



中国科学院大学  
University of Chinese Academy of Sciences

# 第2课 对象技术导论

面向对象程序设计  
Object-Oriented Programming



# 说明

- 仅供选修国科大“面向对象程序设计”课程的学生复习功课时参考，**不得用于其它目的的翻印制作。**
- 这里所提供的只是课堂讲授的部分内容，**万不可误认为课件以外的内容都不要掌握。**



# 助教有话说



“震震”  
有词



# 考核办法

- 平时作业（编程实训、大作业等）+课堂研讨，**70%**  
大作业及课堂研讨60%（其中课堂研讨20%）  
编程实训及其他课后作业10%  
要求：认真思考，鼓励交流，按时提交，迟交扣分
- 期末成绩，**30%**  
开卷

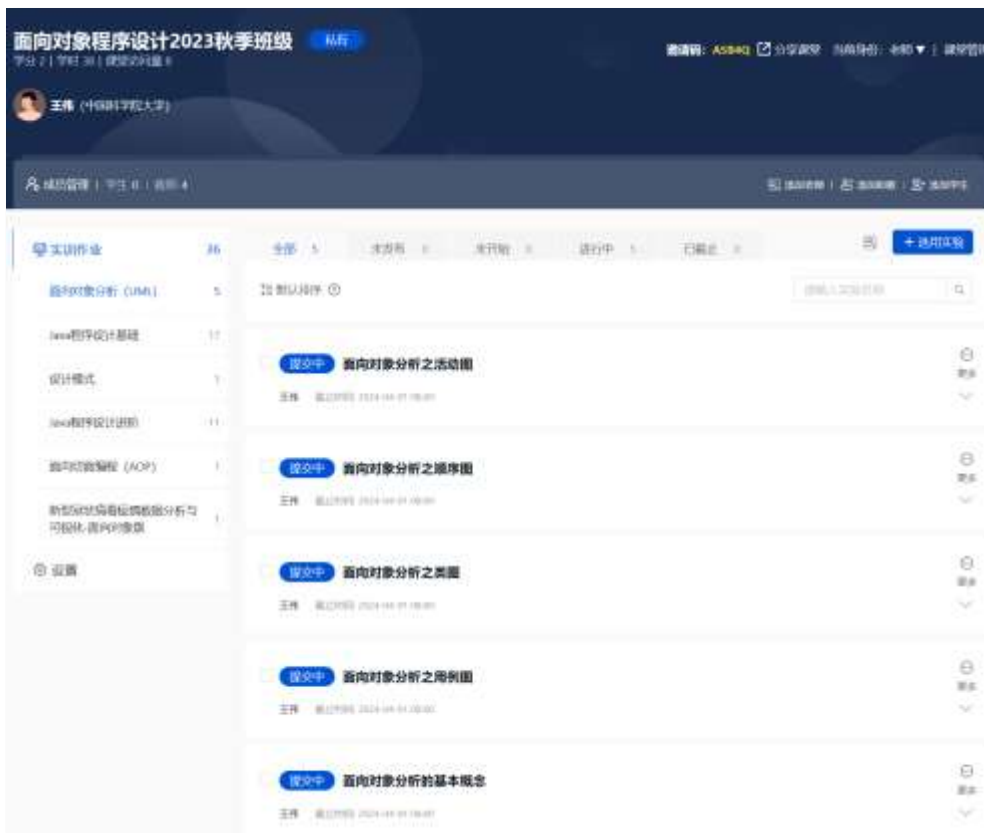


# 编程实训

[www.educoder.net](http://www.educoder.net)

面向对象程序设计  
2023秋季班级

邀请码: **A5B4Q**





# 大作业——设计实现

6	我认为这门课程主要传递了一种面向对象的重要编程思想，而不教你怎么编程的。我觉得老师可以加一些编程教学和编程实践，让理论落地	王伟老师种科研八比喻非常认真负们准备了们每个人趣
---	---	--------------------------

应同学们的强烈要求，大作业继续保留“设计实现”类，利用课上所学知识，设计并开发一个简单小巧实用的工具系统，**推荐与本学期的其他课程大作业或实验室科研任务结合**

## 设计实现类大作业的参考选题：包括但不限于

- **笔记 / 便签：开发一个随手记录零碎信息的应用**
  - 基本要求：支持基本文字排版；支持保存/读取
  - 扩展要求：支持插入图片、视频、公式、表格等；支持Markdown、……
  - 硬核要求：导入导出、历史记录、云存储接入、……
- **文献管理：管理参考文献的元数据及附件**
  - 基本要求：支持对参考文件的增、删、改、查和持久化；支持附件管理
  - 扩展要求：参考文献去重与合并；输出为特定引用格式
  - 硬核要求：自定义引用格式、数据共享、云存储接入、……
- **自拟题目：其他任何难度相当的选题（其他课程大作业、实验室科研实践任务等等）**





# 大作业——顶级开源项目源码阅读与分析

## ● 主题：热门+基础

- 大数据（流计算、图计算）
- 内存计算
- 微服务
- 消息通讯
- 依赖注入
- ...



流处理框架



分布式内存缓存



服务发现与配置管理



网络通讯框架



依赖注入框架



RPC框架

## ● 导师：学术+产业

- 窦文生（博士，软件所研究员）
- 许利杰（博士，软件所副研究员）
- 纪树平（博士，亚马逊研究员）
- 潘旭（百度研究员）
- 唐震（博士，软件所副研究员）



<http://www.tcse.cn/~wsdou>



<http://www.tcse.cn/~xulijie>



<http://www.tcse.cn/~tangzhen12>



# 大作业——大家关心的问题

1. 如何选题：二选一（设计实现、源码阅读与分析）

2. 是否组队：不组队，以个人为单位

3. 提交形式：

- ① 设计实现：随着课程进展，从功能分析与建模、核心流程设计分析、高级设计意图分析依次展开，分3个阶段提交设计报告和源码
- ② 源码阅读与分析：随着课程进展，从功能分析与建模、核心流程设计分析、高级设计意图分析依次展开，分3个阶段提交分析报告
- ③ 作业提交地址：提交至Gitee等源码托管平台，报告格式参考往期优秀报告

