Java 常规教学

计算机协会

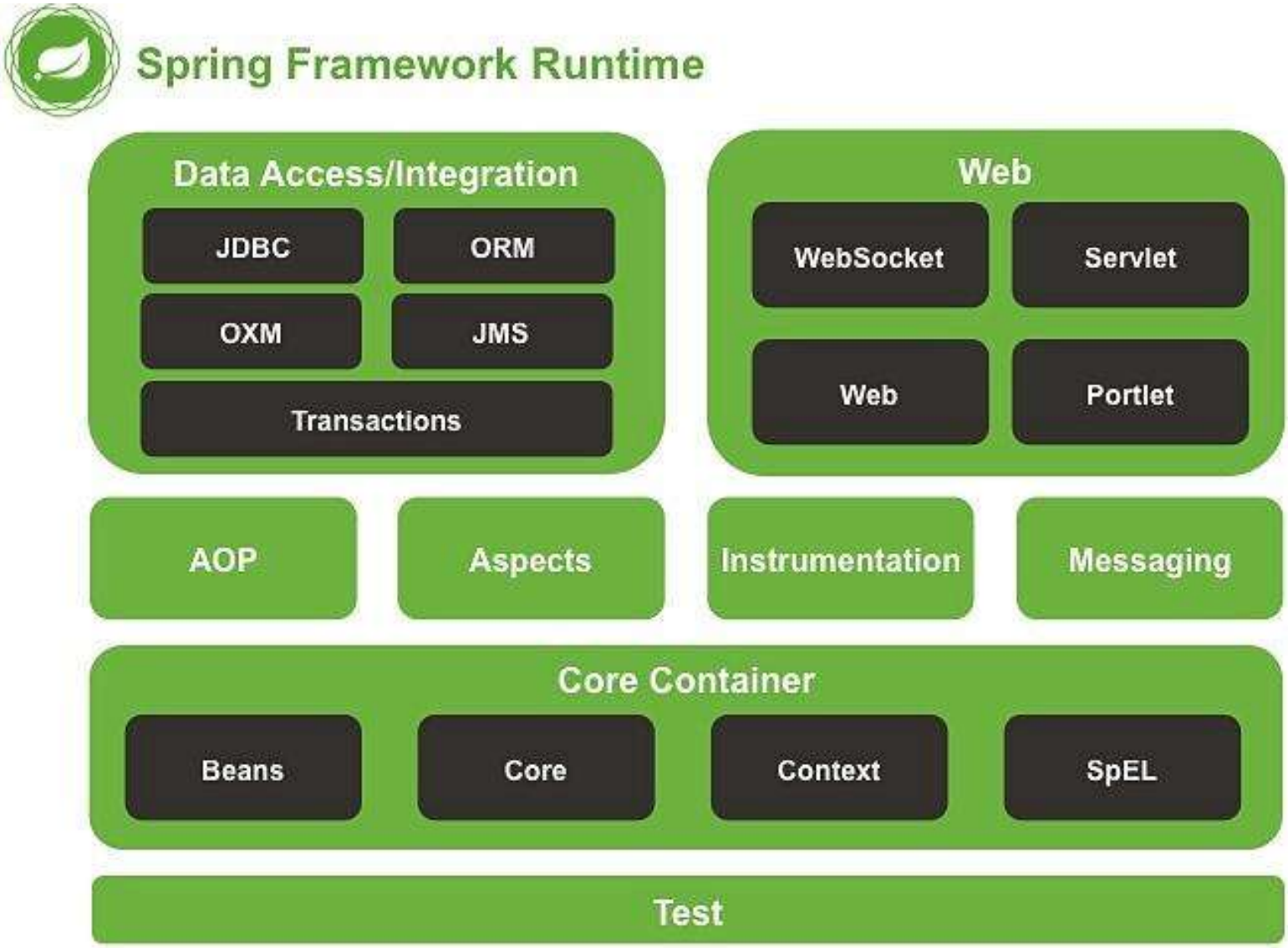
上海财经大学

23 会计学院 ACCA 张华轩

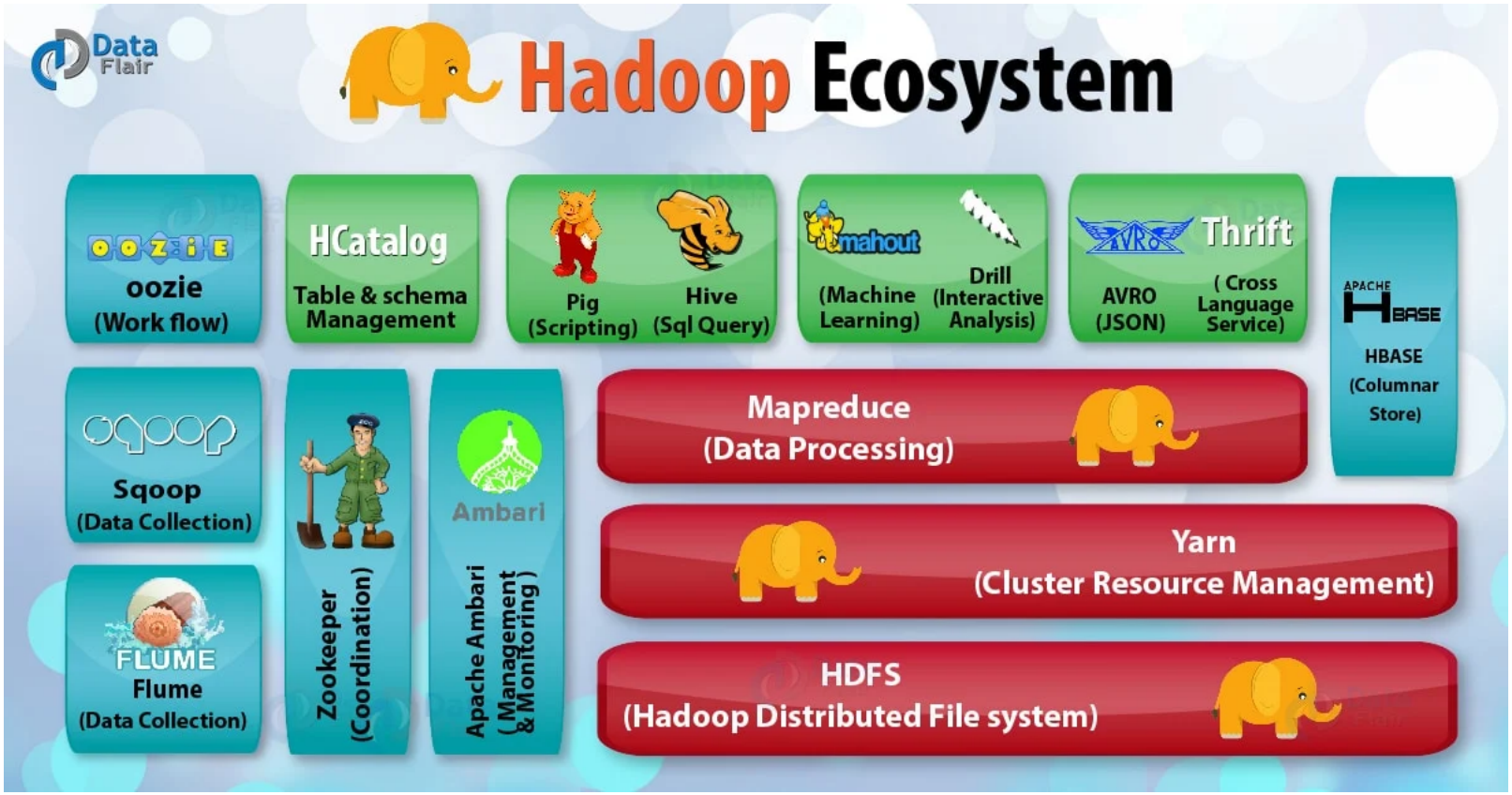
1 Java 生态

2 Java 基础

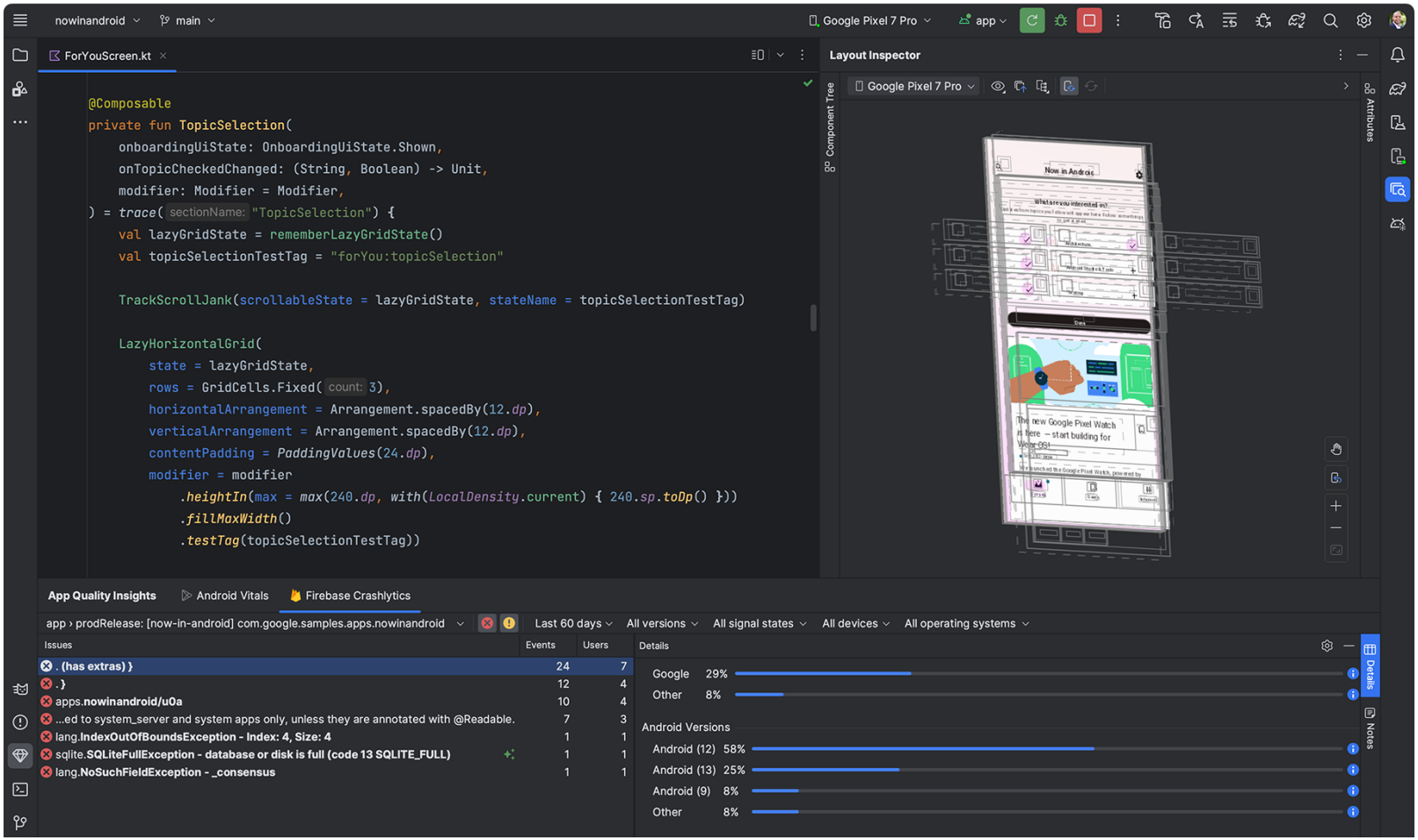
Spring & Spring Boot



Hadoop, Storm(Clojure), Spark(Scala)



Android App Development



1 Java 生态

2 Java 基础

环境配置

？有必要吗

谁还用命令行编译 Java

- JRE(Java Runtime Environment)：Java 运行时环境
- JDK(Java Development Kit)：Java 开发工具包
