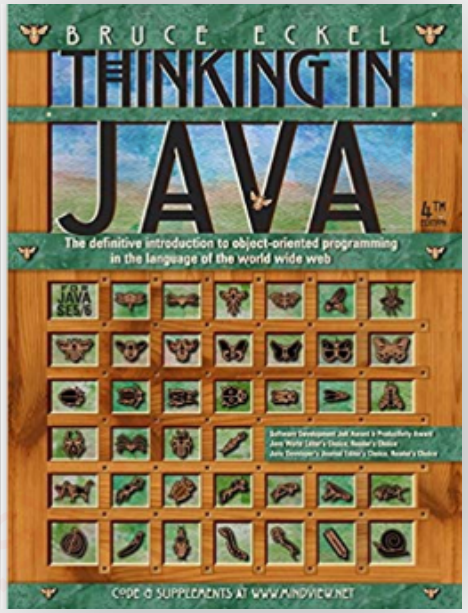
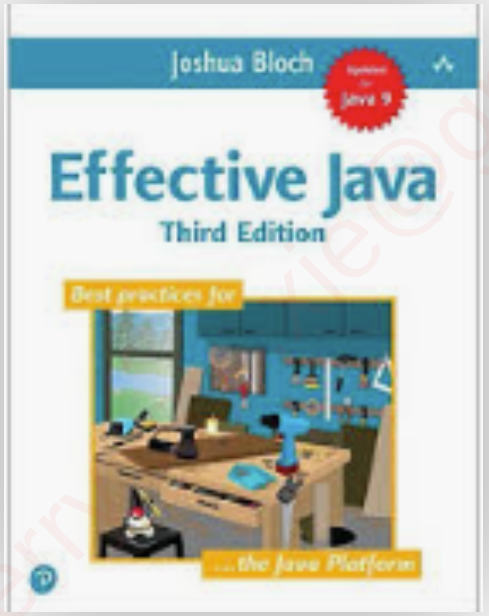
匠人学院
jiangren.com.au

Java Spring Boot
—— Java Basic

Java History

- James Goslin created the Java in 1991.
- Objected Oriented Programming language
- Very very popular among the most popular language list
- Easy to learn and easy to get hands dirty. Why? Garbage collection, JVM, Good IDE Support. Tons of Slack Overflow questions
- Huge community. And a good language to understand OO as well
- Comprehensive and powerful language supporting multi thread programming. NIO programming etc.
- Android using Java, Kotlin


BRUCE ECKEL
THINKING IN
JAVA
4TH EDITION
The definitive introduction to object-oriented programming in the language of the world wide web
CODE IS SUPPLEMENTS AT WWW.MINDVIEW.NET


Joshua Bloch
Updated for Java 9
Effective Java
Third Edition
Best practices for
...the Java Platform


O'REILLY
Head First
Design Patterns
A Brain-Friendly Guide

Java Core

匠人学院
jiangren.com.au

- **OO Concept**
  - Inheritance
  - Polymorphism
  - Encapsulation
  - Composition
  - Container (List, Set, Map)
  - Generics (Diamond operator etc)

