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗
TreeMap	ConcurrentSkipListMap	✗	✓	✗	✗	✗	✓	✗	✓*	✓*	✗
Int2IntMap (Fastutil)		✗	✓	✗	✓	✗	✗	✗	✓	✓	✓
ArrayList	CopyOnWriteArrayList	✓	✗	✓	✗	✓	✗	✓	✗	✗	✓
HashSet	Collections.newSetFromMap (new ConcurrentHashMap<>())	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓
IntArrayList (Fastutil)		✓	✗	✓	✓	✓	✓	✓	✓	✗	✓
PriorityQueue	PriorityBlockingQueue	✓	✗	✓	✗	✗	✓**	✗	✗	✗	✗
ArrayDeque	ArrayBlockingQueue	✓	✗	✓	✗	✓**	✗	✓**	✗	✗	✗

\*  $O(\log n)$  complexity, while all others are  $O(1)$  - constant time

\*\* when using Queue interface methods: offer() / poll()

## How fast are your collections?

Collection class	Random access by index / key	Search / Contains	Insert
ArrayList	$O(1)$	$O(n)$	$O(n)$
HashSet	$O(1)$	$O(1)$	$O(1)$
HashMap	$O(1)$	$O(1)$	$O(1)$
TreeMap	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$

Remember, not all operations are equally fast. Here's a reminder of how to treat the Big-O complexity notation:

$O(1)$  - constant time, really fast, doesn't depend on the size of your collection

$O(\log(n))$  - pretty fast, your collection size has to be extreme to notice a performance impact

$O(n)$  - linear to your collection size: the larger your collection is, the slower your operations will be

BROUGHT TO YOU BY  
**JRebel**

[http://files.zeroturnaround.com/pdf/zt\\_java\\_collections\\_cheat\\_sheet.pdf](http://files.zeroturnaround.com/pdf/zt_java_collections_cheat_sheet.pdf)



# Types of Error in Java

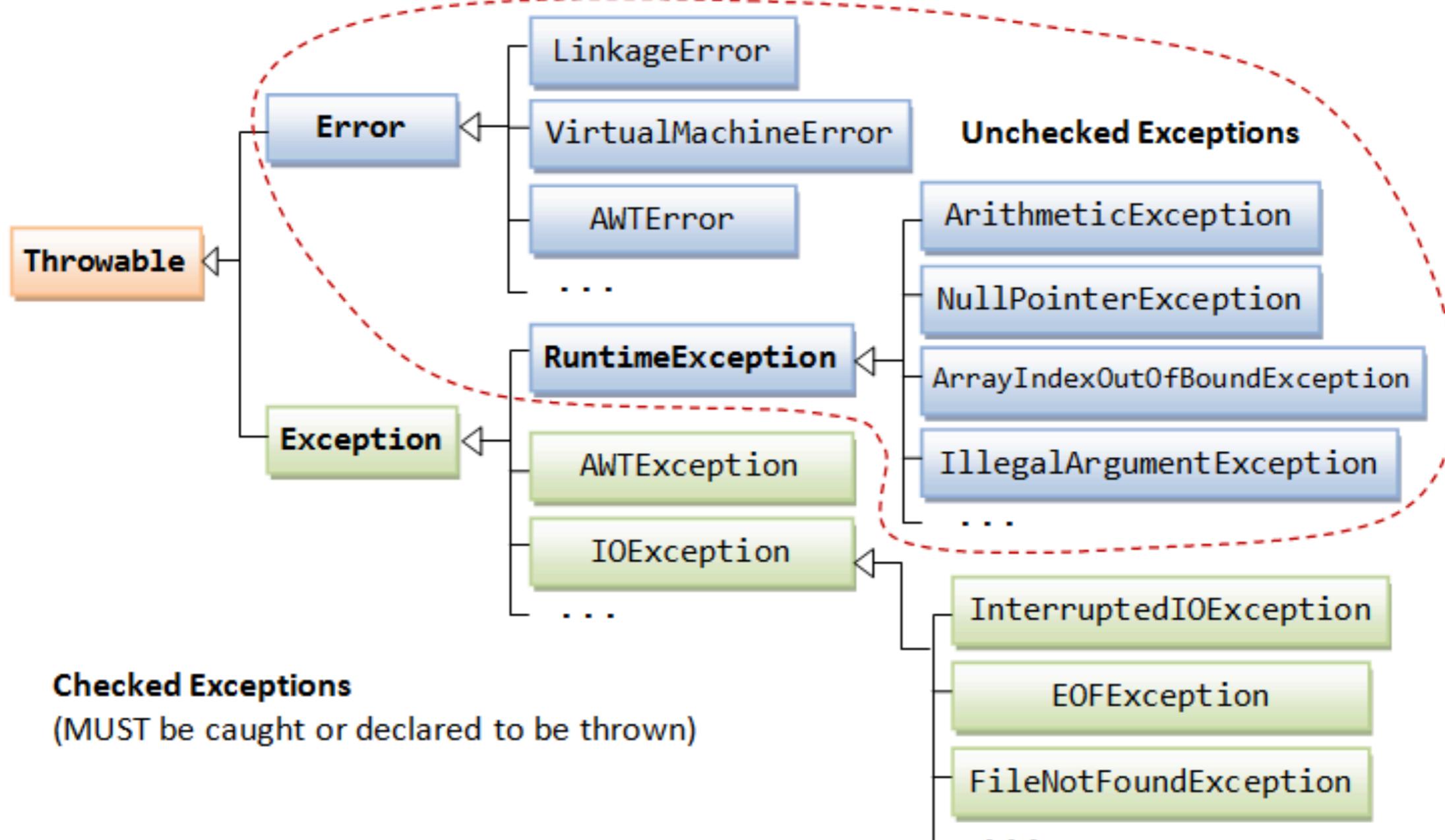


# Types of Errors

- Compile-time errors
- Runtime errors
- Logical errors



# Exception classes



# Lambda expression



# Java 8 Best Practices Cheat Sheet



BROUGHT TO YOU BY



## Default methods

Evolve interfaces & create traits

```
// Default methods in interfaces
@FunctionalInterface
interface Utilities {
    default Consumer<Runnable> m() {
        return (r) -> r.run();
    }

    // default methods, still functional
    Object function(Object o);
}

class A implements Utilities { // implement
    public Object function(Object o) {
        return new Object();
    }

    // call a default method
    Consumer<Runnable> n = new A().m();
}
}
```

Traits: 1 default method per interface  
Don't enhance functional interfaces  
Only conservative implementations

## Lambdas

Syntax:

```
(parameters) -> expression
(parameters) -> { statements; }
```

```
// takes a Long, returns a String
Function<Long, String> f = (l) -> l.toString();

// takes nothing gives you Threads
Supplier<Thread> s = Thread::currentThread;

// takes a string as the parameter
Consumer<String> c = System.out::println;

// use them with streams
new ArrayList<String>().stream();

// peek: debug streams without changes
peek(e -> System.out.println(e));

// map: convert every element into something
map(e -> e.hashCode());

// filter: pass some elements through
filter(e -> ((e.hashCode() % 2) == 0));

// collect all values from the stream
collect(Collectors.toCollection(TreeSet::new))
```

## java.util. Optional

A container for possible null values

```
// Create an optional
Optional<String> optional =
Optional.ofNullable(a);

// process the optional
optional.map(s -> "RebellLabs:" + s);

// map a function that returns Optional
optional.flatMap(s -> Optional.ofNullable(s));

// run if the value is there
optional.ifPresent(System.out::println);

// get the value or throw an exception
optional.get();

// return the value or the given value
optional.orElse("Hello world!");

// return empty Optional if not satisfied
optional.filter(s -> s.startsWith("RebellLabs"));
```

## Rules of Thumb

Expressions over statements  
Refactor to use method references  
Chain lambdas rather than growing them

Fields - use plain objects  
Method parameters, use plain objects  
Return values - use Optional  
Use `orElse()` instead of `get()`

[http://files.zeroturnaround.com/pdf/zt\\_java8\\_best\\_practices.pdf](http://files.zeroturnaround.com/pdf/zt_java8_best_practices.pdf)



# Lambda expression

Functionality as a method arguments

Code as data

Help code more compact



# Demo with Sorting data

Sorting data in List with Comparator

Employee id=1 name=First

Employee id=2 name=Second

Employee id=3 name=Third



# Create class Employee

```
class Employee {  
    private int id;  
    private String name;  
    private double salary;  
  
    //Setter and Getter methods
```



# Sorting with java.util.Comparator

```
Comparator<Employee> sortByName = new Comparator<Employee>() {  
    @Override  
    public int compare(Employee e1, Employee e2) {  
        return e1.getName().compareTo(e2.getName());  
    }  
};
```



# Using with Collections.sort()

```
List<Employee> list = new ArrayList<>();
list.add(new Employee(500, "First", 150000));
list.add(new Employee(504, "Second", 120000));
list.add(new Employee(503, "Third", 100000));
list.add(new Employee(730, "Forth", 45000));

Collections.sort(list, sortByName);
list.forEach(System.out::println);
```



# Run and see output

```
demo.java8.lambda.Employee@e9e54c2  
demo.java8.lambda.Employee@65ab7765  
demo.java8.lambda.Employee@1b28cdfa  
demo.java8.lambda.Employee@eed1f14
```

*Try to improve format of output !!  
more readable*



# Override `toString()`

```
class Employee {  
    private int id;  
    private String name;  
    private double salary;  
  
    @Override  
    public String toString() {  
        return "Employee [id=" + id + ", name=" + name + ", salary=" +  
salary + "]";  
    }  
}
```

```
Employee [id=500, name=First, salary=150000.0]  
Employee [id=504, name=Second, salary=120000.0]  
Employee [id=503, name=Third, salary=100000.0]  
Employee [id=730, name=Forth, salary=45000.0]
```



# Sorting with Lambda expression

```
Comparator<Employee> sortByNameWithLambda =  
    (Employee e1, Employee e2) ->  
        e1.getName().compareTo(e2.getName());  
  
Collections.sort(list, sortByName);  
list.forEach(System.out::println);
```



# Workshop

Sort by name (Z to A)

Sort by ID (0 to 9)



# Java Stream API



# Java Stream API

A new package **java.util.stream**  
New functionality which contains classes for  
processing sequences of elements



# Java Stream API

## Working with List and Array

```
String[] datas = new String[] {"P", "U", "I"};
Stream<String> streams = Arrays.stream(datas);
streams.forEach(System.out::println);
```

```
Stream<String> streams2 = Stream.of("P", "U", "I");
streams2.forEach(System.out::println);
```



# Java Stream API

## Sequential and Parallel

```
List<String> myList = Arrays.asList("P", "U", "I");
```

```
myList.stream().forEach(System.out::println);
```

```
myList.parallelStream().forEach(System.out::println);
```



# Operations

Iterating  
Filtering  
Mapping  
Matching  
Collecting



# Iterating

```
myList.stream().forEach(System.out::println);
```



# Filtering

```
myList.stream()  
    .filter(element -> element.contains("P"))  
    .forEach(System.out::println);
```



# Matching

Try to validate all elements of sequence

```
boolean isValid = myList.stream()  
    .anyMatch(element -> element.contains("P"));  
  
out.println(isValid);
```



# Mapping

Try to convert elements by apply a special function

```
List<String> results =  
    myList.stream().map(element -> element.toLowerCase())  
        .collect(Collectors.toList());  
  
out.println(results);
```



# Collecting

Try to convert stream to collection or map

```
List<String> results =  
    myList.stream().map(element -> element.toLowerCase())  
        .collect(Collectors.toList());  
  
out.println(results);
```



# Workshop

United States of America => **USA**

united states of america => **USA**

Light Amplification by Stimulation of Emitted Radiation => **LASER**

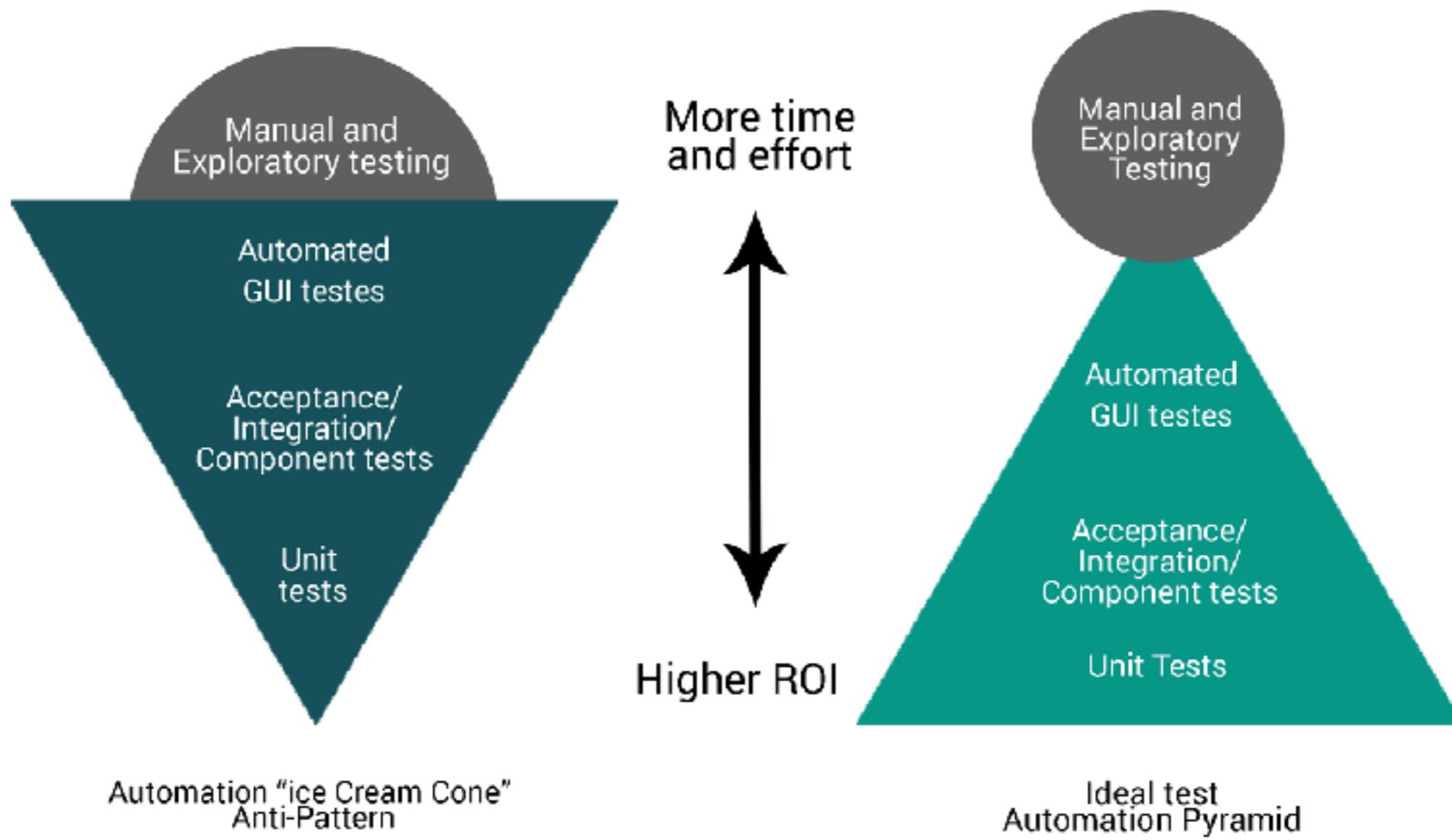
Jordan Of the World => **JTW**



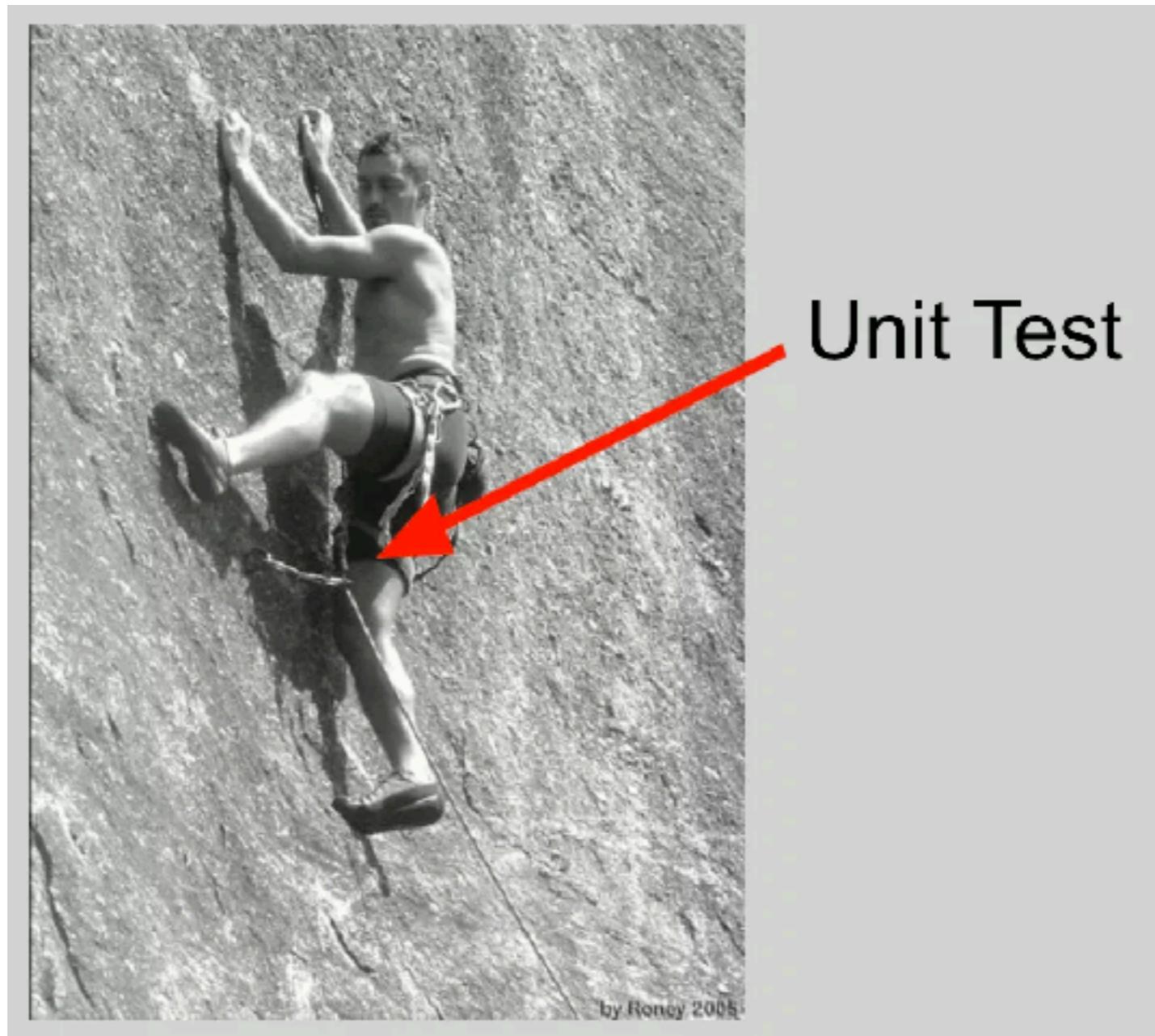
# Automated testing with Java



# Automation test



# Unit test



<https://less.works/less/technical-excellence/unit-testing.html>



# Purpose of Unit test

Facilitates changes  
Simplified integration  
Documentation  
Design tool

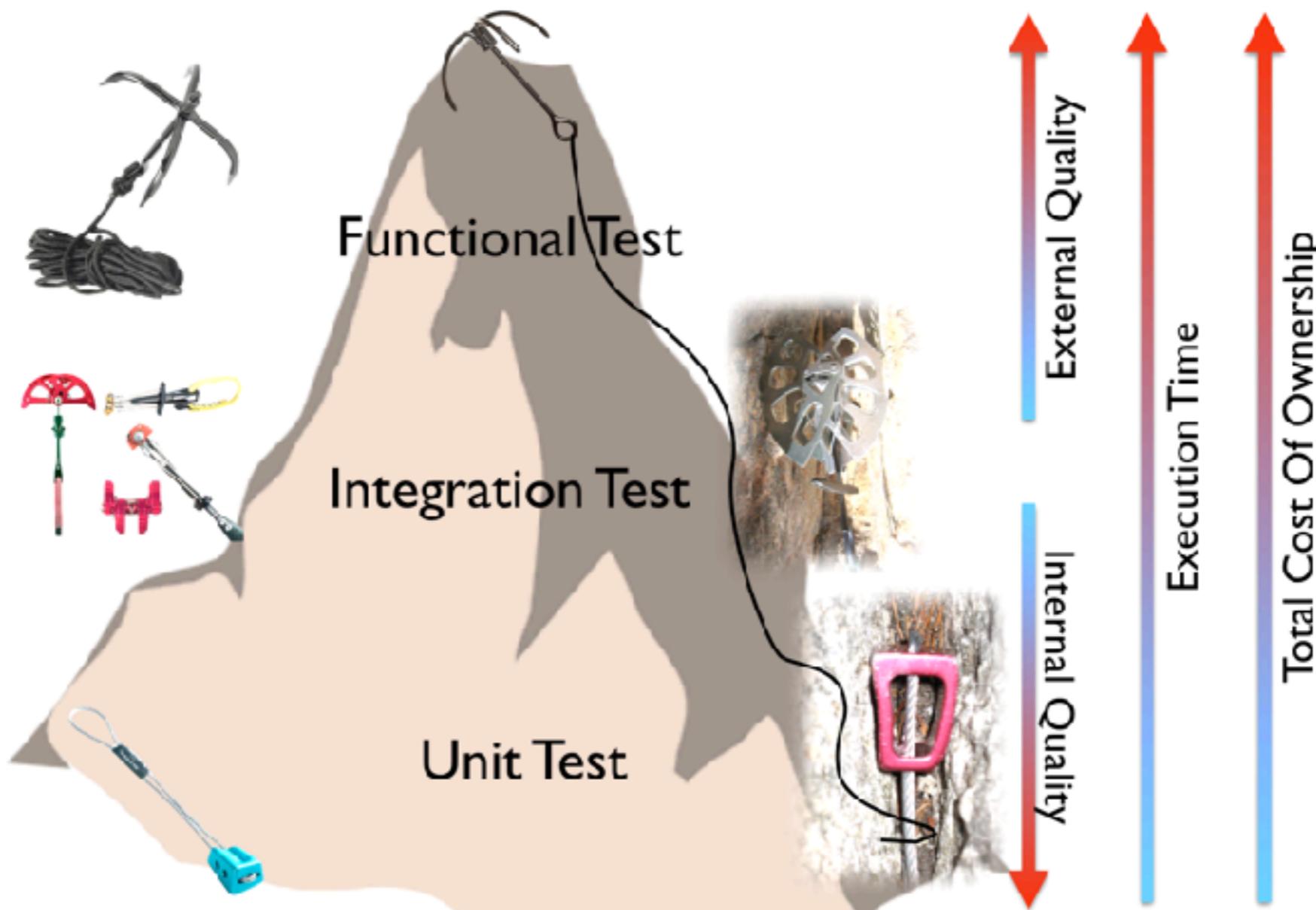


# Why Unit test ?

Total cost of ownership  
Internal vs External quality  
Quality of feedback



# Quality of feedback



<https://less.works/less/technical-excellence/unit-testing.html>



# Golden rule of Unit test

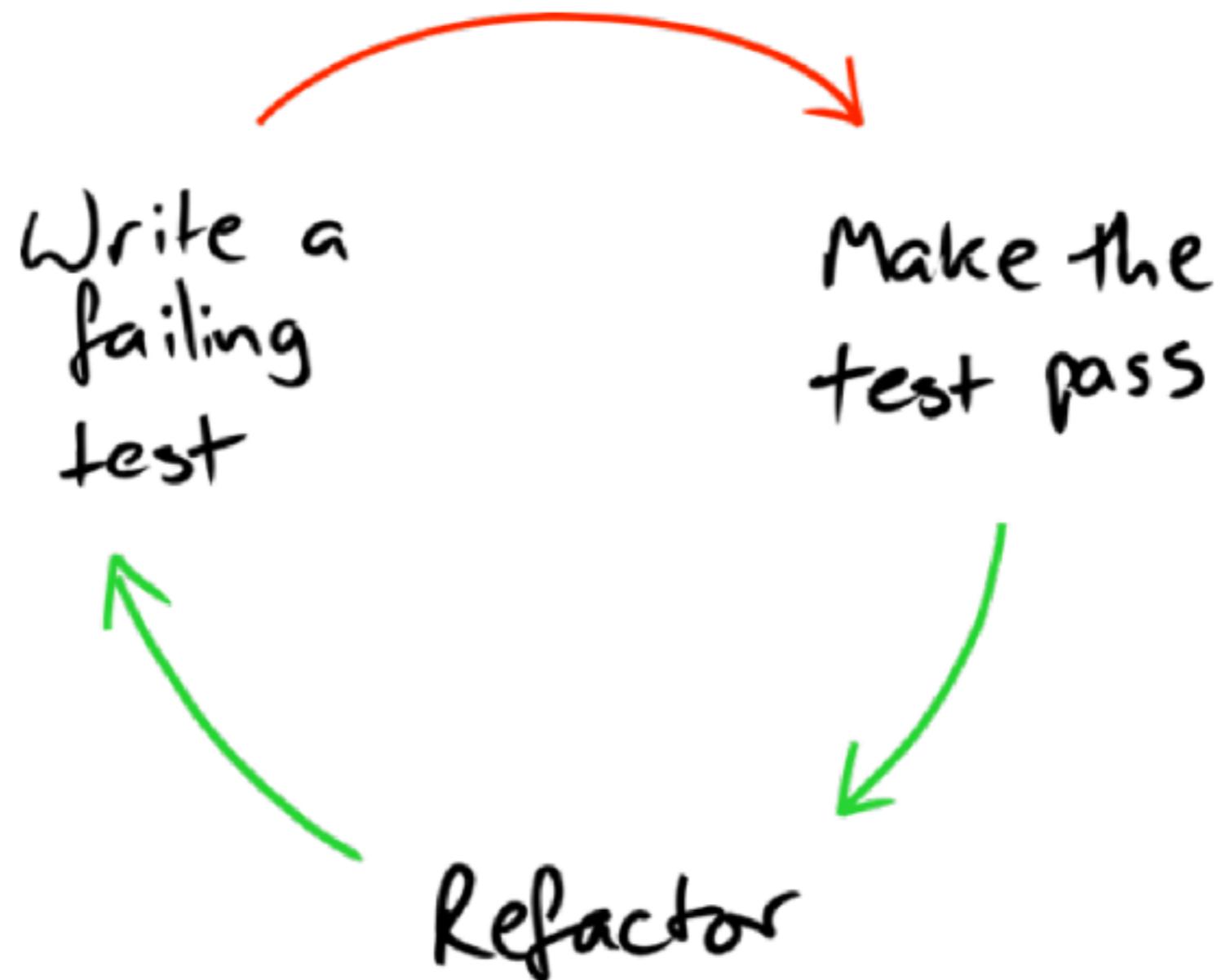
“Each unit test case  
should be **very limited** in scope”



