第一章

面向对象的软件构造概述及Java语言简介

软件无处不在

软件无处不在，为我们的生活创造了无限精彩。在当今的信息时代，世界正在变得更加“智慧”，万事万物间感知化、互联化和智能化的程度不断加深。

课程内容

面向对象 的 软件构造 导论

一种思维方式

一种构造方法

一门引导大家如何用面向对象的思维方式，去进行软件构造的课程。

课程内容

- 面向对象的软件构造概述及Java语言基础(第一周)
- 类与对象、接口与继承（第二周）
- 设计模式（第三周）
- 单元测试及代码质量保障（第三周）
- 集合、流与输入输出、Swing介绍（第四周）
- 多线程与同步（第五周）
- 泛型与反射、网络编程（第七周）

课程学习的目标

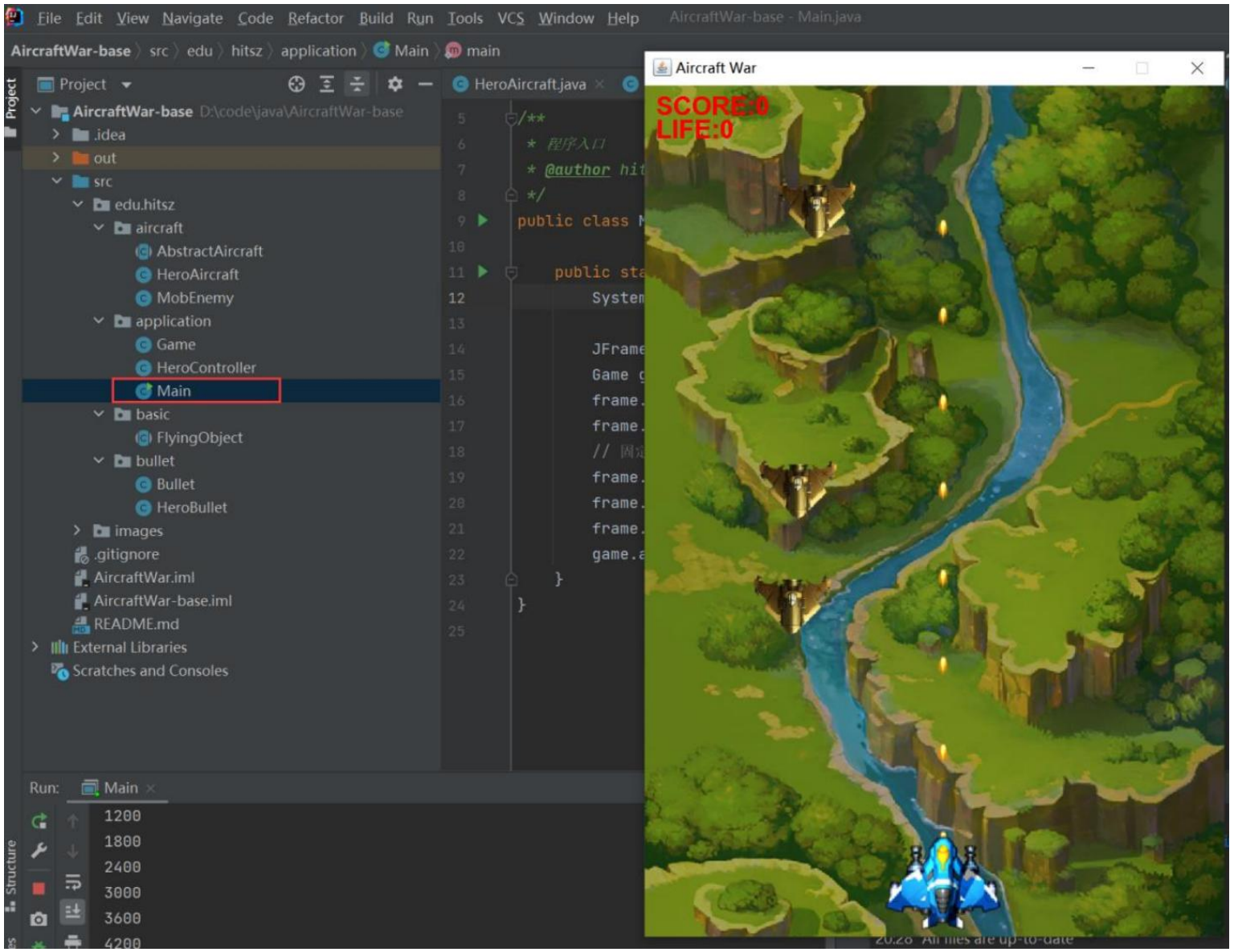
- 面向对象的思维方式
- 面向对象的软件构造的方法流程
- Java语言及主要类库的熟练掌握
- 初步掌握设计模式
- 测试驱动的开发
- 独立完成一个面向对象的软件构造的案例

教学方法

- 课堂授课（24学时）
  - 讲授基本知识点和思路
  - 课内讨论和课外答疑
  - 提供相关阅读资料

- 实验教学（16学时）
  - 培养学生运用课程中教授的面向对象编程方法进行软件构造
  - 包含了6个实验
  - 按照实验指导书的要求独立完成

课程案例


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
AircraftWar-base - Main.java
AircraftWar-base src edu hitsz application Main main
Project
AircraftWar-base D:\code\java\AircraftWar-base
.idea
out
src
edu.hitsz
aircraft
AbstractAircraft
HeroAircraft
MobEnemy
application
Game
HeroController
Main
basic
FlyingObject
bullet
Bullet
HeroBullet
images
.gitignore
AircraftWar.iml
AircraftWar-base.iml
README.md
External Libraries
Scratches and Consoles
HeroAircraft.java
Run: Main
1200
1800
2400
3000
3600
4200
Aircraft War
SCORE:0
LIFE:0

课程参考书

- Cay S. Horstmann 著；林琪，苏钰涵等译. Java核心技术卷1（原书第11版）. 机械工业出版社，ISBN：9787111636663，2019

- Cay S. Horstmann 著；陈昊鹏译. Java核心技术卷2（原书第11版）. 机械工业出版社，ISBN：9787111643432，2019

- Erich Gamma等著；李英军，马晓星，蔡敏，刘建中等译. 设计模式：可复用面向对象软件的基础，机械工业出版社，ISBN：9787111618331，2019

课程形式和要求

先修课程：高级语言程序设计、计算机专业导论

授课 & 实验

最终成绩：

- 10% 平时作业及考核
- 40% 实验
- 50% 期末考试

授课团队

- 任课老师：包俊 副教授
- 助教：待定
- 课程QQ群：530276553
- 办公室：T6803
- 手机：18055536566
- Email: baojun@hit.edu.cn

软件构造-2025年秋

扫一扫二维码，加入群聊

QQ

授课团队

- 教育背景
  - 上海交通大学
  - 英国爱丁堡大学

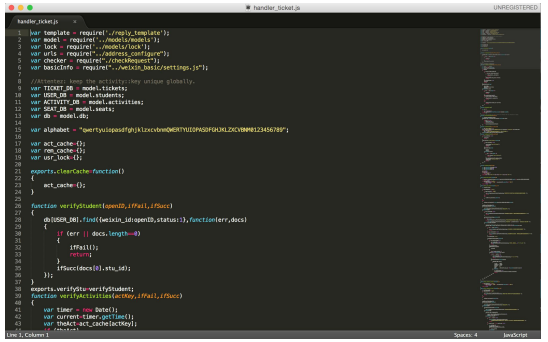
- 研究领域
  - **计算机视觉**
  - **智能设备**

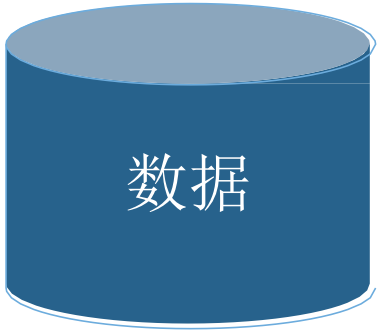
第一章

- 软件构造基本流程及目标
- 面向对象思想
- 设计模式
- Java语言简介
- Java开发环境搭建及程序示例

软件的定义？

- 软件 = 程序 + 数据 + 文档
  - 程序：计算机可以接受的一系列指令，运行时可以提供所要求的功能和性能。
  - 数据：使得程序能够适当地操作信息的数据结构。
  - 文档：描述程序的研制过程、方法和使用的图文资料。