Java 8 & Above

- Lambda
- Optional
- Functional programming
- Pass method as parameter

Java Database

- ORM hibernate
- SQL
- Liquibase vs Flyway
- PostgreSQL
- H2

Spring & Spring boot

- Spring data JPA
- Lombok
- MapStruct

System integration

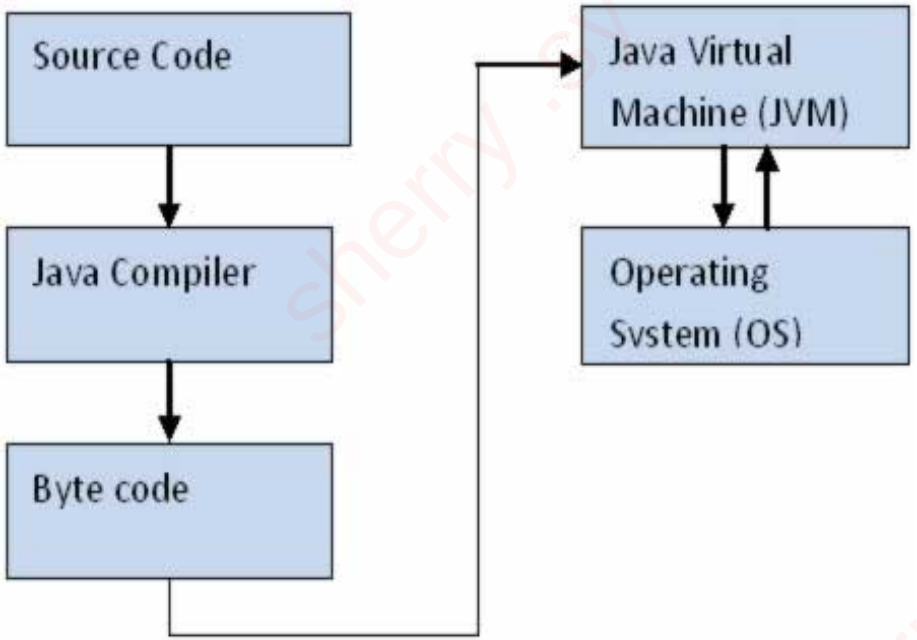
- SAML Oauth2, JWT (gumtree)
- Spring Security

Environment

- Java 11
- IDEA

How does Java work

- Compile Time. Using javac to class files
- Run Time. Using java to run class files



Java 参考资料

- JDK是最好的参考资料
- Google

Variables and Types

Although Java is object oriented, not all types are objects. It is built on top of basic variable types called primitives.

Here is a list of all primitives in Java:

- byte
- short
- int
- long
- float
- double
- char
- boolean

String

String probably is one of the most important object in Java.

- Immutable
- bear in mind always use equals to compare two strings if possible
- Familiar with String manipulations. indexOf, subString, concat, etc.
- Regular Expression

Array

Array is another important data structure in almost any programming language

- Immutable
- Different type of arrays
- Familiar with Array manipulations.

Conditions

- If else then
- Switch case break
- ? : operator
- Operators : == , !=, >, < ,&&, ||

Loop

- for Loop, 2 ways
- While Loop
- forEach. Java8

```
for(data_type item : collection) { ... }
```

Methods and Functions

Important concept in any languages

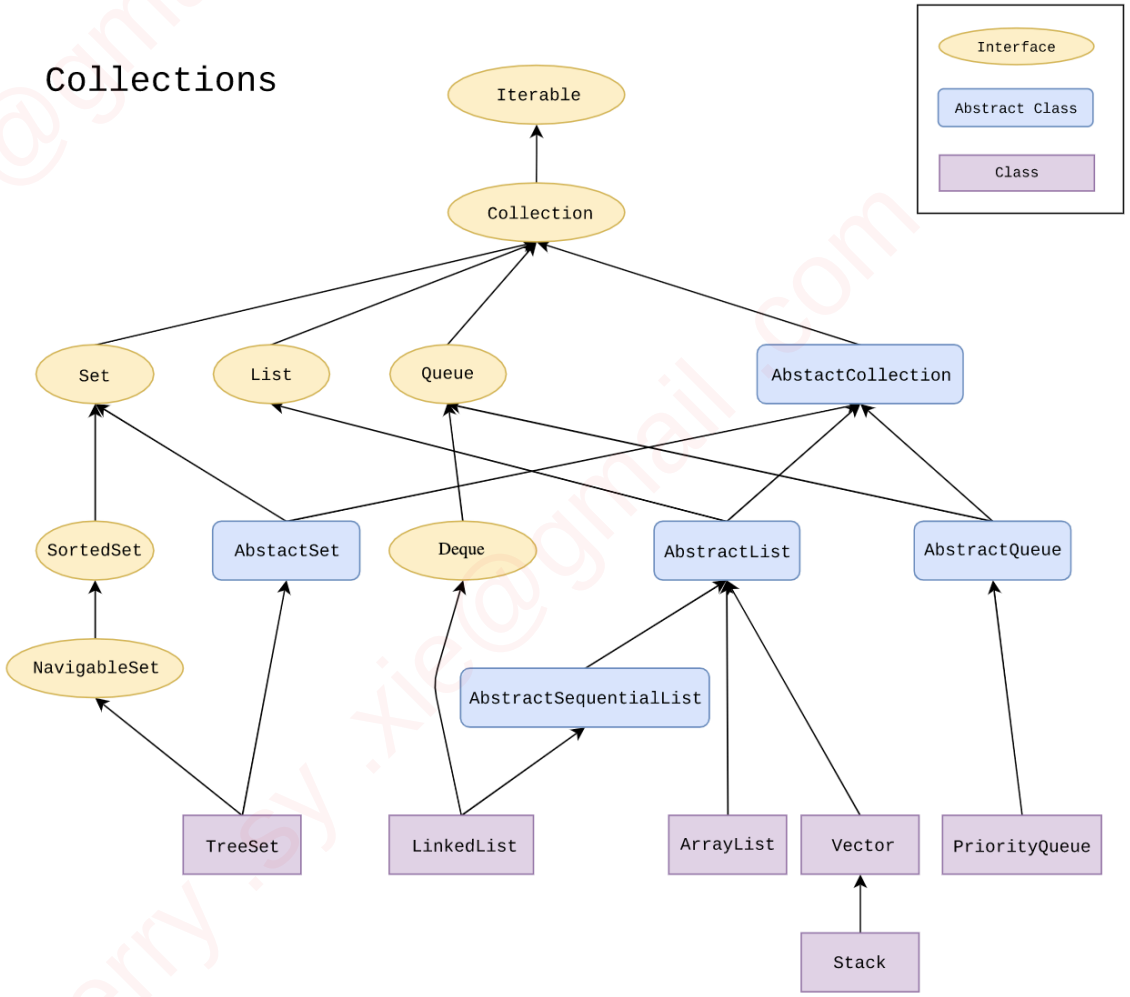
Method signature includes the following:

- Access modifiers. Public private protected
- Return value
- Parameters
- passes the references by value

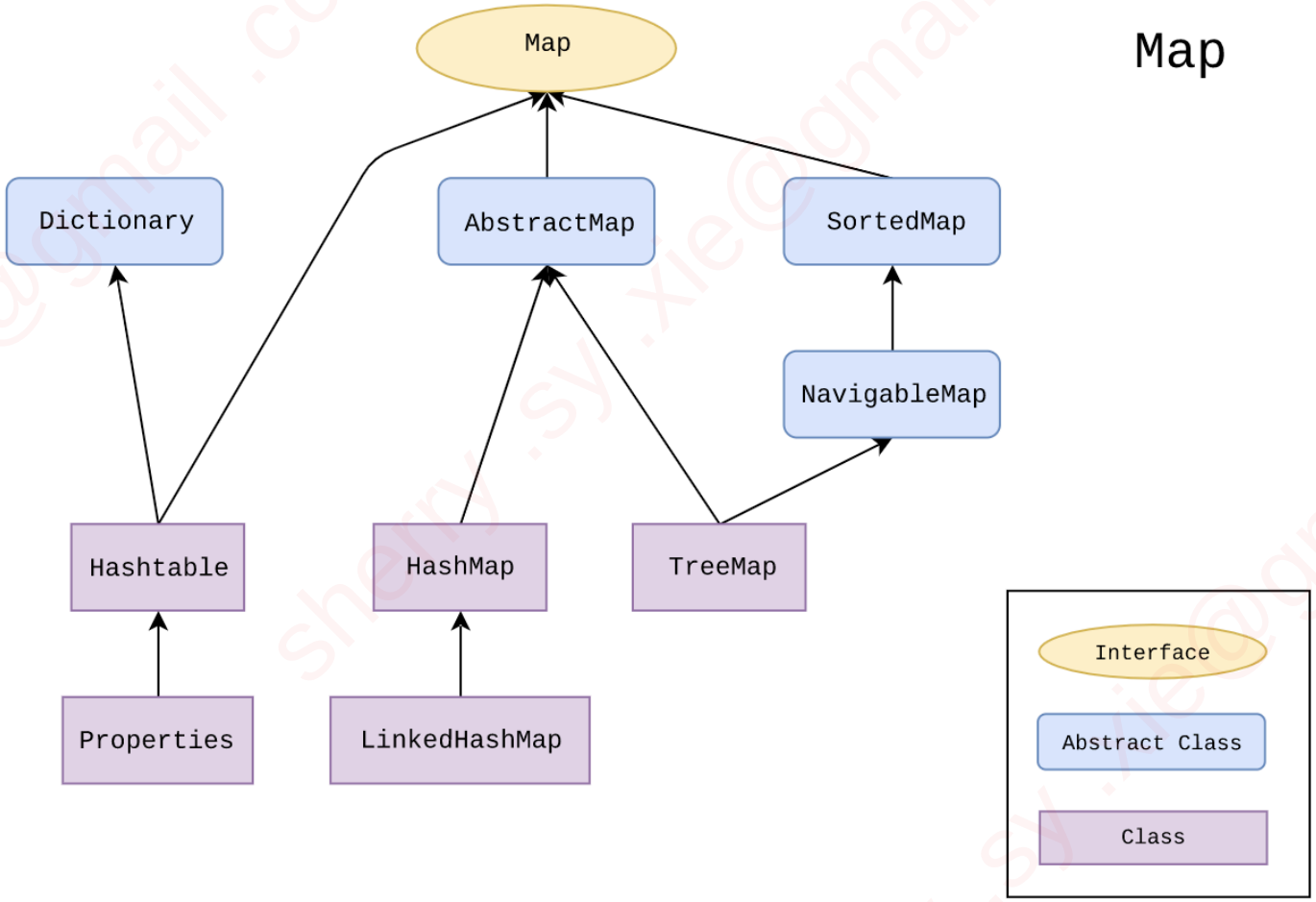
Java Collections Framework - Collections