# Test-Driven Development (TDD)

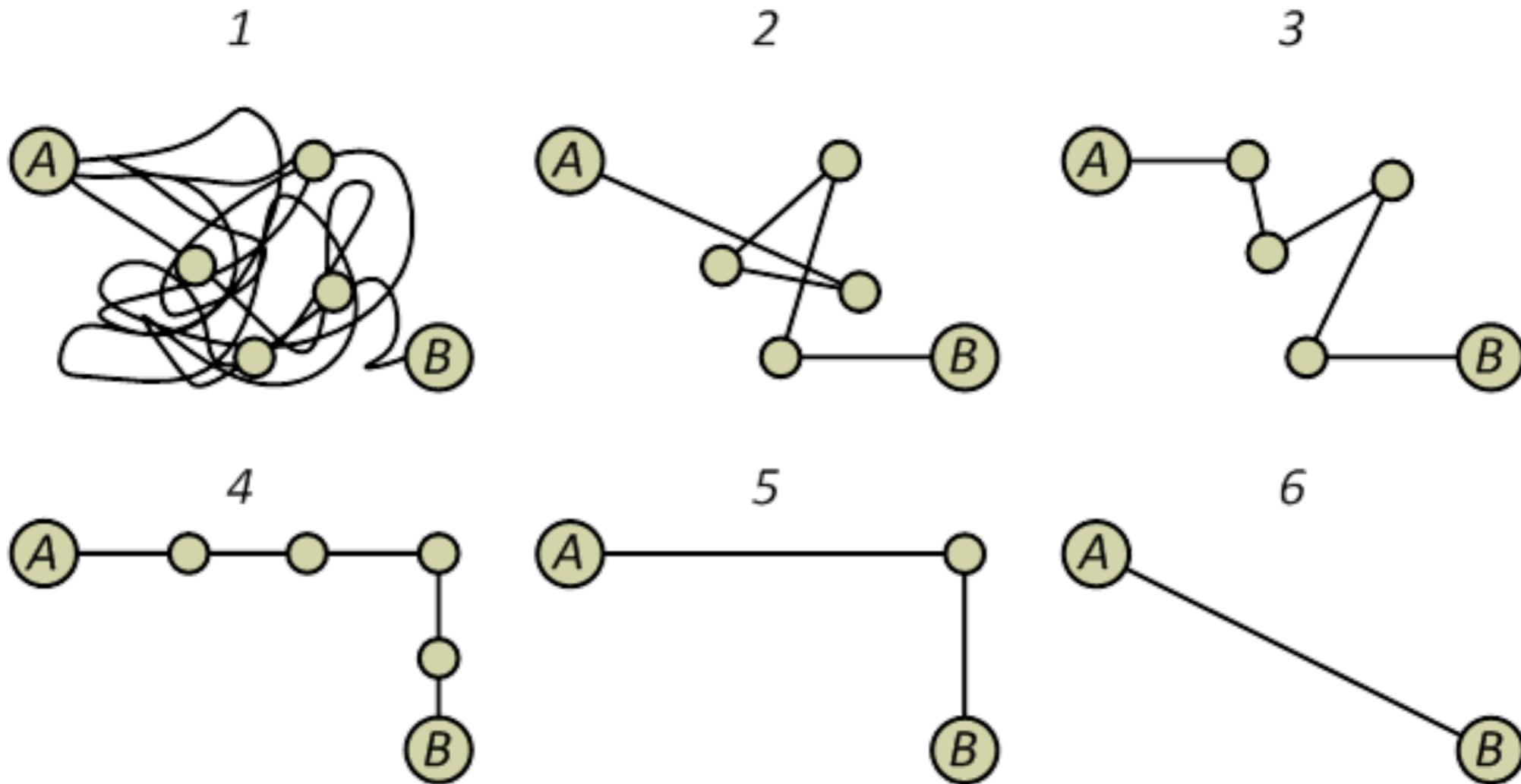


# TDD Cycle



# Refactor

Change the structure of code without change the behaviour of code

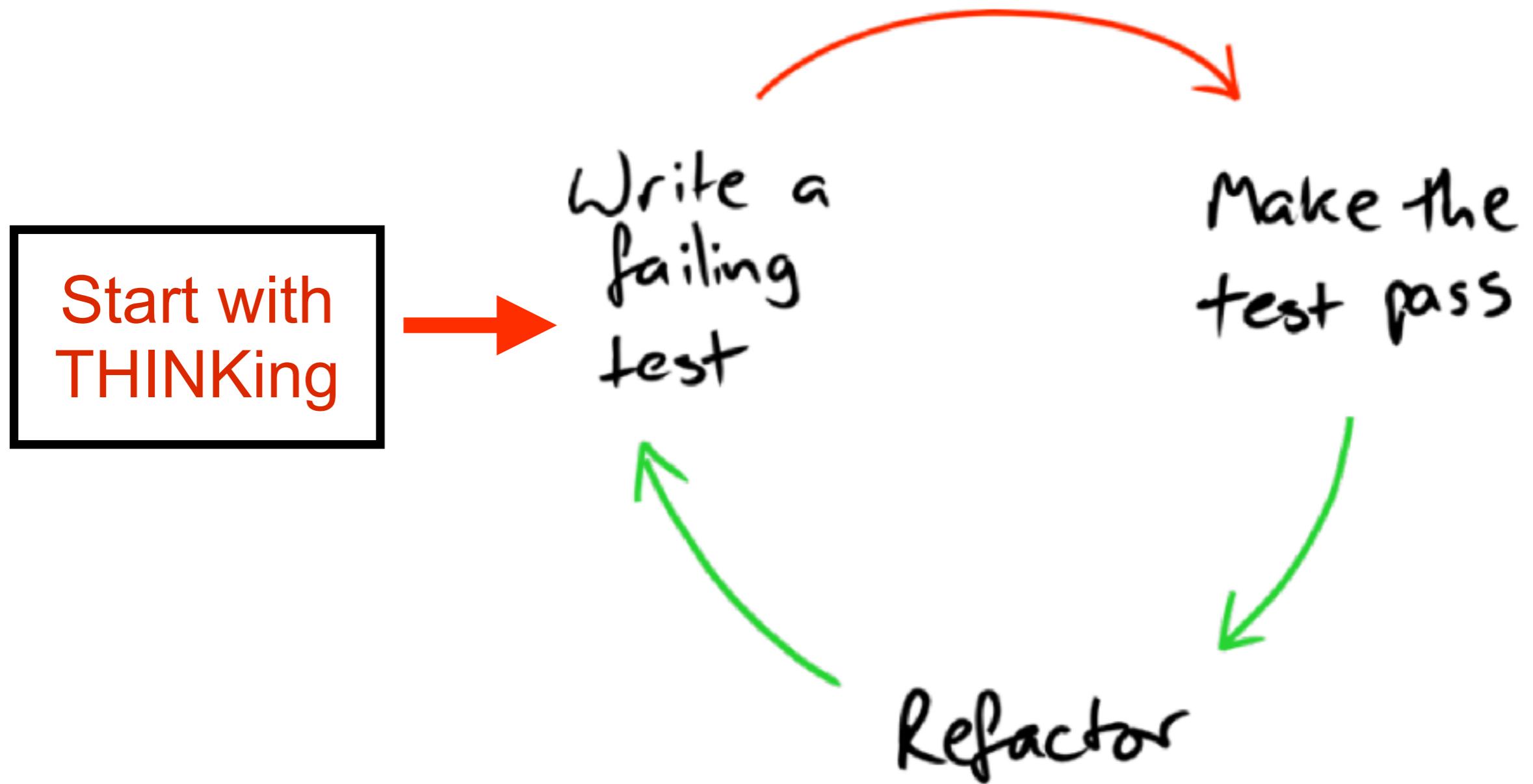


# Simple Rule of TDD

1. Write a failing test before write any code
2. Remove duplication code



# TDD Cycle



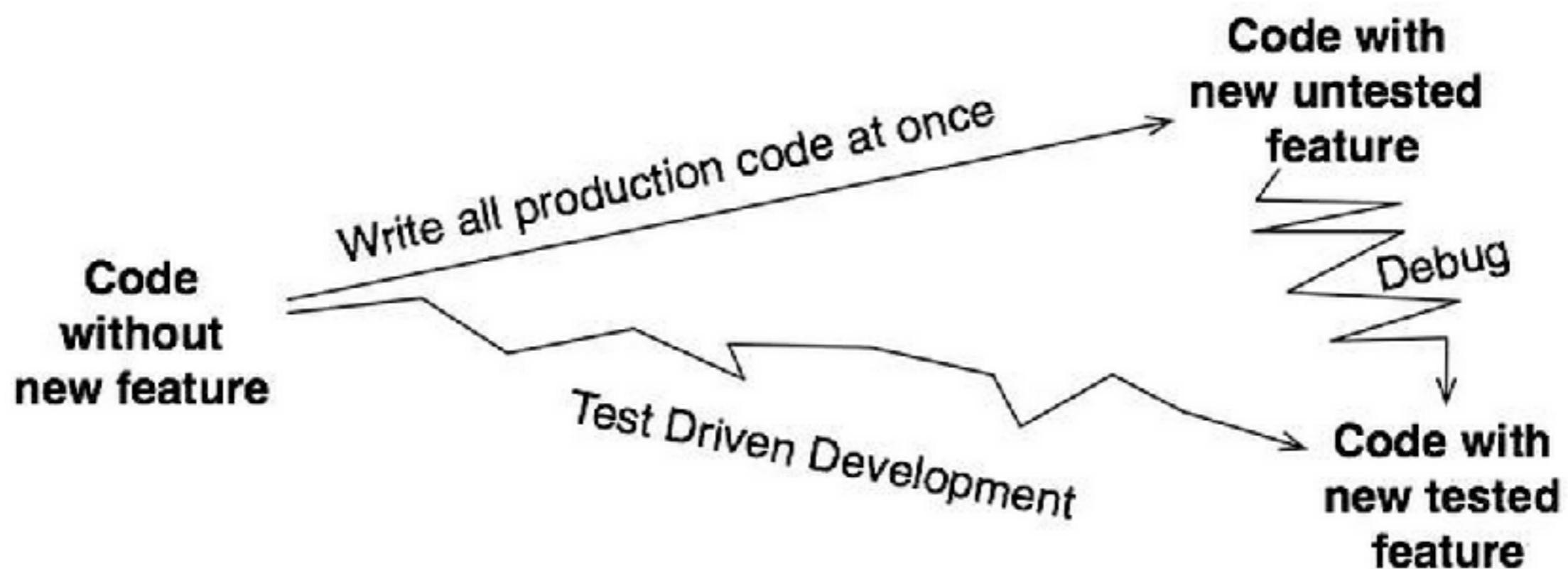
# TDD cycle should be

Short  
Rhythmic  
Incremental  
Design-focus  
Discipline



# TDD with DLP

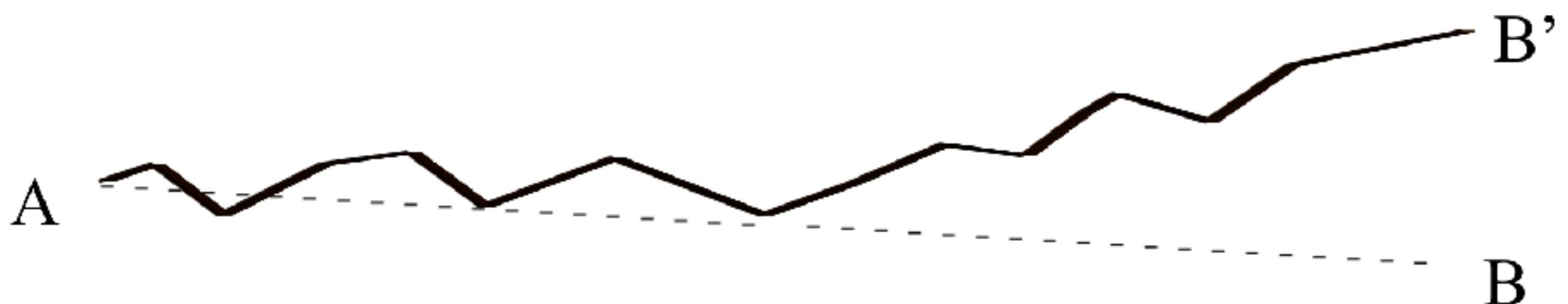
## Debug Later Programming



<https://www.wingman-sw.com/>



# Let's start with Small Step



# Good Unit Test

**F**ast  
**I**solation  
**R**epeatable  
**S**elf-validate  
**T**imely



# **Let's start to write Unit tests with Java**

# **JUnit**



**BUT we need to separate the  
Test code from Production  
code !!**



# We need new Project Structure



<https://maven.apache.org/>



# POM file (1)

Project Object Model  
Represent in XML format  
Filename is **pom.xml**



# POM file (2)

```
1<project xmlns="http://maven.apache.org/
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>demo1</groupId>
4   <artifactId>demo1</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6 </project>
```



# POM file (2)

Add new dependency => junit

```
1<project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instantiation"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     pom-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>demo1</groupId>
7   <artifactId>demo1</artifactId>
8   <version>0.0.1-SNAPSHOT</version>
9
10  <dependencies>
11    <dependency>
12      <groupId>junit</groupId>
13      <artifactId>junit</artifactId>
14      <version>4.12</version>
15      <scope>test</scope>
16    </dependency>
17  </dependencies>
18</project>
```

<https://github.com/junit-team/junit4/wiki/use-with-maven>



# Write first unit test case



The ratio of time spent in code  
**Read vs Write is +10:1**



# Run your test in command line

\$mvn clean test

\$mvn clean package



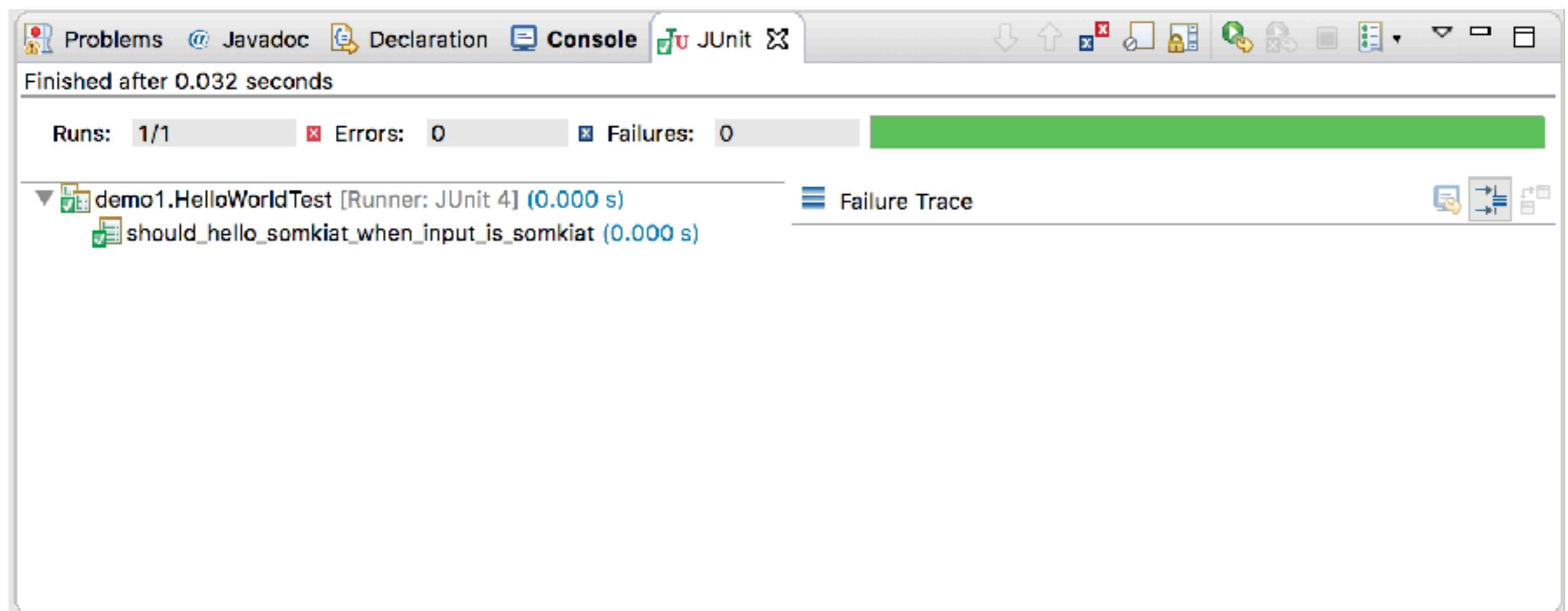
# Make your test pass

```
public class Reception {  
    public String hello(String name) {  
        return "Hello " + name;  
    }  
}
```



# Run test again

Result should be pass (Green)

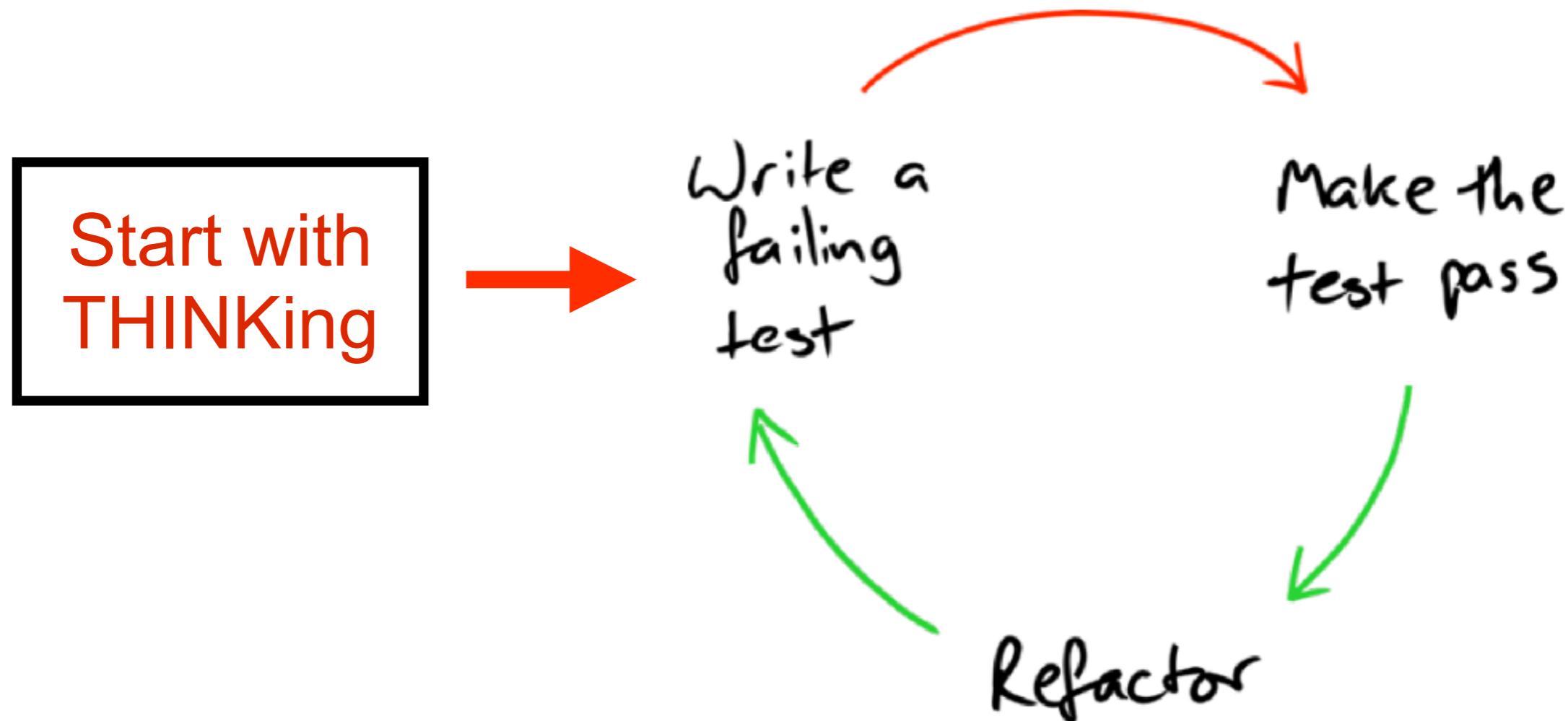


# **Write next test case ...**



# Simple Rule of TDD (again)

1. Write a failing test before write any code
2. Remove duplication code

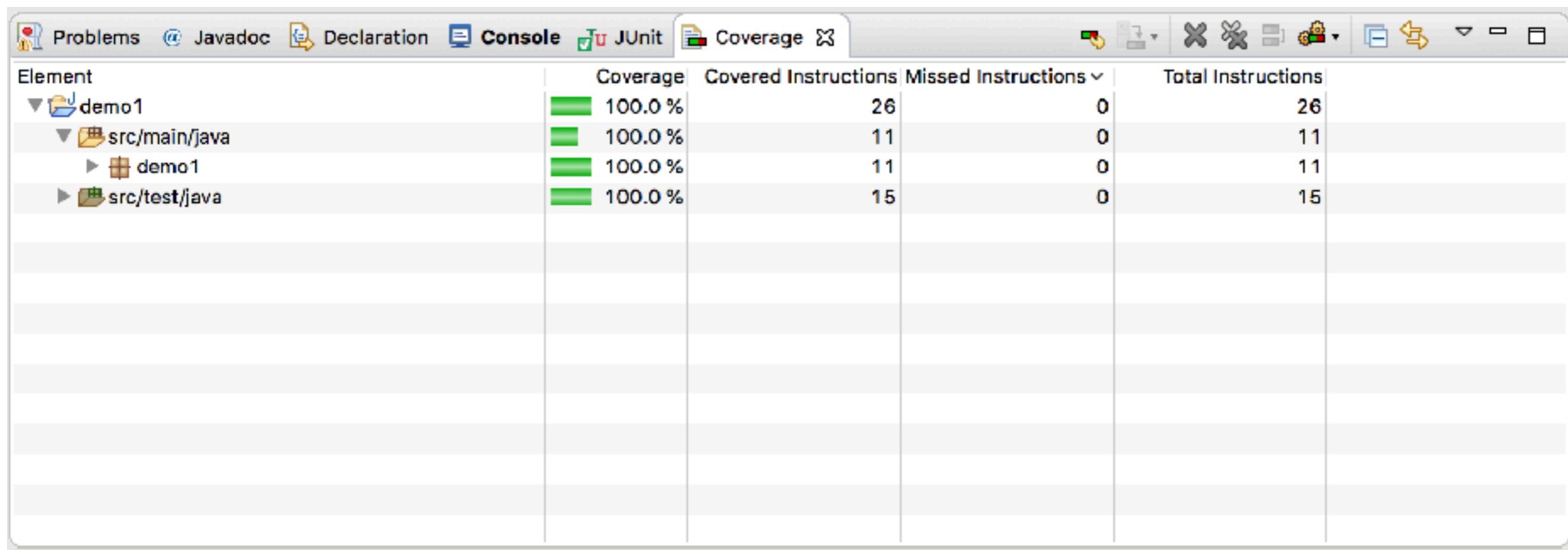


# Code Coverage



# Show result of code coverage

## Summary result report



The screenshot shows a software interface with a toolbar at the top containing icons for Problems, Javadoc, Declaration, Console, JUnit, and Coverage. The Coverage tab is selected. Below the toolbar is a table titled 'Element' with columns for Coverage, Covered Instructions, Missed Instructions, and Total Instructions. The table lists four items under the 'demo1' project: 'src/main/java/demo1' with 100.0% coverage, 26 covered instructions, 0 missed instructions, and 26 total instructions; 'src/main/java/demo1' with 100.0% coverage, 11 covered instructions, 0 missed instructions, and 11 total instructions; 'src/main/java/demo1' with 100.0% coverage, 11 covered instructions, 0 missed instructions, and 11 total instructions; and 'src/test/java' with 100.0% coverage, 15 covered instructions, 0 missed instructions, and 15 total instructions.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
demo1	100.0 %	26	0	26
src/main/java	100.0 %	11	0	11
demo1	100.0 %	11	0	11
src/test/java	100.0 %	15	0	15



# Show result of code coverage

See result in each file/class

```
1 package demo1;  
2  
3 public class Reception {  
4  
5     public String hello(String name) {  
6         return "Hello " + name;  
7     }  
8  
9 }
```