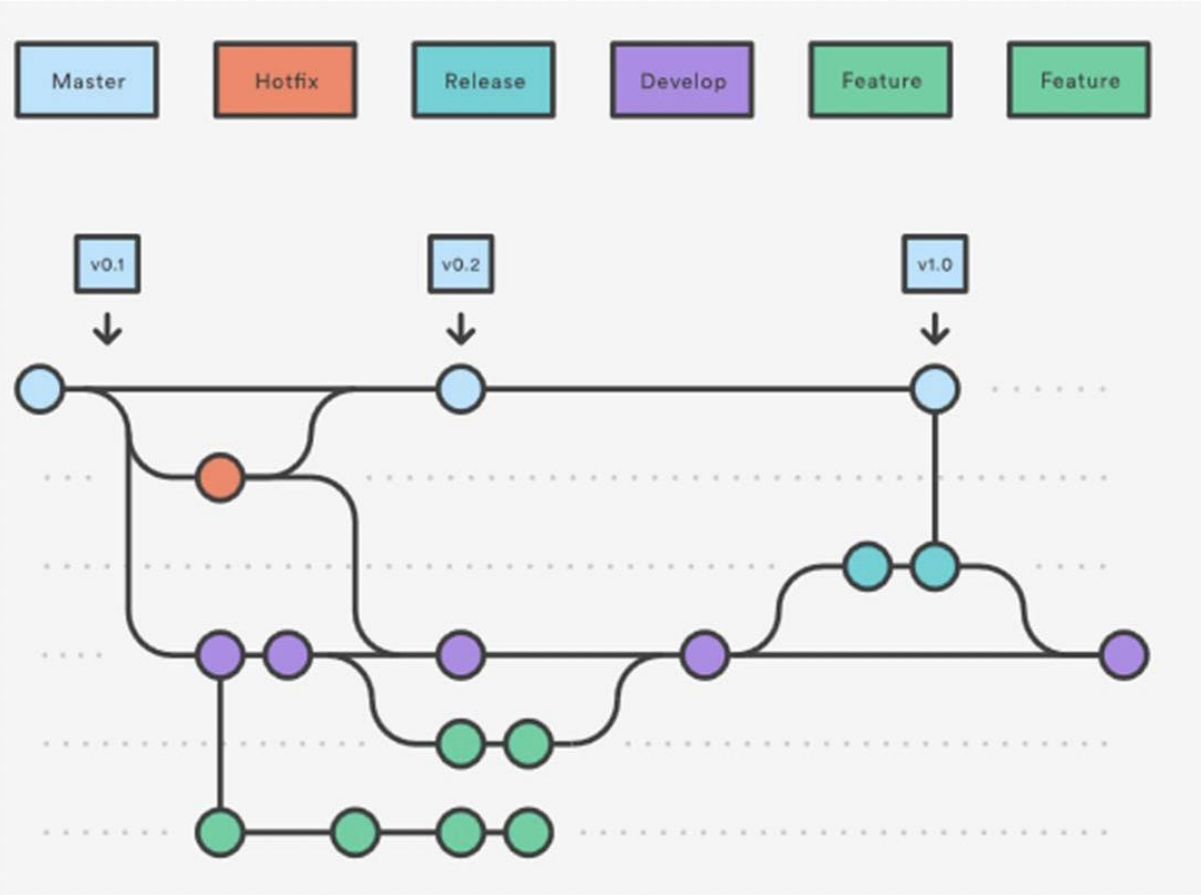
软件构造的基本流程

软件开发生命周期（SDLC，Software Development Life Cycle）从无到有，From 0 to 1：

软件构造的基本流程

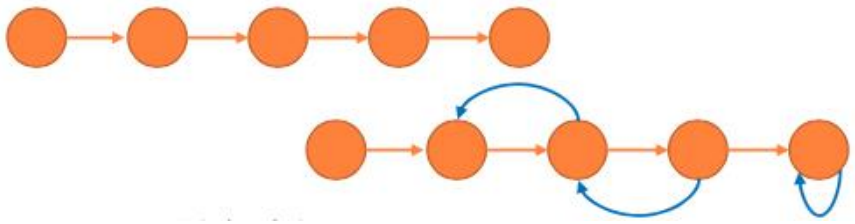
软件开发生命中的多版本迭代

从有到好，From 1 to n ：



传统软件开发过程模型

- 两种基本模型
  - 线性模型
  - 迭代过程

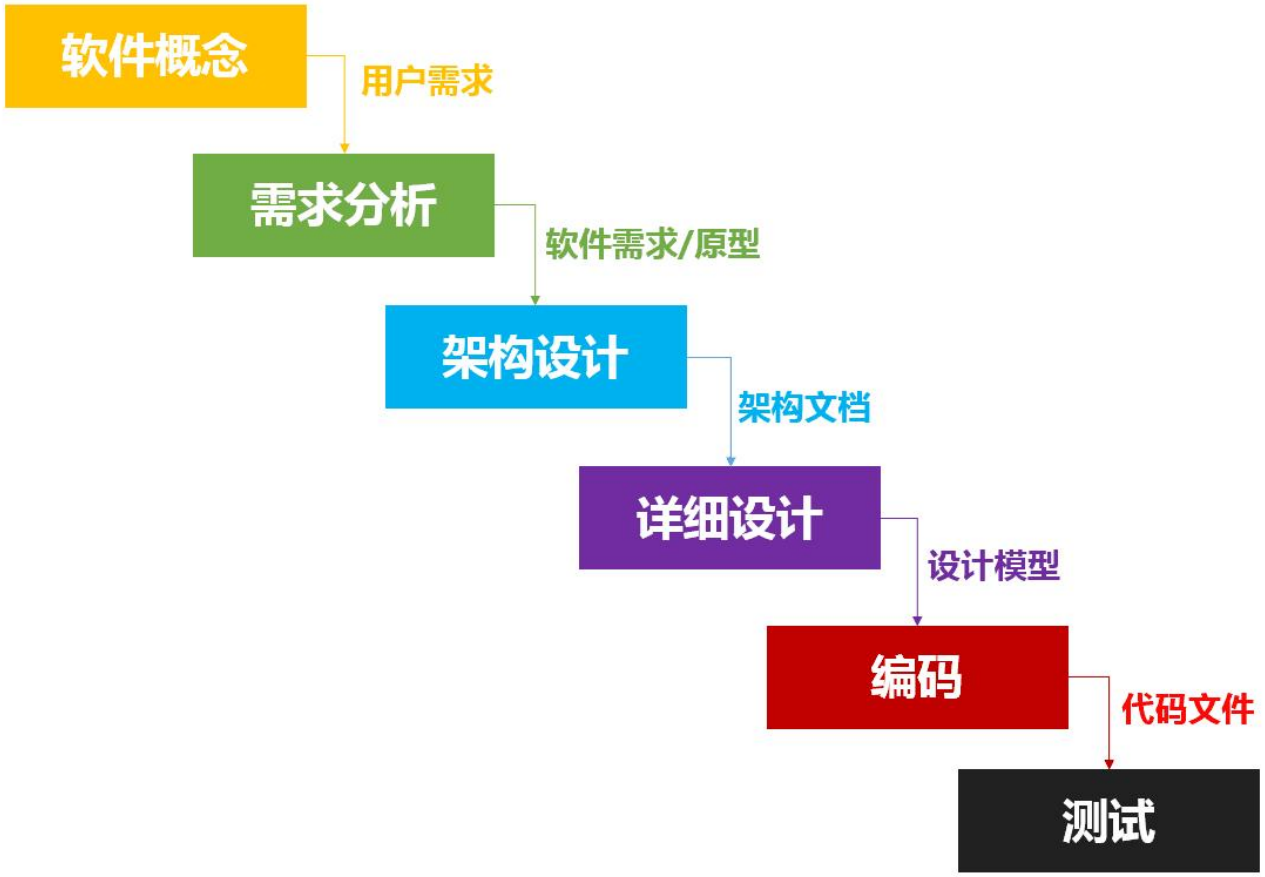


- 传统软件开发过程模型
  - 瀑布过程
  - 增量过程
  - 原型过程

- 流行的软件开发过程模型
  - 敏捷开发
  - 测试驱动开发

瀑布过程

瀑布模型是最典型的预见性开发方法，严格遵循预先计划的需求分析、设计、编码、集成、测试、维护的步骤顺序进行。



瀑布过程特点

- 线性推进
- 阶段划分清楚
- 整体推进
- 无迭代
- 管理简单
- 无法适应需求增加/变化

瀑布过程

举例：飞机大战

1. 需求分析

- 角色设定
- 游戏规则
- 界面、音效要求
- 游戏难度设计
- ...
- 确定需求
- 撰写需求分析报告

2. 软件设计

- 整体架构设计
- 模块化设计
- 类与接口的设计
- …

3.软件实现

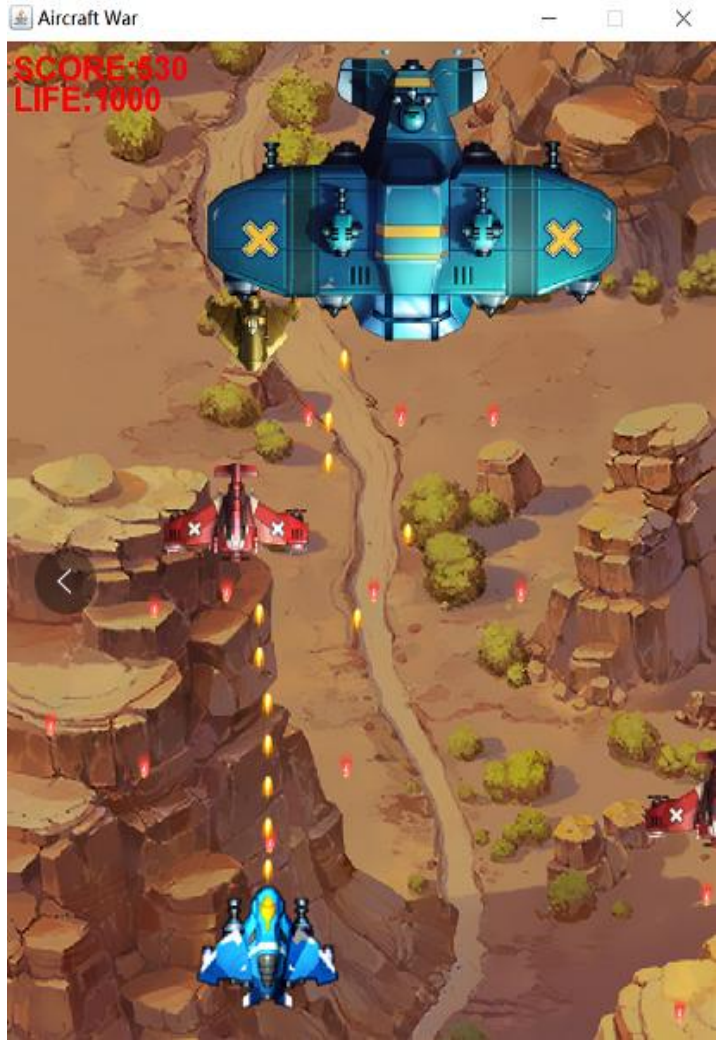
- 程序员编写代码
- …

4.软件测试

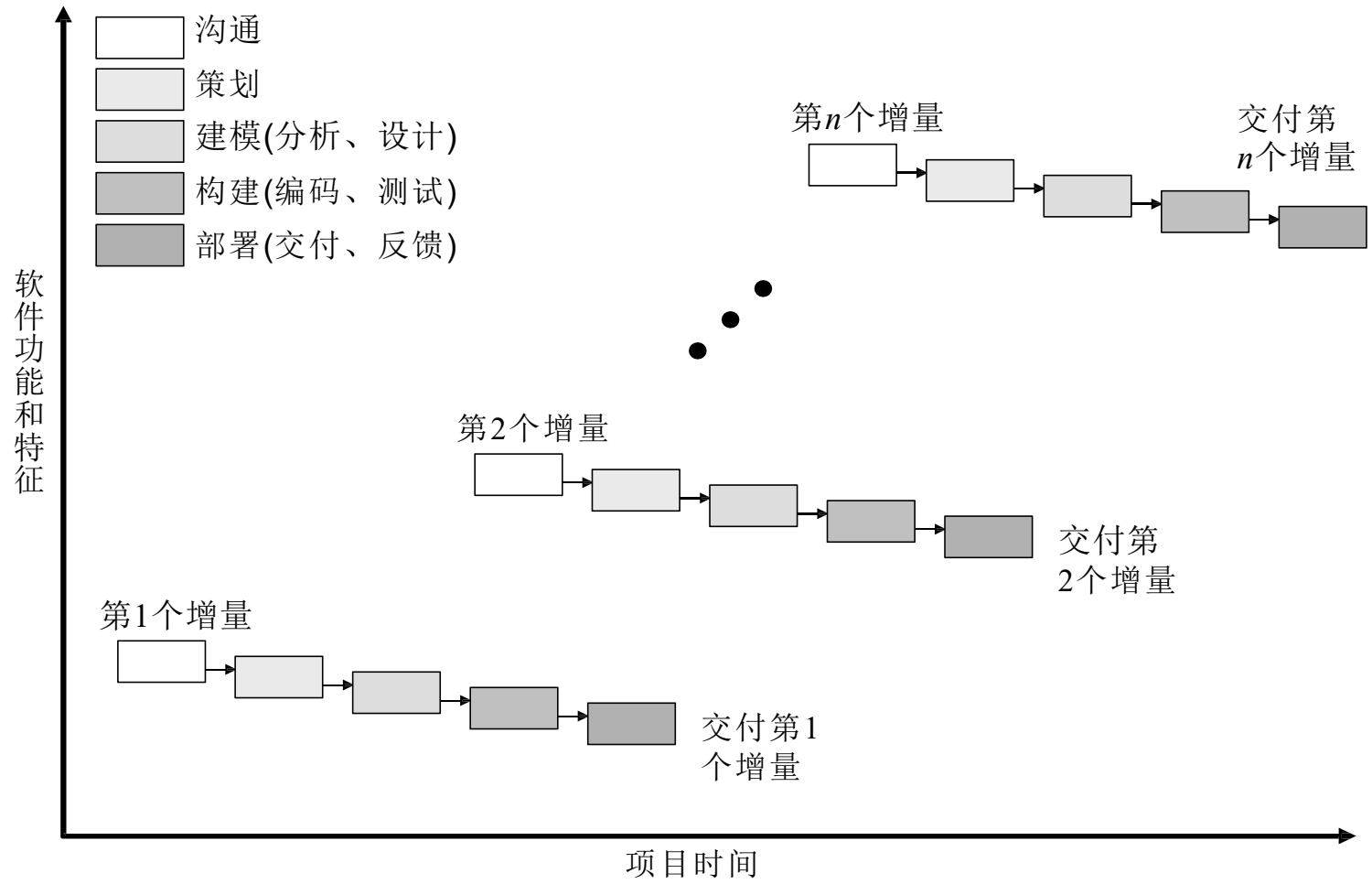
- 测试软件功能完整性、正确性
- 黑盒测试、白盒测试
- …

5.软件运行与维护

- 软件上线
- 版本更新
- …



增量过程


沟通
策划
建模(分析、设计)
构建(编码、测试)
部署(交付、反馈)
软件功能和特征
第n个增量
交付第n个增量
第2个增量
交付第2个增量
第1个增量
交付第1个增量
项目时间