A Java collection framework provides an architecture to store and manipulate a group of objects. A Java collection framework includes the following:

- Interfaces
- Classes
- Algorithm: Algorithm refers to the methods which are used to perform operations such as searching and sorting, on objects that implement collection interfaces



Map



序列化 和反序列化

- I/O操作
- 对象序列化：将对象以二进制的形式保存在硬盘上或者进行网络传输；
- 反序列化：将二进制的文件/数据 转化为对象读取；
- 实现 serializable 接口可以实现对象序列化，其中没有需要实现的方法，implements Serializable 只是为了标注该对象是可被序列化的。
- 场景：文件读写（Json，xml格式），网络传输，数据库读写

Java 8 Lambda

A lambda expression is like syntactic sugar for an anonymous class with one method whose type is inferred. However, it will have enormous implications for simplifying development.

The main syntax of a lambda expression is “parameters -> body”. The compiler can usually use the context of the lambda expression to determine the functional interface being used and the types of the parameters. There are four important rules to the syntax:

- Declaring the types of the parameters is optional.
- Using parentheses around the parameter is optional if you have only one parameter.
- Using curly braces is optional (unless you need multiple statements).
- The “return” keyword is optional if you have a single expression that returns a value.

Java 8 Lambda

```
() -> System.out.println(this)

(String str) -> System.out.println(str)

str -> System.out.println(str)

(String s1, String s2) -> { return s2.length() - s1.length(); }

(s1, s2) -> s2.length() - s1.length()
```

Java 8 Optional

Optional class in the java.util package for avoiding null return values (and thus NullPointerException)

Optional 的三种构造方式: Optional.of(obj)，Optional.ofNullable(obj) 和明确的 Optional.empty()

```
Optional<User> user = .....
if (user.isPresent()) {
  return user.getOrders();
} else {
  return Collections.emptyList();
}

return user.map(u -> u.getOrders()).orElse(Collections.emptyList())
```

Java 8 Optional

存在即返回, 无则提供默认值
return user.orElse(null);
//而不是 return user.isPresent() ? user.get() : null;

return user.orElse(UNKNOWN_USER);

存在即返回, 无则由函数来产生
return user.orElseGet(() -> fetchAUserFromDatabase());
//而不要 return user.isPresent() ? user:
fetchAUserFromDatabase();