# Workshop



# Range of number

Create test case = **RangeTest**

Input	Expected output
[1,5]	1,2,3,4,5
[1,5)	1,2,3,4
(1,5]	2,3,4,5
(1,5)	2,3,4

**Thinking before coding**



# Code Smell & Refactoring

[https://en.wikipedia.org/wiki/Code\\_smell](https://en.wikipedia.org/wiki/Code_smell)



# Code Smell



# Code Smell



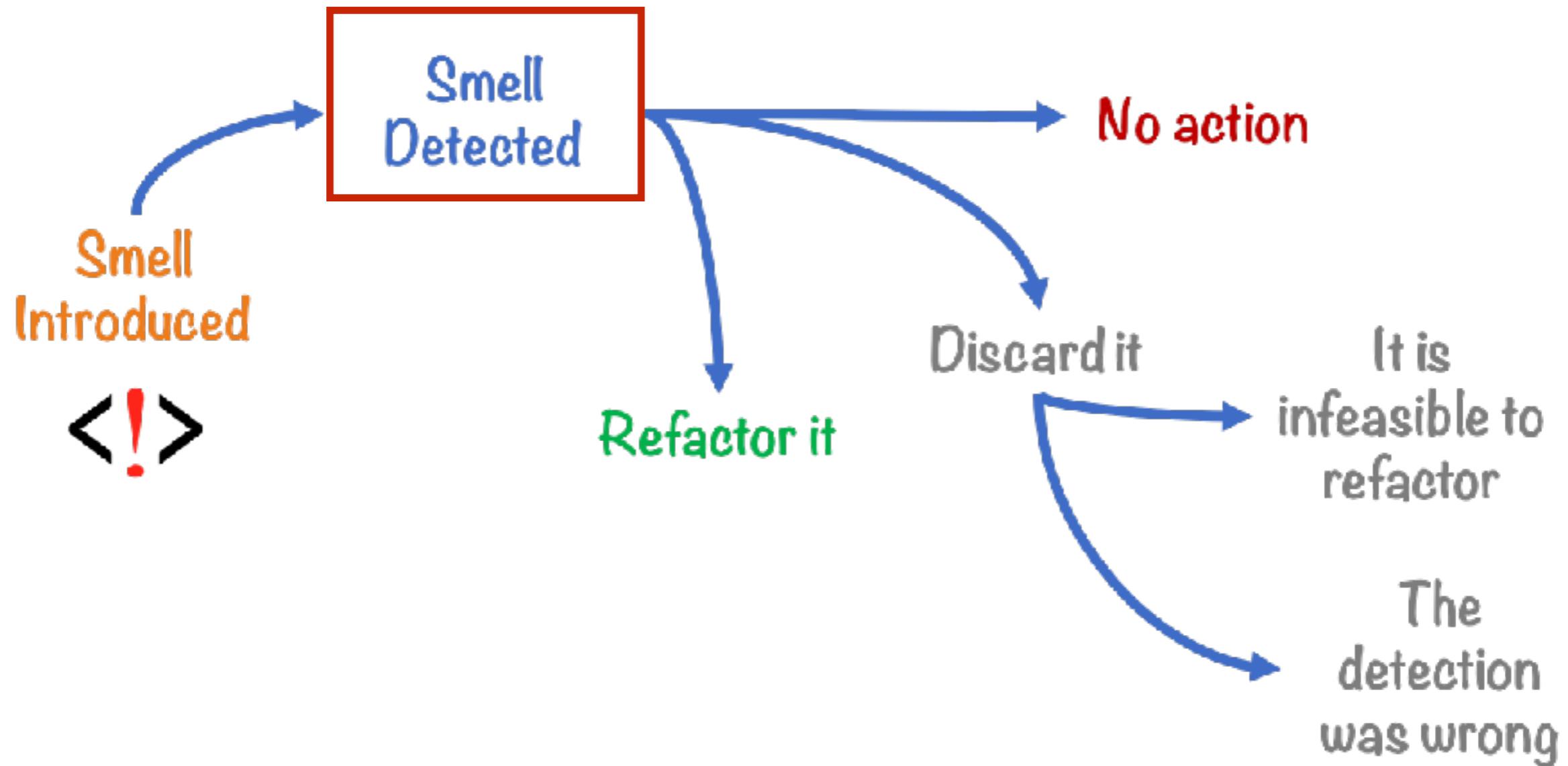
# Code Smell make problems

```
41 @... static MappedField validateQuery(final Class clazz, final Mapper mapper, final StringBuilder origProp, final FilterOperator op, final
42     MappedField mf = null;
43     final String prop = origProp.toString();
44     boolean hasTranslations = false;
45     if (!origProp.substring(0, 1).equals("$")) {
46         final String[] parts = prop.split(regex: "W.");
47         if (clazz == null) { return null; }
48         MappedClass mc = mapper.getMappedClass(clazz);
49         //CHECKSTYLE:OFF
50         for (int i = 0; ; ) { --Eek!
51             //CHECKSTYLE:ON
52             final String part = parts[i];
53             boolean fieldIsArrayOperator = part.equals("$");
54             nf = mc.getMappedField(part);
55             //translate from java field name to stored field name
56             if (mf == null && !fieldIsArrayOperator) {
57                 mf = mc.getMappedFieldByJavaFieldName(part);
58                 if (validateNames && mf == null) {
59                     throw new ValidationException(format("The field '%s' could not be found in '%s' while validating - %s; if you wi:
60                 }
61                 hasTranslations = true;
62                 if (mf != null) {
63                     parts[i] = mf.getNameInStore();
64                 }
65             }
66             i++;
67             if (mf != null && mf.isMap()) {
68                 //skip the map key validation, and move to the next part
69                 i++;
70             }
71             if (i >= parts.length) {
72                 break;
73             }
74             if (!fieldIsArrayOperator) {
75                 //catch people trying to search/update into @Reference/@Serialized fields
76                 if (validateNames && !canQueryPast(nf)) {
77                     throw new ValidationException(format("Cannot use dot-notation past '%s' in '%s'; found while validating - %s", pa:
78                 }
79                 if (mf == null && mc.isInterface()) {
80                     break;
81                 } else if (mf == null) {
82                     throw new ValidationException(format("The field '%s' could not be found in '%s'", prop, mc.getClazz().getName()));
83                 }
84                 //get the next MappedClass for the next field validation
85                 mc = mapper.getMappedClass((mf.isSingleValue() ? mf.getType() : mf.getSubClass()));
86             }
87         }
88         //record new property string if there has been a translation to any part
89         if (hasTranslations) {
90             origProp.setLength(0); // clear existing content
91             origProp.append(parts[0]);
92             for (int i = 1; i < parts.length; i++) {
93                 --Parameter mutation!
```

What's a prop?  
What's a part?  
Why all the null checks?  
Control the loop  
Comments, because code is unclear  
Parameter mutation!



# Do you know Code smell ?



# Code Smell (example)

Duplication code

Long method

Large class (God class)

Comments

Long parameter list

Switch/If statements

Divergent change (one class)

Shotgun surgery (one class)

<https://sourcemaking.com/refactoring/smells>



**Not taking too much  
Show me the code**



# Code Smell Workshop

<https://github.com/emilybache/Tennis-Refactoring-Kata>



# SOLID principle



# SOLID principle



**SOLID**

Software development is not a Jenga game.



# 1. Single Responsibility Principle



## Single Responsibility Principle

Just because you *can* doesn't mean you *should*.



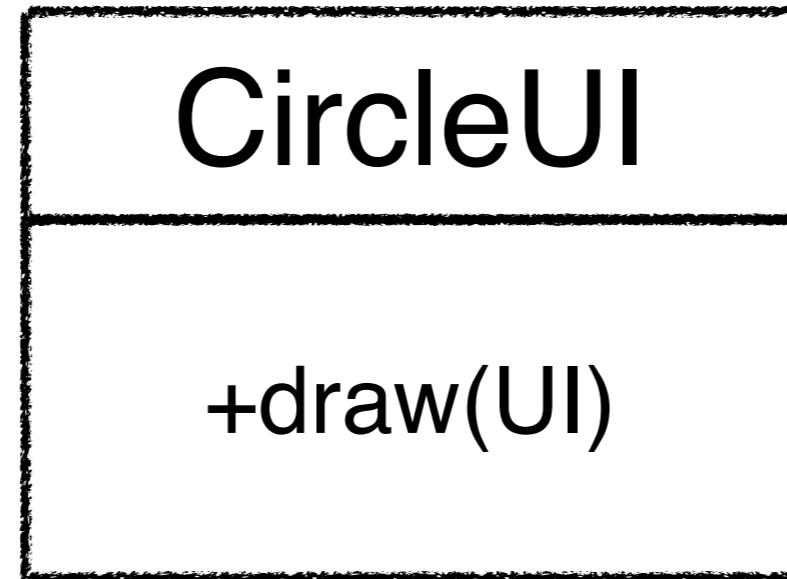
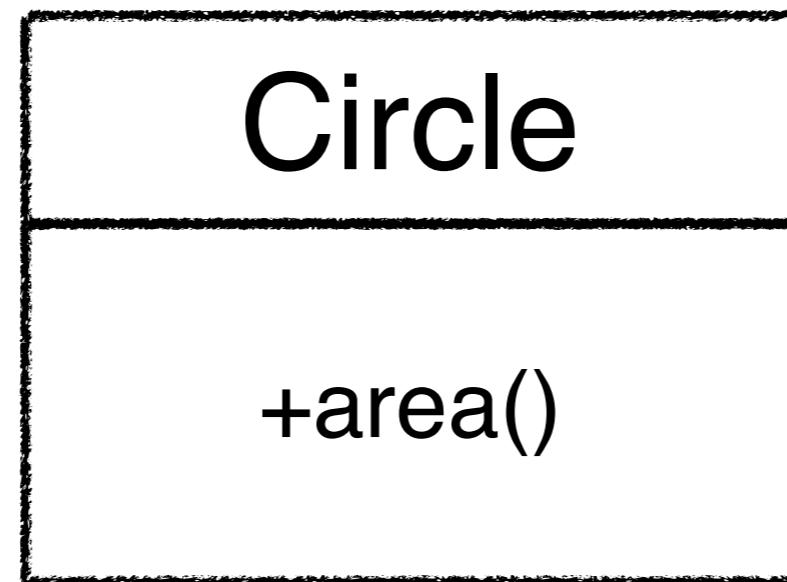
# Example

Circle

+area()  
+draw(UI)



# Example



## 2. Open-Closed Principle



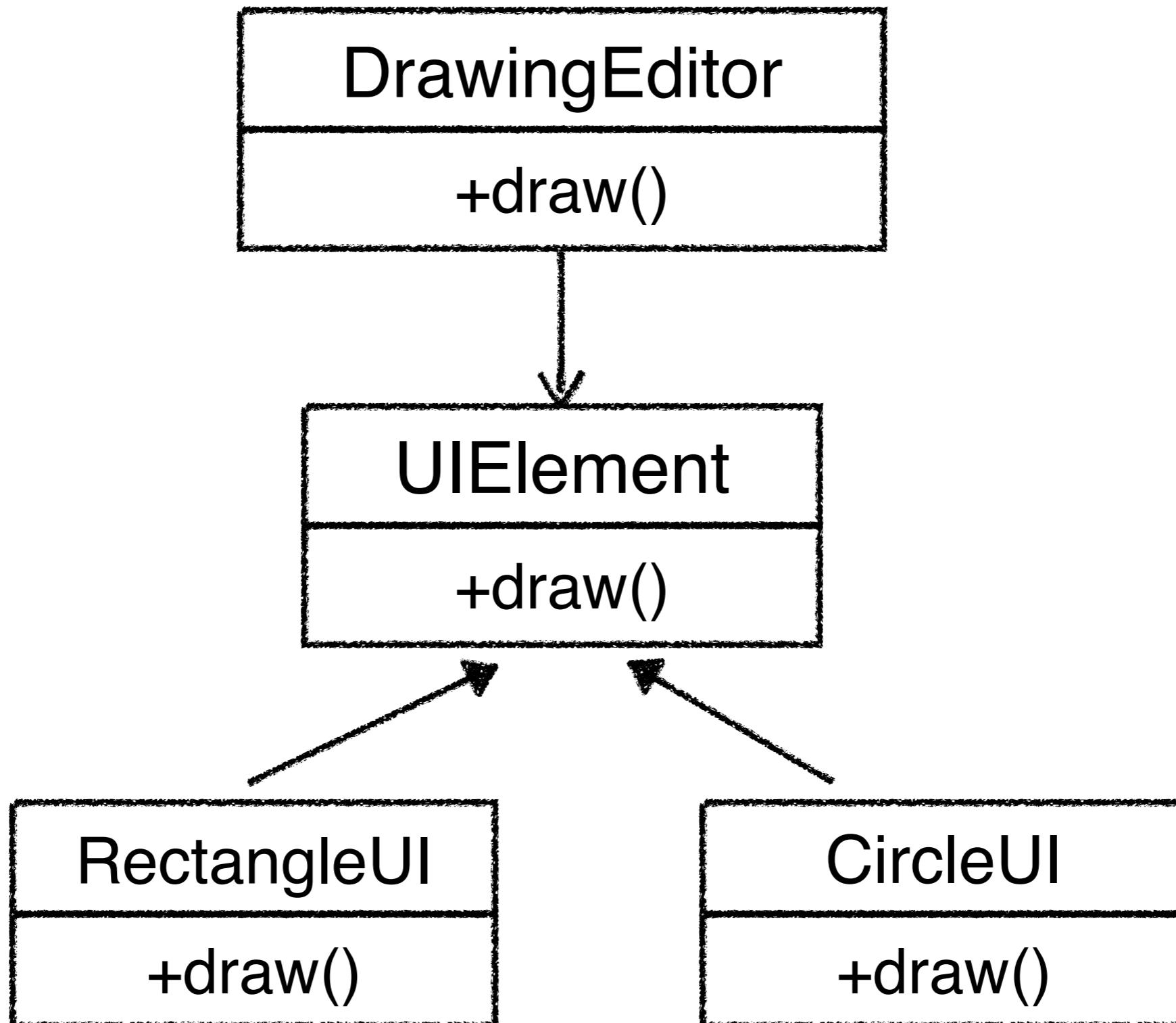
# Example

## DrawingEditor

```
+draw()  
-drawCircle()  
-drawRectangle()
```



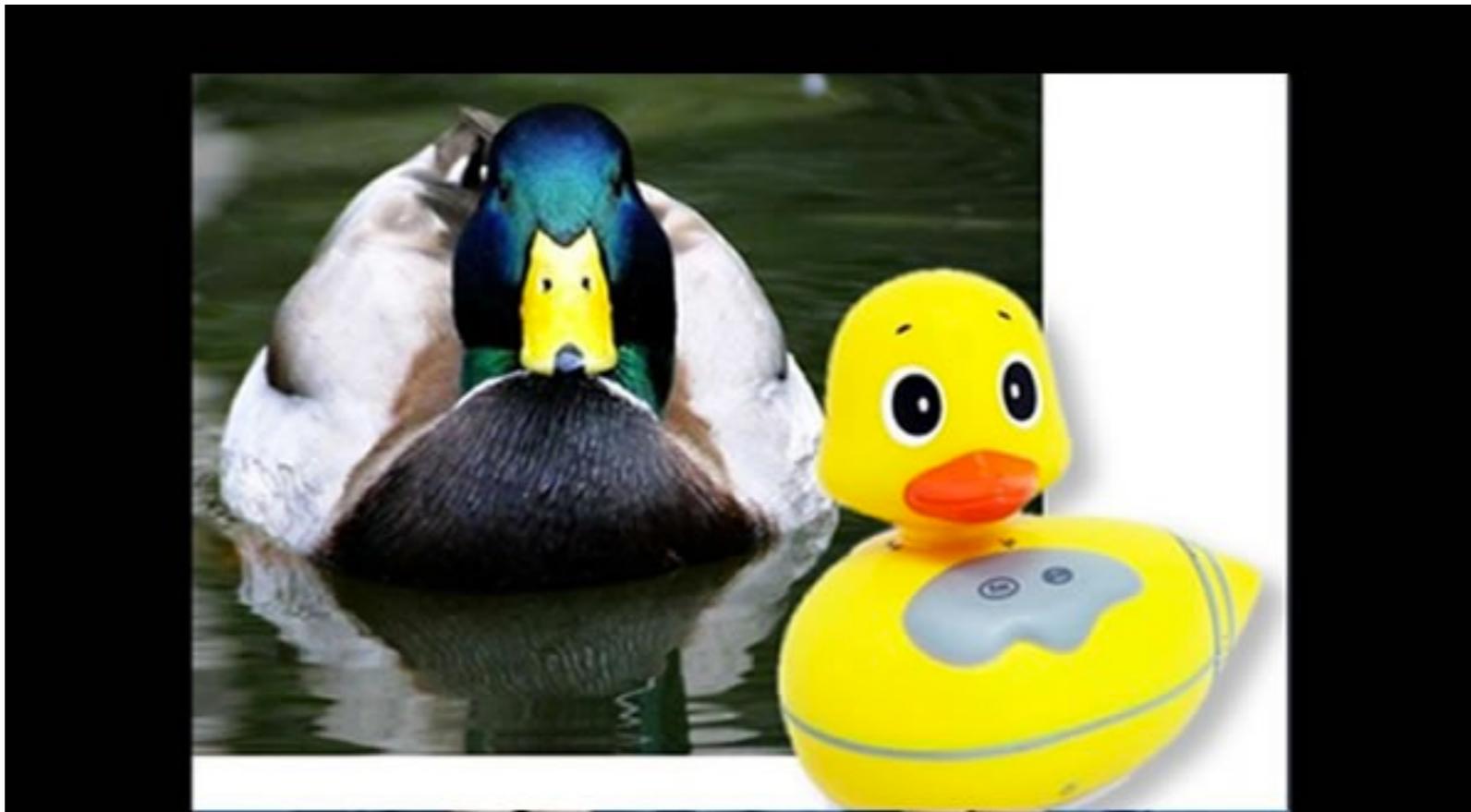
# Example



# Workshop



# 3. Liskov Substitution Principle

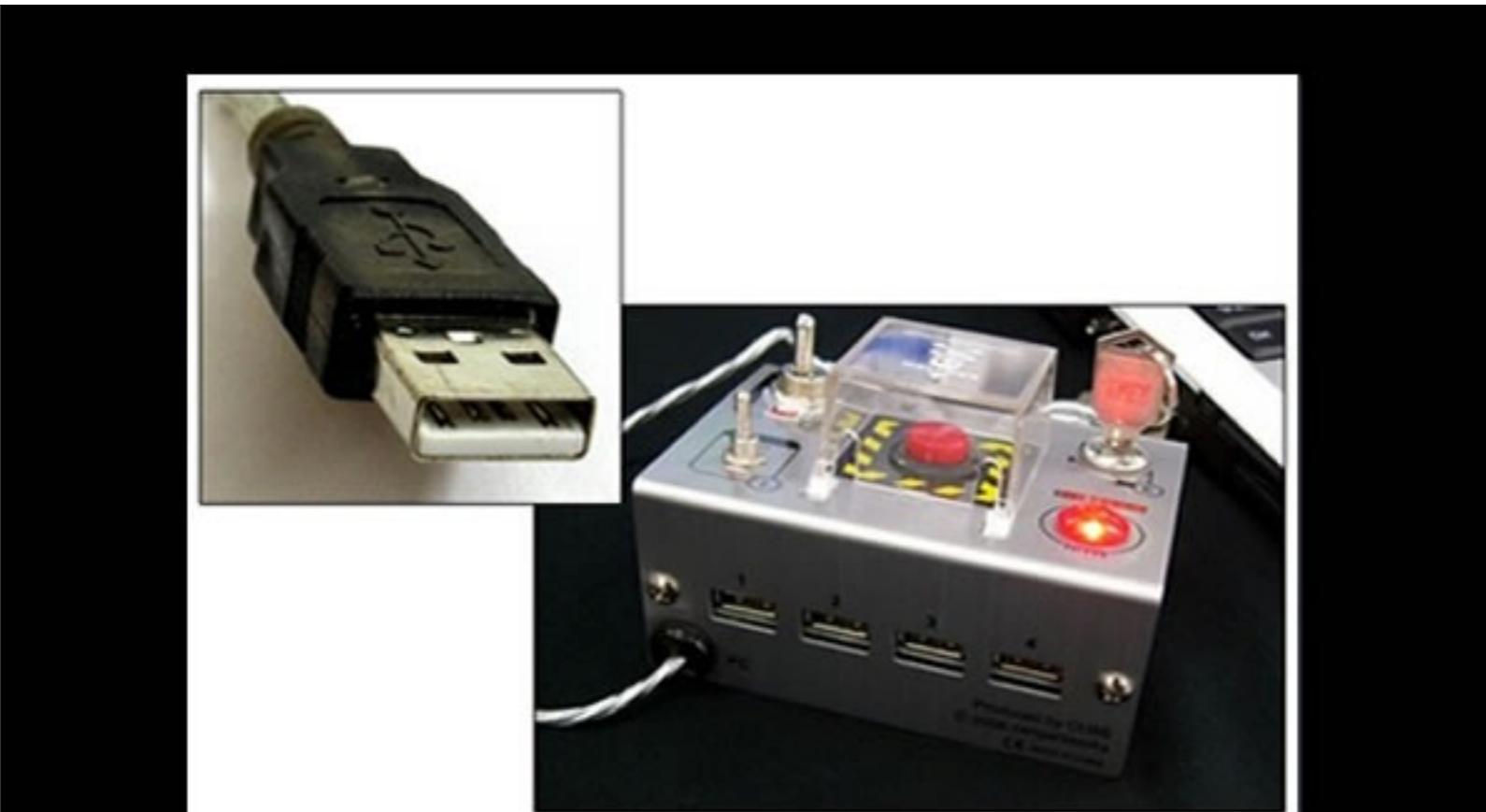


## Liskov Substitution Principle

If it looks like a duck and quacks like a duck but needs batteries,  
you probably have the wrong abstraction.



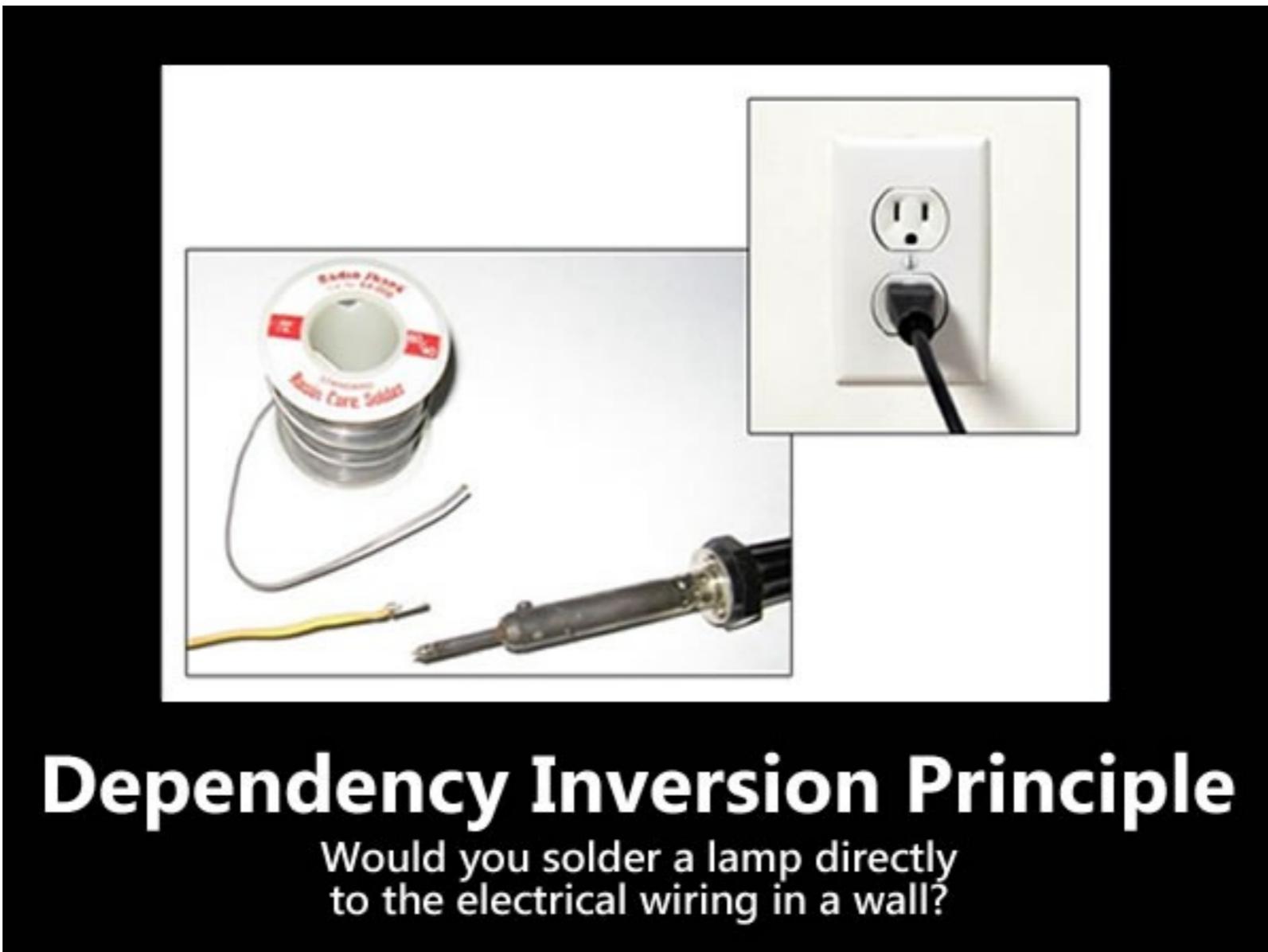
# 4. Interface Segregation Principle



**Interface Segregation Principle**  
You want me to plug this in *where?*



# 5. Dependency Inversion Principle



# Workshop



# Spring Boot



# Agenda

- RESTful API
- Java frameworks
- Building RESTful API with Spring Boot
- Spring Boot with Spring Data (JDBC/JPA)
- Testing with Spring Boot



# **REST**

**RE**presentation **S**tate **T**ransfer

The style of software architecture behind  
**RESTful** services

Defined in 2000 by Roy Fielding

[https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)



# Goals

Scalability  
Generality of interfaces  
Independent deployment of components



# **RESTful service**



# REST Request Messages

RESTful request is typically in form of  
**Uniform Resource Identifiers (URI)**



# REST Request Messages

RESTful request is typically in form of  
**Uniform Resource Identifiers (URI)**

Structure of URI depend on specific service



# REST Request Messages

RESTful request is typically in form of  
**Uniform Resource Identifiers (URI)**

Structure of URI depend on specific service

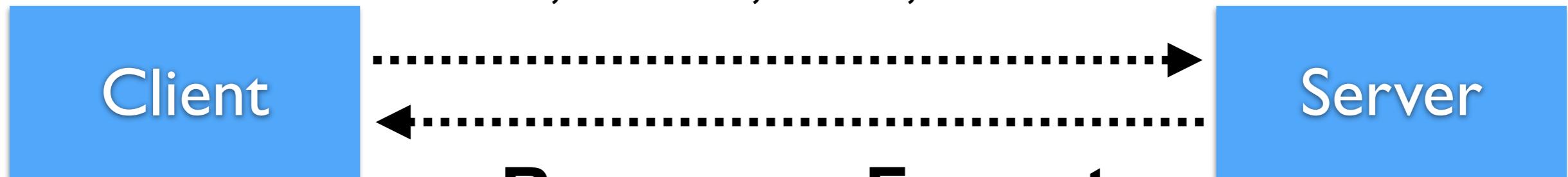
Request can include parameter and data in  
body of request as XML, JSON etc.