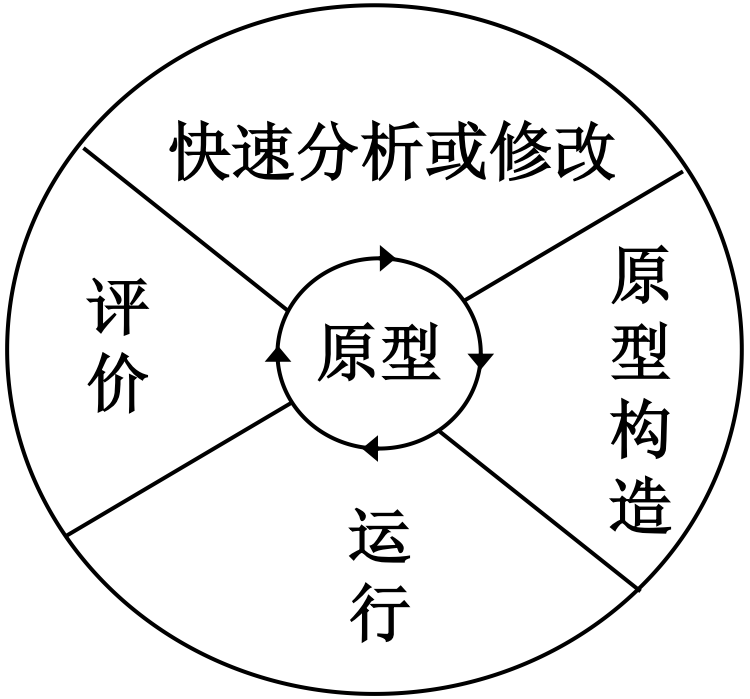
原型过程

- 原型过程是快速建立起来的可以在计算机上运行的程序，它所能完成的功能往往是最终产品能完成的功能的一个子集。原型在开发中的作用：

（1）**获得用户的真正需求**

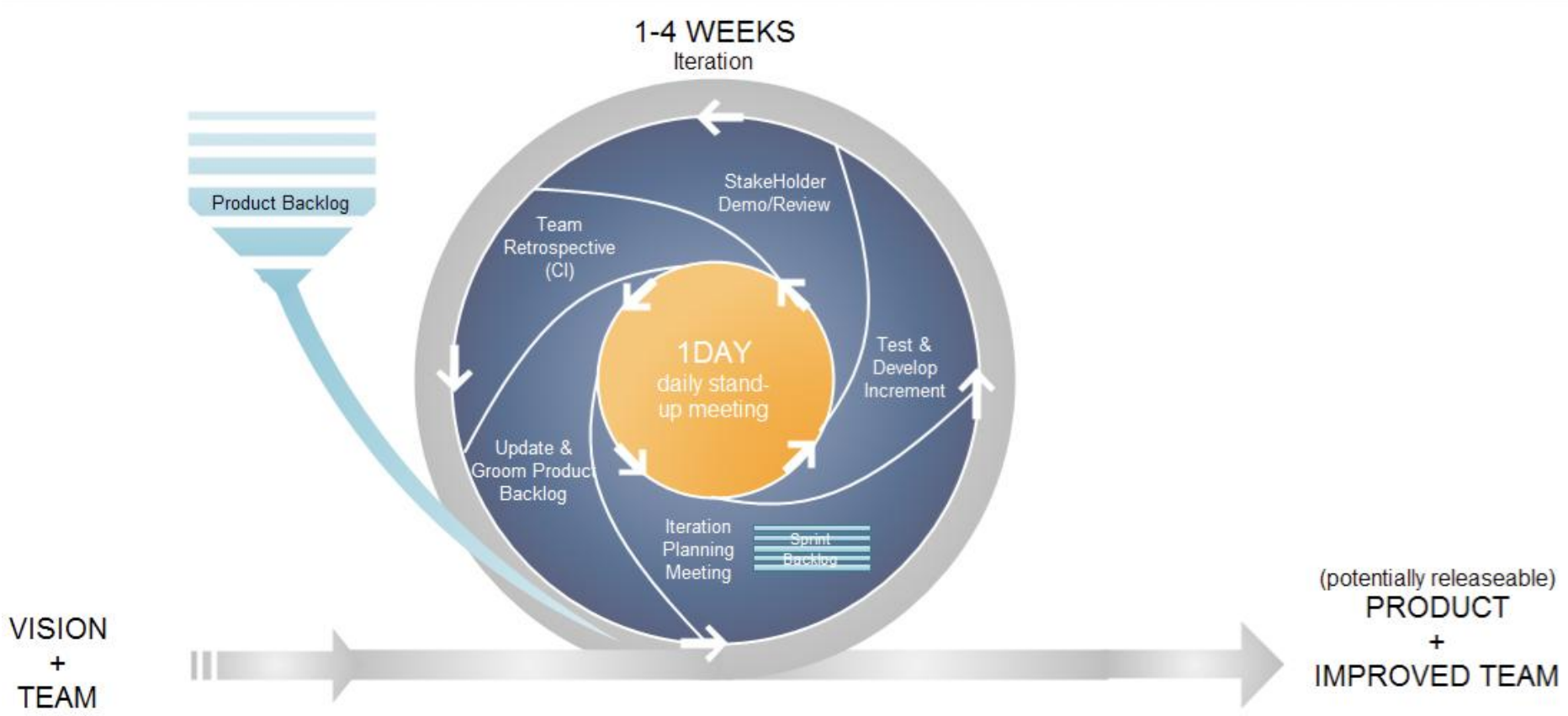
（2）可用于为一个项目或项目中某些部分，确定技术、成本和进度的可能性

在原型系统不断调整以满足各种利益相关者需求的过程中，采用迭代技术，同时也使开发者逐步清楚用户的需求。



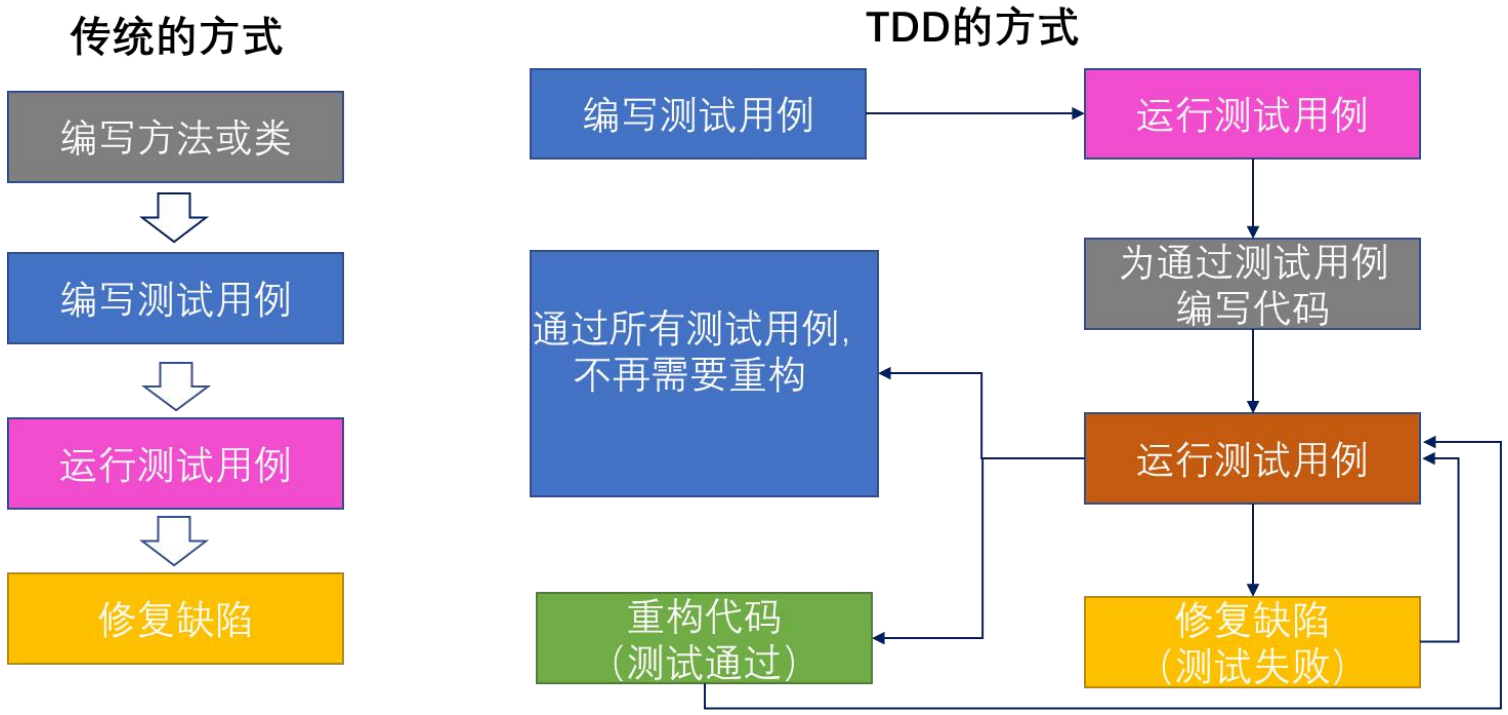
敏捷开发

敏捷开发：通过快速迭代和小规模的持续改进，以快速适应变化。



测试驱动的开发

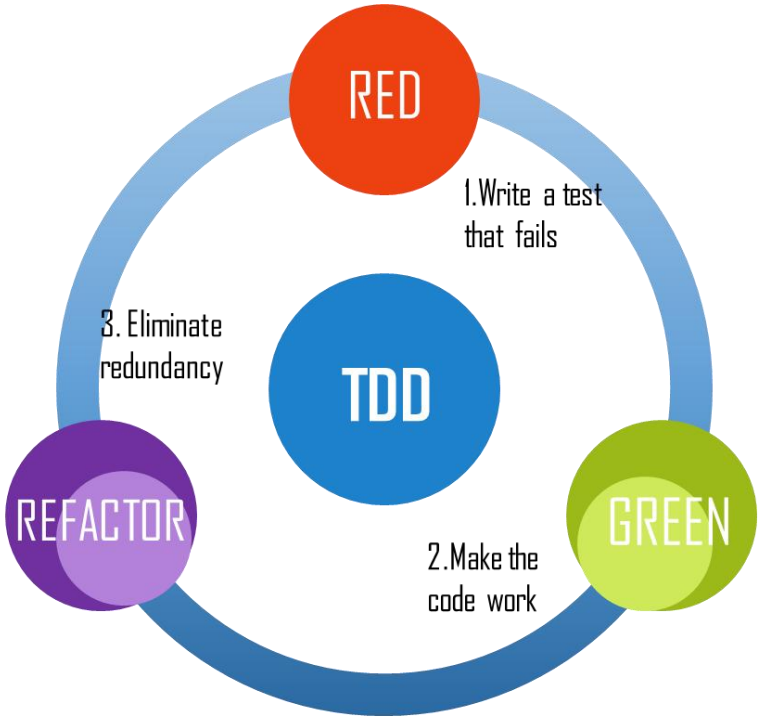
测试驱动开发（Test-Driven Development，简称TDD）是一种不同于传统软件开发流程的新型的开发方法。它要求在编写某个功能的代码之前先编写测试代码，然后只编写使测试通过的功能代码，通过测试来推动整个开发的进行。这有助于编写简洁可用和高质量的代码，并加速开发过程。



测试驱动的开发

基本过程（红灯-绿灯-重构）

1） 明确当前要完成的功能需求，制定TODO测试列表；
2） 快速完成针对此功能需求的测试用例编写；
3） 测试代码编译不通过（RED）；
4） 编写对应的功能代码；
5） 测试通过（GREEN）；
6） 对代码进行重构，并保证测试通过(REFACTOR)；
7） 循环完成所有功能的开发。



The mantra of Test-Driven Development(TDD) is "red, green, refactor",

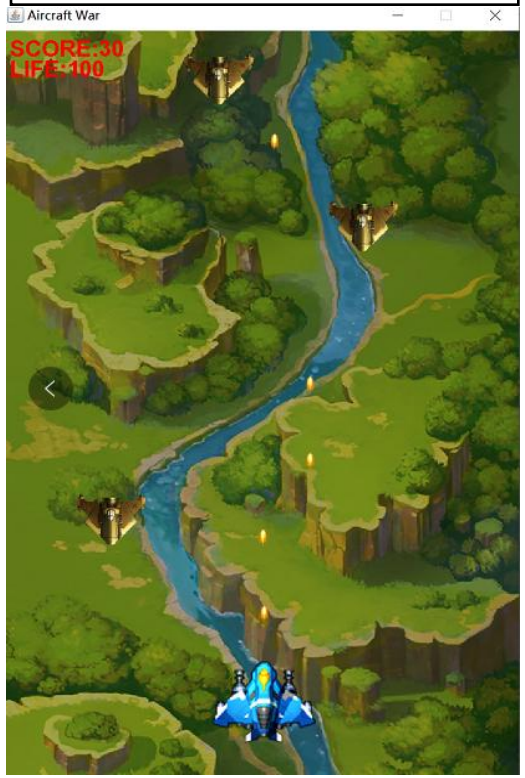
测试驱动的开发

优势

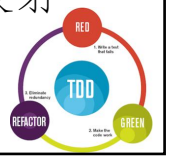
① **强化对需求的理解**，有助于从使用者的角度设计简单易用的接口。
② **强化模块与接口的概念**，促使我们实现松耦合的设计，并更多地依赖于接口，提高系统的可扩展性和抗变性。
③ 频繁运行单元测试，**尽早发现错误**，极大地降低后续测试及修复的成本。
④ 提供持续的回归测试，**代码易于维护和重构**。
⑤ 可实时验证功能正确性的测试代码就是最好的代码文档，且与最新代码同步。
⑥ 减轻压力、降低忧虑、提高我们对代码的信心，提高程序开发效率。

测试驱动的开发

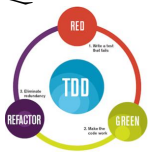
模板程序
- 游戏主界面
- 英雄机、普通敌机
- 飞机移动
- 英雄机子弹发射
- 碰撞检测



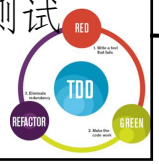
实验一
- 添加所有敌机和道具类
- 制造精英敌机
- 精英敌机子弹发射
- 加血道具生效
- 重构代码



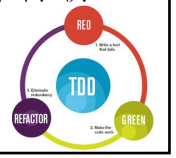
实验二
- 采用**单例模式**制造英雄机
- 采用**工厂模式**制造敌机和道具
- 重构代码



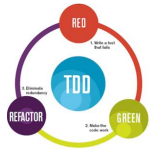
实验三
- 添加JUnit单元测试