- Java IDE：Eclipse, IntelliJ IDEA, VS Code(new!)

https://www.jetbrains.com/zh-cn/idea

IntelliJ 启动!

Java 版本演进

Java SE 8 (LTS)	52	2014 年 3 月 18 日	OpenJDK 目前由 Red Hat 维护 Oracle 于 2022 年 3 月停止更新（商用） Oracle 于 2030 年 12 月停止更新（非商用） Azul 于 2030 年 12 月停止更新 IBM Semeru 于 2026 年 5 月停止更新 Eclipse Adoptium 于 2026 年 5 月或之后停止更新 Amazon Corretto 于 2026 年 5 月或之后停止更新	Oracle 于 2030 年 12 月停止更新 Red Hat 于 2026 年 11 月停止更新
Java SE 9	53	2017 年 9 月 21 日	OpenJDK 于 2018 年 3 月停止更新	不适用
Java SE 10	54	2018 年 3 月 20 日	OpenJDK 于 2018 年 9 月停止更新	不适用
Java SE 11 (LTS)	55	2018 年 9 月 25 日	OpenJDK 目前由 Red Hat 维护 Azul 于 2026 年 9 月停止更新 IBM Semeru 于 2024 年 10 月停止更新 Eclipse Adoptium 于 2024 年 10 月或之后停止更新 Amazon Corretto 于 2027 年 9 月或之后停止更新 微软于 2024 年 10 月或之后停止更新	Oracle 于 2026 年 9 月停止更新 Azul 于 2026 年 9 月停止更新 Red Hat 于 2024 年 10 月停止更新