# REST Request & Response

## Request Method

GET, POST, PUT, DELETE



## Response Format

XML or JSON



# HTTP Methods meaning

Method	Meaning
GET	Read data
POST	Create/Insert new data
PUT/PATCH	Update data or insert if a new id
DELETE	Delete data



# Response format ?



# Good APIs ?



# Java Framework for RESTful



## unirest

Lightweight HTTP Request Client Libraries



## Dropwizard



## Restlet



## ACT.framework

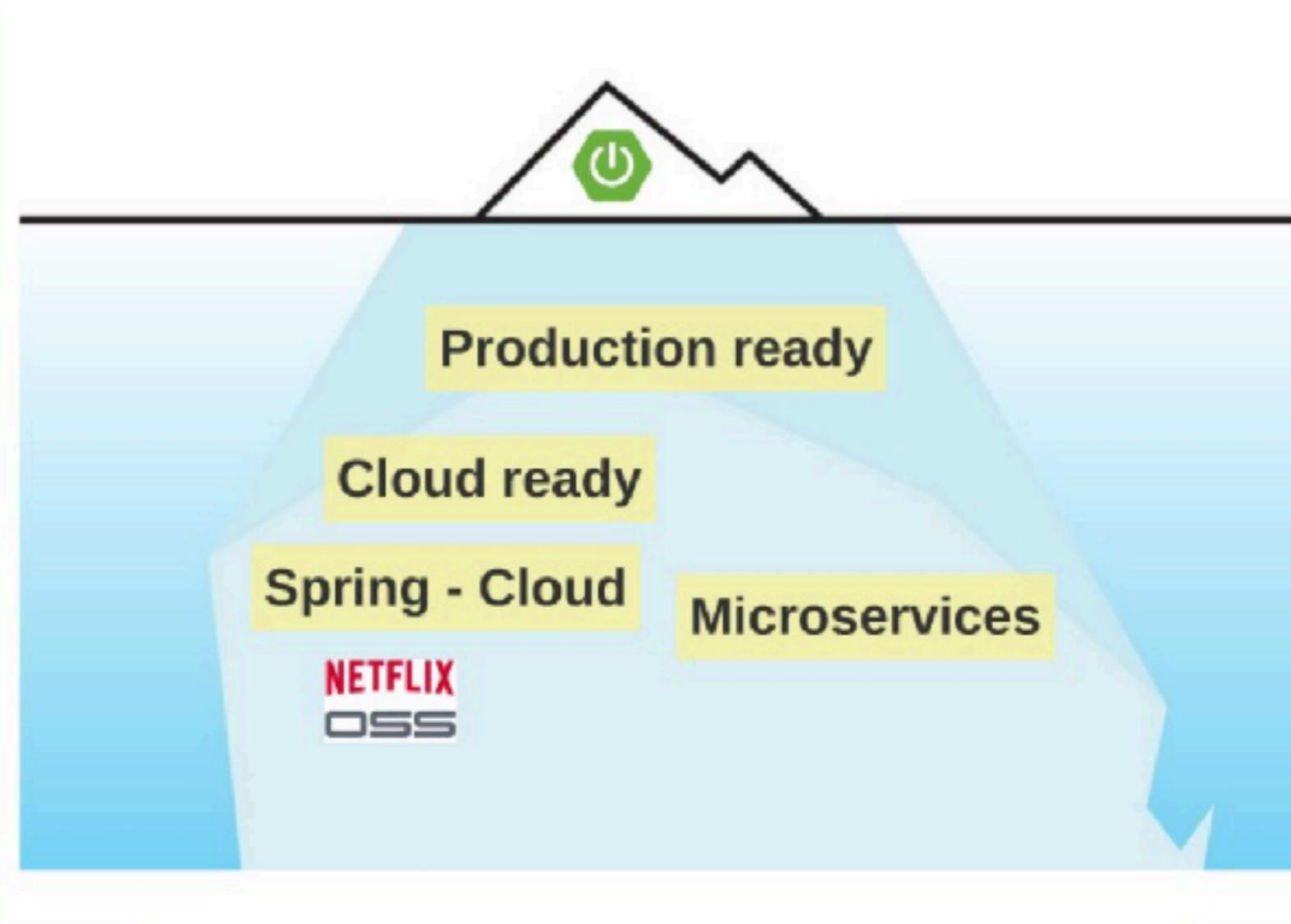


# Hello Spring Boot 2.x



# Why ?

Application skeleton generator  
Reduce effort to add new technologies



# What ?

Embedded application server

Integration with tools/technologies (starter)

Production tools (monitoring, health check)

Configuration management

Dev tools

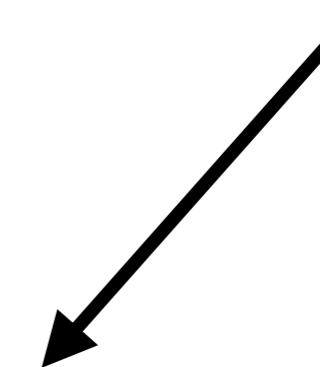
No source code generation, no XML



# How ?

Apache Tomcat

JAR file

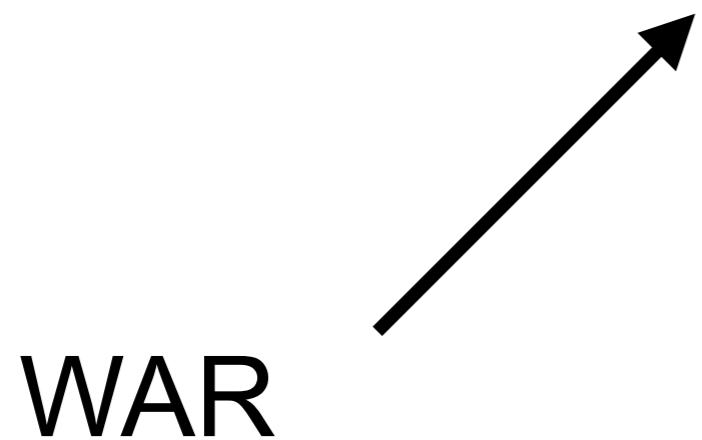


\$mvn spring-boot:run  
\$java -jar demo.jar



# How ?

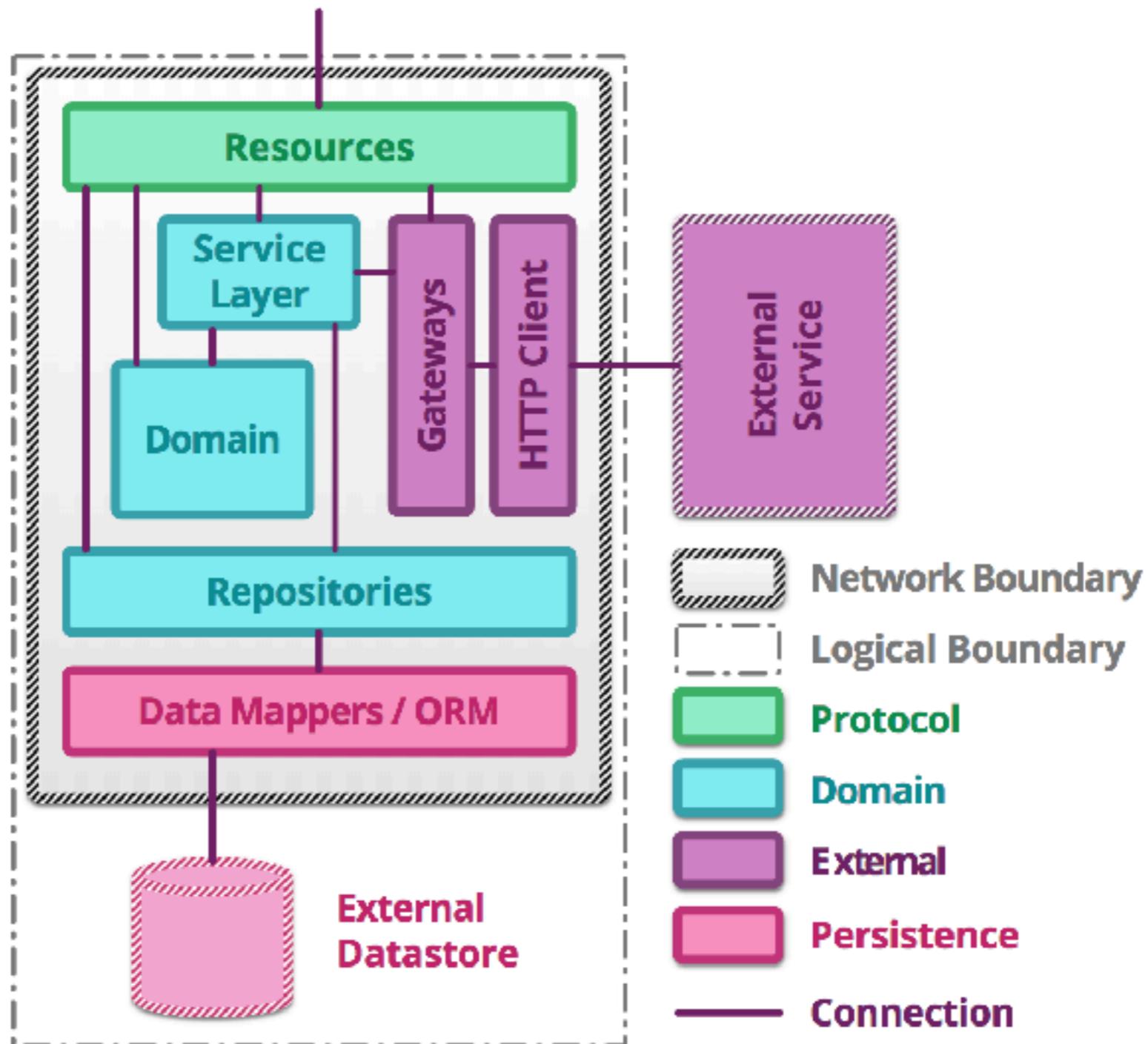
Apache Tomcat



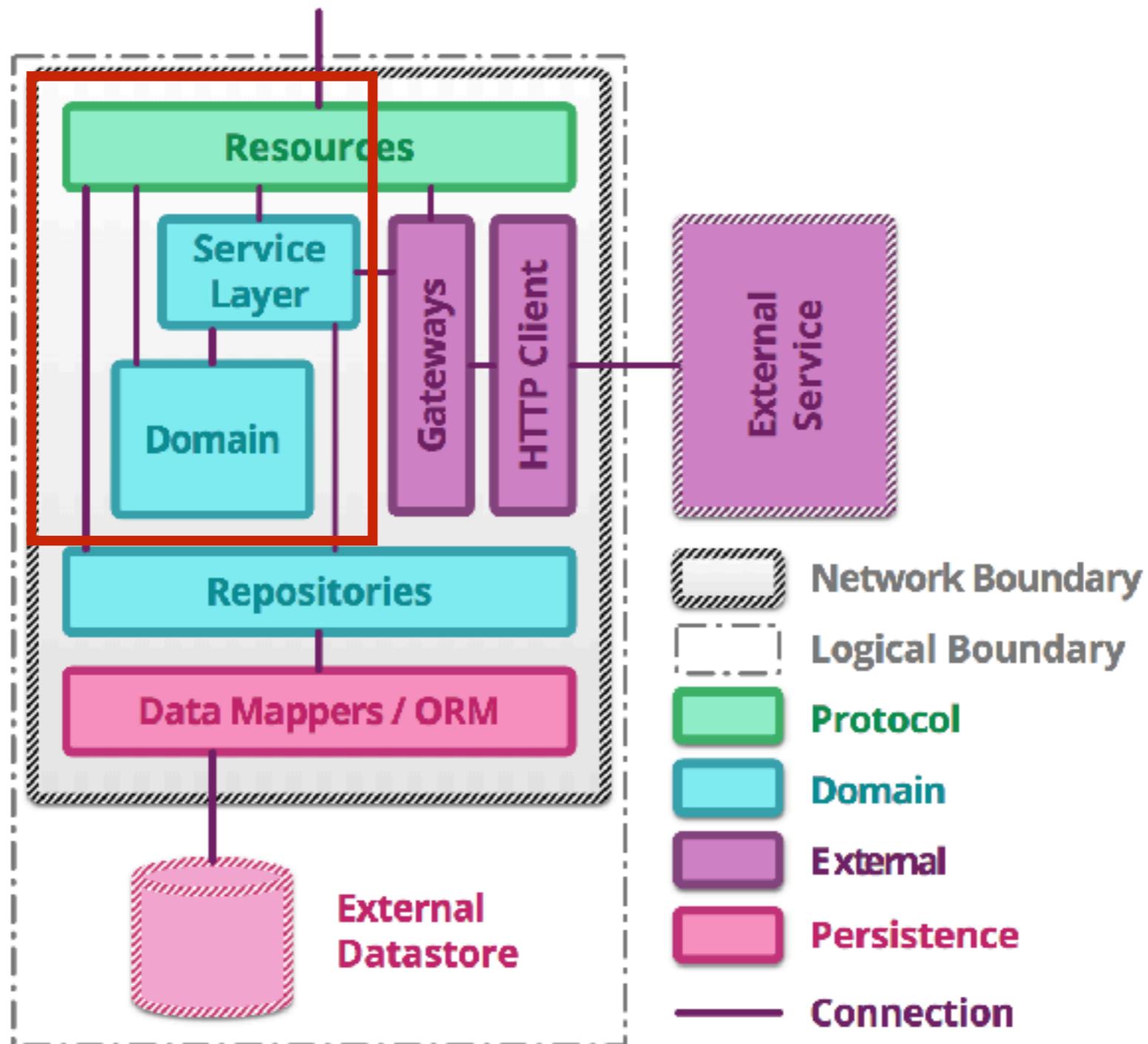
# **Building RESTful API with Spring Boot**



# Service Structure



# Service Structure



# Create project with Spring Initializr



# Spring Initializr

<https://start.spring.io/>

SPRING INITIALIZR bootstrap your application now

Generate a  with  and Spring Boot

**Project Metadata**

Artifact coordinates

Group

Artifact

**Dependencies**

Add Spring Boot Starters and dependencies to your application

Search for dependencies

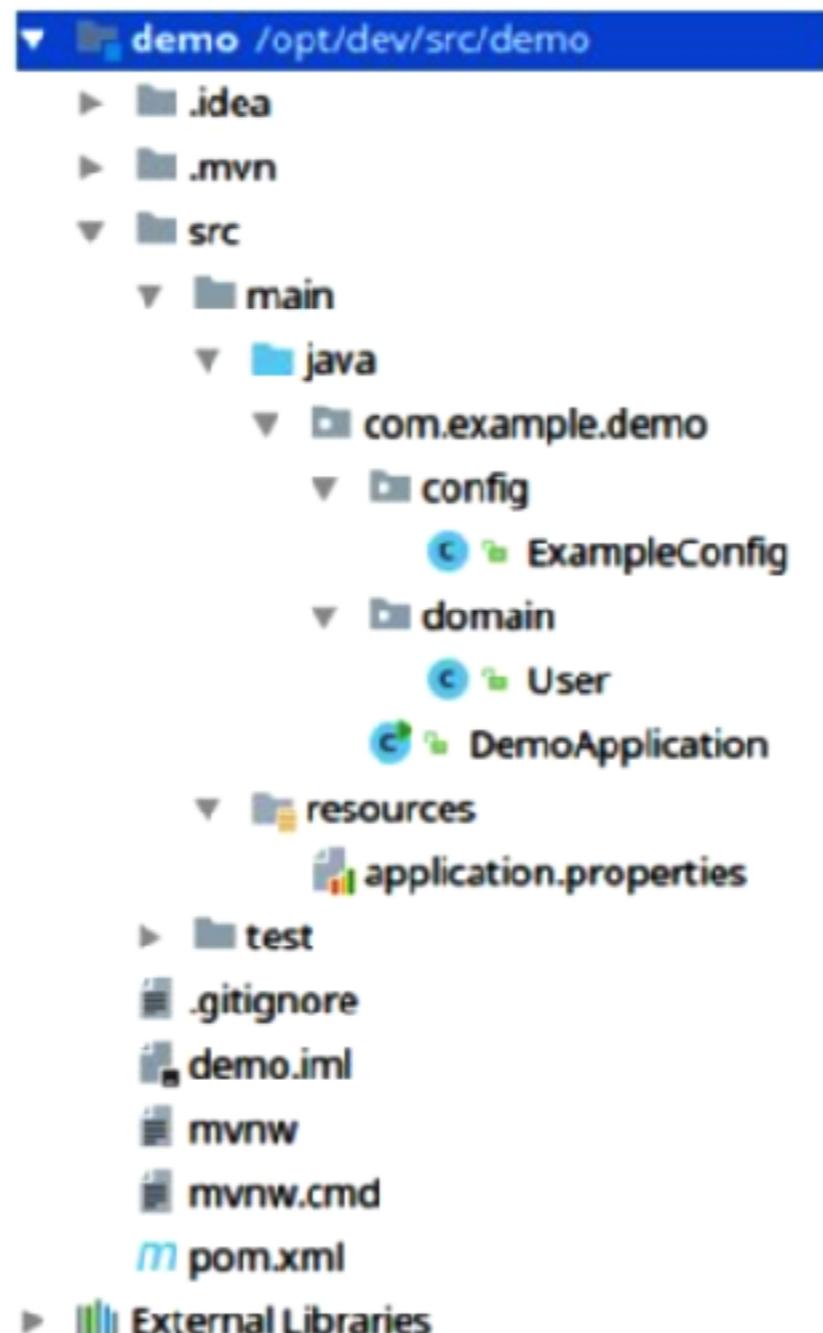
Selected Dependencies

Don't know what to look for? Want more options? [Switch to the full version.](#)

start.spring.io is powered by [Spring Initializr](#) and [Pivotal Web Services](#)



# Structure of Spring Boot



# Spring Boot main class

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```



# Run project

```
./mvnw package  
$java -jar target/<file name>.jar
```

```
2018-06-07 13:03:30.412  INFO 12828 --- [  
oApplication           : Starting DemoApplication on  
D 12828 (started by somkiat in /Users/somkiat/Down  
2018-06-07 13:03:30.418  INFO 12828 --- [  
oApplication           : No active profile set, fall
```



# Custom banner

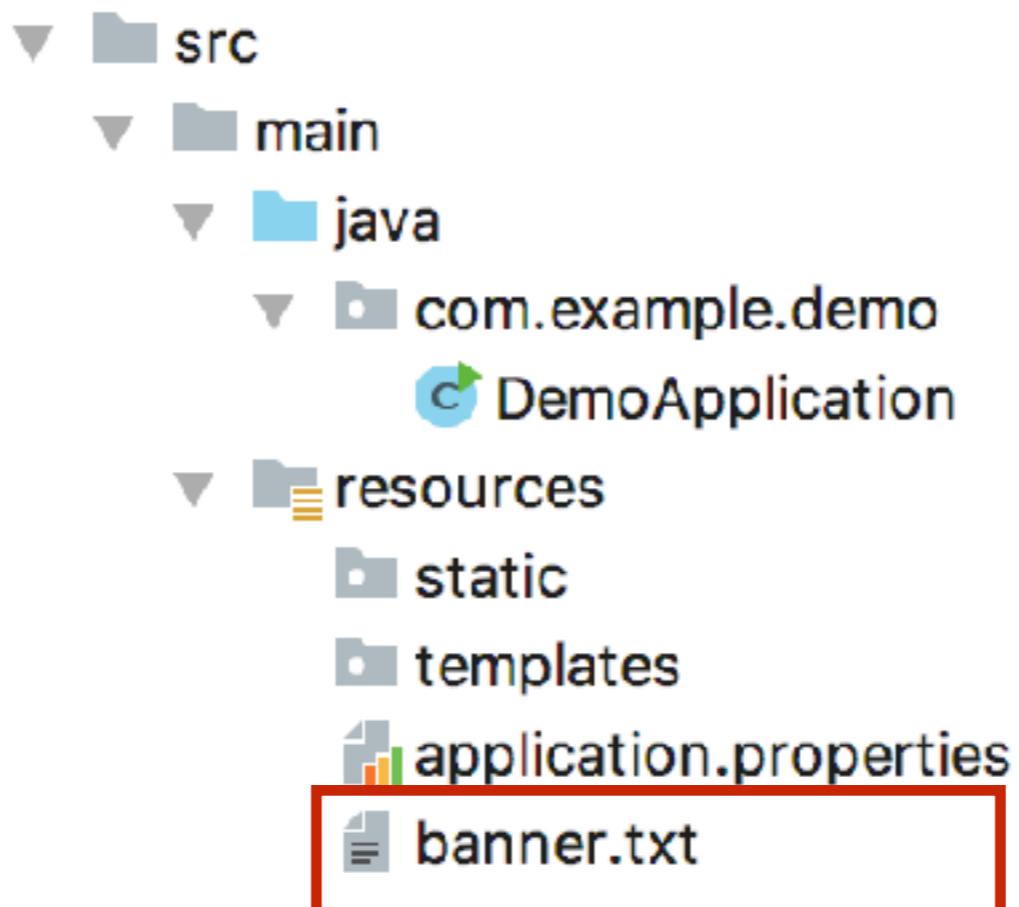
The Spring Boot logo consists of a grid of characters including slashes, parentheses, brackets, and underscores, arranged in a pattern that forms the words "Spring" and "Boot". Below the grid, the text ":: Spring Boot ::" is written in green, and "(v2.0.2.RELEASE)" is written in gray.

```
2018-06-07 13:03:30.412  INFO 12828 --- [  
  oApplication : Starting DemoApplication on  
D 12828 (started by somkiat in /Users/somkiat/Down  
2018-06-07 13:03:30.418  INFO 12828 --- [  
  oApplication : No active profile set, fall
```



# Custom banner (1)

Create file banner.txt or banner.png in resources folder



# Custom banner (2)

```
@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        ImageBanner banner = new ImageBanner(
            new ClassPathResource("try.png"));

        new SpringApplicationBuilder()
            .sources(DemoApplication.class)
            .banner(banner)
            .run();
    }
}
```



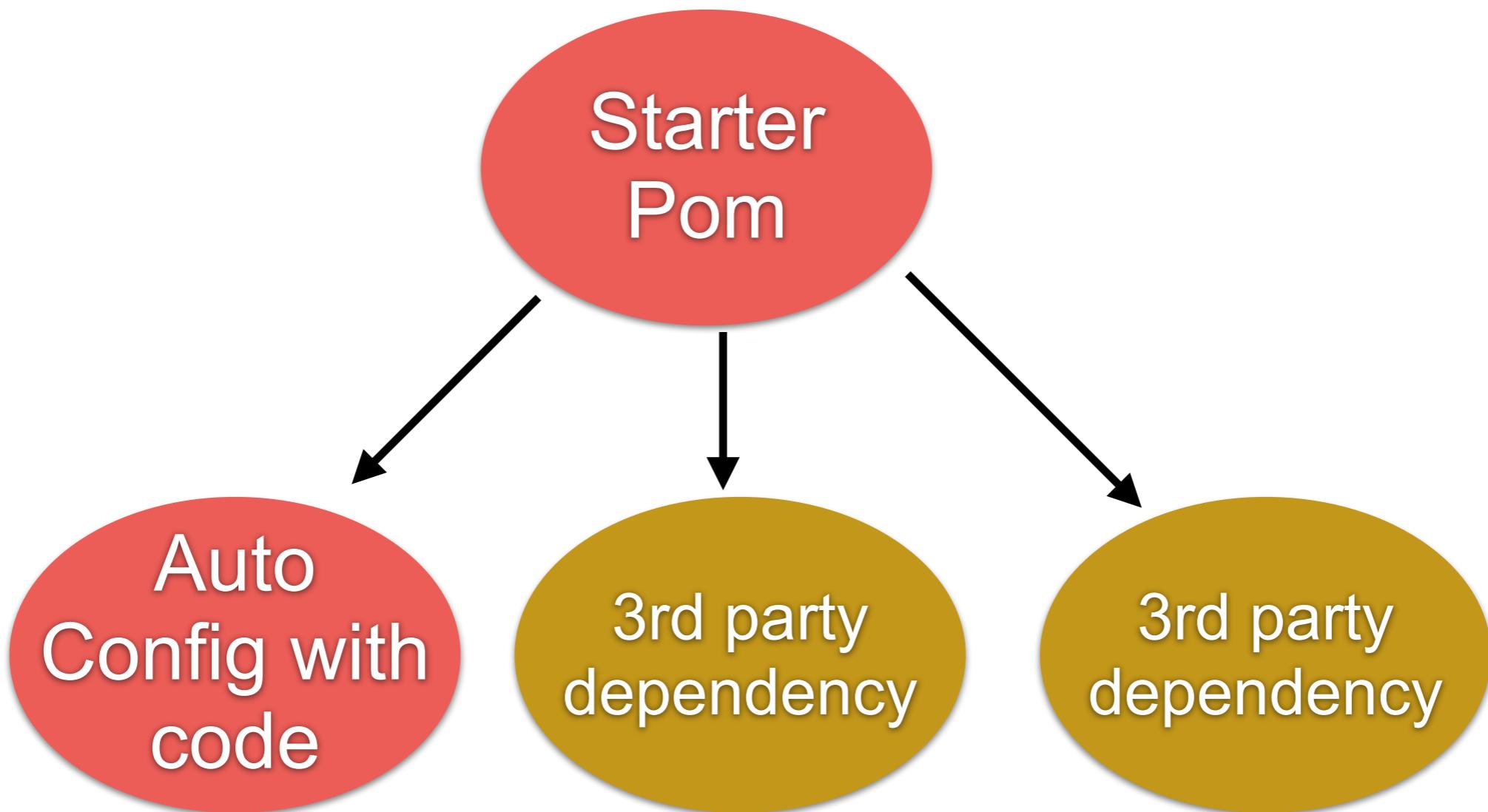
# Disable banner

```
@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        new SpringApplicationBuilder()
            .sources(DemoApplication.class)
            .bannerMode(Banner.Mode.OFF)
            .run(args);
    }
}
```



# Anatomy of Starter



# Configuration management

Properties/XML/YAML

Config server (App, Spring cloud config, git)

17 ways !!!