- 重构代码



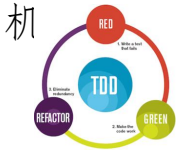
实验四
- 采用**策略模式**实现子弹不同弹道发射
- 采用**数据访问对象模式**实现得分排行榜
- 重构代码



实验五
- 使用Swing添加游戏难度选择和排行榜界面
- 使用多线程实现音效的开启/关闭
- 使用多线程实现火力道具
- 重构代码



实验六
- 采用**观察者模式**实现炸弹道具
- 采用**模板模式**实现三种游戏难度
- 实现BOSS敌机
- 重构代码



软件构造的目标

可理解性

- 遵循特定的命名规范
- 有足够的注释与说明
- 代码布局合理：缩进/空行/分块/对齐等
- 文件组织方式合理
- 方法不应过长或过短，应当容易理解
- 避免多层嵌套 – 增加复杂度

Example #a：
z = ((3*x^2) + (4*x) – 5) – ((2*y^2) – (7*y) + 11) / ((3*x^2) + (4*x) – 5)

Example #b：
a = ((3*x^2) + (4*x) – 5)
b = ((2*y^2) – (7*y) + 11)
z = (a – b) / a

可维护性

- 软件发生变化时，是否可以以很小的代价适应变化？
- 模块化编程，高内聚低耦合
- 灵活使用设计模式

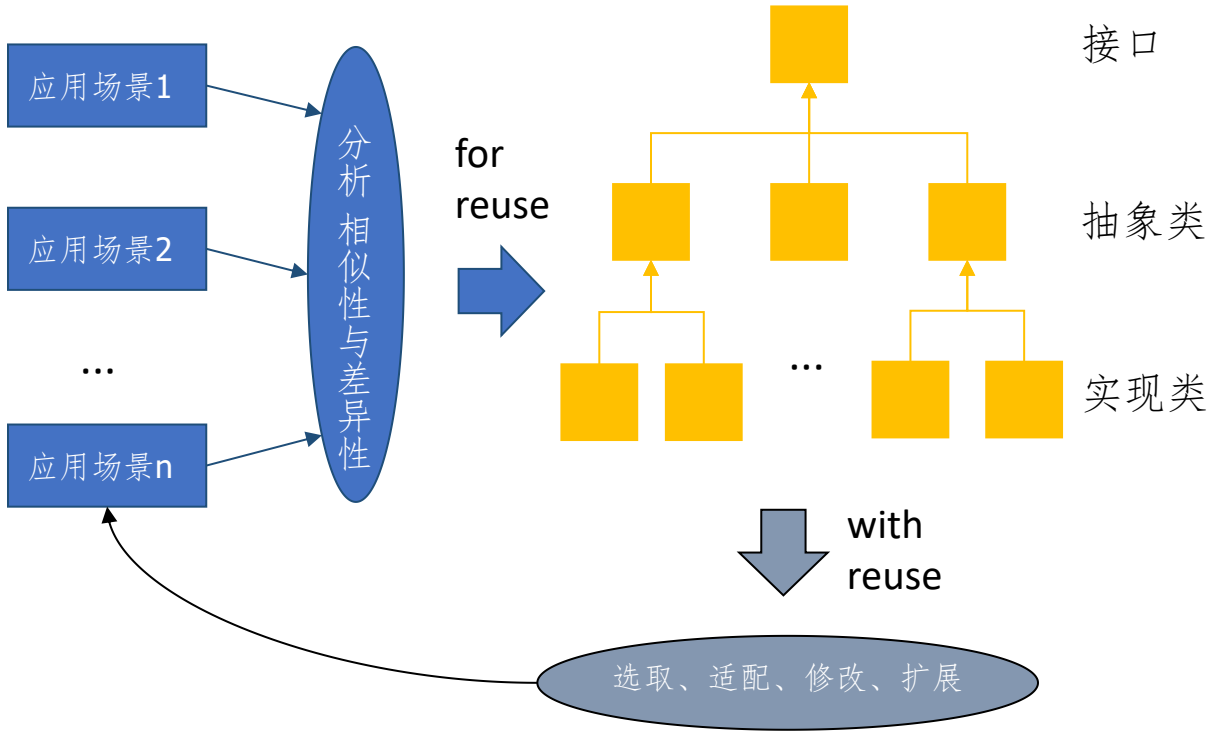
软件构造的目标

可复用性

- 构造可在不同应用中重复使用的软件模块/API
- 模块化，泛型化，抽象化
- 提供完备的细粒度操作，保证功能完整性，不同场景复用不同操作

时空性能

- 内存性能
- 算法性能
- I/O性能



https://github.com/google/guava
https://github.com/spring-projects/spring-framework

软件构造的目标

◘可理解性

- 遵循特定的命名规范

```java
// 类名大写开头，方法名动宾结构
public class OrderProcessor {
    public void calculateTotalPrice() {
        int itemCount = 5; // 小写开头变量
        // ...
    }
}
```

```java
public class c1 {
    public void abc() {
        int a = 5; // 无法理解变量用途
        // ...
    }
}
```

- 有足够的注释与说明

```java
/**
 * 计算订单折扣（满100减20）
 * @param originalAmount 原始金额
 * @return 折后金额
 */
public double applyDiscount(double originalAmount) {
    return originalAmount >= 100 ? originalAmount - 20 : originalAmount;
}
```

```java
public double calc(double amt) {
    return amt >= 100 ? amt - 20 : amt; // 20代表什么？100是什么标准？
}
```