雷正宇：<https://842376130.gitbook.io/fastjson-learning-report/>

曾鸿斌：<https://yuankong11.gitbook.io/fastjson/>

4. 课堂研讨：按学号、每人1次

5. 如何评分：

- ① 设计实现题的得分上限高于源码阅读题
- ② 课堂研讨表现占比 20%





# 2023年秋季课程安排

- 第1周, 导言
  - 第2周, 认识软件系统的复杂性
  - 第3周, 有组织的复杂系统
  - 第4周 **中秋假期**
  - 第5周 **国庆补课**, 程序语言的发展
  - 第6周, 建模方法和建模语言
  - 第7周, 抽象
  - 第8周, 封装和模块化
  - 第9周, 复用与解耦: 层次结构
  - 第10周, 复用与解耦: 类型
  - 第11周, 复用与解耦: 抽象类和接口
  - 第12周, 持久化 (时间)
  - 第13周, 持久化 (空间)
  - 第14周, 并发
  - 第15周, 课堂研讨 (设计模式)
  - 第16周, 课堂研讨1 (大作业汇报)
  - 第17周, 课堂研讨2 (大作业汇报)
  - 第18周, 课堂研讨3 (大作业汇报)
  - 第19/20周, 考试
- 16-18周, 补课 课堂研讨4 (大作业汇报)

第3周给出“源码阅读与分析”选题清单

第13周末提交第二次作业

第5周末确定选题

“源码阅读与分析”的选题需要具体到项目的特定模块

第9周末提交第一次作业

第18周末提交第三次作业



# fastjson 学习报告, 雷正宇

<https://842376130.gitbook.io/fastjson-learning-report/>



中国科学院大学





# fastjson 学习报告，曾鸿斌

<https://sbzeng.gitbook.io/fastjson/>



## Fastjson简析

这是笔者在中国科学院大学上王伟老师的面向对象程序设计一课时作业要求——对Fastjson进行分析以考察面向对象思想在其中的应用。

因此，本文的重点不是Fastjson的介绍与分析，也不是具体源码实现的解析与算法设计的讨论，而是对需求建模、主要功能流程设计、类及类间关系、面向对象设计原则及设计模式的分析。总而言之，是对面向对象思想的探讨。

因此如果读者感兴趣的是Fastjson的实现解析，可以参照高东海的[fastjson-source-code-analysis](#)，这篇文章提供了源码的解析。读者在阅读过程中也可以参照他的[fastjson](#)分支，其中添加了大量的注释，一定程度上解决了Fastjson的代码中注释过少带来的阅读不便。

如果读者在阅读过程中有任何意见或者建议，可以发邮件与我联系。我的邮箱是：[zenghongbin16@mails.ucas.ac.cn](mailto:zenghongbin16@mails.ucas.ac.cn)。如果读者对此感兴趣，也很欢迎各位的加入，您同样可以通过邮件联系我。

此外，每节的开头都有主要内容的介绍，希望这可以帮助大家抓住重点。


曾鸿斌

2019.1.9

Next

一. Fastjson简介

→

 Last updated 9 months ago



A majestic, snow-capped mountain peak dominates the background, its rugged ridges partially covered in white snow. The sky is a pale, overcast grey, and several dark silhouettes of birds are captured in flight, scattered across the upper and middle portions of the frame. The foreground shows a rocky, scree-covered slope leading up towards the base of the mountain.

# 对象技术导论（一）

## 认识软件系统的复杂性



# 认识软件系统的复杂性 为什么软件系统是复杂的？



# 我们更关注“工业级”的软件系统

## ● “不复杂” 的软件

- # 由编程新手或独立程序员提出、构建、维护和使用的软件系统
- # 系统的生命周期短，可以轻易被抛弃或被代替，不必尝试去修复Bug、升级功能

## ● “工业级” 软件

- # 表现出丰富行为，反馈式系统，由真实世界的事件驱动，或发出驱动事件
- # 维护着海量规模信息记录的完整性
- # 允许大量并发的更新和查询
- # 系统可以控制真实世界的实体
- # 具有很长的生命周期
- # 随着时间的推移，用户逐渐依赖于这些软件系统的正常工作

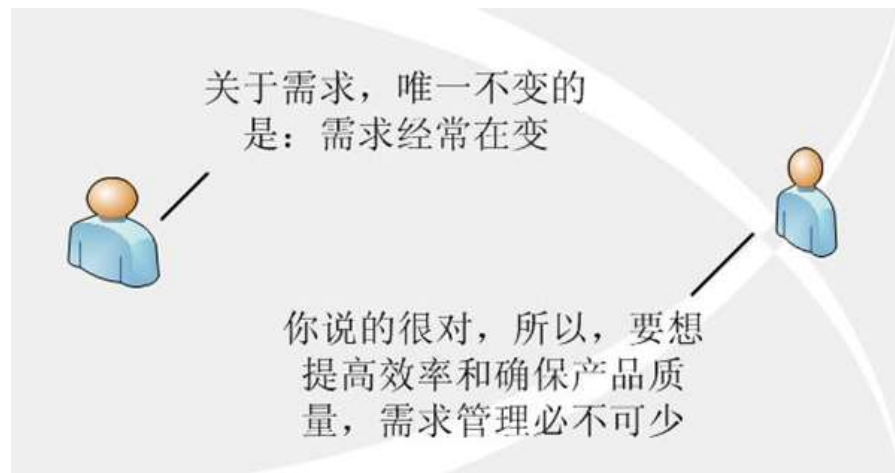
...