Latest: Java SE 25(LTS)

JDK

主流 JDK 分支

- OpenJDK：由社区支持、开源
- Oracle JDK：Oracle 主导、商用（最后的免费商用版本是 8u202）
- Adoptium Eclipse Temurin：由 Eclipse 基金会支持、开源
- Amazon Corretto：由 Amazon 主导、免费商用

https://adoptium.net/zh-CN/temurin/releases

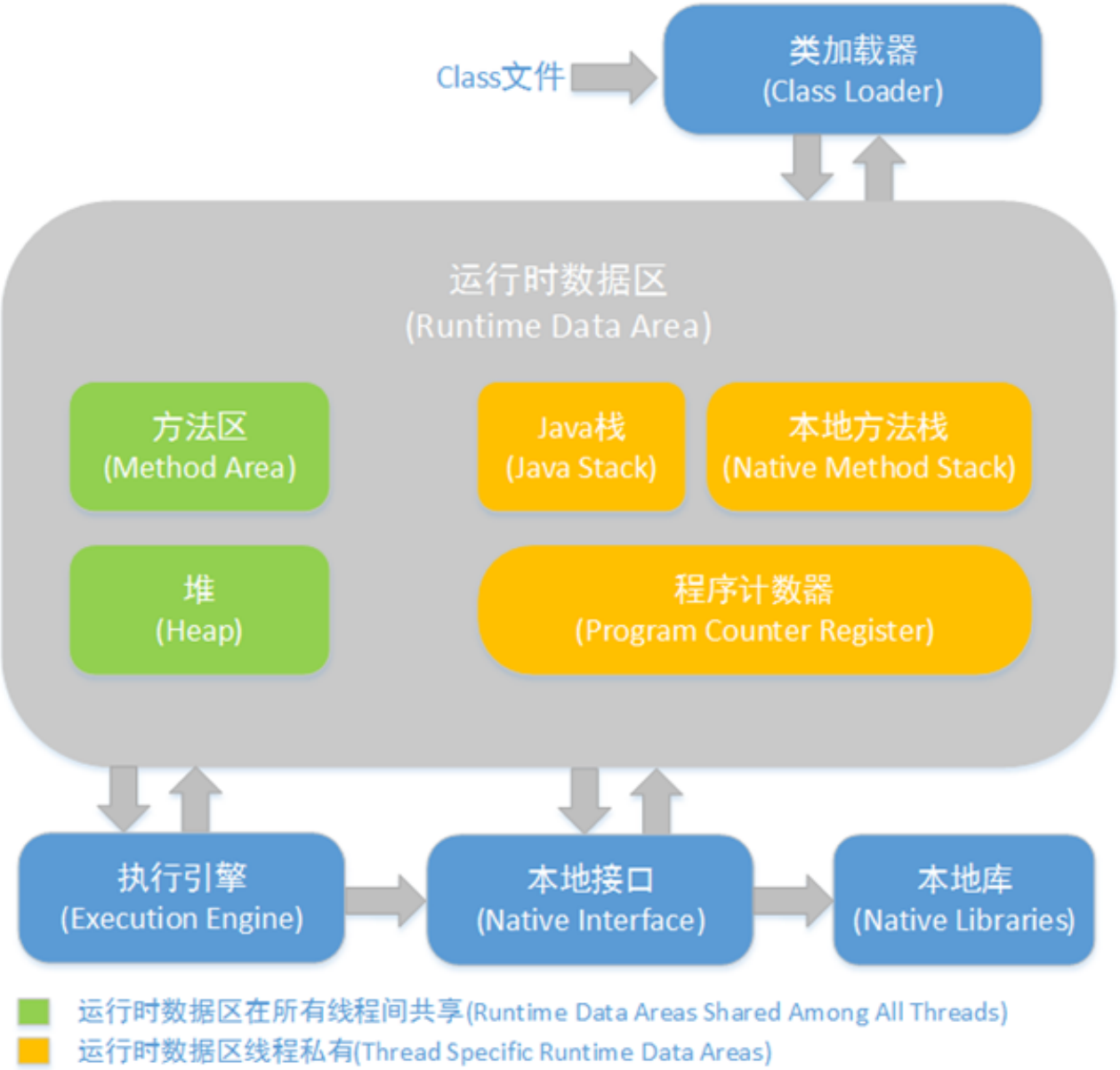
JVM

Java 虚拟机 (Java Virtual Machine)

- 是什么：用于执行 Java 字节码的虚拟机，拥有独立的运行机制，所有 Java 程序都运行在 JVM 内部，但其运行的 Java 字节码未必由 Java 编译而成
- 为什么：一次编译，到处运行；自动内存管理；自动垃圾回收功能（GC）
- 怎么样：汇编-> 字节码，本地运行时->JVM 运行时，只需要针对特定平台提供 JVM 实现