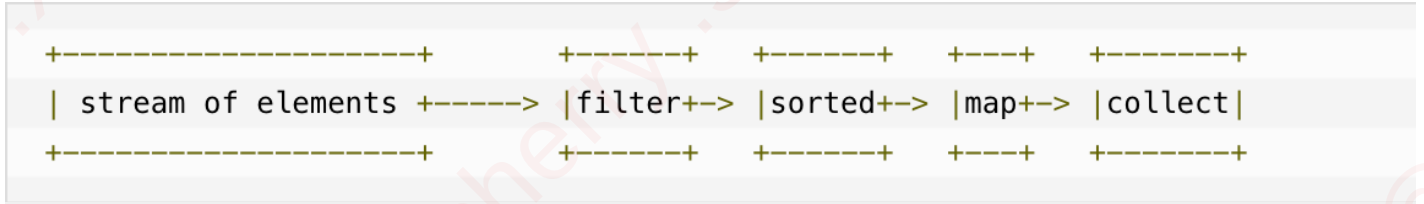
存在才对它做点什么
user.ifPresent(System.out::println);

//而不要下边那样
if (user.isPresent()) {
  System.out.println(user.get());
}

TIPS: 使用 Optional 时尽量不直接调用 Optional.get() 方法, Optional.isPresent() 更应该被视为一个私有方法, 应依赖于其他像 Optional.orElse(), Optional.orElseGet(), Optional.map() 等这样的方法.

Java 8 Stream

- Java 8 API添加了一个新的抽象称为流Stream，可以以一种声明的方式处理数据。Stream 使用一种类似用 SQL 语句从数据库查询数据的直观方式来提供一种对 Java 集合运算和表达的高阶抽象。
- 这种风格将要处理的元素集合看作一种流， 流在管道中传输， 并且可以在管道的节点上进行处理， 比如筛选， 排序，聚合等。
- 元素流在管道中经过中间操作（intermediate operation）的处理，最后由最终操作(terminal operation)得到前面处理的结果。




```
List<Integer> transactionsIds =
widgets.stream()
             .filter(b -> b.getColor() == RED)
             .sorted((x,y) -> x.getWeight() - y.getWeight())
             .mapToInt(Widget::getWeight)
             .sum();
```

Java 8 Stream

Stream（流）是一个来自数据源的元素队列并支持聚合操作

- 元素是特定类型的对象，形成一个队列。 Java中的Stream并不会存储元素，而是按需计算
- **数据源** 流的来源。 可以是集合，数组，I/O channel， 产生器generator 等。
- **聚合操作** 类似SQL语句一样的操作， 比如filter, map, reduce, find, match, sorted等。

和以前的Collection操作不同， Stream操作还有两个基础的特征：

- **Pipelining**: 中间操作都会返回流对象本身。 这样多个操作可以串联成一个管道， 如同流式风格（fluent style）。 这样做可以对操作进行优化， 比如延迟执行(laziness)和短路(short-circuiting)。
- **内部迭代**： 以前对集合遍历都是通过Iterator或者For-Each的方式, 显式的在集合外部进行迭代， 这叫做外部迭代。 Stream提供了内部迭代的方式， 通过访问者模式(Visitor)实现。

Java 8 Stream Examples

```
public void generateStream() {
  List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");
  List<String> filtered = strings.stream().filter(string -> !
string.isEmpty()).collect(Collectors.toList());
  filtered.forEach(i -> System.out.println(i));
}
```

map

map 方法用于映射每个元素到对应的结果

```
public void mapExample() {
  List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);
  // 获取对应的平方数
  List<Integer> squaresList = numbers.stream().map( i ->
i*i).distinct().collect(Collectors.toList());
}
```

filter

filter 方法用于通过设置的条件过滤出元素。

```
public void sortedExample() {
  Random random = new Random();
  random.ints().limit(10).sorted().forEach(System.out::println);
}
```

limit

limit 方法用于获取指定数量的流。

sorted

sorted 方法用于对流进行排序

Java 8 Stream Examples

Collectors

Collectors 类实现了很多归约操作，例如将流转换成集合和聚合元素。

Collectors 可用于返回列表或字符串

```
public void collectionExample() {
  List<String>strings = Arrays.asList("abc", "",
"bc", "efg", "abcd","", "jkl");
  List<String> filtered =
strings.stream().filter(string -> !
string.isEmpty()).collect(Collectors.toList());

  System.out.println("筛选列表: " + filtered);
  String mergedString =
strings.stream().filter(string -> !
string.isEmpty()).collect(Collectors.joining(", "));
  System.out.println("合并字符串: " +
mergedString);
}
```