软件构造的目标

可理解性

- 代码布局合理：

  缩进/空行/分块/对齐等

```
public void processOrder(Order order){
if(order.isValid()){updateInventory(order);}
double total=calculateTotal(order);applyDiscount(total);}
```

```
public void processOrder(Order order) {
    // 验证区块
    if (order.isValid()) {
        updateInventory(order);
    }

    // 计算区块（用空行分隔）
    double total = calculateTotal(order);
    applyDiscount(total);
}
```

- 文件组织方式合理

```
/src
  /service
    OrderService.java
  /model
    Order.java
  /util
    PriceCalculator.java
```

```
/src
  AllClasses.java // 包含Order/Customer/Utils等所有类
```

软件构造的目标

可理解性

- 方法不应过长或过短，应当容易理解

```java
public void processOrder(Order order) {
    validateOrder(order);
    updateInventory(order);
    generateInvoice(order);
}

private void validateOrder(Order order) {
    // 10行以内验证逻辑
}
```

```java
public void doEverything() {
    // 验证订单
    // 更新库存
    // 计算价格
    // 生成发票
    // ... 300行代码
}
```

- 避免多层嵌套 – 增加复杂度

```java
public void process(Order order) {
    if (order != null) {
        for (Item item : order.getItems()) {
            if (item != null) {
                if (item.inStock()) {
                    // 处理代码...
                }
            }
        }
    }
}
```

```java
public void process(Order order) {
    if (!order.isValid()) return; // 卫语句

    for (Item item : order.getItems()) {
        if (item.inStock()) {
            processItem(item);
        }
    }
}
```

软件构造的目标

可维护性

- 软件发生变化时，是否可以以很小的代价适应变化？ -- 接口解耦

```
class Notifier {
    void notify() {
        EmailSender sender = new EmailSender(); // 直接绑定具体类
        sender.send();
    }
}
```

```
interface Sender { void send(); }
class EmailSender implements Sender { public void send() { /* 邮件发送 */ } }
class SMSSender implements Sender { public void send() { /* 短信发送 */ } }

// 使用方依赖接口
class Notifier {
    private Sender sender;
    void notify() { sender.send(); }
}
```

软件构造的目标

可维护性

- 模块化编程

```
class UserService {
    void register() {
        // 注册逻辑...
        // 混合日志记录
        System.out.println("记录日志");
    }
}
```

```
// 用户模块
class UserService { void register() { /* 注册逻辑 */ } }
// 日志模块
class Logger { void log() { /* 日志记录 */ } }

// 各自独立修改
```

- 高内聚低耦合

```
// 上帝厨房（低内聚）
class GodKitchen {
    // 混杂煮饭和炒菜逻辑
    void doEverything() {
        measureRice();  // 煮饭步骤1
        heatOil();      // 炒菜步骤1 ← 毫不相关!
        addWater();     // 煮饭步骤2
        stirFood();     // 炒菜步骤2
    }

    // 高耦合：直接访问其他模块细节
    void fixCooker() {
        // 需要知道电饭煲内部电路结构
        checkHeatingElement(RiceCooker.innerCircuit);
    }
}
```

```
// 高内聚的电饭煲模块 - 只关注煮饭功能
class RiceCooker {
    void cookRice() {
        /* 1. 量米 2. 加水 3. 启动 */
    }
}

// 高内聚的炒菜模块 - 只关注烹饪功能
class Stove {
    void stirFry() {
        /* 1. 热油 2. 翻炒 3. 调味 */
    }
}

// 低耦合的厨房控制器（通过参数交互）
class Kitchen {
    void makeMeal(RiceCooker rc, Stove s) {
        rc.cookRice();  // 仅通过接口调用
        s.stirFry();    // 不依赖内部实现
    }
}
```

软件构造的目标

可复用性

- 模块

```java
// 字符串工具类（可在任何项目复用）
class StringUtils {
    public static boolean isBlank(String s) {
        return s == null || s.trim().isEmpty();
    }
}
```

```java
class OrderService {
    // 只能在本类使用
    private boolean validateName(String name) {
        return name != null && !name.trim().isEmpty();
    }
}
```

- 泛型

```java
class Box<T> {  // 可装任何类型
    private T content;
    public void put(T item) { this.content = item; }
}
```

```java
class StringBox {  // 只能装字符串
    private String content;
    public void put(String item) { this.content = item; }
}
```

- 抽象

```java
interface Payment {
    void pay(double amount);  // 微信/支付宝/银联均可实现
}
```

```java
class WeChatPayment {
    void payByWeChat(double amt) { /* 仅限微信 */ }
}
```

软件构造的目标

可复用性

- 提供完备的细粒度操作，保证功能完整性，不同场景复用不同操作

```
class DateTool {
    // 提供多个细粒度方法
    static LocalDate parse(String date) { /*...*/ }
    static String format(LocalDate date) { /*...*/ }
    static boolean isWeekend(LocalDate date) { /*...*/ }
}
```

```
class MyDateUtils {
    // 只有单一功能无法扩展
    static LocalDate onlyParse(String date) { /*...*/ }
}
```

单选题 1分

设置

以下哪项**不是**测试驱动开发的优点?

A 有利于设计易用的接口

B 适用于快速迭代开发，灵活处理需求变化

C 易于重构，有利于代码维护

D 频繁测试，能尽早发现错误

提交

单选题 1分

设置

以下哪项**不是**测试驱动开发的优点?

A 有利于设计易用的接口

B 适用于快速迭代开发，灵活处理需求变化

C 易于重构，有利于代码维护

D 频繁测试，能尽早发现错误

提交

第一章

- 软件构造基本流程及目标
- 面向对象思想
- 设计模式
- Java语言简介
- Java开发环境搭建及程序示例

软件工程方法

结构化方法(面向过程)

- 复杂世界—>复杂处理过程（事情的发生发展）
- 设计一系列功能（或算法）以解决某一问题
- 寻找适当的方法存储、处理数据

面向对象方法

- 任何系统都是由能够完成一组相关任务的对象构成
- 如果对象依赖于一个不属于它负责的任务，那么就需要访问负责此任务的另一个对象（调用其他对象的方法）
- 一个对象不能直接操作另一个对象内部的数据，它也不能使其它对象直接访问自己的数据
- 所有的交流都必须通过方法调用

面向对象三大特性

- **封装(Encapsulation)：**
  - 隐藏对象的属性和实现细节，仅对外公开访问方法；
  - 增强安全性和简化编程

学生
- 私有信息和操作
- 公开操作

- **继承(Inheritance)**
  - 子类继承父类的特征和行为
  - 实现代码的复用

- 学生--计算机学院学生

- **多态(Polymorphism)**
  - 同一个行为具有多个不同表现形态的能力（“一个接口，多个方法”）
  - 提高了程序的扩展性和可维护性

- 上课：
  - 计算机学院-编译原理
  - 理学院-物理学

面向对象三大特性

封装(Encapsulation)：

- 隐藏对象的属性和实现细节，仅对外公开访问方法；
- 增强安全性和简化编程

学生
- 私有信息和操作
- 公开操作

```java
public class Student {
    // 私有属性（隐藏实现细节）
    private String idCardNumber; // 私有信息
    private double gpa;

    // 公开方法（受控访问）
    public void setGPA(double grade) {
        if (grade >= 0 && grade <= 4.0) { // 安全性检查
            this.gpa = grade;
        }
    }

    public String getAcademicLevel() { // 简化编程
        return gpa > 3.5 ? "优秀" : "普通";
    }
}
```

面向对象三大特性

继承(Inheritance)

- 子类继承父类的特征和行为
- 实现代码的复用

- 学生--计算机学院学生

```java
// 父类：通用学生特征
public class Student {
    protected String name;

    public void attendClass() {
        System.out.println("学生上课");
    }
}

// 子类：继承并扩展
public class CSStudent extends Student { // 继承关系
    @Override
    public void attendClass() {
        System.out.println("计算机学院-编译原理"); // 特定行为
    }

    public void program() { // 新增方法
        System.out.println("编写代码");
    }
}
```

```java
public class CSStudent {
    private String name;

    public void attendClass() {
        System.out.println("计算机学院-编译原理");
    }
    // 重复Student类的通用属性和方法
}

public class PhysicsStudent {
    private String name; // 重复字段

    public void attendClass() {
        System.out.println("理学院-物理学");
    }
    // 大量重复代码
}
```

面向对象三大特性

多态(Polymorphism)

```java
public class Student {
    public void attendClass() { // 通用方法定义
        System.out.println("上课");
    }
}

// 不同子类不同实现
public class CSStudent extends Student {
    @Override
    public void attendClass() { // 多态表现
        System.out.println("计算机学院-编译原理");
    }
}

public class PhysicsStudent extends Student {
    @Override
    public void attendClass() { // 多态表现
        System.out.println("理学院-物理学");
    }
}

// 使用多态
Student student1 = new CSStudent();
Student student2 = new PhysicsStudent();

student1.attendClass(); // 输出"计算机学院-编译原理"
student2.attendClass(); // 输出"理学院-物理学"
```

- 上课：
  - 计算机学院-编译原理
  - 理学院-物理学

```java
public class Student {
    private String major;

    public void attendClass() {
        // 使用条件判断而非多态
        if ("计算机".equals(major)) {
            System.out.println("计算机学院-编译原理");
        } else if ("物理".equals(major)) {
            System.out.println("理学院-物理学");
        }
        // 新增专业需要修改此方法
    }
}
```

面向对象思想 vs 结构化编程思想

举例：去深圳北站

- 方案一：自己开车
  - 1. 上车启动
  - 2. 出校门左转，留仙大道
  - 3. 如果有红灯，等待，否则继续
  - 4. 在致远中路路口右转
  - 5. …

以上是典型的结构化编程的思维过程

结构化方法承袭了传统的编程思想与编程方法，以**实现计算机的计算功能为前提**。

面向对象思想 vs 结构化编程思想

举例：去深圳北站

- 方案二：打车
  - 1.用软件叫车（创建出租车对象）
  - 2.告诉司机：我要去深圳北站（调用对象方法）
  - 3.闭目养神、静等到达

面向对象思想：一切都是对象、方法的封装。

面向对象思想模拟**客观世界的事物及事物之间的联系**为前提。

面向对象思想 vs 结构化编程思想

举例：带领你的球队打一场比赛

- 方案一：我来指挥
  - 1. 小A队员，持球进攻。
  - 2. 小B队员，出来帮助挡拆。
  - 3. 小C队员、跑空位接应。
  - 4. 小A传给小D
  - 5. 小D投篮，小B抢篮板
  - 6. … …

结构化思想

实现程序或算法的过程

面向对象思想 vs 结构化编程思想

举例：带领你的球队打一场比赛

- 方案二：排兵布阵
  - 1. 小A队员做后卫（创建对象）
  - 2. 小B队员做中锋（创建对象）。
  - 3. 小C、小D、小E（创建对象）
  - 4. 比赛进行
  - 5. 小C受伤了，小F替换他（对象的销毁、创建）