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html>



# Create project



# Spring boot application

## hello.HelloApplication.java

```
package hello;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HelloApplication {

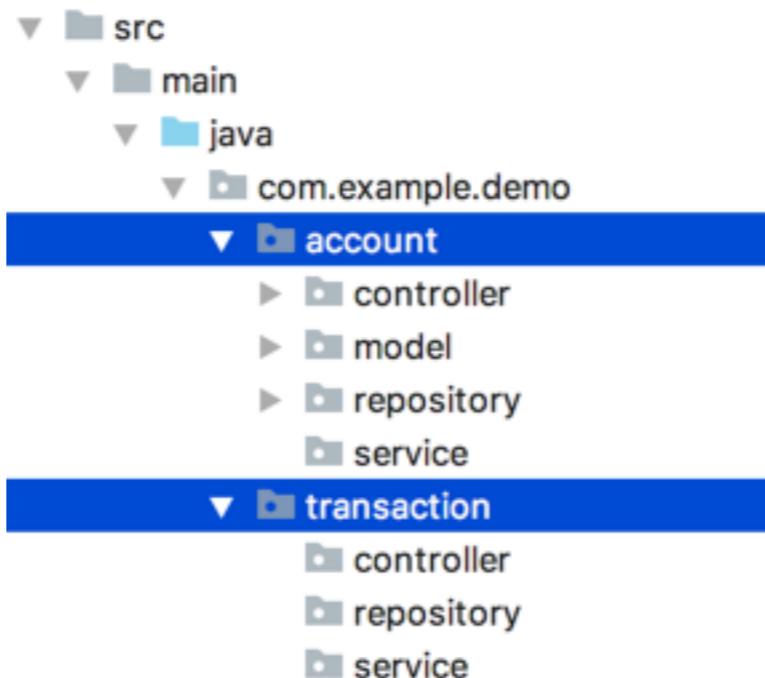
    public static void main(String[] args) {
        SpringApplication.run(HelloApplication.class, args);
    }

}
```



# Spring Boot Structure

Separate by function/domain



# Create model class

hello.model.Hello.java

```
package hello.domain;

public class Hello {

    private String message;

    public Hello(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```



# 6. Create REST Controller

## hello.controller.HelloController.java

```
package hello.controller;

import hello.domain.Hello;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/hello/{name}")
    public Hello sayHi(@PathVariable String name) {
        return new Hello("Hello " + name);
    }

}
```



# 7. Compile and Packaging

\$mvnw clean package



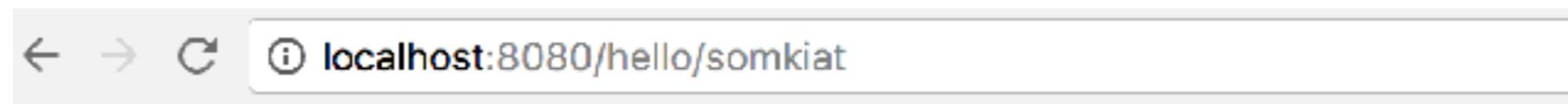
# 8. Run

```
$java -jar target/hello.jar
```



# 9. Open in browser

<http://localhost:8080/hello/somkiat>

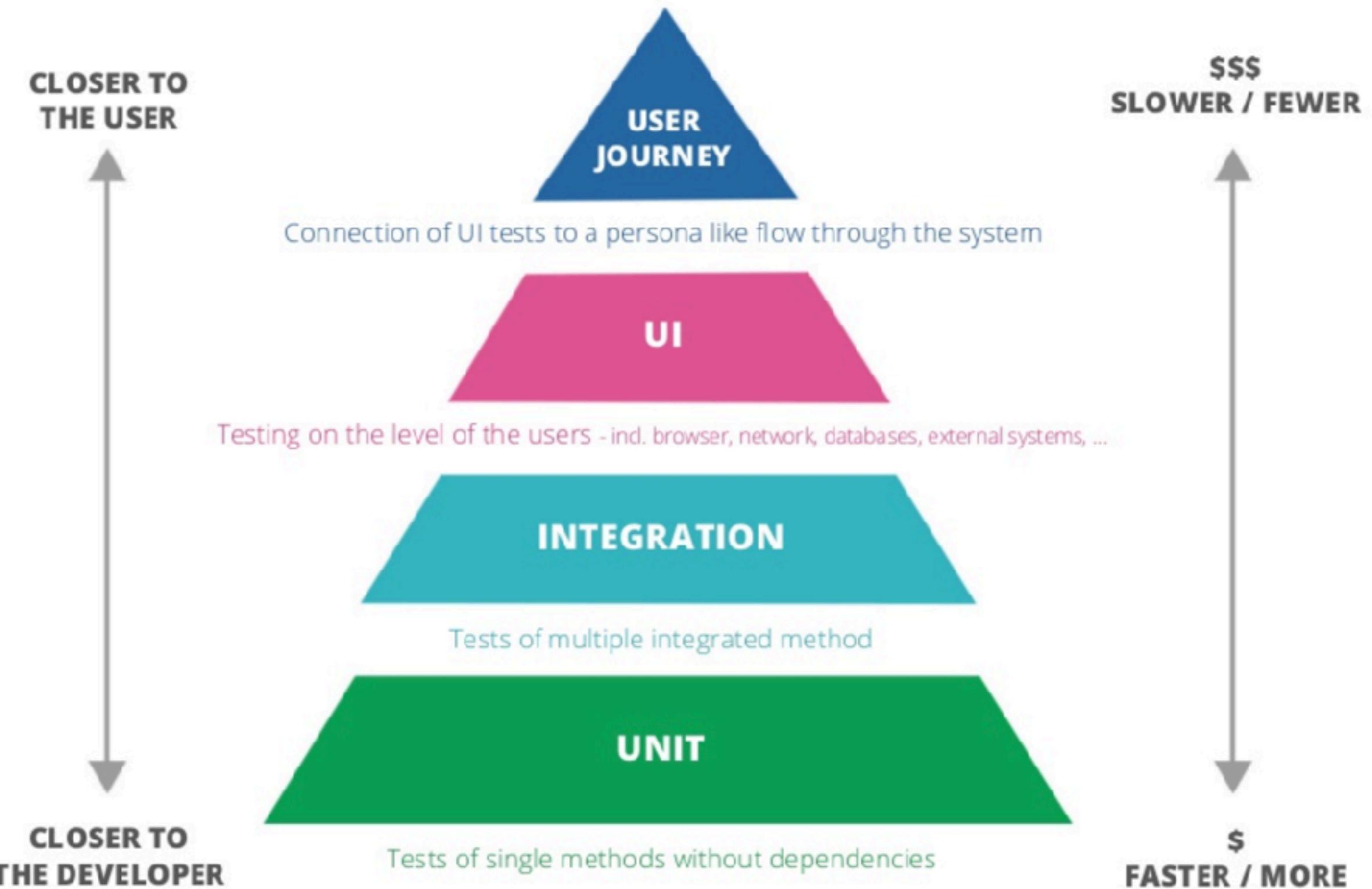


{ "message": "Hello somkiat" }



# How to test the Hello service ?





# Unit tests

How to use model ?

```
public class HelloTest {  
  
    @Test  
    public void success_to_create_model_with_constructor() {  
        Hello hello = new Hello("Somkiat");  
        assertEquals( expected: "Somkiat", hello.getMessage());  
    }  
  
}
```



# API/Controller tests

How to use controller ?

Spring boot provides MockMvc to test Controller

```
package hello.controller;

import hello.domain.Hello;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/hello/{name}")
    public Hello sayHi(@PathVariable String name) {
        return new Hello("Hello " + name);
    }

}
```



# API/Controller tests

```
@RunWith(SpringRunner.class)
@WebMvcTest/controllers = HelloController.class)
public class HelloControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void shouldReturnHelloSomkiat() throws Exception {
        mockMvc.perform(get(urlTemplate: "/hello/somkiat"))
            .andExpect(jsonPath(expression: "$.message")
                .value(expectedValue: "Hello somkiat"))
            .andExpect(status().is2xxSuccessful());
    }

}
```



# API/Controller tests

With Spring Boot Testing + TestRestTemplate

```
@RunWith(SpringRunner.class)
@SpringBootTest(
    webEnvironment = WebEnvironment.RANDOM_PORT)
public class AccountControllerTest {

    @Autowired
    private TestRestTemplate testRestTemplate;

    @Autowired
    private AccountRepository accountRepository;

    @Before
    public void initData() {
        accountRepository.save(new Account("01"));
        accountRepository.save(new Account("02"));
    }
}
```



# API/Controller tests

With Spring Boot Testing + TestRestTemplate

```
@Test  
public void  
    เรียกข้อมูลบัญชีของหมายเลข_0868696209_ต้องเจอสองบัญชีนั่น () {  
    List<AccountResponse> accountResponses  
        = testRestTemplate.getForObject(  
            "/account/0868696209", List.class);  
  
    assertEquals(2, accountResponses.size());  
}
```



# Compile with testing

\$mvn clean package

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
```



# % of Code/Test coverage



# Add coverage to pom.xml (1)

```
<build>
    <finalName>hello</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>

        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.5.1</version>
            <configuration>
                <source>${java.version}</source>
                <target>${java.version}</target>
                <encoding>${project.build.sourceEncoding}</encoding>
            </configuration>
        </plugin>
    </plugins>
</build>
```