# 为什么软件在本质上是复杂的?

## 1. 问题域的复杂性

- # 领域自身的复杂性（卫星导航系统、交通指挥系统、气象监测系统、电子商务系统…）
- # 功能要求、非功能要求（性能、安全、可靠性、可用性…）
- # 软件系统用户和开发者之间的沟通问题
- # 需求经常变更（软件改变了用户“问题域”的行为规则，进而带来更好的需求理解和表述）





# 为什么软件在本质上是复杂的？

## 2. 管理开发过程的困难性

- # 代码管理问题（几十年前，数千行的汇编程序；现在，高级语言编写的系统代码规模达到百万、千万行）
- # 成百上千的独立模块、第三方程序包…
- # 开发团队的沟通、协作问题，如何维持设计的一致性和完整性？



# 为什么软件在本质上是复杂的？ cont.

## 3. 软件中随处可能出现的灵活性

两个世纪前，英国的羊毛纺织厂，养羊、挖煤、修铁路，一个都不能少！

软件行业，这种情况依然存在！

# 软件开发人员可以表达任何形式的抽象，这种灵活性“诱使”开发人员打造几乎所有的初级构建模块

# 缺少品质规范和标准

软件行业至今仍是劳动密集型产业！



# 为什么软件在本质上是复杂的？ cont.

## 4. 描述离散系统行为的问题

# 大型软件系统中，存在大量的程序变量、控制线程以及分布式的子系统，这些变量值、地址、调用栈…等等构成了软件系统的当前状态——**数字计算机系统的离散状态，即系统状态的变化不能够用连续函数来建模**

# 因此，系统外部的输入、子系统之间的数据交互，都可能导致系统状态的非预期变化

### 宰赫兰导弹事件，毫秒的误差

在1991年2月的第一次海湾战争中，一枚伊拉克发射的飞毛腿导弹准确击中美国在沙地阿拉伯的宰赫兰基地，当场炸死28个美国士兵，炸伤100多人，造成美军海湾战争中唯一一次伤亡超过百人的损失

**调查发现，由于软件系统的数据舍入误差，使基地的爱国者反导弹系统失效，未能在空中拦截飞毛腿导弹**





# 为什么软件在本质上是复杂的? cont.

## ● 《没有银弹》[2]

1987年, IBM大型机之父Brooks发表在《Computer》上的软件工程经典论文里指出:

“爱因斯坦认为自然界必定存在着简单的解释, 因为造物主不是反复无常或者随心所欲。但是, 软件工程师没有这样的信仰来提供安慰。许多必须控制的复杂性是随心所欲的复杂性。”

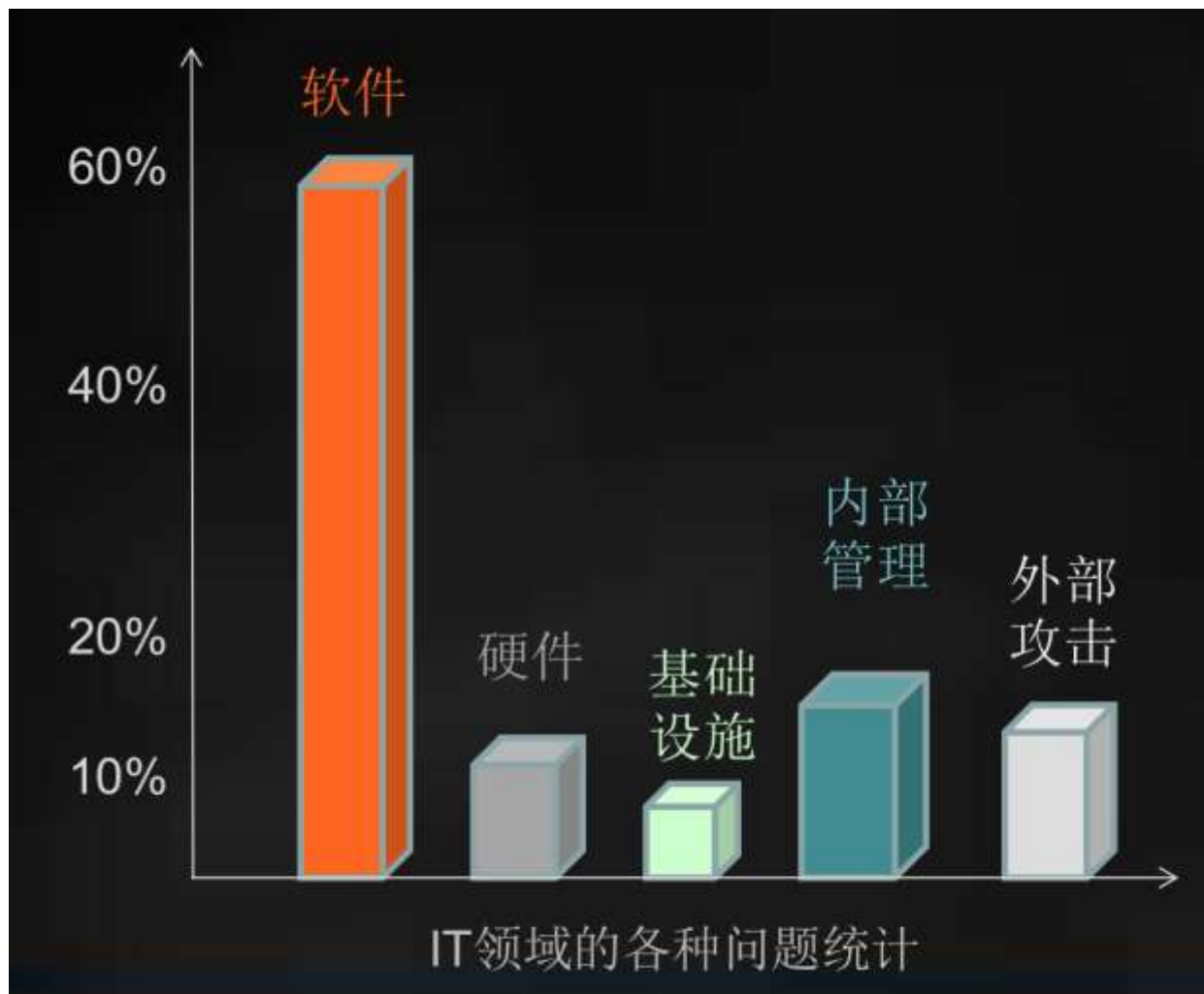
*“Einstein argued that there must be simplified explanations of nature, because God is not capricious or arbitrary. No such faith comforts the software engineer. Much of the complexity that he must master is arbitrary complexity, forced without rhyme or reason by the many human institutions and systems to which his interfaces must conform. These differ from interface to interface, and from time to time, not because of necessity but only because they were designed by different people, rather than by God.”*



[2] Brooks F P. No Silver Bullet Essence and Accidents of Software Engineering. Computer, 1987, 20(4):10-19  
[中文相关材料] <http://www.cnblogs.com/cute/archive/2012/09/27/2705253.html>.