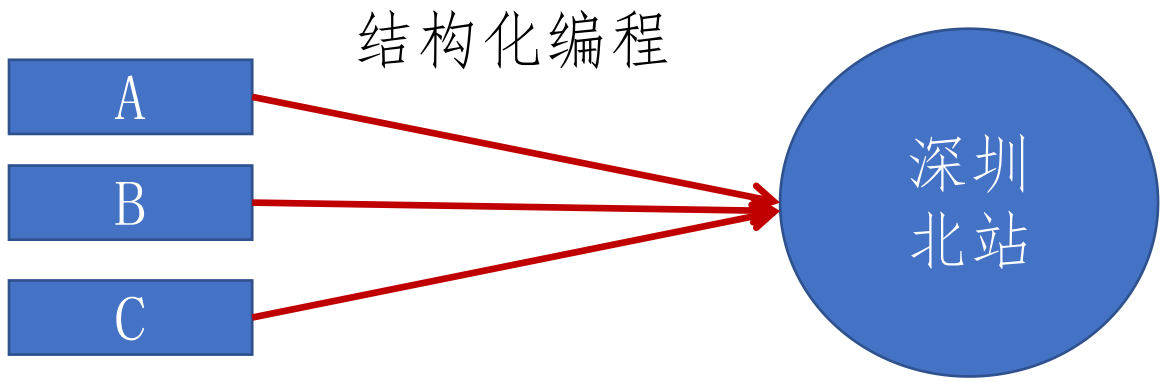
面向对象思想

一切都是对象、方法的封装

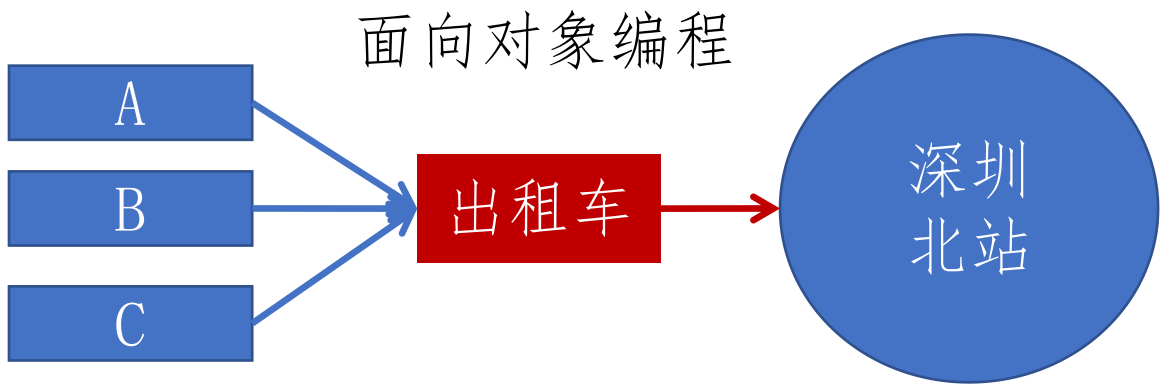
面向对象思想 vs 结构化编程思想

- **面向对象基本思想**
  - 要点1:任何事物都是对象，对象有属性和方法。复杂对象可以由相对简单的对象以某种方式构成。
  - 要点2:通过类比发现对象间的相似性，即对象间的共同属性，是构成对象类的依据。
  - 要点3:对象间的相互联系是通过传递“消息”来完成的。 通过对象之间的消息通信驱动对象执行一系列的操作从而完成某一任务。
- **面向对象优点：**模块化、自然性、并发性、重用性
- 面向对象方法使得软件具有良好的体系结构、便于软件构件化、软件复用和良好的扩展性和维护性，抽象程度高，因而具有较高的生产效率。

面向对象技术在软件构造中的优势：解耦、封装



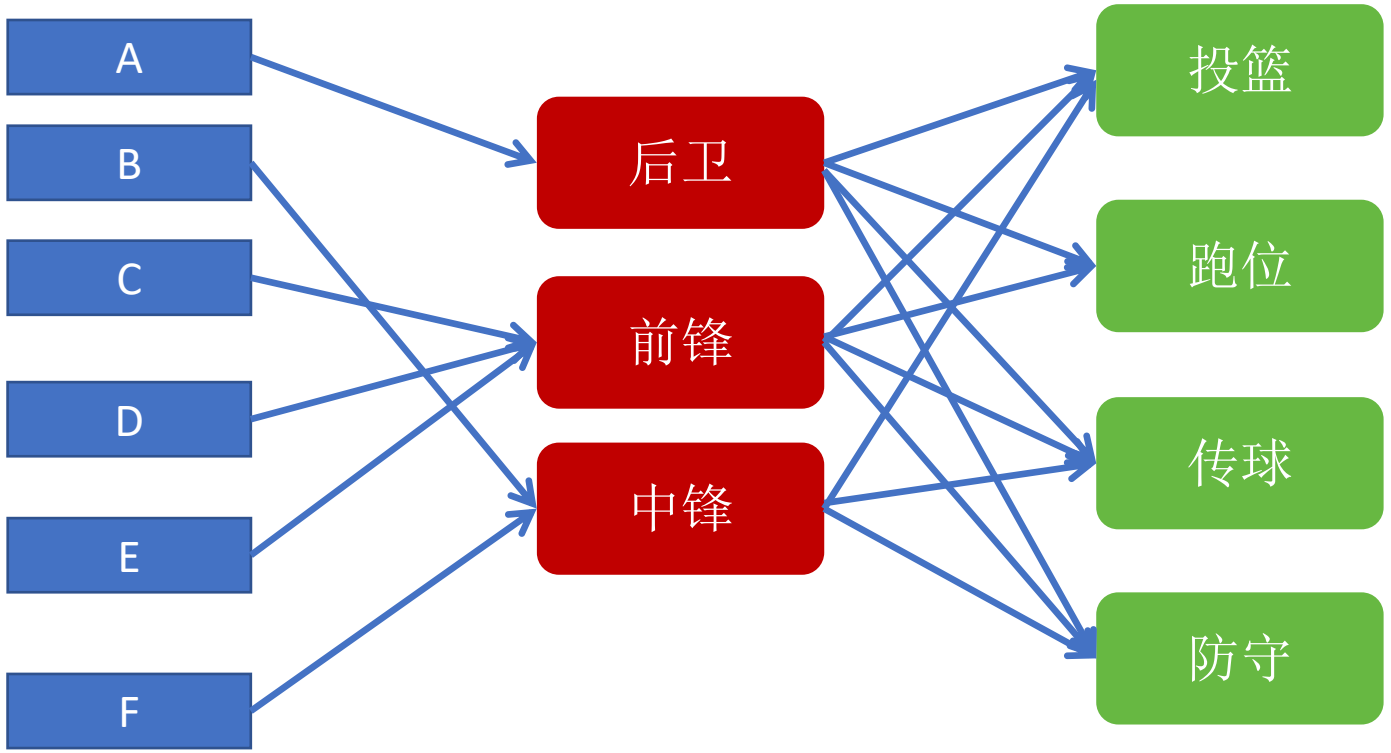
在结构化的编程中，每个人都要包括前往深圳北站的方法，如果发生变化，例如“留仙大道修路”。则需要更改所有人的程序代码。



在面向对象编程中，通过创建对象“出租车”，将去北站的过程单独**“封装”**起来。实现了ABC对于前往深圳北站方法的“解耦”。如果要更改，只需改变出租车类的方法。

面向对象技术在软件构造中的优势：复用

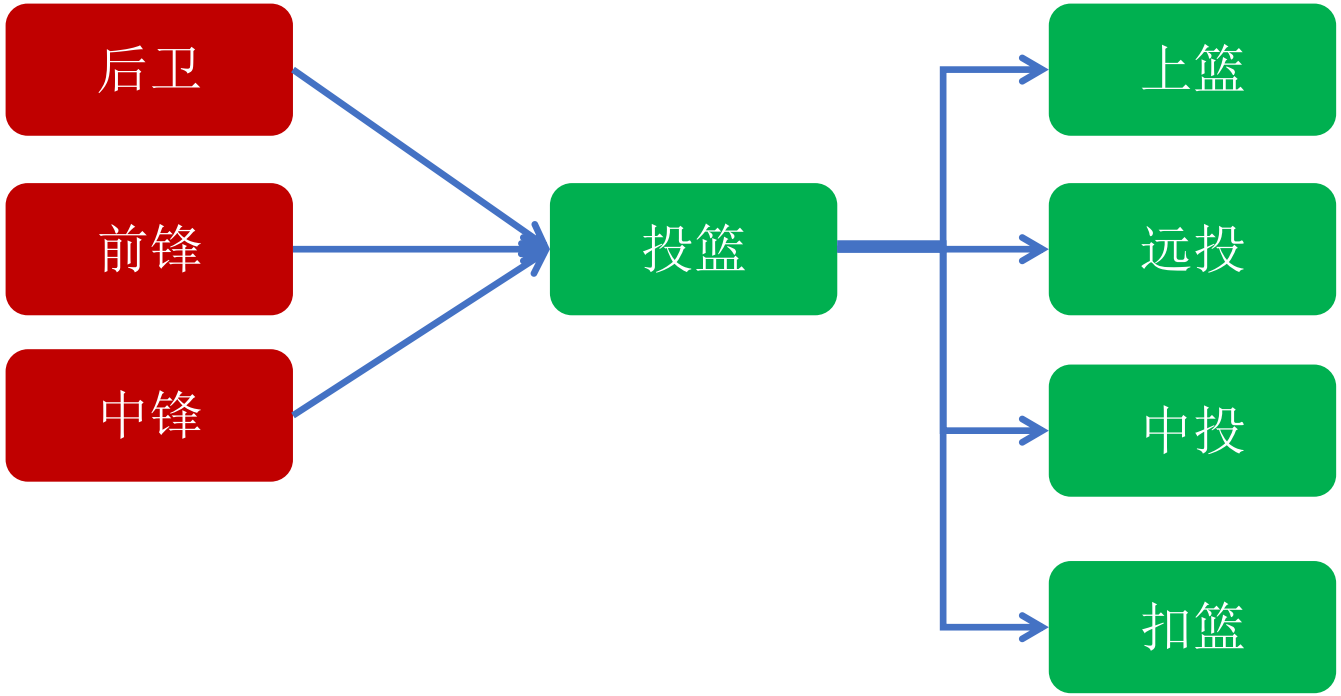
- **结构化编程：**每构造一个新的队员，编码他的投篮、跑位、传球和防守等一系列动作。
- **面向对象编程：**根据新队员所属的“类”，复用之前的代码模块。甚至可以快速的创建新的球员类型。



面向对象技术在软件构造中的优势：多态

面向对象编程的“多态”可以支持用不同的方式实现同一个接口，这样的好处是可以将实现细节与上层对象分开。

想象一下：篮球场上的教练只需要对着持球队员喊：“投篮”，他们就能按照事先设计好的方式，采用各种不同的策略执行这个“投篮”指令了。



第一章

- 软件构造基本流程及目标
- 面向对象思想
- 设计模式
- Java语言简介
- Java开发环境搭建及程序示例

软件构造的基本流程

软件开发生命周期（SDLC，Software Development Life Cycle）从无到有，From 0 to 1：

设计模式的发展历史

“每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动。”