统计

另外，一些产生统计结果的收集器也非常有用。

它们主要用于int、double、long等基本类型上

```
public void statixExample() {
  List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);

  IntSummaryStatistics stats = numbers.stream().mapToInt((x) ->
x).summaryStatistics();

  System.out.println("列表中最大的数 : " + stats.getMax());
  System.out.println("列表中最小的数 : " + stats.getMin());
  System.out.println("所有数之和 : " + stats.getSum());
  System.out.println("平均数 : " + stats.getAverage());
}
```

Exception handling

- 异常机制，是指程序不正常时的处理方式。具体来说，异常机制提供了程序退出的安全通道。当出现错误后，程序执行的流程发生改变，程序的控制权转移到异常处理器。

```
try {
  // 有可能抛出异常的代码
} catch (Exception e) {
  // 异常处理
} finally {
  // 无论是否捕获到异常都会执行的程序
}
```

Exception handling

- Throwable类是整个Java异常体系的超类，都有的异常类都是派生自这个类。包含Error和Exception两个直接子类。
- Error表示程序在运行期间出现了十分严重、不可恢复的错误，在这种情况下应用程序只能中止运行，例如JAVA虚拟机出现错误。在程序中不用捕获Error类型的异常。一般情况下，在程序中也不应该抛出Error类型的异常。
- Exception是应用层面上最顶层的异常类，包含RuntimeException（运行时异常）和 Checked Exception（受检异常）。

✓ RuntimeException是一种Unchecked Exception，即表示编译器不会检查程序是否对RuntimeException作了处理，在程序中不必捕获RuntimExcepti on类型的异常，也不必在方法体声明抛出RuntimeException类。一般来说，RuntimeException发生的时候，表示程序中出现了编程错误，所以应该找出错误修改程序，而不是去捕获RuntimeException。常见的RuntimeException有NullPointException、ClassCastException、IllegalArgumentException、IndexOutOfBoundException等。

✓ Checked Exception是相对于Unchecked Exception而言的，所有继承自Exception并且不是RuntimeException的异常都是Checked Exception。JAVA 语言规定必须对checked Exception作处理，编译器会对此作检查，要么在方法体中声明抛出checked Exception，要么使用catch语句捕获checked Exception进行处理，不然不能通过编译。常用的Checked Exception有IOException、ClassNotFoundException等。

正确的Exception handling

原则性问题-异常处理机制初衷：

- 1.将不可预期异常的处理代码和正常的业务逻辑处理代码分离。
- 2.异常只应该用于处理非正常情况，不要用异常处理来代替正常流程控制。（对于完全可知的、处理方式清晰的错误，程序本应该提供相应的错误代码，而不是笼统称为异常）
- 3.先捕获小异常，再捕获大异常（Exception e 用此表示未知异常）
- 4.对于完全已知的错误应该编写处理这种错误的代码，增加程序健壮性。

企业应用做法：程序先捕获原始异常，然后抛出一个新的业务异常，新的业务异常中包含对用户的提示信息-----这种做法叫异常转译。因为核心是：在合适的层级处理异常。

Log

- In production, do not use System.out as log tool
- Log4j, logback
- Use log configuration
- And suggest to use slf4j: slf4j是一套包装Logging 框架的界面程式，以外观模式实现。可以在软件部署的时候决定要使用的 Logging 框架，目前主要支援的有Java Logging API、log4j及logback等框架。 https://zh.wikipedia.org/wiki/SLF4J