JVM 对内提供统一接口，对外对接本地接口

Java 虚拟机运行时


Class文件
类加载器
(Class Loader)
运行时数据区
(Runtime Data Area)
方法区
(Method Area)
Java栈
(Java Stack)
本地方法栈
(Native Method Stack)
堆
(Heap)
程序计数器
(Program Counter Register)
执行引擎
(Execution Engine)
本地接口
(Native Interface)
本地库
(Native Libraries)
运行时数据区在所有线程间共享(Runtime Data Areas Shared Among All Threads)
运行时数据区线程私有(Thread Specific Runtime Data Areas)

基本结构

最简单的类

```java
/**
* 可以用来自动创建文档的注释
**/
public class Solution {
    public static void main(String[] args) {
        // 向屏幕输出文本:
        System.out.println("Hello, world!");
        /* 多行注释开始
        注释内容
        注释结束 */
    }
} // class定义结束
```

基本结构

Annotation for Cpp

- **入口方法**：main 方法为入口方法，是程序的唯一入口
- **外壳类** (shell class)：Solution 类在这里没有任何作用
- **退出码** (exit code)：main 方法不返回任何值

任何 Java 源文件都是类/类的集合!

Dive-in

入口方法

- public **修饰符**：main 方法是唯一的程序入口，为了让 JVM 可以调用，必须使用 public 修饰符暴露出来
- static **静态方法**：入口方法需要在类创建之前调用
- void **返回值**：JVM 不接受返回值，所以返回 void

Java 访问修饰符

访问修饰符

- public：公共访问，允许该类或成员被任何其他类访问
- protected：受保护访问，仅限于同包内的类和子类
- **包访问**：不加修饰符时，默认包访问，仅限同包内访问
- private：私有访问，仅限类内部访问。适用于不希望外部直接访问的属性或方法

Example

protected 子类访问

```
1  class Parent {
2      protected String message = "Hello from Parent";
3
4      protected void displayMessage() {
5          System.out.println(message);
6      }
7  }
8
9  class Child extends Parent {
10     public void show() {
11         displayMessage(); // 子类可以访问父类的受保
   护方法
12     }
13 }
```

Example(Continue)

protected 子类访问

```
1 public class Main {
2     public static void main(String[] args) {
3         Child child = new Child();
4         child.show(); // 输出: Hello from Parent
5     }
6 }
```

Example

protected 包内访问

```
1  package mypackage;
2  class Parent {
3      protected String message = "Hello from Parent";
4
5      protected void displayMessage() {
6          System.out.println(message);
7      }
8  }
9
10 class NonSubclass {
11     public void accessParent() {
12         Parent parent = new Parent();
13         parent.displayMessage(); // 可访问包内的受保护方法
14     }
15 }
```

Example(Continue)

protected 包内访问

```
1 public class Main {
2     public static void main(String[] args) {
3         NonSubclass example = new NonSubclass();
4         example.accessParent(); // 输出: Hello from
  Parent
5     }
6 }
```

Why protected?

Annotation for Cpp

- Java：同包内的类和子类均可以访问
- C++：仅限子类访问

Java 访问修饰符的设计大体参考 C++

为什么允许包内访问?

Java 包机制

Java Package

- **包的定义**：Java 包（Package）是一种命名空间机制，用于将相关的类和接口分组
- **包的作用**：
  - **命名空间**：提供独立的命名空间，防止类和接口命名冲突
  - **访问控制**：包允许定义类和成员的访问级别（public、protected、default 和 private），实现不同的封装级别。
  - **模块化**：将相关的类按功能分组，建立模块化的项目结构

Example

Java Package

```
1  // 在 mypackage 包中定义一个类
2  package mypackage;
3
4  public class MyClass {
5      public void display() {
6          System.out.println("Hello from MyClass in
   mypackage");
7      }
8  }
9
10 // 在其他类中导入该包
11 import mypackage.MyClass;
```

设计模式：组合

组合而非继承

组合模型中，一个对象（称为复合对象）可以包含另一个对象（称为组件对象），复合对象可以使用组件对象的行为

降低继承可能带来的耦合性

Thanks!

Reference

[1] Cay S. Horstmann. “Core Java: Fundamentals, Volume 1”. In: (2018). ISSN: 978-0135166307.

[2] 廖雪峰. “廖雪峰 Java 教程”. In: https://liaoxuefeng.com/books/java/introduction/index.html (2024).