-- Christopher Alexander

美国著名建筑大师、加利福尼亚大学伯克利分校环境结构中心主任克里斯托夫·亚历山大（Christopher Alexander）著作《建筑模式语言：城镇、建筑、构造》（A Pattern Language: Towns Building Construction）

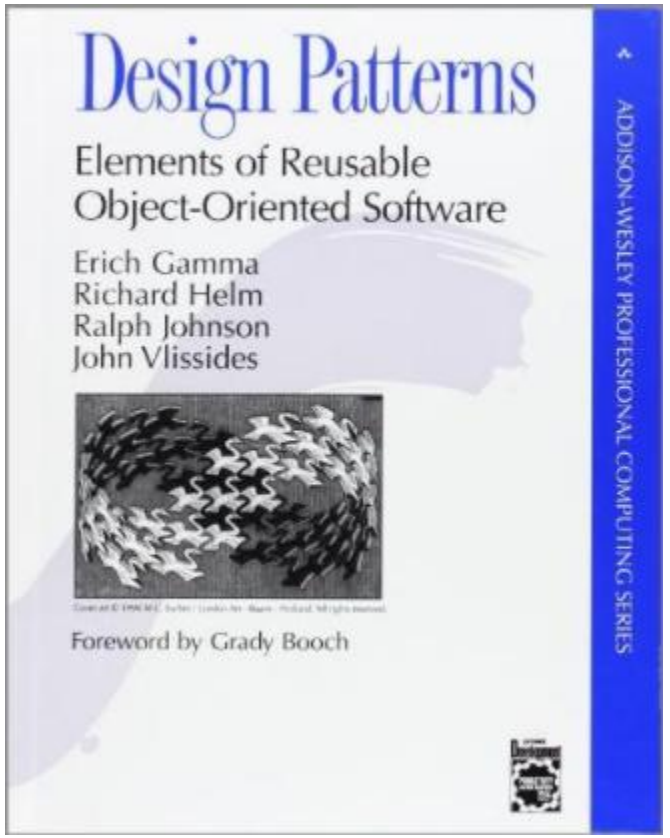
模式的核心：

提供相似问题的解决方案

设计模式的发展历史

“模式”起源于建筑，但同样适用于面向对象的软件设计。

1995 年，由 Erich Gamma、Richard Helm、Ralph Johnson 和 John Vlissides 四人合著出版了一本名为《设计模式 - 可复用面向对象软件的基础》（Design Patterns - Elements of Reusable Object-Oriented Software）的书。



书中收录了 23 个设计模式，首次提到了软件开发中**设计模式**的概念。这是设计模式领域里程碑的事件，导致了软件设计模式的突破。

四位作者合称 **GOF**（四人帮，全拼 Gang of Four）。

设计模式的概念

- 设计模式（Design Pattern）
  - 是一套被反复使用、经过分类编目的代码设计经验的总结。它描述了在软件设计过程中的一些不断重复发生的问题，以及该问题的核心解决方案。
  - 其目的是为了提高代码的可重用性、代码的可阅读性和代码的可靠性。
- 设计模式的本质是面向对象设计原则的实际运用，是对类的封装性、继承性和多态性以及类的关联关系和组合关系的充分理解。
- 注意
  - 一个抽象的方案
  - 模式不是具体的函数库，宏定义，模板类

第一章

- 软件构造基本流程及目标
- 面向对象思想
- 设计模式
- Java语言简介
- Java开发环境搭建及程序示例

什么是Java

- 一种面向对象的编程语言
  - 具有令人赏心悦目的语法和易于理解的语义
- 一个高质量的执行环境
  - 安全性、可移植性、自动垃圾回收
- 一个庞大的库
  - 炫酷的绘图功能、网络连接功能、数据库存取功能

Java语言发展历史

1991年，由 James Gosling带领的Sun公司的工程师小组计划设计出一种代码短小、紧凑且与平台无关的语言。这个项目被命名为“Green”。

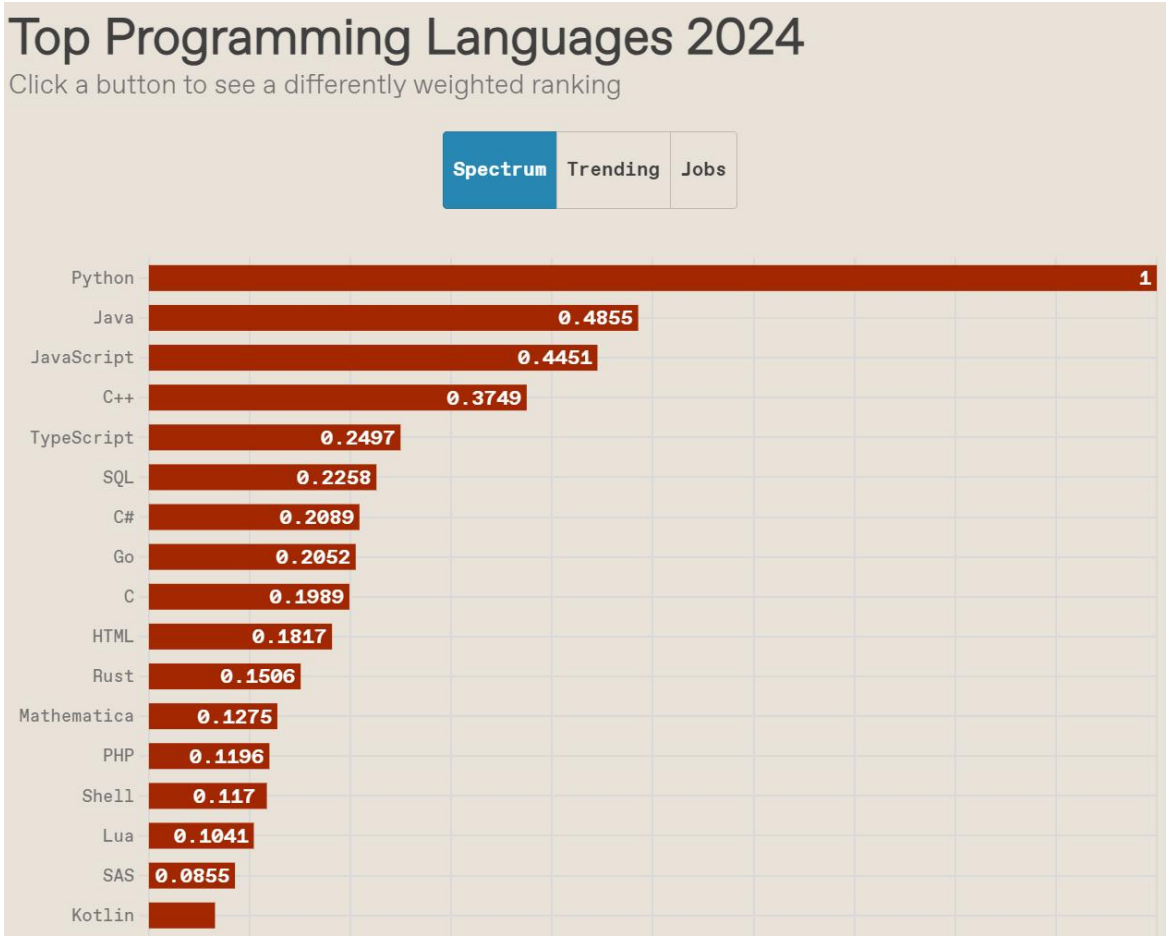
James Gosling

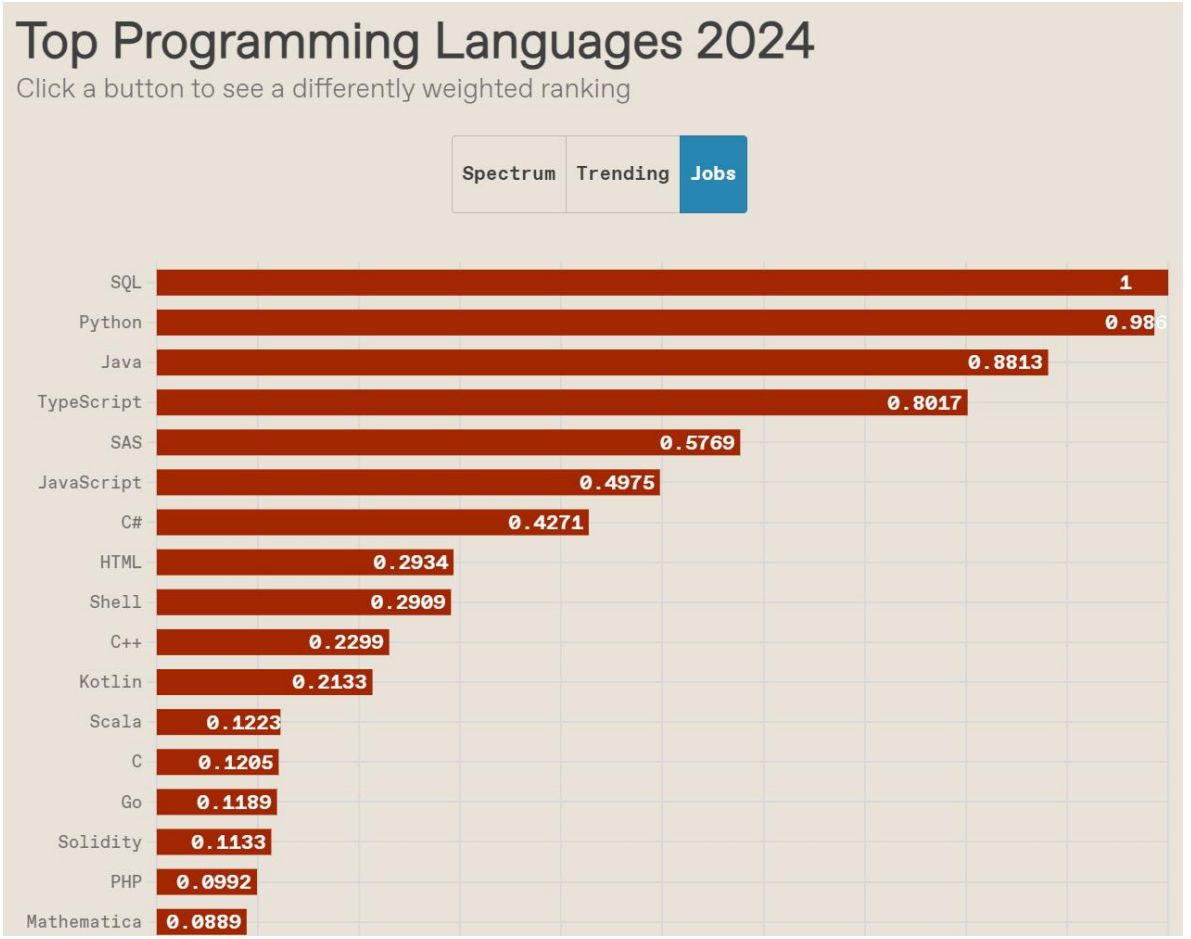
Green项目开发团队

Java语言发展历史

版本	年份	新语言特性	类与接口数量
1.0	1996	语言本身	211
1.1	1997	内部类	477
1.2	1998	strictfp 修饰符	1524
1.3	2000		1840
1.4	2002	断言	2723
5.0	2004	泛类型、“for each”循环、可变元参数、自动装箱、元数据、枚举、静态导入	3279
6	2006		3793
7	2011	基于字符串的选择语句、菱形运算符、二进制字面量、异常处理增强	4024
8	2014	lambda表达式、包含默认方法的接口、流和日期/时间库	4240
9	2017	模块、其他的语言和类库增强	6005