# 我们更关注“工业级”的软件系统







# “系统越复杂，就越容易全面崩溃”

## ● 阿丽亚娜五号运载火箭由于软件系统缺陷而导致安全事故

首次测试发射在1996年6月4日，发射后37秒自身毁灭，原因是控制火箭飞行的软件故障

具体原因是由于火箭导航系统SRI的超负荷运作所致，而这套导航系统则是工程师从4号火箭上原封不动地照搬而来…但是，5号火箭的速度是4号火箭的5倍，导航系统SRI计算的数据在4号火箭上从未出现过

火箭点火升空后测算所得的速度以64位浮点数值表示，导航系统SRI无法将其转换为16位的整数值，最终导致系统的内存溢出，并复写了部分数据，指引火箭偏离既定轨道、最终爆炸



视频地址

<http://v.ku6.com/show/lfO3WygujSbTA4P9fa55qw...html>



# “系统越复杂，就越容易全面崩溃”

- 程序错误和漏算方位角，一箭十九星成为太空垃圾



2017年11月28日，俄罗斯航天集团从位于俄远东地区的东方发射场发射了一枚携有19颗卫星的“联盟-2.1B”运载火箭，但火箭未能将这些卫星送入预定轨道。这些卫星来自俄罗斯、加拿大、挪威、日本、德国、美国等国，其中约半数没有航天发射保险。

俄航天集团总经理科马罗夫指出，分析事故原因后，发现了此前从未被研究过的问题，即佛盖特上面级控制系统软件算法的特殊性。

该算法是大约20年前制定出来的，有62次成功发射记录，但此前佛盖特上面级都是从别的航天发射场升空的。他指出，“联盟号火箭-佛盖特上面级-东方发射场”这样的搭配还是首次出现。

# “系统越复杂，就越容易全面崩溃”

## ● 波音737MAX发生两起空难

2018年10月29日，印尼狮航一架波音737Max飞机坠毁…

2019年3月10日，埃塞俄比亚航空公司一架波音737Max飞机坠毁…

波音公司承认在这两起空难中MCAS均被迎角传感器的错误数据激活，并同驾驶员争夺飞机的控制权，导致飞机最终失控

波音公司随后又承认737Max飞机的软件系统还存在一个“相对较小”、但对飞行安全存在重要影响的问题，该问题会影响到飞机襟翼和其他保持飞行稳定的硬件，FAA已将该问题对于飞行安全的影响列为“紧急”项，并责令波音修复







# “系统越复杂，就越容易全面崩溃”



国内版 国际版

tesla autopilot crash



Sign in



ALL IMAGES VIDEOS

Filter

Image size Color Type Layout People Date License



# “系统越复杂，就越容易全面崩溃”



2012年11月11日，**淘宝**网络瘫痪，**支付宝**提示系统繁忙，一致性放松导致大量订单**“超卖”**（光明网）

2012年1月1日，**12306铁路售票网**访问量过大系统瘫痪，旅客购票遇难题（新京报）



2011年4月26日，**伦敦奥运订票系统**在截止时间前的一小时则飙升到超过预期的峰值，最后时刻瘫痪（法新社）

2007年10月30日，**北京奥运会门票第二阶段预售**，订票人数超预期，瞬间访问数量过大，造成系统“瘫痪”...（新华网）





# “系统越复杂，就越容易全面崩溃”

## 网络化带来的复杂性导致软件系统面临巨大风险



2014年，系统升级失败导致Dropbox整个平台宕机2天



2016年，代码更新错误导致移动与Web端8小时不可用



2015年，复杂更新导致系统停止服务40小时



2016年，大量资源需求导致欧洲用户难以登录邮箱



2015年，内部DNS错误导致iTunes和AppStore宕机12小时



2017年，可疑登录功能更新导致美国、欧洲用户无法登陆



2015年，内部网络错误导致公共云服务宕机超过2小时



2017年，由于人为运维操作错误，导致亚马逊S3服务中断长达5小时，大面积云服务应用无法使用





# “系统越复杂，就越容易全面崩溃”

## ● 大众汽车帝国的软件危机

### 传统汽车 → 软件定义汽车

- 1、碎片化的70多个独立电子控制单元（ECU） → 简化为3个集中式的域控制器（辅助驾驶控制、整车控制、娱乐控制）
- 2、开发人员面向70多个独立ECU操作系统开发 → 开发人员面向vw.OS开发，vw.OS统一管理整车所有芯片、内存、传感器、输入输出，所有上层应用程序由其管理和调度
- 3、vw Automotive Cloud，类似APP Store

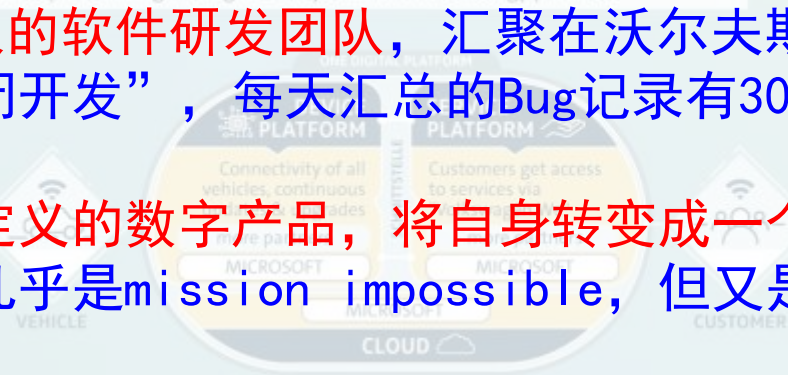
VOLKSWAGEN  
AUTOMOTIVE CLOUD

大众集团、合作伙伴、供应商等将近1万人的软件研发团队，汇聚在沃尔夫斯堡大众总部的74号大厅，夜以继日的“封闭开发”，每天汇总的Bug记录有300左右…

对于传统汽车而言，将汽车变成一个软件定义的数字产品，将自身转变成一个以软件能力为核心的智能硬件科技企业，几乎是mission impossible，但又是一个不得不做的事情。