# Add coverage to pom.xml (2)

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>cobertura-maven-plugin</artifactId>
  <version>2.7</version>
  <configuration>
    <formats>
      <format>html</format>
      <format>xml</format>
    </formats>
  </configuration>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>cobertura</goal>
      </goals>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>org.ow2.asm</groupId>
      <artifactId>asm</artifactId>
      <version>5.0.3</version>
    </dependency>
  </dependencies>
</plugin>
```



# Run test again

\$mvn clean package

Cobertura Report generation was successful.

Cobertura 2.1.1 - GNU GPL License (NO WARRANTY) - See COPYRIGHT file

Cobertura: Loaded information on 3 classes.

time: 125ms

Cobertura Report generation was successful.

---

BUILD SUCCESS

---



# Coverage report

open target/site/cobertura/index.html

**Packages**

All  
[hello](#)  
[hello.controller](#)  
[hello.domain](#)

**Coverage Report - All Packages**

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	3	63% 7/11	N/A N/A	1
hello	1	33% 1/3	N/A N/A	1
hello.controller	1	100% 2/2	N/A N/A	1
hello.domain	1	66% 4/6	N/A N/A	1

Report generated by [Cobertura](#) 2.1.1 on 3/6/18 12:40 AM.

---

**All Packages**

**Classes**

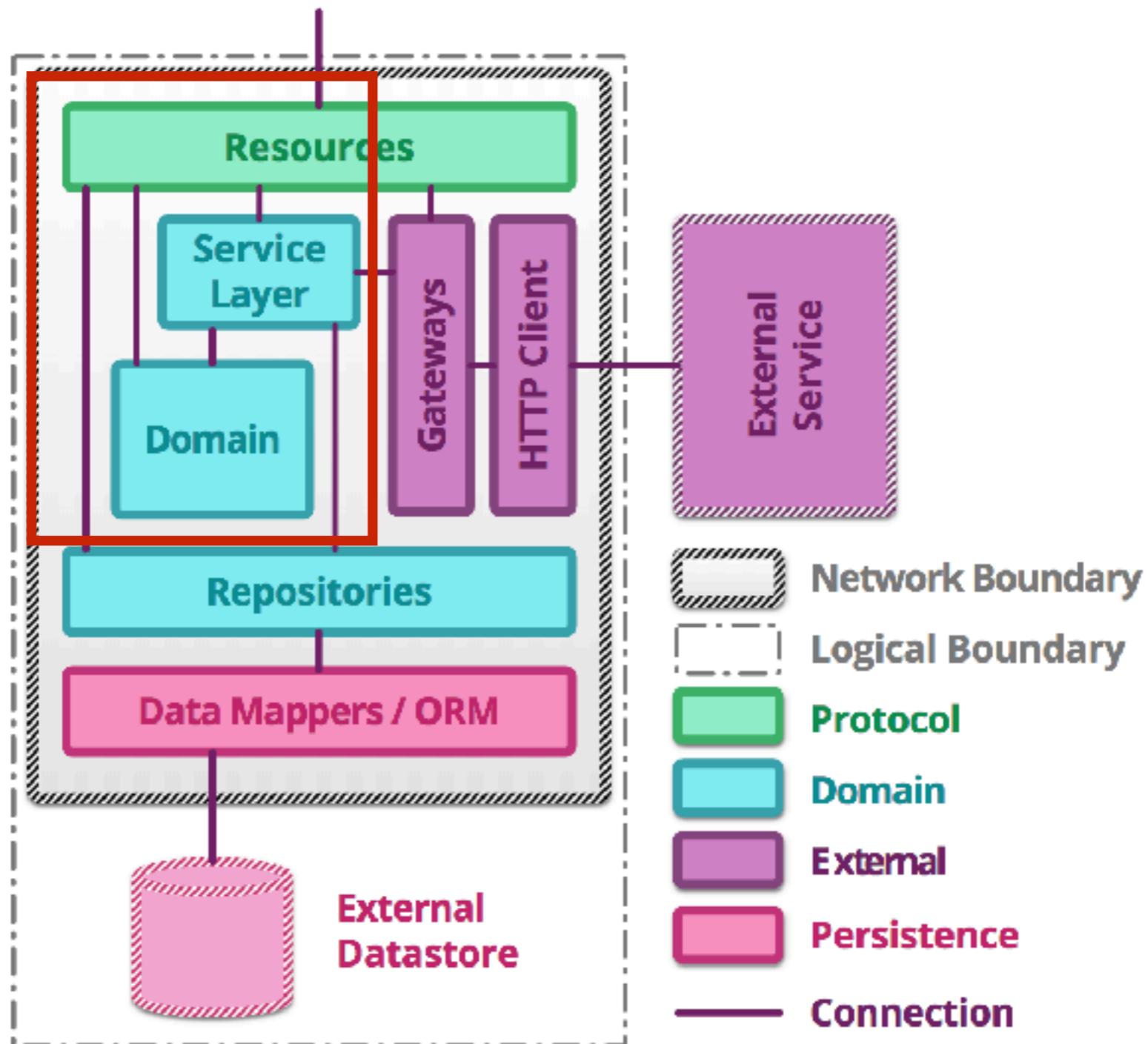
[Hello \(66%\)](#)  
[HelloApplication \(33%\)](#)  
[HelloController \(100%\)](#)



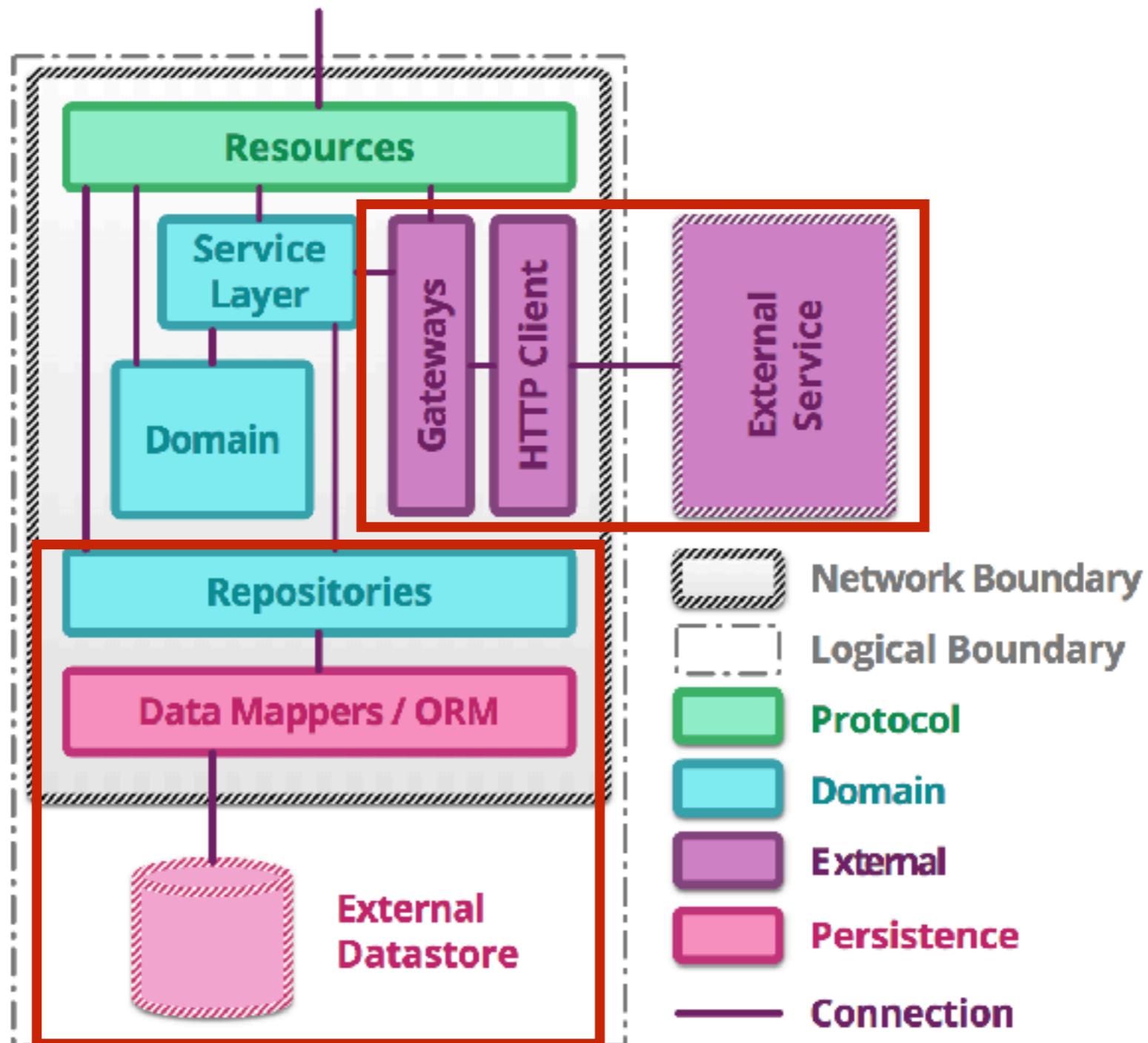
# How to improve % of coverage ?



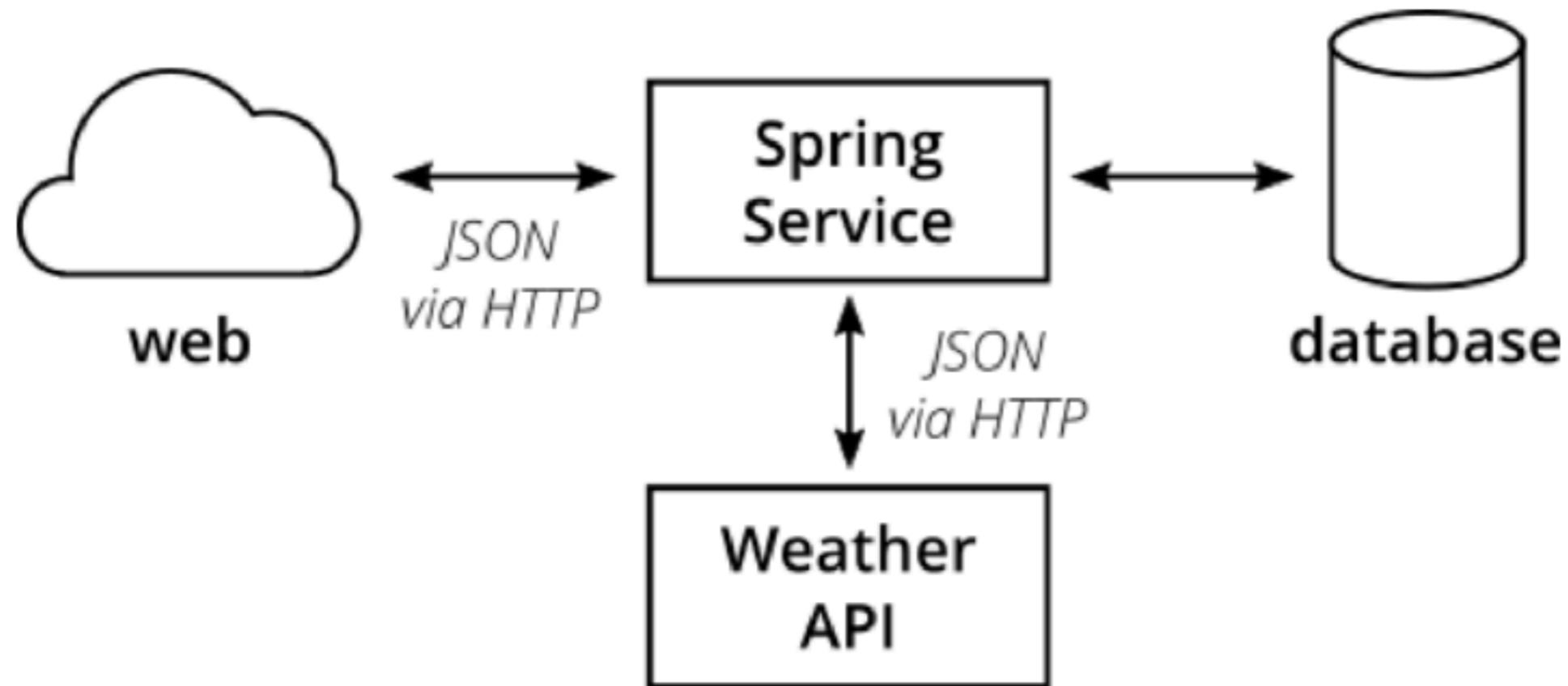
# Service Structure



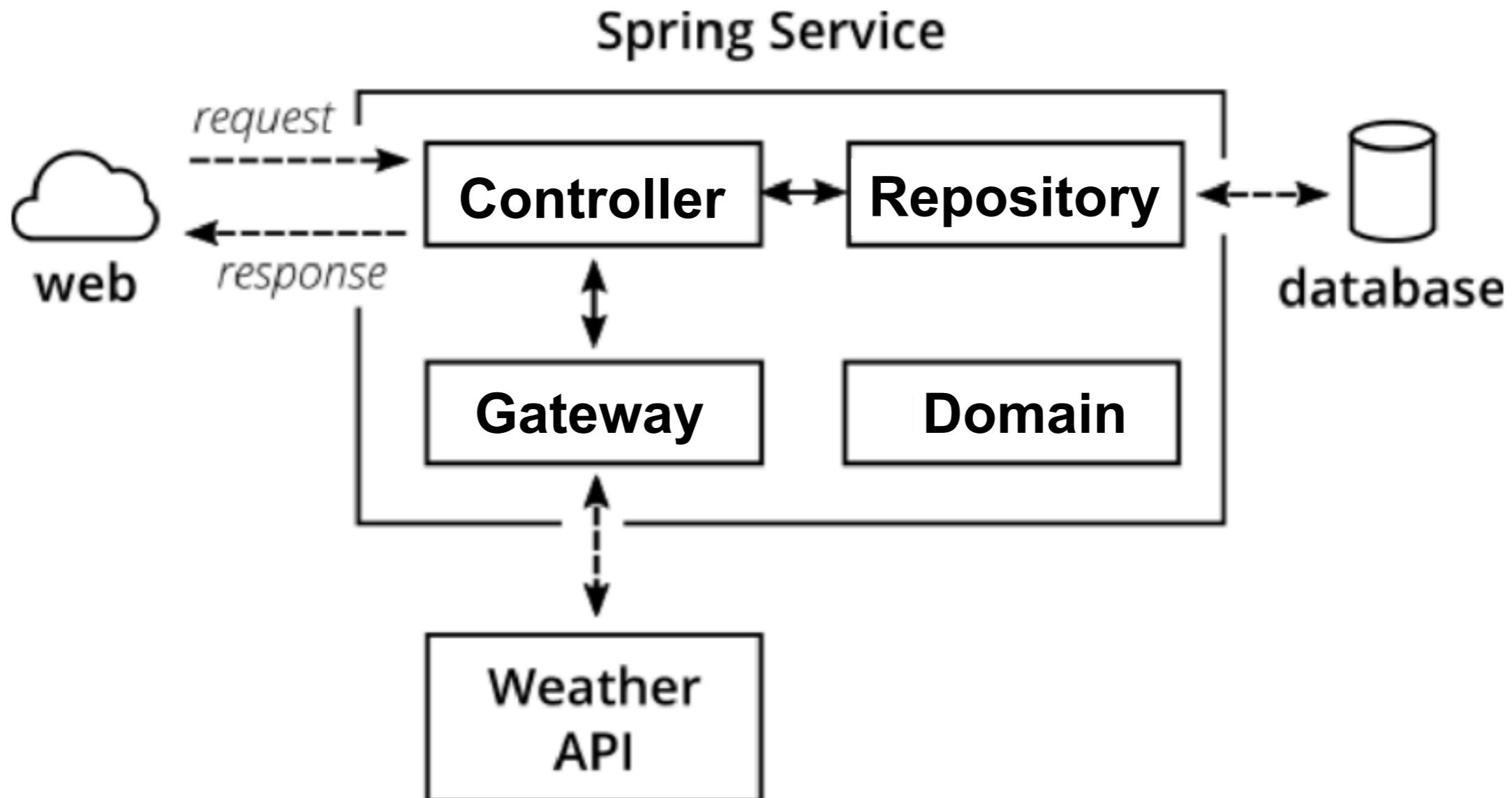
# Service Structure



# Sample application



# Project structure

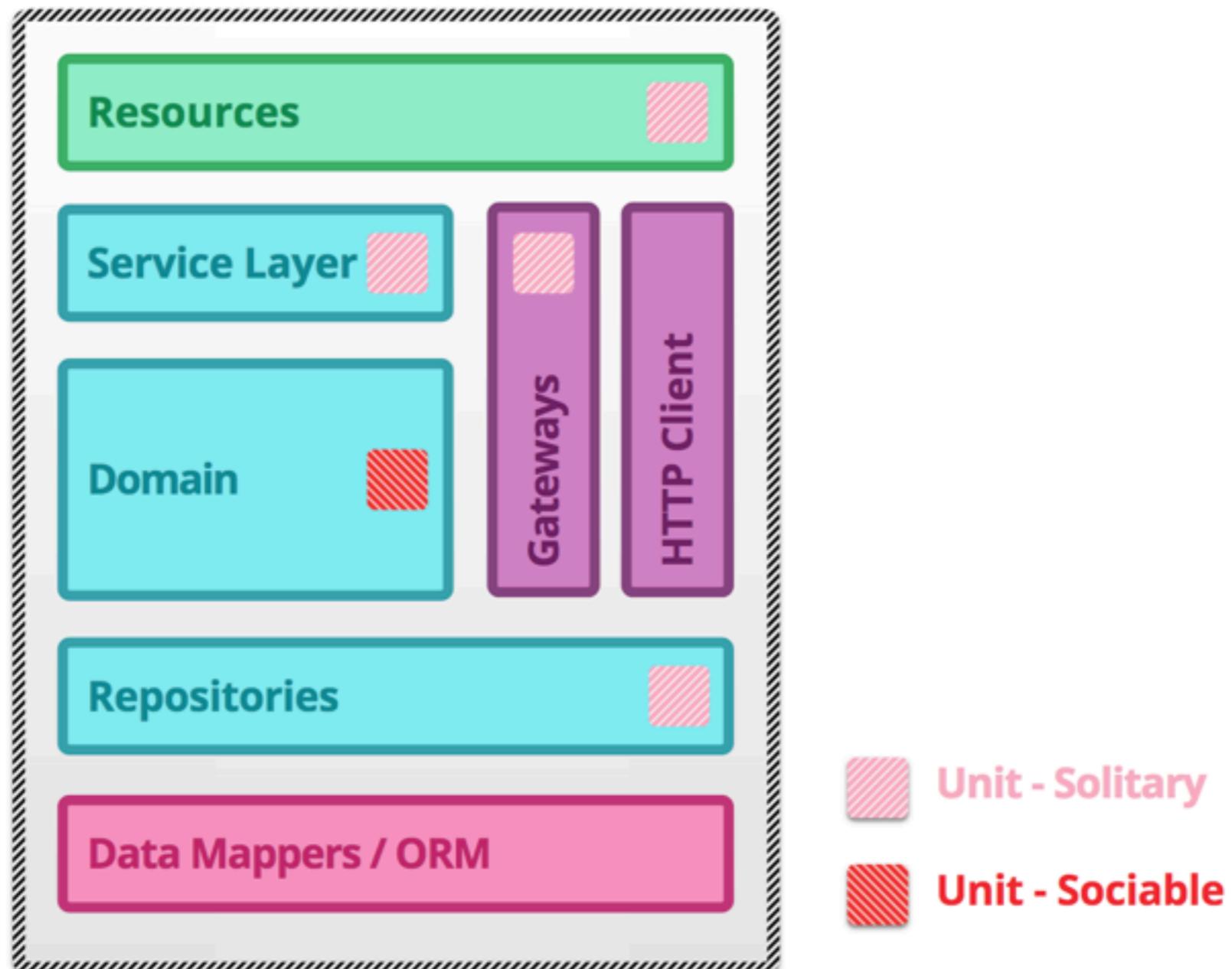


# How to test ?

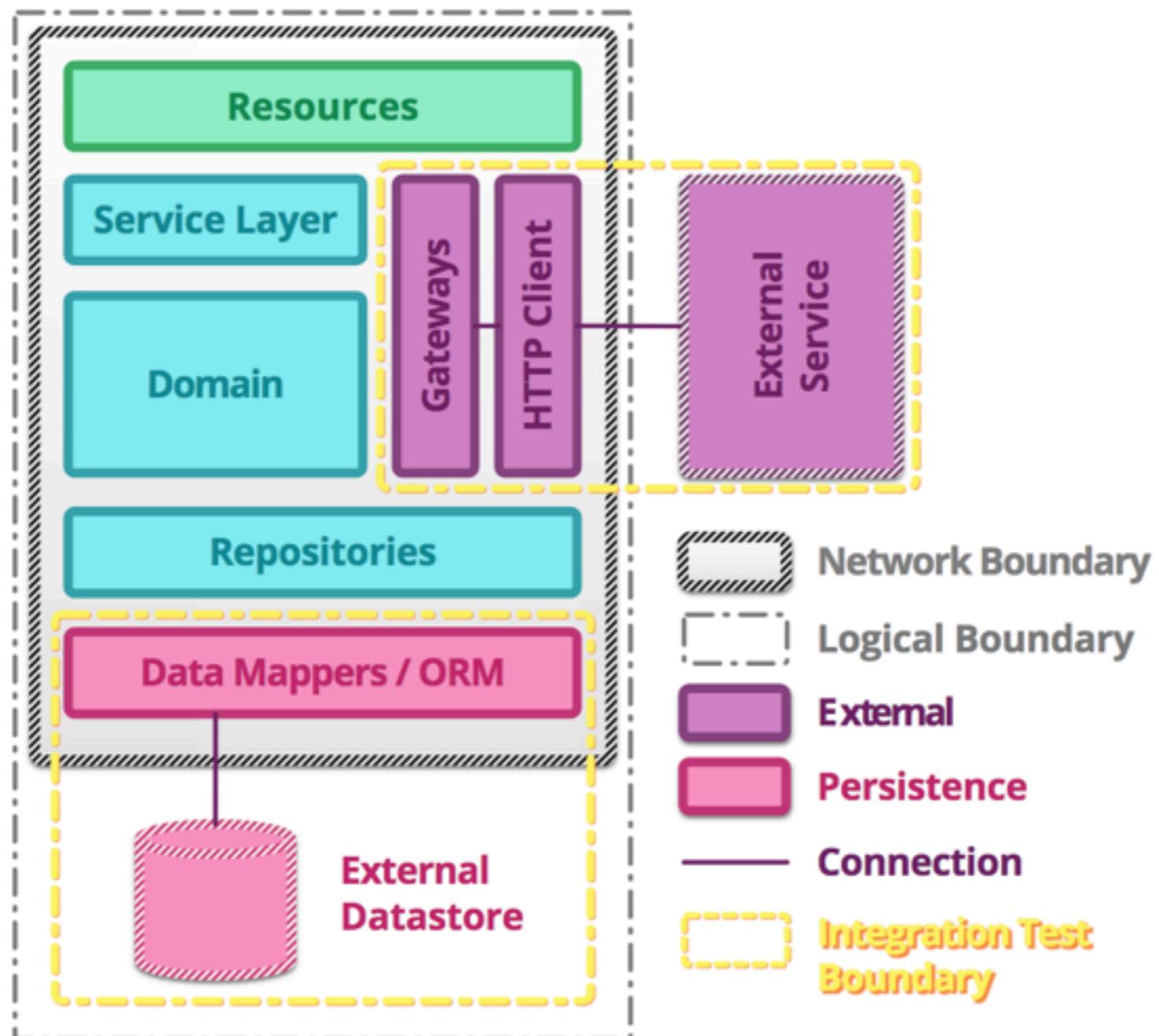




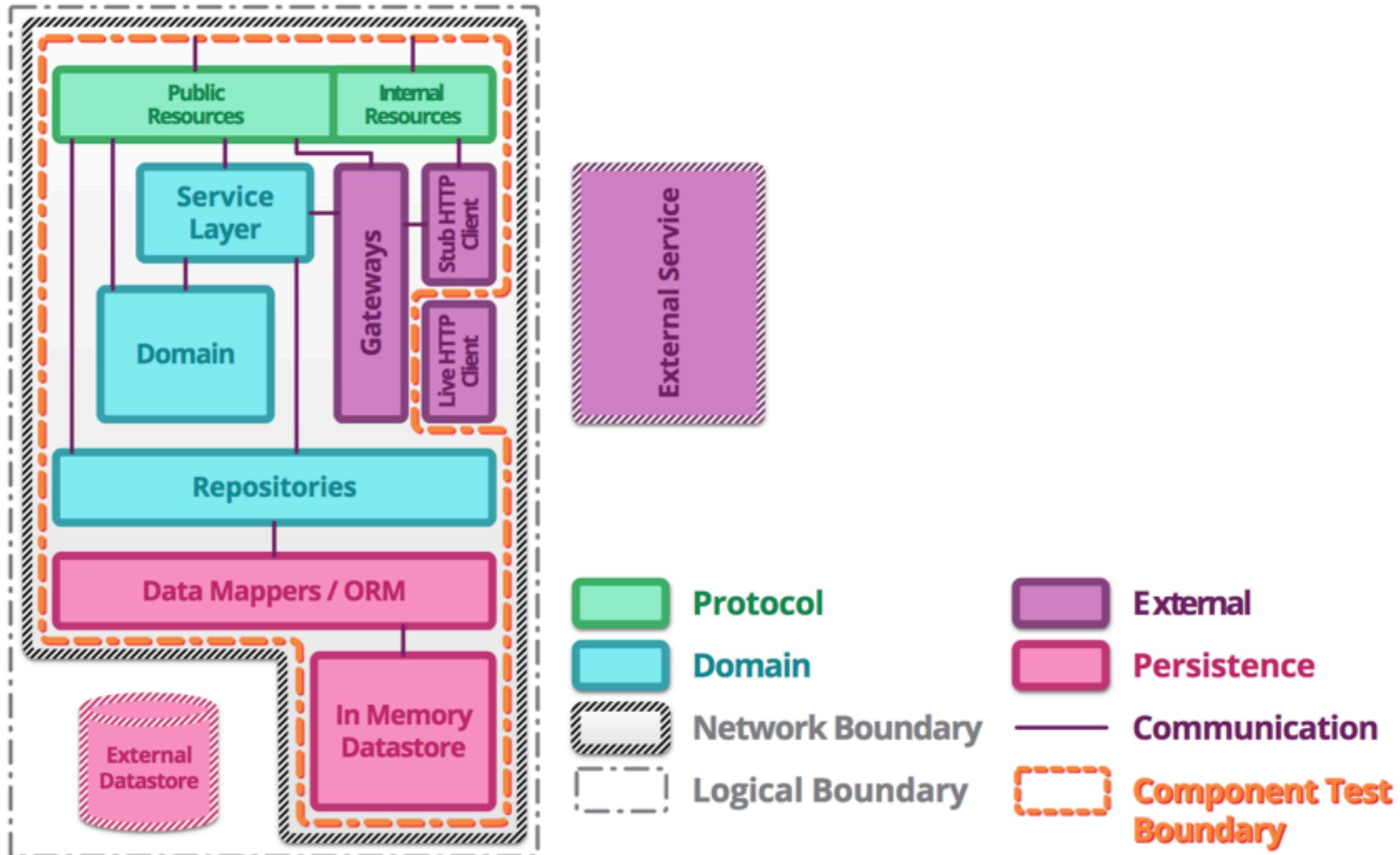
# Unit testing



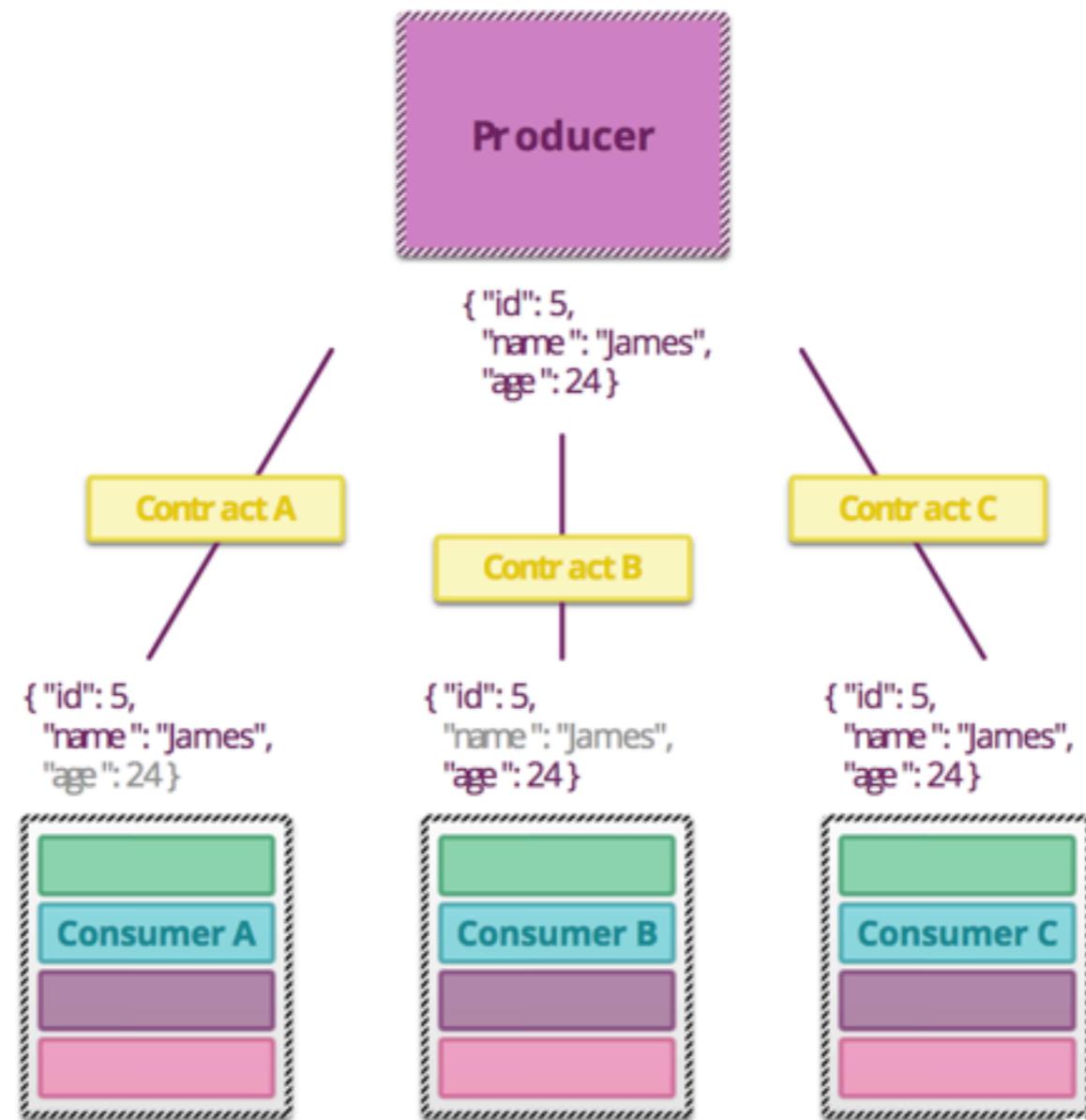
# Integration testing



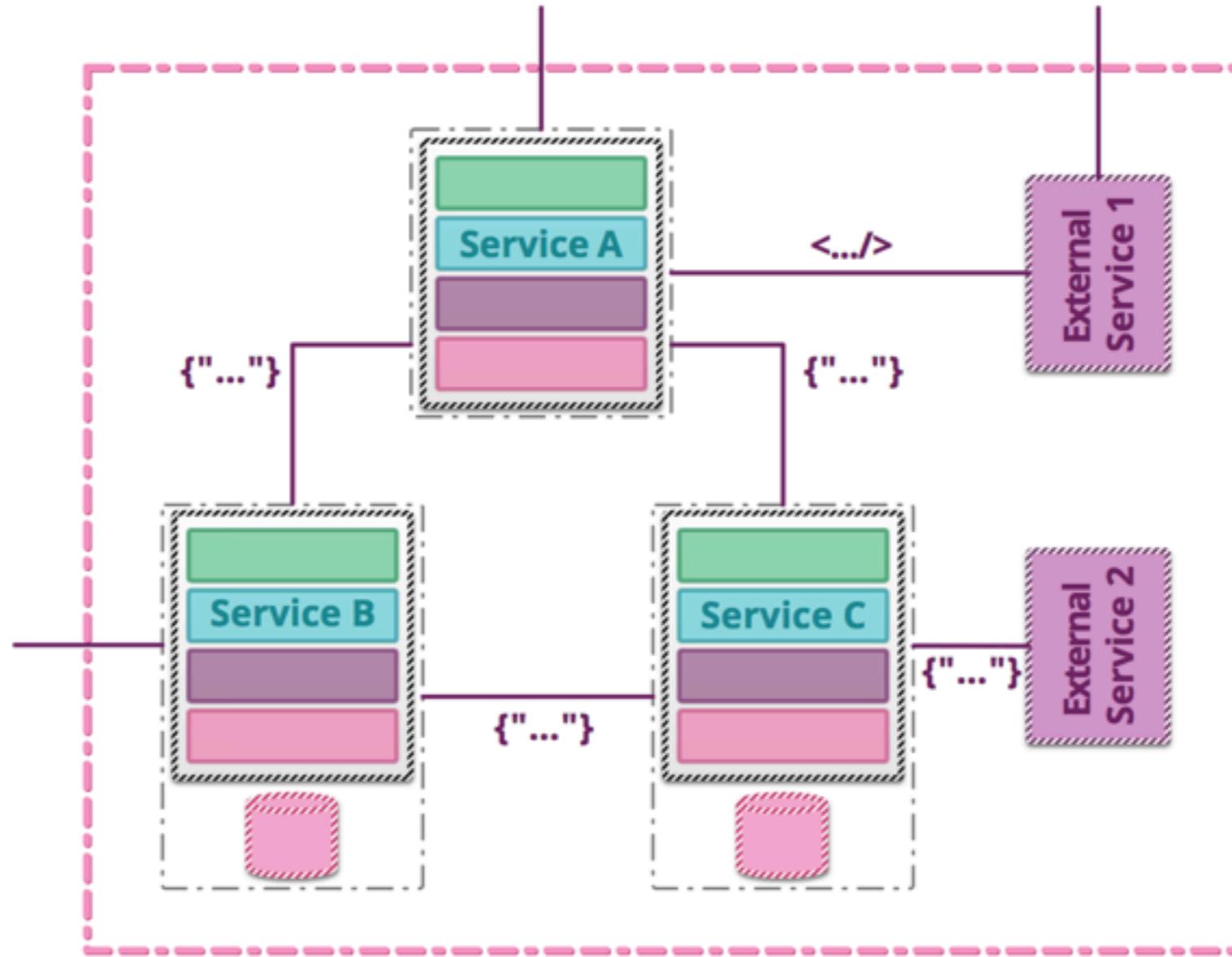
# Component testing



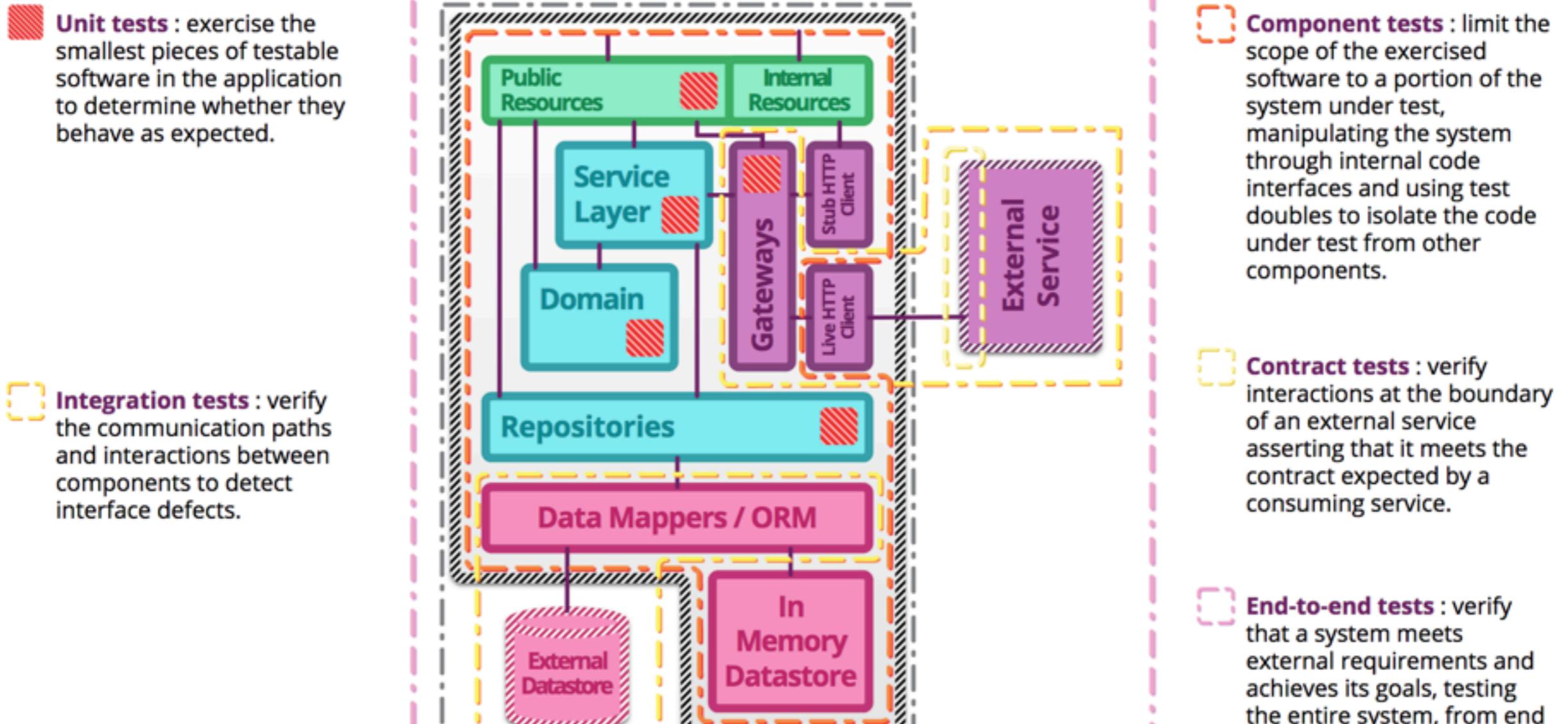
# Contract testing



# End-to-End testing



# Summary



**Unit tests** : exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

**Integration tests** : verify the communication paths and interactions between components to detect interface defects.

**Component tests** : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

**Contract tests** : verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.

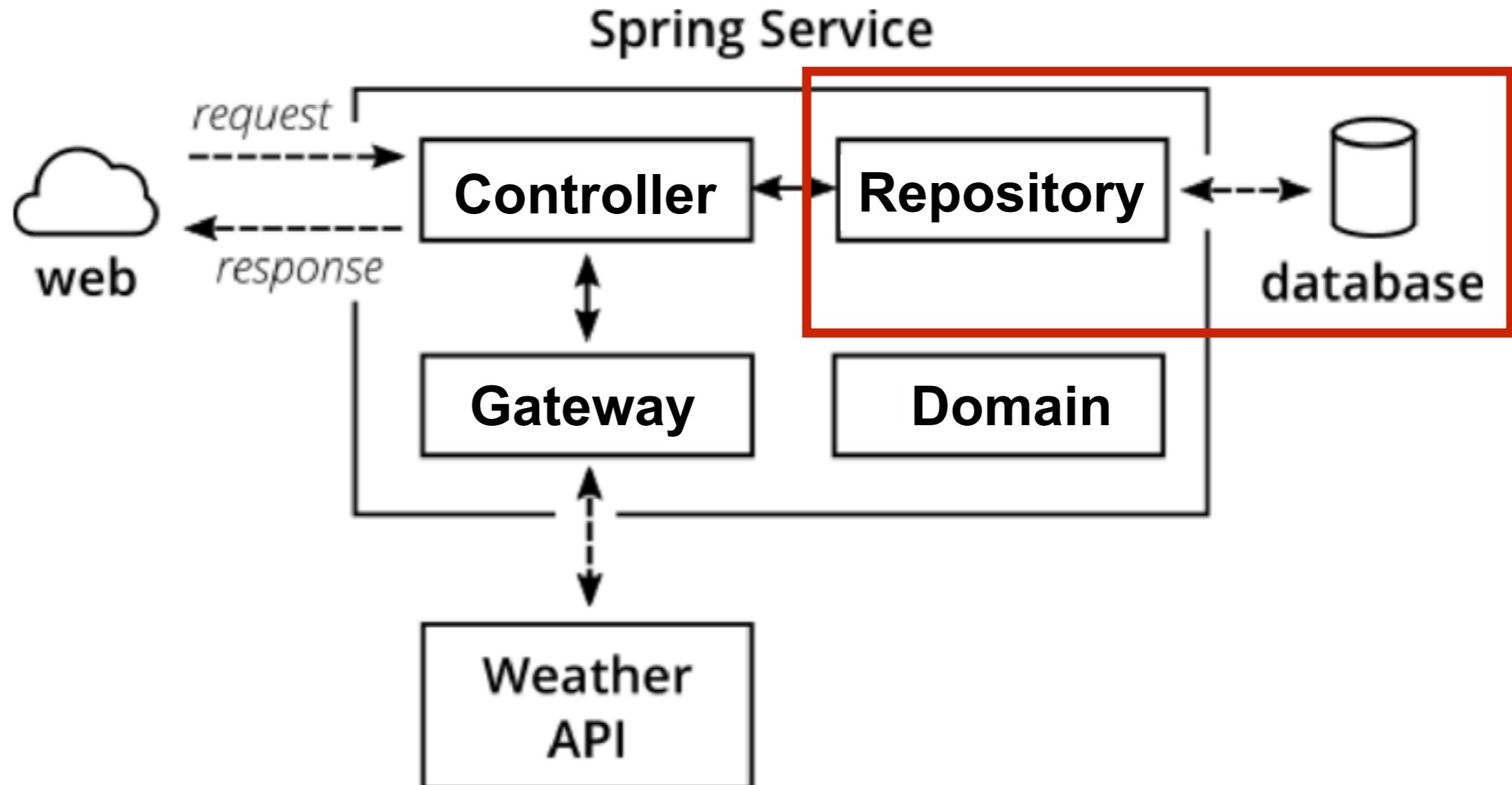
**End-to-end tests** : verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.



# **Let's workshop with Repository**



# Working with repository



# Working with repository

We're using Spring Data



# Modify pom.xml

Add library of Spring Data

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```



# Modify pom.xml

Add library of Persistence/data store

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.1.1</version>
</dependency>
```