Java语言是最热门的语言之一

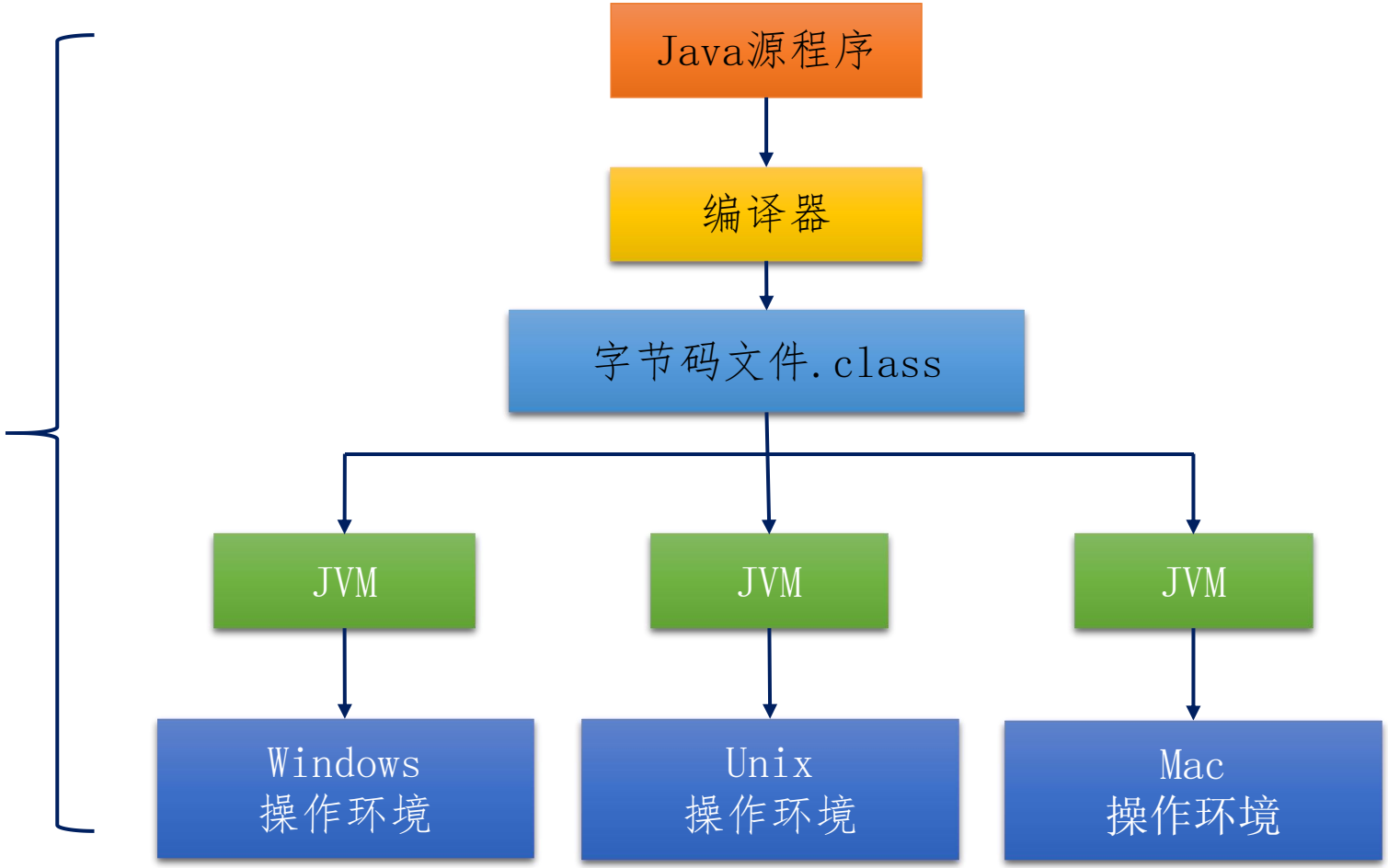




数据来源 https://spectrum.ieee.org/top-programming-languages-2024

Java 语言特点

- 简单性
- 面向对象
- 安全性
- 平台无关
- 多线程
- 网络功能
- 执行效率



Java与C/C++主要区别

- 跨平台
  - C/C++语言编译为机器码，Java语言编译为字节码，通过JVM编译为机器码。
- 废除指针
  - C/C++语言有指针类型，Java语言没有指针类型。
- 继承
  - C++允许多继承，Java语言仅允许单继承。
- 速度
  - C/C++运行速度快，Java运行速度慢。

运行Java程序的基本过程

- 利用编辑器编写Java源程序
  - 源文件名：主类名.java
- 利用编译器(javac)将源程序编译成字节码
  - 字节码文件名：源文件名.class
- 利用虚拟机(解释器,java)运行
  - 运行过程：载入，代码校验，解释执行
  - 如 java 源文件名.class

第一章

- 软件构造基本流程及目标
- 面向对象思想
- 设计模式
- Java语言简介
- Java开发环境搭建及程序示例

Java Development Kit (JDK)

- JDK是 Java 语言的软件开发工具包，包含一批用于Java开发的组件
  - Javac：编译器，将后缀名为.java的源代码编译成后缀名为.class的字节码
  - Java：运行工具，运行.class的字节码

- 根据操作系统版本选取合适的JDK进行安装
  - https://www.oracle.com/java/technologies/javase-jdk11-downloads.html

- 安装后需要配置JAVA_HOME和PATH两个环境变量
  - JAVA_HOME指明JDK安装路径
  - PATH使得系统可以在任何路径下识别java命令

程序示例——使用命令行工具

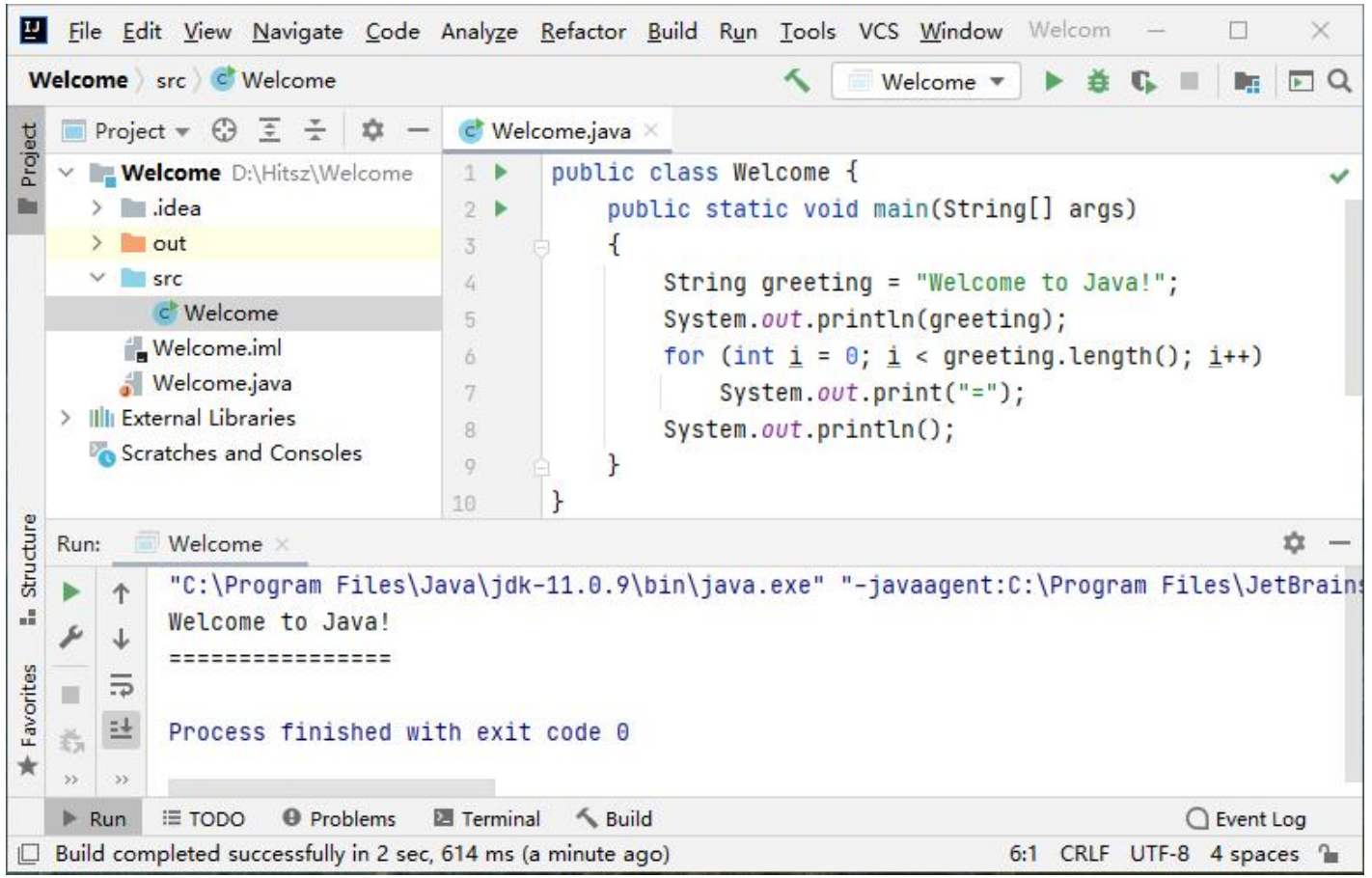
- JDK安装完成后，可以从命令行编译并运行 Java 程序。
  - 创建示例代码
  - 打开一个终端窗口
  - 进入示例代码目录
  - 键入下面的命令：
    - javac Welcome.java
    - java Welcome

示例代码 Welcome/ Welcome.java

```
public class Welcome
{
   public static void main(String[] args)
  {
      String greeting = "Welcome to Java!";
      System.out.println(greeting);
      for (int i = 0; i < greeting.length(); i++)
         System.out.print("=");
      System.out.println();
   }
}
```

Java集成开发环境

- IntelliJ IDEA、Eclipse、NetBeans 等



多选题 1分

设置

面向对象的三大特性包括：

A 封装

B 继承

C 模块化

D 多态

E 流程化

提交

多选题 1分

设置

面向对象的三大特性包括：

A 封装

B 继承

C 模块化

D 多态

E 流程化

提交

第一章

- 软件构造基本流程及目标
- 面向对象思想
- 设计模式
- Java语言简介
- Java开发环境搭建及程序示例