VOLKSWAGEN AUTOMOTIVE CLOUD

The Group is building the digital ecosystem with technology partners





# “系统越复杂，就越容易全面崩溃”

## ● 大众汽车帝国的软件危机

### vm. OS，新世纪的IBM OS/360？

# 20世纪60年代，随着计算机用途的增多，人们逐渐意识到为每一台新的计算机重新编写相同功能的应用软件是一件低效的工作。

# 添加**操作系统**可以让用户更加方便地用高级编程语言编写程序和管理计算机，应用程序也可以移植到升级后的硬件上，大大方便了应用程序的开发、使用和维护。

# 前瞻到这一趋势，IBM决定为其第一款多功能大型机S/360开发配备后来软件工程领域颇具传奇色彩的OS/360操作系统。**OS/360操作系统**的设计和研发是当时计算机软件领域的一个重要突破。它的设计需要兼顾许多未曾尝试的领域，比如为充分利用中央处理器而实现多程序同时运行，兼顾系列中不同配置的机型，以及各种机型和其程序所需的不同文件的读写要求，这使得它的设计**复杂性呈指数增长**，导致该操作系统直到S/360大型机问世两年后才得以完成。

【来源】操作系统的兴衰，[https://dl.ccf.org.cn/institute/instituteDetail?id=4537806934083584&\\_ack=1](https://dl.ccf.org.cn/institute/instituteDetail?id=4537806934083584&_ack=1)



# “系统越复杂，就越容易全面崩溃”

## ● 大众汽车帝国的软件危机

### vm. OS，新世纪的IBM OS/360？

# IBM为了加快进度，一度添加了很多人手到该项目中，但随着开发人员的不断增多，整个工程的进度反而减缓。软件开发与传统的工程开发有所区别，将软件分割成不同模块同时开发，需不断协调和整合。当模块分割过细时，协调不同模块的进度将变得过于复杂，以至于彻底抵消分工所带来的效率。

# 负责OS/360开发的项目经理是1956年从哈佛大学博士毕业的布鲁克斯。他根据上述观察写出了软件工程领域的经典论文《没有银弹：软件工程中的根本与次要工作》和家喻户晓的《人月神话》一书。

# S/360大型机和OS/360操作系统的成功使得IBM牢牢把握住了20世纪70年代美国计算机市场。因为操作系统的软件兼容性，购买了S/360计算机的公司会更加频繁地进行系统升级，为IBM带来的利润甚至超过了计算机本身的销售。IBM OS/360的后续版本一直延续到IBM最新一代大型机zSeries上的z/OS操作系统，而且都实现了对早期版本的兼容。

【来源】操作系统的兴衰，[https://dl.ccf.org.cn/institute/instituteDetail?id=4537806934083584&\\_ack=1](https://dl.ccf.org.cn/institute/instituteDetail?id=4537806934083584&_ack=1)



# 认识软件系统的复杂性

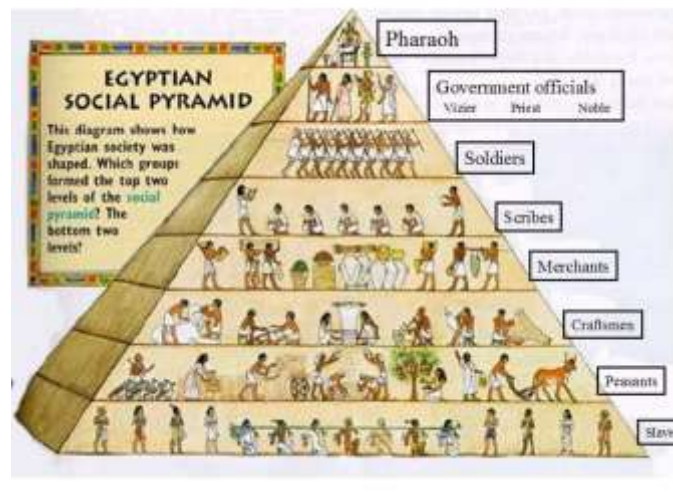
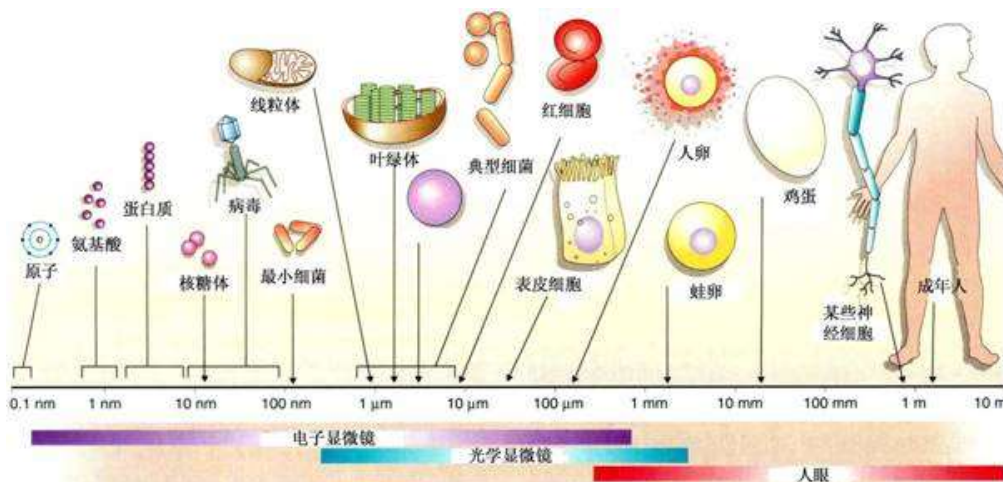
## 复杂系统有哪些特点？

# 复杂系统的5个属性<sup>[3]</sup>

## 1. 层次结构

# “复杂系统常以层次结构的形式存在，即由一些相关的子系统组成，子系统又有自己的子系统，直至达到某种最低层次的基本组件”<sup>[4]</sup>

# 动物的结构、物质的结构、社会系统的结构……我们更善于理解这些有层次结构的系统



[3] (美)布奇(Booch, G.). 面向对象分析与设计: 第3版. 电子工业出版社, 2012.