# Add data store config

In src/main/resources

```
spring.datasource.url= jdbc:postgresql://127.0.0.1:15432/postgres
spring.datasource.username= user
spring.datasource.password= password
spring.datasource.platform= POSTGRESQL

spring.jpa.show-sql= true
spring.jpa.hibernate.ddl-auto= create-drop
spring.jpa.database-platform= org.hibernate.dialect.PostgreSQLDialect
```



# Create repository interface

hello.repository.PersonRepository.java

```
public interface PersonRepository  
    extends CrudRepository<Person, String> {  
  
    Optional<Person> findByFirstName(String name);  
}
```



# Create Entity class

hello.repository.Person.java

```
@Entity
public class Person {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String firstName;
    private String lastName;

    public Person() {
    }

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```



# Create new controller

hello.repository.HelloControllerWithRepository.java

```
public class HelloControllerWithRepository {  
  
    private final PersonRepository personRepository;  
  
    @Autowired  
    public HelloControllerWithRepository(PersonRepository personRepository) {  
        this.personRepository = personRepository;  
    }  
  
    @GetMapping("/hello/data/{name}")  
    public Hello sayHi(@PathVariable String name) {  
        Optional<Person> foundPerson = personRepository.findByName(name);  
        String result = foundPerson  
            .map(person -> String.format("Hello %s", person.getFirstName()))  
            .orElse( other: "Data not found");  
        return new Hello(result);  
    }  
}
```



# Run test and package

\$mvn clean package

Cobertura Report generation was successful.

Cobertura 2.1.1 - GNU GPL License (NO WARRANTY) - See COPYRIGHT file

Cobertura: Loaded information on 3 classes.

time: 125ms

Cobertura Report generation was successful.

---

BUILD SUCCESS

---



# Run your application

```
$java -jar target/hello.jar
```



# Coverage report

## Packages

All  
[hello](#)  
[hello.controller](#)  
[hello.domain](#)  
[hello.repository](#)

## Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	6	25%  2/8	N/A	N/A
<a href="#">hello</a>	1	33%  1/3	N/A	N/A
<a href="#">hello.controller</a>	2	20%  2/10	N/A	N/A
<a href="#">hello.domain</a>	1	66%  4/6	N/A	N/A
<a href="#">hello.repository</a>	2	0%  0/9	N/A	N/A

Report generated by Cobertura 2.1.1 on 3/6/18 8:52 AM.

## All Packages

## Classes

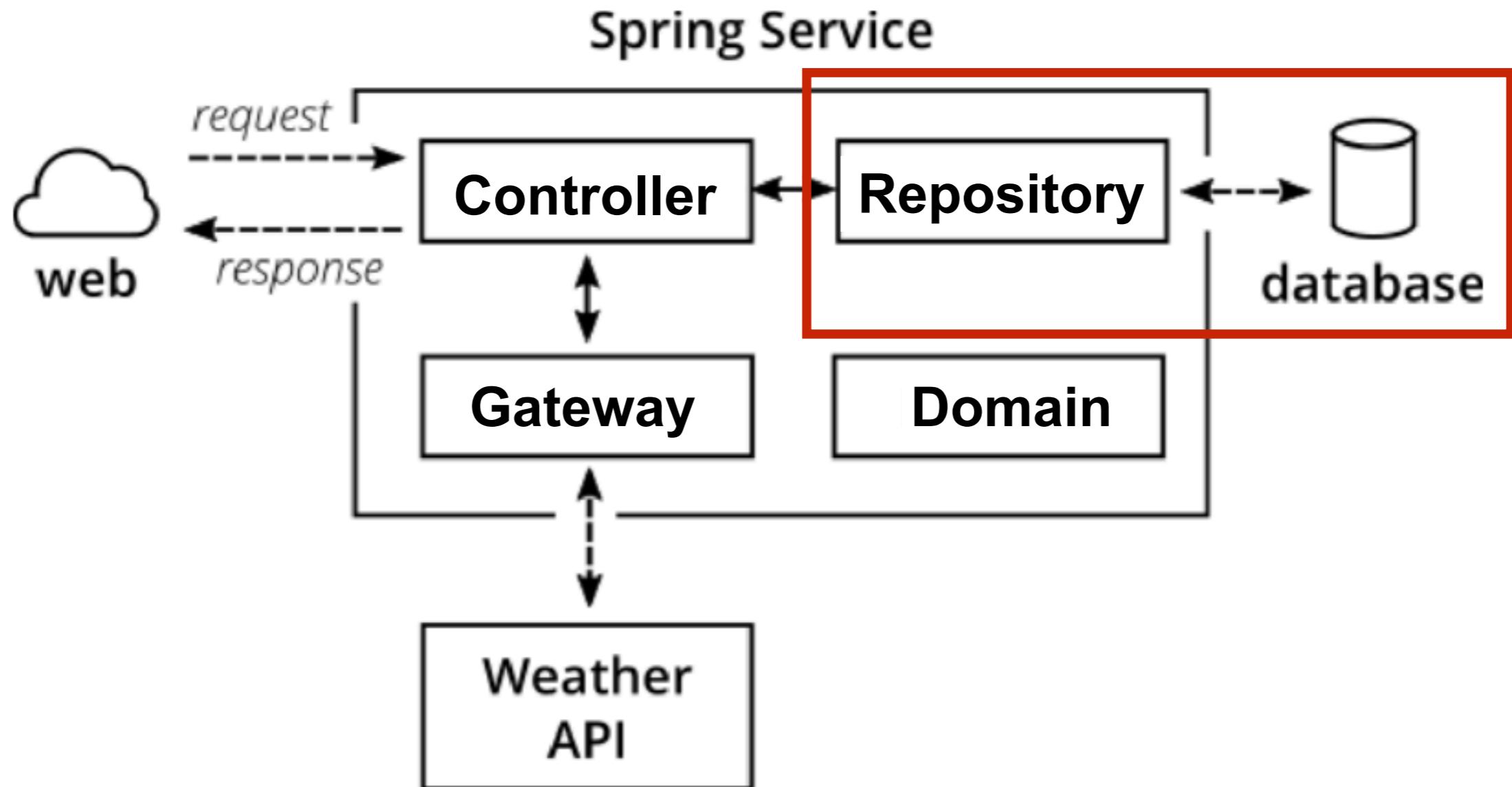
[Hello](#) (66%)  
[HelloApplication](#) (33%)  
[HelloController](#) (100%)  
[HelloControllerWithRepository](#) (0%)  
[Person](#) (0%)  
[PersonRepository](#) (N/A)



# How to test ?



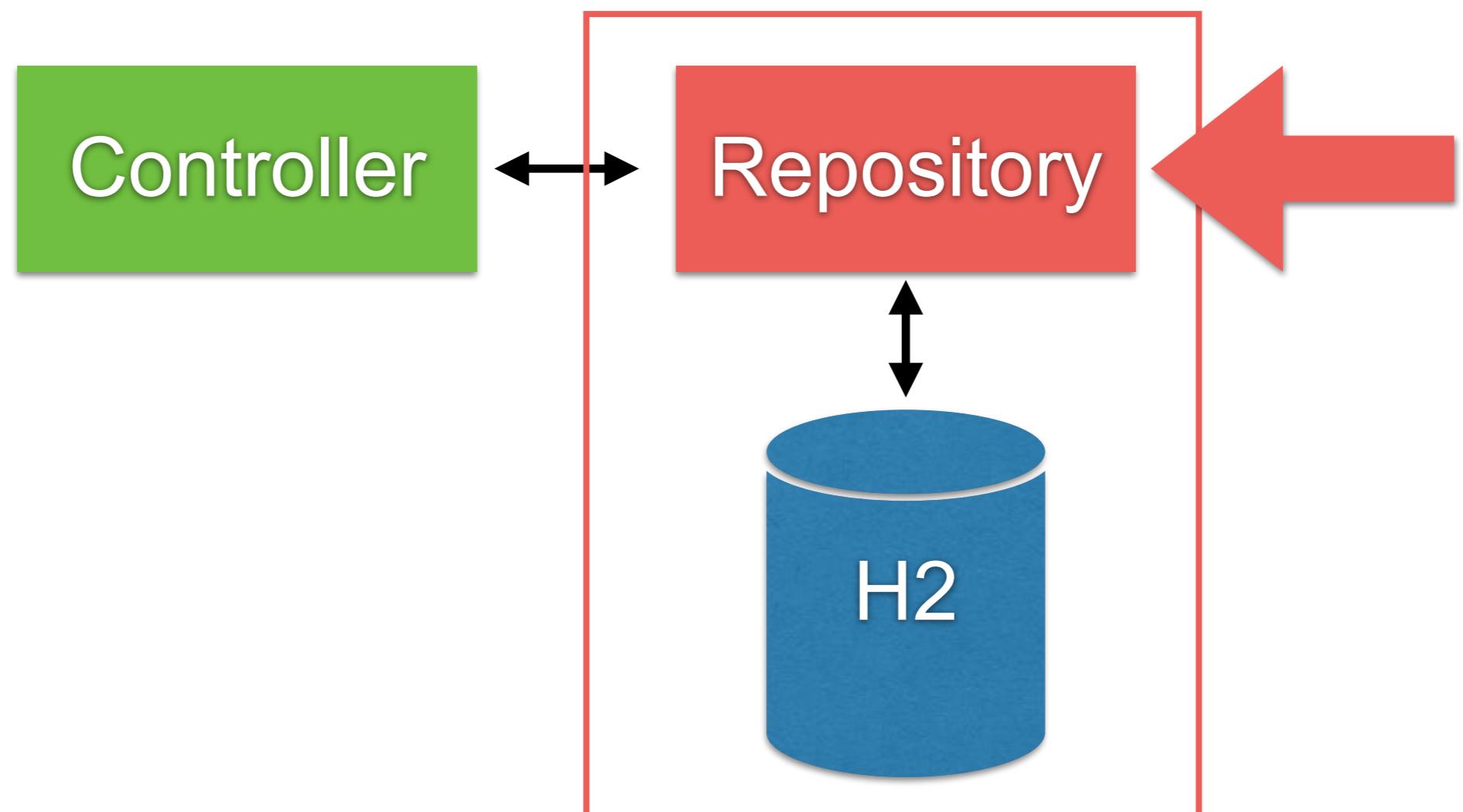
# How to test with Repository ?



# Repository Testing

Using `@DataJpaTest`

Spring Testing



# Spring boot provide DataJpaTest

should be add H2 library to pom.xml

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>
```



# Repository Testing (1)

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class PersonRepositoryTest {

    @Autowired
    private PersonRepository personRepository;

    @After
    public void clearData() {
        personRepository.deleteAll();
    }
}
```



# Repository Testing (2)

Add a test case

```
@Test  
public void shouldSaveAndGetData() throws Exception {  
    //Arrange  
    Person somkiat = new Person("somkiat", "pui");  
    personRepository.save(somkiat);  
  
    Optional<Person> shouldSomkiat  
        = personRepository.findByFirstName("somkiat");  
  
    assertEquals(expected: "somkiat",  
        shouldSomkiat.get().getFirstName());  
}
```



# Run test and package

\$mvn clean package

Cobertura Report generation was successful.

Cobertura 2.1.1 - GNU GPL License (NO WARRANTY) - See COPYRIGHT file

Cobertura: Loaded information on 3 classes.

time: 125ms

Cobertura Report generation was successful.

---

BUILD SUCCESS

---

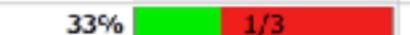


# Coverage report

## Packages

All  
[hello](#)  
[hello.controller](#)  
[hello.domain](#)  
[hello.repository](#)

## Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	6	53%  15/28	N/A	N/A
<a href="#">hello</a>	1	33%  1/3	N/A	N/A
<a href="#">hello.controller</a>	2	20%  2/10	N/A	N/A
<a href="#">hello.domain</a>	1	66%  4/6	N/A	N/A
<a href="#">hello.repository</a>	2	88%  8/9	N/A	N/A

Report generated by [Cobertura](#) 2.1.1 on 3/6/18 9:32 AM.

## All Packages

### Classes

[Hello \(66%\)](#)  
[HelloApplication \(33%\)](#)  
[HelloController \(100%\)](#)  
[HelloControllerWithRepository \(0%\)](#)  
[Person \(88%\)](#)  
[PersonRepository \(N/A\)](#)

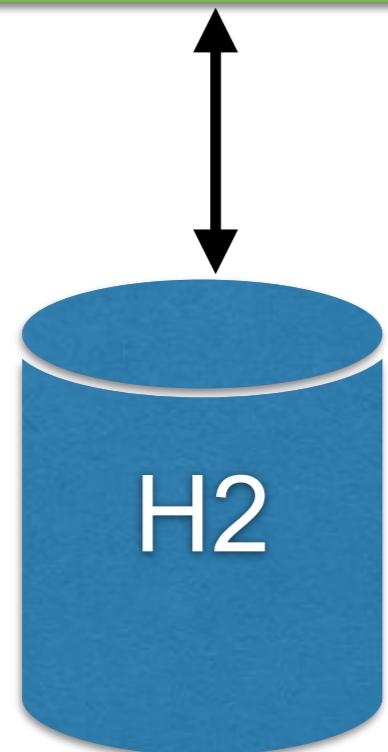


# Working with Real Database

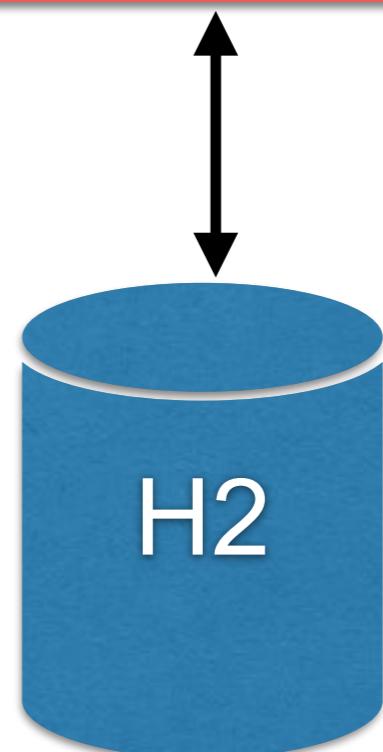


# Working with Database

Production



Testing



# Initial database 1

## Using @PostConstruct

```
@PostConstruct  
public void initData() {  
    Account account1 = new Account();  
    account1.setAccountId("01");  
    accountRepository.save(account1);  
    Account account2 = new Account();  
    account2.setAccountId("02");  
    accountRepository.save(account2);  
}
```



# Initial database 2

**Schema** (resources/schema.sql)

**Data** (resources/data.sql)

## Schema.sql

```
CREATE TABLE account(
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    account_Id VARCHAR(16) NOT NULL UNIQUE,
    mobile_No VARCHAR(10),
    name VARCHAR(50),
    account_Type CHAR(2)
);
```

## Data.sql

```
INSERT INTO account (account_Id) VALUES ('01');
INSERT INTO account (account_Id) VALUES ('02');
```



# Initial database 2

Disable auto generate DDL from JPA in file application.yml

```
spring:  
  jpa:  
    show-sql: true  
    hibernate:  
      ddl-auto: none
```



# Initial database 2

## Problem with naming strategy !!

```
spring:  
  jpa:  
    show-sql: true  
    hibernate:  
      ddl-auto: none  
      naming:  
        physical-strategy:  
          org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy  
        implicit-strategy:  
          org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
```

<https://github.com/spring-projects/spring-boot/tree/master/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/orm/jpa/hibernate>



# Run and see from logging

Execute file schema.sql and data.sql

```
o.s.jdbc.datasource.init.ScriptUtils      : Executing SQL script from URL [file:/Users/somkiat/dat
o.s.jdbc.datasource.init.ScriptUtils      : Executed SQL script from URL [file:/Users/somkiat/data
49 ms.
o.s.jdbc.datasource.init.ScriptUtils      : Executing SQL script from URL [file:/Users/somkiat/dat
o.s.jdbc.datasource.init.ScriptUtils      : Executed SQL script from URL [file:/Users/somkiat/data
ms.
```



# Run service

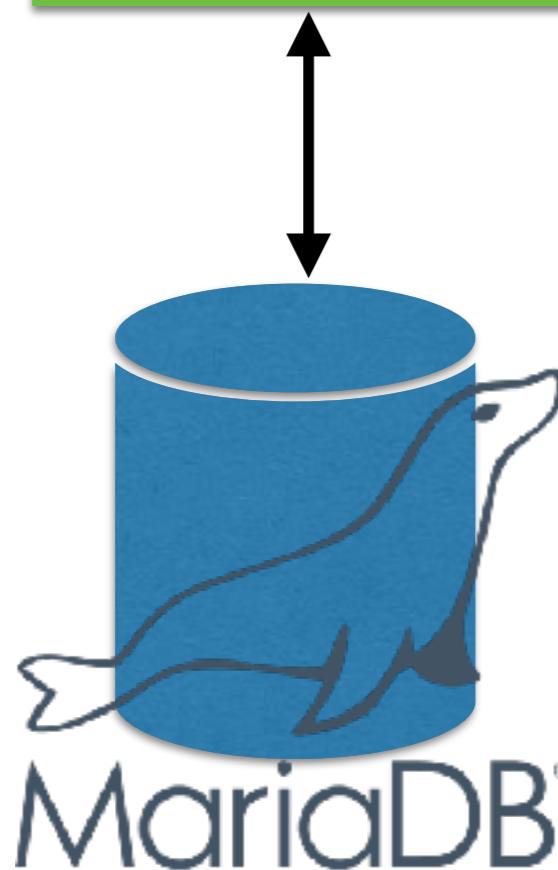
<http://localhost:8080/account/0868696209>

```
← → ⌂ ⓘ localhost:8080/account/0868696209
[
  - {
    accountNo: "01",
    mobileNo: null,
    name: "",
    accountType: ""
  },
  - {
    accountNo: "02",
    mobileNo: null,
    name: "",
    accountType: ""
  }
]
```

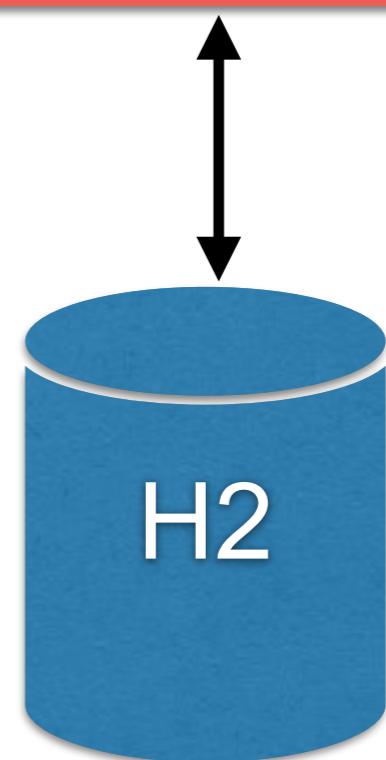


# Working with Database

Production



Testing



# Add database dependency

In file pom.xml

```
<dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
    <version>2.2.5</version>
</dependency>
```

<https://mariadb.com/kb/en/library/about-mariadb-connector-j/>



# Config database in application

In file resources/application.yml

```
spring:
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://35.198.254.195:3306/account
    username: user01
    password: password

    initialization-mode: always
    testWhileIdle: true
    validationQuery: SELECT 1

  jpa:
    show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL5InnoDBDialect
```



# Config database in application

## Required config for database

```
spring:
```

```
  datasource:  
    driver-class-name: org.mariadb.jdbc.Driver  
    url: jdbc:mariadb://35.198.254.195:3306/account  
    username: user01  
    password: password
```

```
    initialization-mode: always
```

```
    testWhileIdle: true
```

```
    validationQuery: SELECT 1
```

```
jpa:
```

```
  show-sql: true
```

```
  properties:
```

```
    hibernate:
```

```
      dialect: org.hibernate.dialect.MySQL5InnoDBDialect
```



# Config database in application

## Required config for database

```
spring:  
  datasource:  
    driver-class-name: org.mariadb.jdbc.Driver  
    url: jdbc:mariadb://35.198.254.195:3306/account  
    username: user01  
    password: password
```

```
  initialization-mode: always
```

```
  testWhileIdle: true  
  validationQuery: SELECT 1  
  
  jpa:  
    show-sql: true  
    properties:  
      hibernate:  
        dialect: org.hibernate.dialect.MySQL5InnoDBDialect
```

By default not run



# Run service

<http://localhost:8080/account/0868696209>

```
← → ⌂ ⓘ localhost:8080/account/0868696209
[
  - {
    accountNo: "01",
    mobileNo: null,
    name: "",
    accountType: ""
  },
  - {
    accountNo: "02",
    mobileNo: null,
    name: "",
    accountType: ""
  }
]
```



# Check your test ?

\$mvnw clean test



# Step to test

Stop the real database before run test

See result



# Error ?

Spring boot try to connect to the real database ?



# Fix Error

Create file /resources/application.yml in test folder

spring: **config of H2 database**

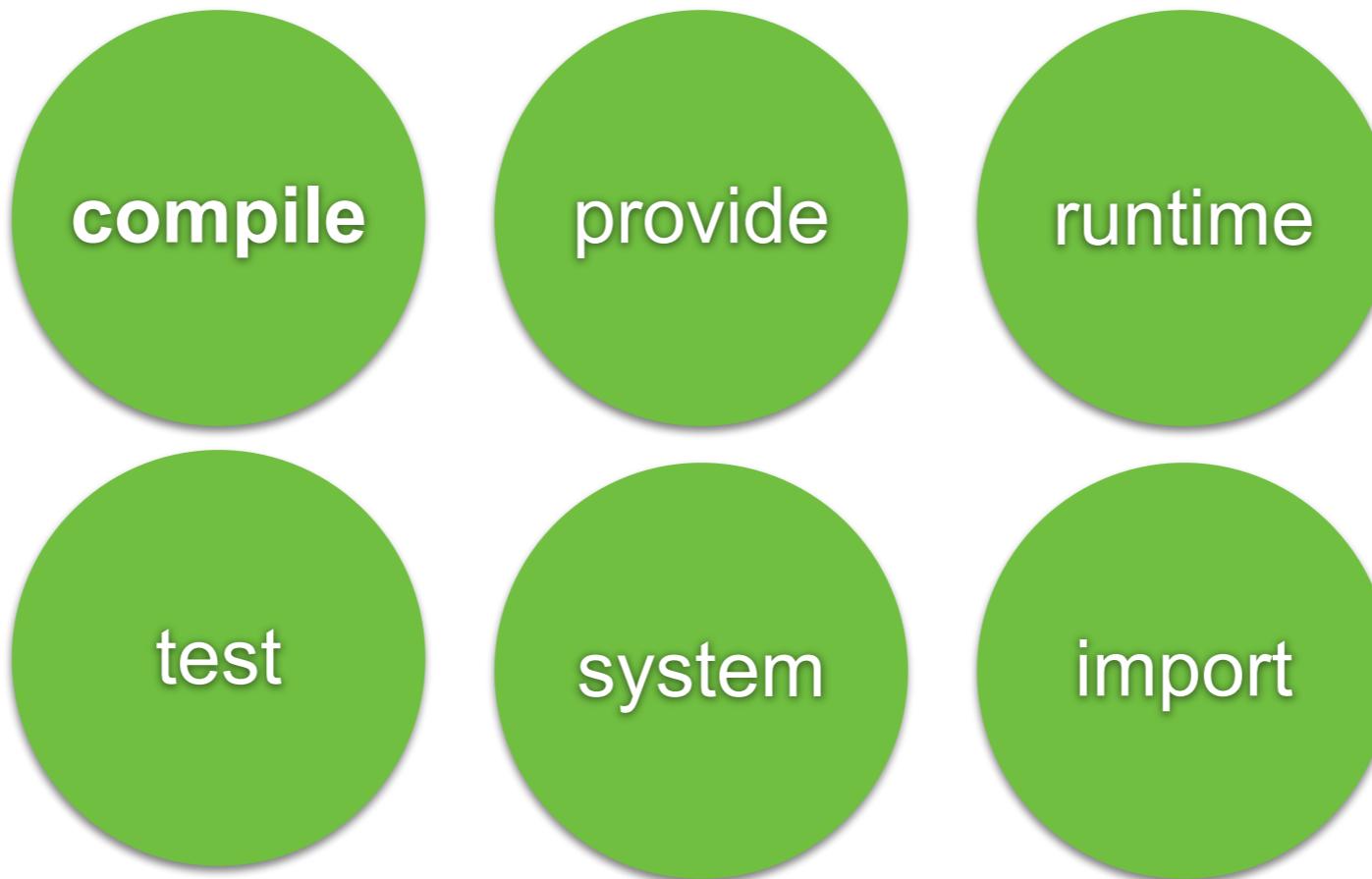
```
datasource:  
  driver-class-name: org.h2.Driver  
  url: jdbc:h2:mem:db;DB_CLOSE_DELAY=-1  
  username: sa  
  password: sa
```

```
jpa:  
  show-sql: true  
  properties:  
    hibernate:  
      dialect: org.hibernate.dialect.H2Dialect
```



# Question ?

Dependency Scopes in file pom.xml  
runtime ?



[https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#Dependency\\_Scope](https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#Dependency_Scope)



# Question ?

show\_sql=true not working ?