- Log应该包含什么内容？ Debug log，interface log…
- 生产环境的中的log到底在哪里？

单元测试

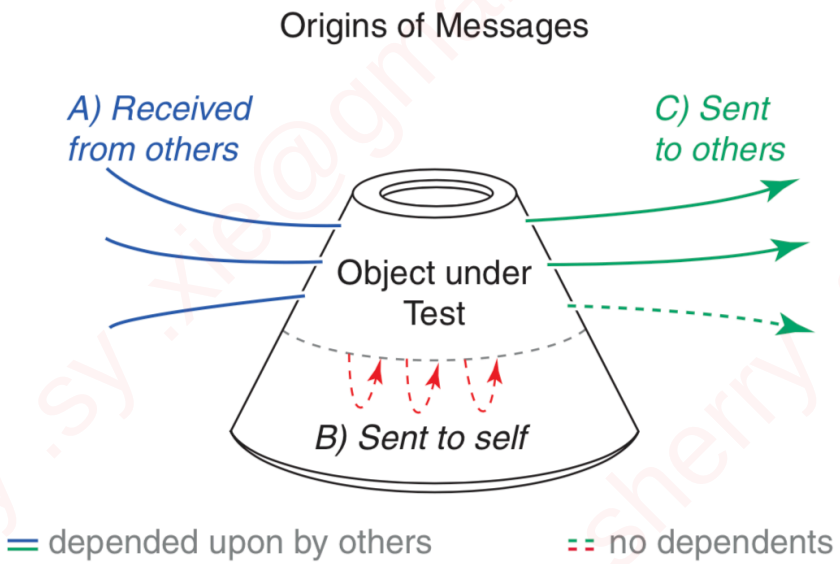
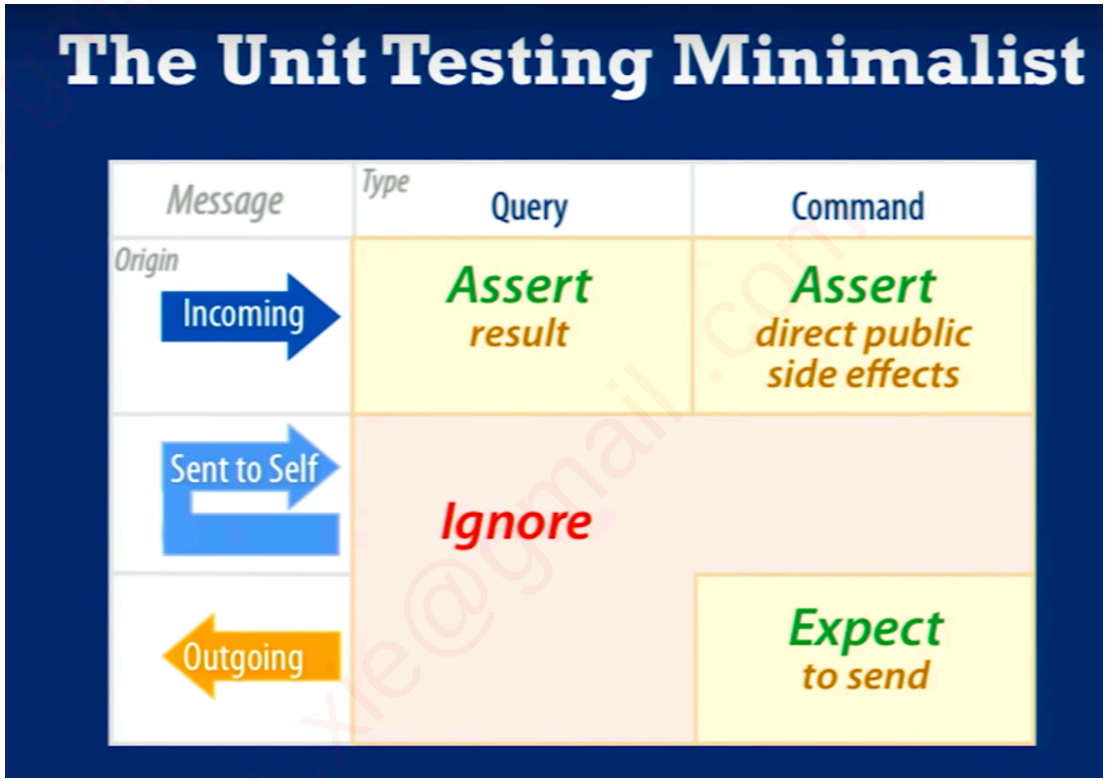


Figure 9.1 Objects under test are like space capsules, messages breach their boundaries.



- Book: Practical Object-Oriented Design (*POODR*) by Sandy Metz
- https://www.youtube.com/watch?v=URSWYvyc42M

匠人学院
jiangren.com.au

Q & A