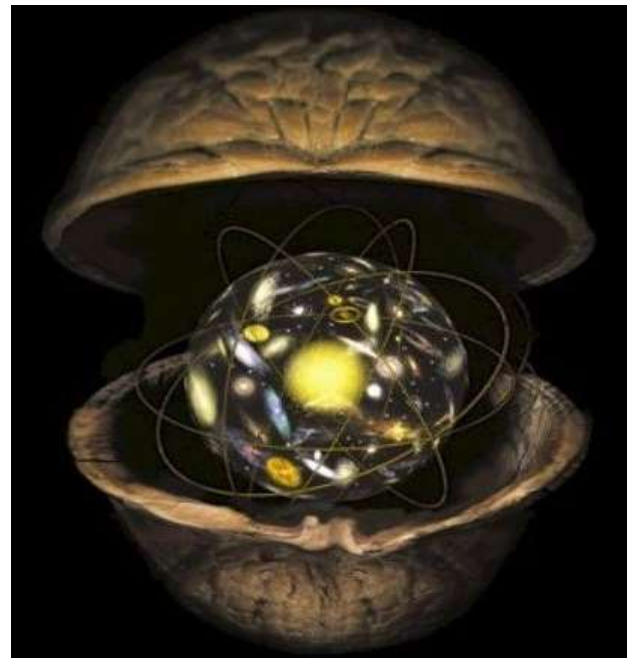
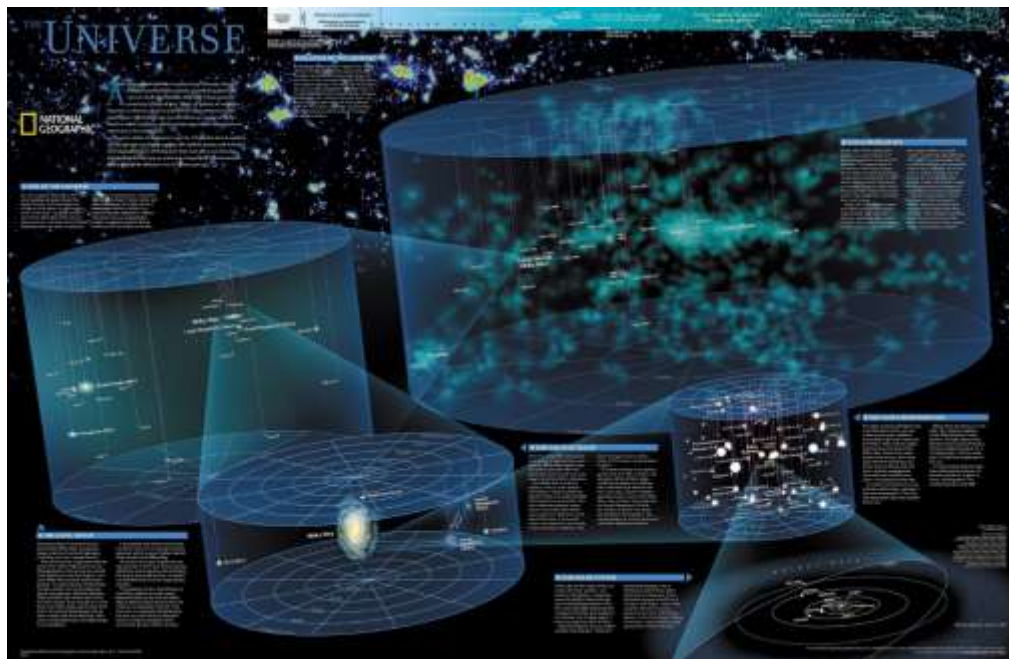
[4] Courtois P J. On time and space decomposition of complex structures. Communications of the ACM, 1985, 28(28):590-603.