```
jpa:  
  show_sql: true  
  hibernate:  
    ddl-auto: none  
  properties:  
    hibernate:  
      format_sql: true  
    dialect: org.hibernate.dialect.MySQL5InnoDBDialect
```

```
logging:  
  level:  
    org.hibernate.SQL: DEBUG
```

**Beautiful sql format**

**Default log level = INFO**

<http://www.baeldung.com/sql-logging-spring-boot>



# Result of logging message

```
13:18:36.336 INFO 79616 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : initialization completed in 37 ms
13:18:36.864 INFO 79616 --- [nio-8080-exec-2] o.h.h.i.QueryTranslatorFactory      : HHH000397: Using ASTQueryTranslatorFactory
13:18:37.138 DEBUG 79616 --- [nio-8080-exec-2] org.hibernate.SQL
select account0_.id as id1_0_, account0_.account_id as account_2_0_, ac
 as account_3_0_, account0_.mobile_no as mobile_n4_0_, account0_.name a
unt account0_
```



# Log Levels

Level	Color
FATAL	Red
ERROR	Red
WARN	Yellow
<b>INFO (default)</b>	Green
DEBUG	Green
TRACE	Green

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-logging.html>



# Testing with Mocking



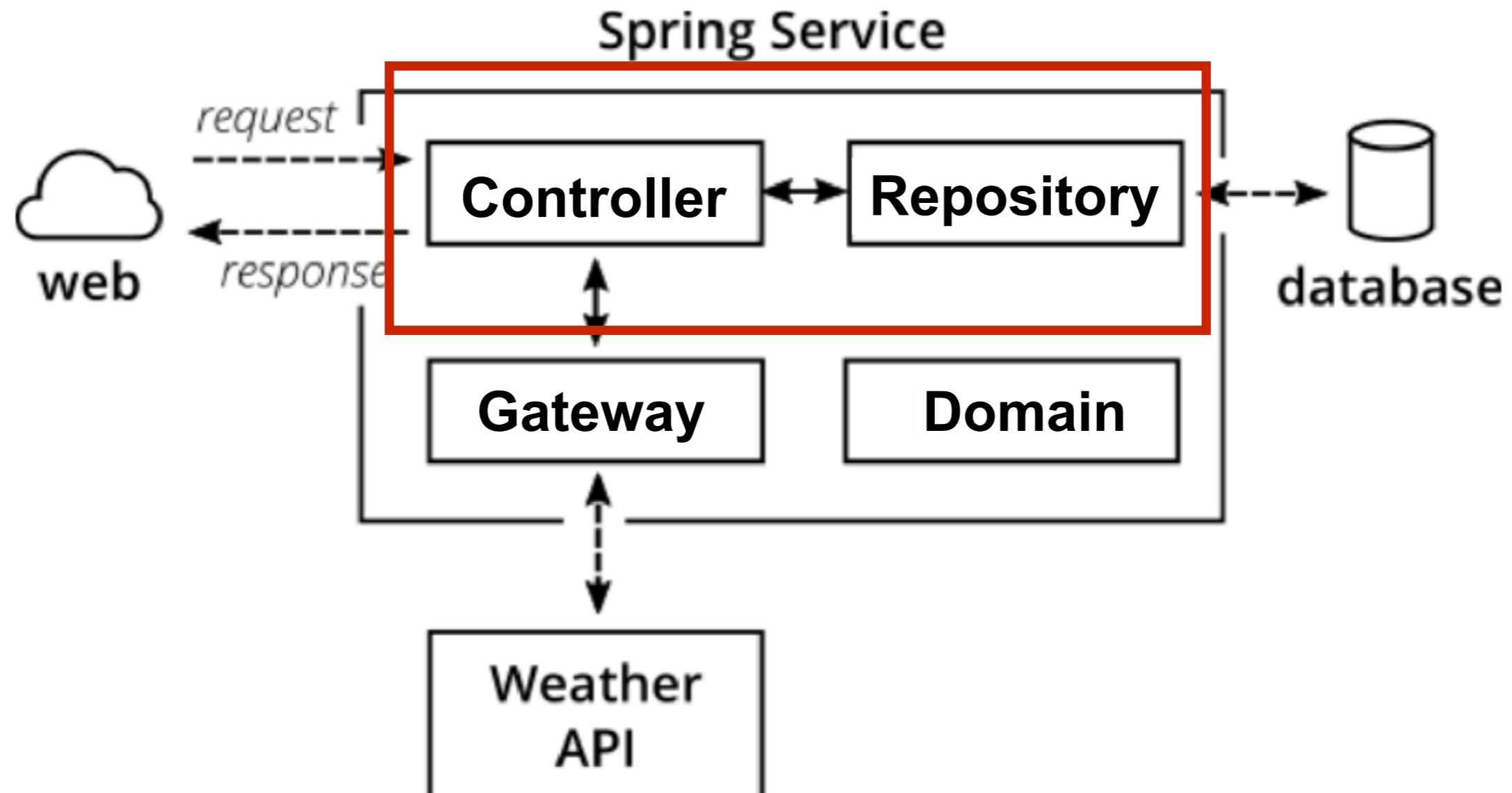
# Controller/Service Testing

Unit testing ?

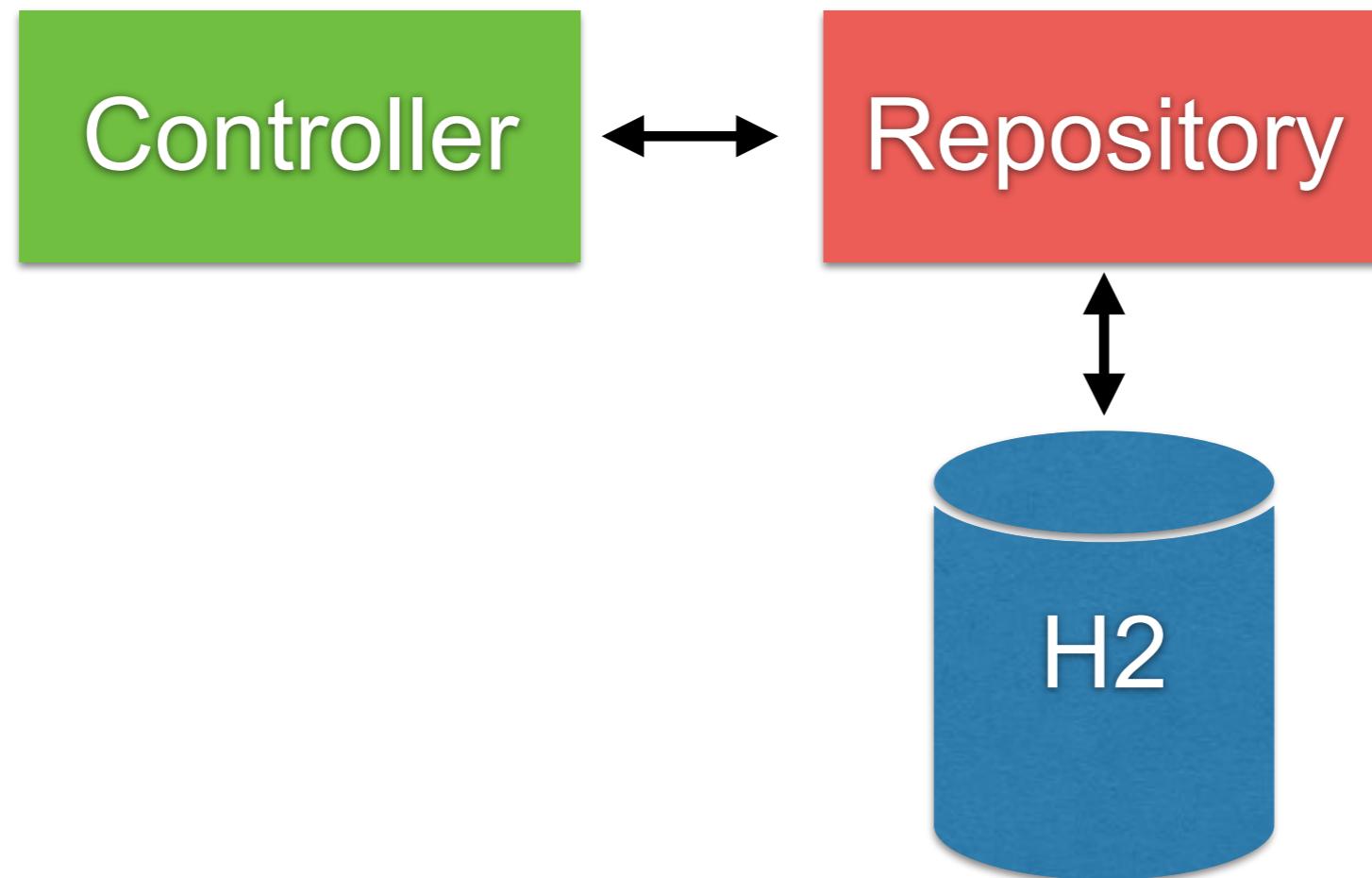
Spring Unit testing with MockMvc ?



# Controller Testing with Unit test



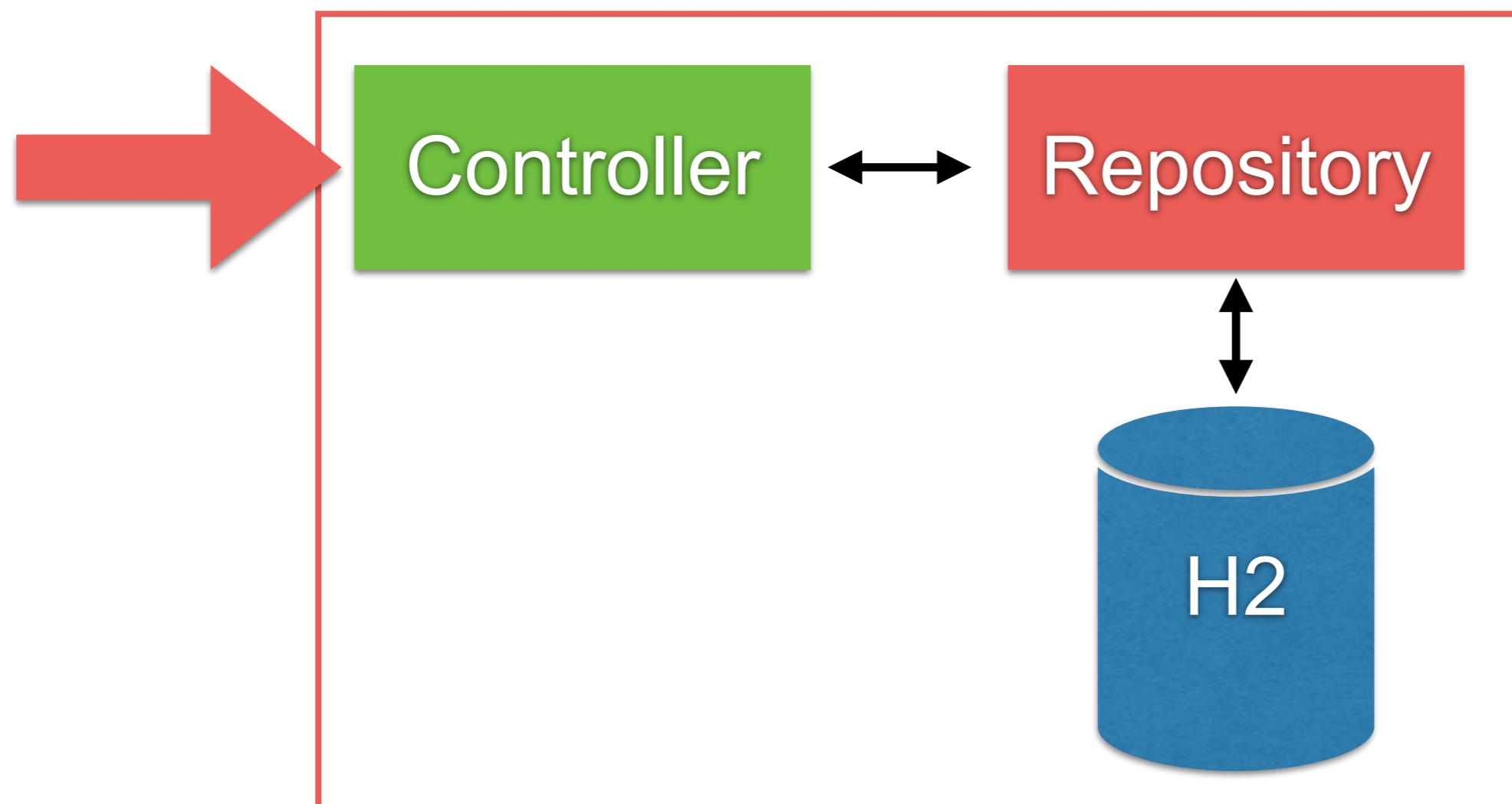
# Controller Testing



# Controller Testing

Using `@SpringBootTest`

Spring Testing



# Unit test

Use Test Double

In java, use Mockito library



<http://site.mockito.org/>



# Unit test with Mockito (1)

```
public class HelloControllerWithRepositoryTest {  
    private HelloControllerWithRepository controllerWithRepository;  
  
    @Mock  
    private PersonRepository personRepository;  
  
    @Before  
    public void init() {  
        initMocks(testClass: this);  
        controllerWithRepository  
            = new HelloControllerWithRepository(personRepository);  
    }  
}
```



# Unit test with Mockito (2)

```
@Test
public void shouldReturnHelloSomkiat() {
    //Arrange
    Person somkiat = new Person("somkiat", "pui");
    given(personRepository.findByName("somkiat"))
        .willReturn(Optional.of(somkiat));

    // Action
    Hello hello = controllerWithRepository.sayHi( name: "somkiat");

    // Assert
    assertEquals( expected: "Hello somkiat", hello.getMessage());
}
```



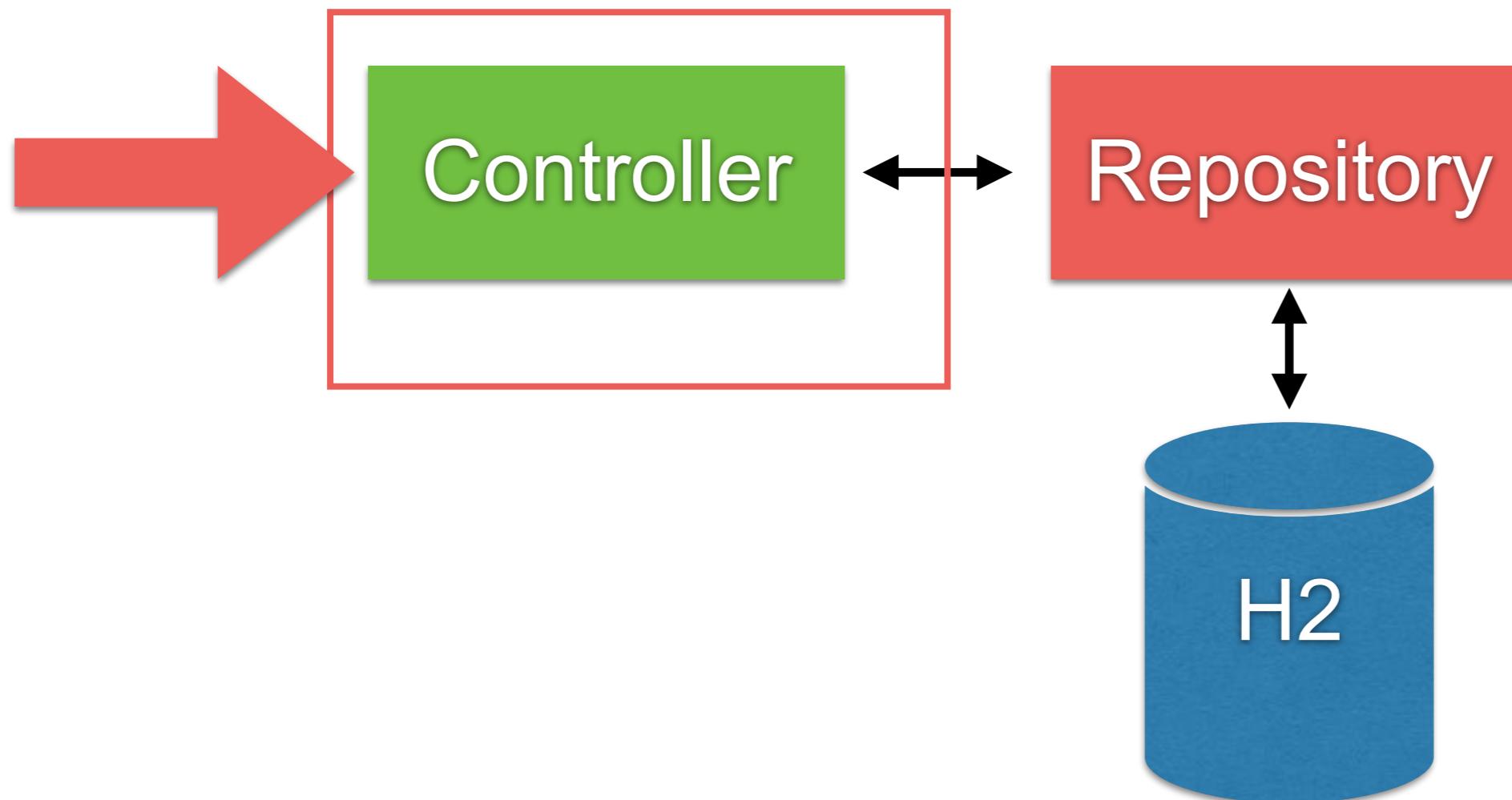
# Try by yourself



# Controller Testing with mock

Using `@SpringBootTest`

Spring Testing



# Controller Testing with mock (1)

```
@RunWith(SpringRunner.class)
@SpringBootTest(
    webEnvironment = WebEnvironment.RANDOM_PORT)
public class AccountControllerMockTest {

    @Autowired
    private TestRestTemplate testRestTemplate;

    @MockBean
    private AccountRepository accountRepository;
```



# Controller Testing with mock (2)

```
@Test  
public void เรียกข้อมูลบัญชีของหมายเลข_0868696209_ต้องเจอสองบัญชีนั่น()  
{  
    // Arrange  
    List<Account> accountIterable = new ArrayList<>();  
    accountIterable.add(new Account("01"));  
    accountIterable.add(new Account("02"));  
    given(accountRepository.findAll()).willReturn(accountIterable);  
  
    // Act  
    List<AccountResponse> accountResponses  
        = testRestTemplate.getForObject(  
            "/account/0868696209", List.class);  
  
    // Assert  
    assertEquals(2, accountResponses.size());  
}
```

# Controller Testing with mock (3)

```
@Test
public void
    เรียกข้อมูลบัญชีของหมายเลข_0868696209_ต้องเจอสองบัญชีนั่น() {
        // Arrange
        List<Account> accountIterable = new ArrayList<>();
        accountIterable.add(new Account("01"));
        accountIterable.add(new Account("02"));
        given(accountRepository.findAll()).willReturn(accountIterable);

        // Act
        List<AccountResponse> accountResponses
            = testRestTemplate.getForObject(
                "/account/0868696209", List.class);

        // Assert
        assertEquals(2, accountResponses.size());
    }
```

Call service with rest template



# Coverage report

## Packages

All  
[toystore](#)  
[toystore.controller](#)  
[toystore.domain](#)  
[toystore.repository](#)

## Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	6	90% 25/28	50% 1/2	1.1
<a href="#">toystore</a>	1	33% 1/3	N/A	1
<a href="#">toystore.controller</a>	2	93% 15/16	50% 1/2	2
<a href="#">toystore.domain</a>	1	100% 4/4	N/A	1
<a href="#">toystore.repository</a>	2	100% 9/9	N/A	1

Report generated by [Cobertura](#) 2.1.1 on 3/6/18 5:20 PM.

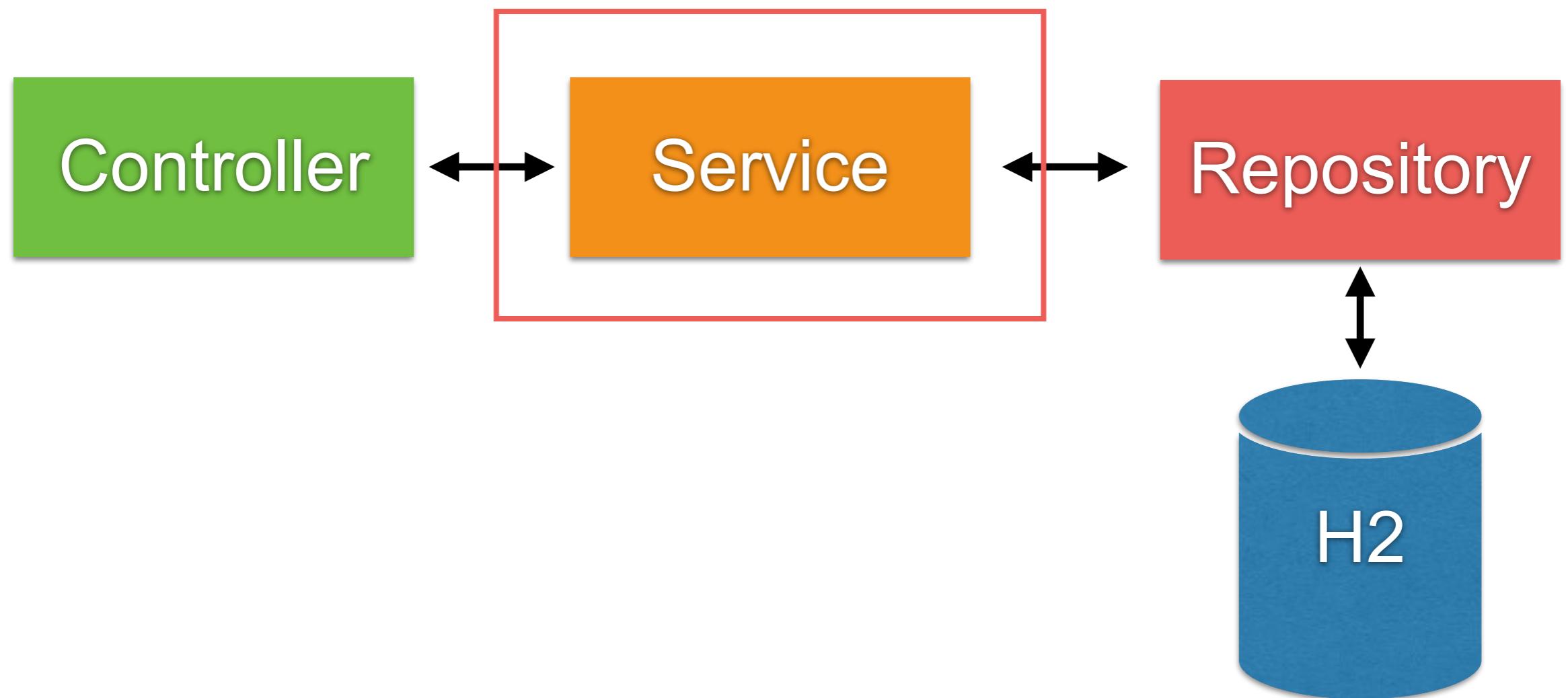
## All Packages

## Classes

[Hello](#) (100%)  
[HelloController](#) (83%)  
[HelloWithRepositoryController](#) (100%)  
[Person](#) (100%)  
[PersonRepository](#) (N/A)  
[ToyStoreApplication](#) (33%)



# Controller Testing with Unit test



# Q/A



# More topics



# Concurrency and Threading



# Key points

Performance  
Responsiveness  
Throughput