# 复杂系统的5个属性 cont.

## 2. 相对基础

- # 对基本组件的选择，很大程度上取决于系统观察者的判断
- # 对于一个观察者来说很基础的东西，对另一个观察者可能具有很高的抽象

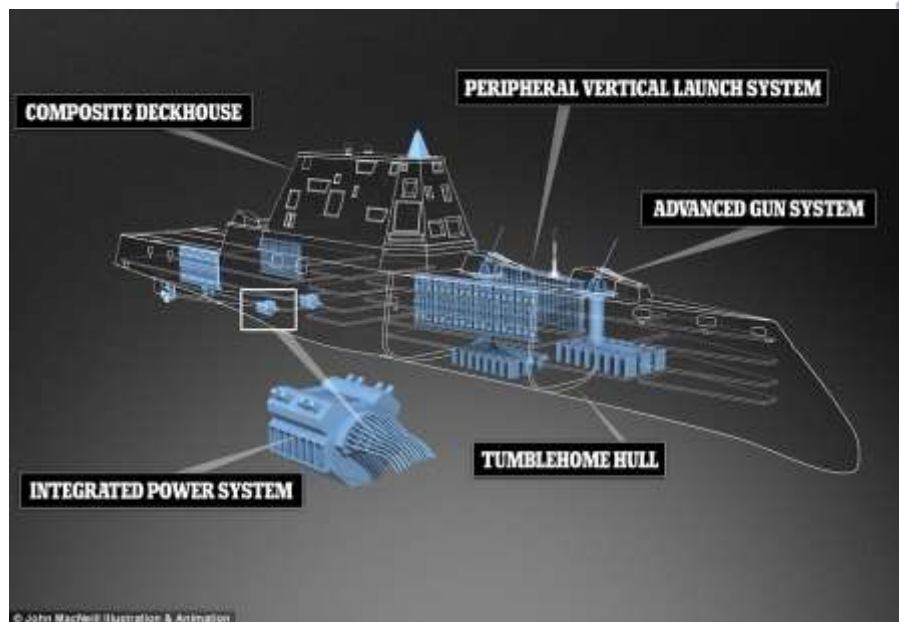




# 复杂系统的5个属性 cont.

## 3. 关注点分离(separation of concerns)

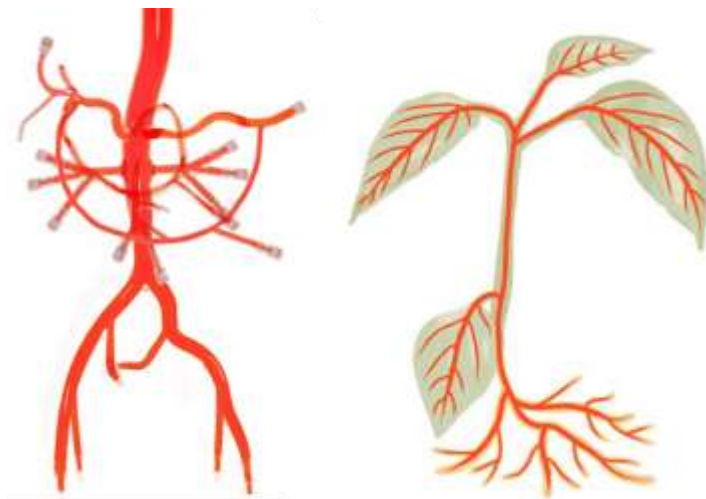
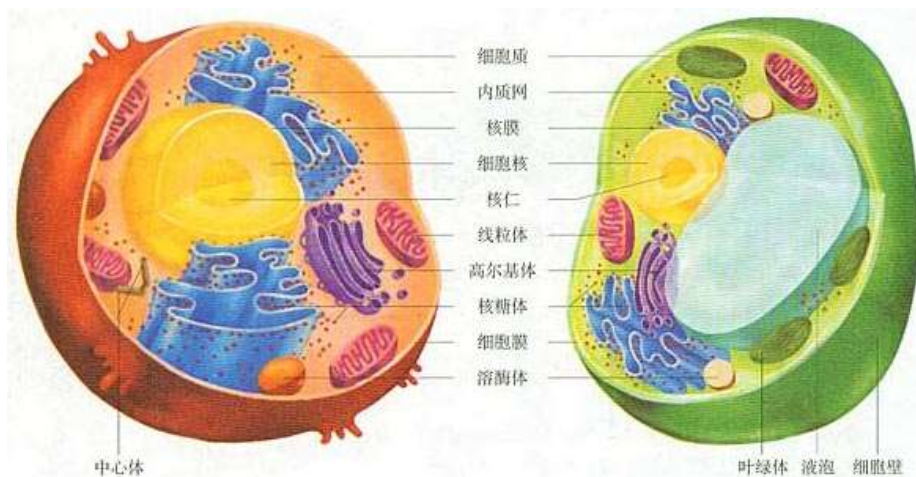
- # 层次系统的可分解性，意味着子系统内部的联系通常比子系统之间的联系更强，子系统内部的高频行为与子系统之间的外部低频行为实现了分离
- # 让我们能够以相对隔离的方式来研究每个部分，也让子系统的行为对其他子系统以及系统整体的影响有限



# 复杂系统的5个属性 cont.

## 4. 共同模式

- # 复杂系统往往是以一种经济的表达方式来实现的，即层次结构通常只是由少数不同类型的子系统，按照不同的组合和安排方式构成
- # 这种共同的模式，其本质是对组件、子系统的复用，例如，细胞、脉管系统，在植物和动物系统中都存在



# 复杂系统的5个属性 cont.

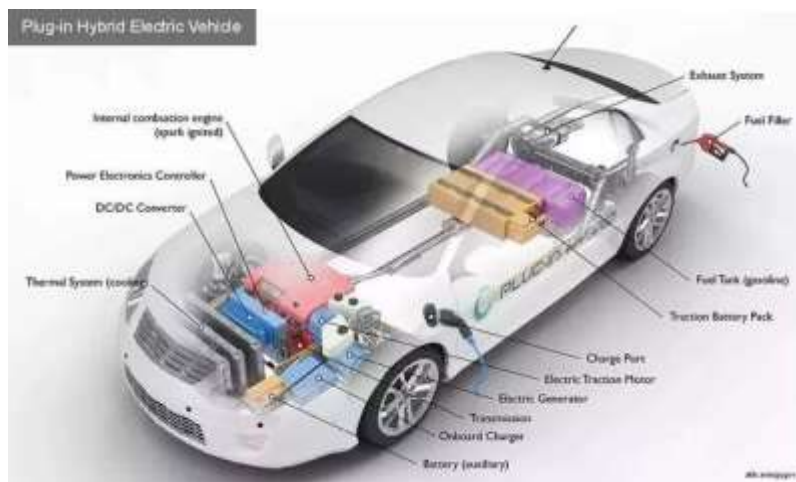
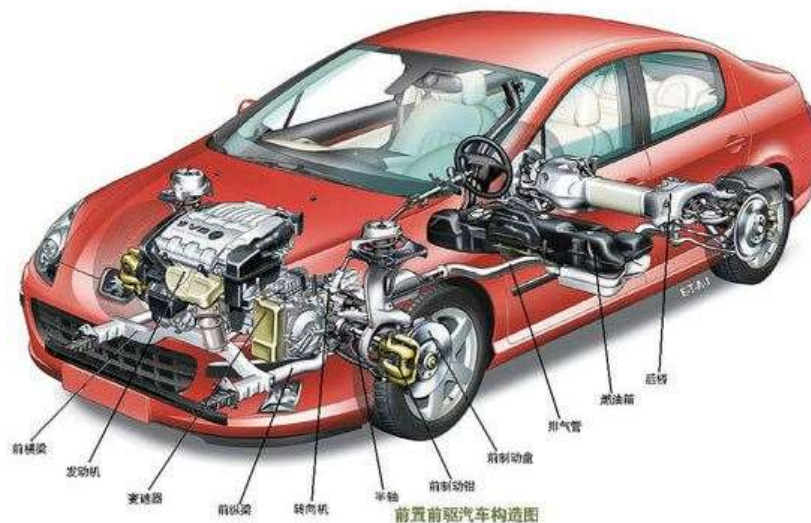
## 4. 共同模式

# 工业产品的例子:

燃油车

混动车

纯电车...



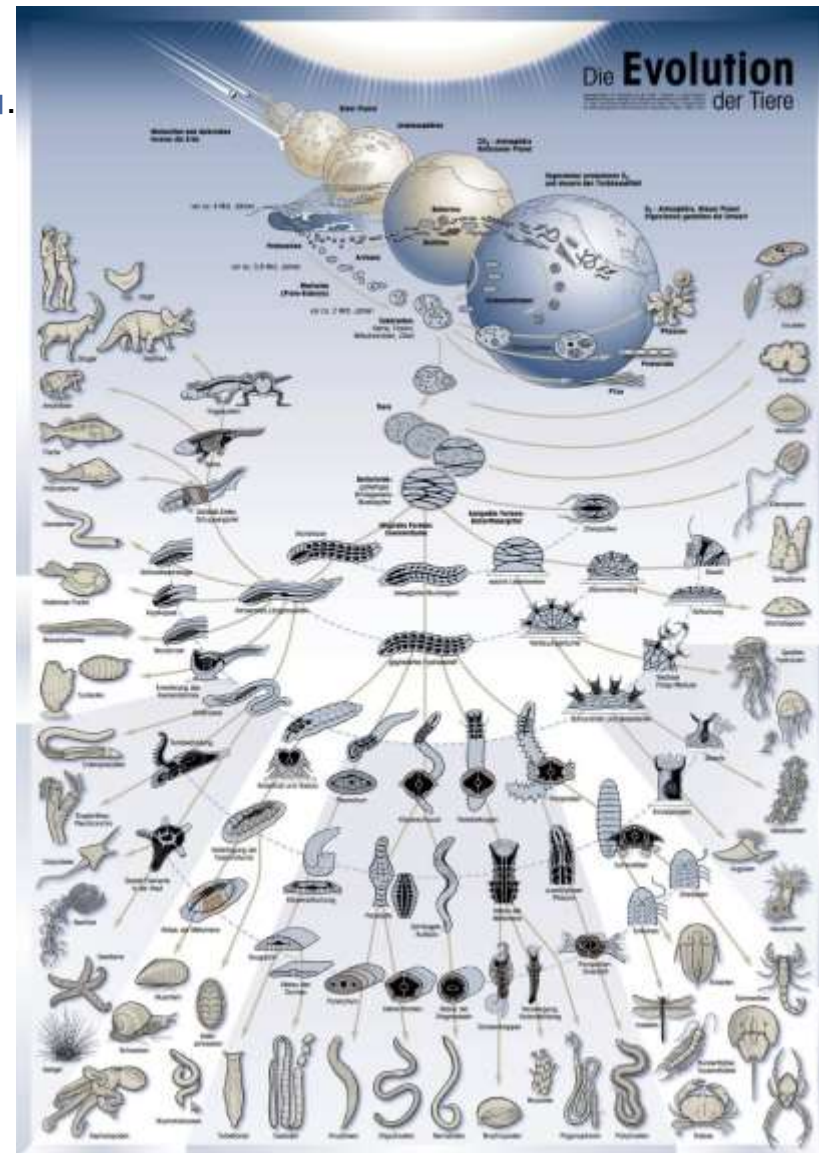
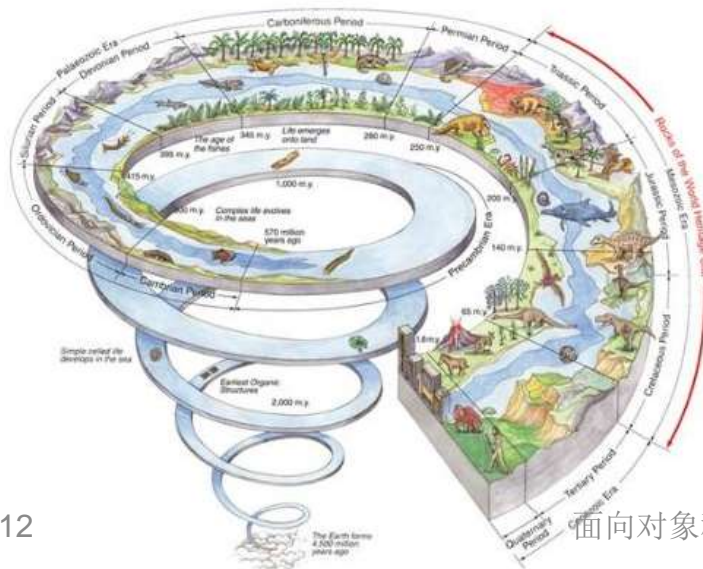


# 复杂系统的5个属性cont.

## 5. 稳定的中间形式（“螺旋”演变）

# 从头设计的复杂系统往往不能工作，打补丁的方式也难以有效，必须从能工作的简单系统开始，持续演进...

# 随着系统的演变，简单系统变得复杂，进一步又变为基础组件，在这些基础组件上构建更复杂的系统







# 小结

## ● “工业级” 软件系统，本质是复杂系统

- # 问题域复杂
- # 开发管理复杂
- # “软件定义一切” 的诱惑
- # 难以刻画离散系统行为

## ● 复杂系统的 “特质”

- # 层次结构
- # 相对基础
- # 关注点分离（分而治之）
- # 共同模式
- # 持续演变（由稳定的 “简单” 到 “稳定” 的复杂）



# Java程序语言基础（一）

## 导论



# Java语言简介

- Java是Sun Microsystems公司推出的面向对象程序设计语言  
原名：Oak，1995年更名为Java

发明者：James Gosling

[http://en.wikipedia.org/wiki/James\\_Gosling](http://en.wikipedia.org/wiki/James_Gosling)



- 2006年Sun Microsystems宣布Java开源
- 2010年Oracle收购Sun

## Java时间轴

<http://oracle.com.edgesuite.net/timeline/java/>





# Java语言特点

- 在Java语言白皮书中，Java被描述为 “是一种简单、面向对象、分布式、解释、健壮、安全、结构中立、可移植、高性能、多线程、动态的语言。

“A simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language”

—James Gosling, Henry McGilton. The Java™ Language Environment, A White Paper.





# 部分特点

## ● 简单易学

去除C++中不容易理解的部分（如指针）

不需要维护内存分配

## ● 面向对象

有利于程序的开发、复用和维护

## ● 平台无关性

Java编译器生成与体系结构无关的字节码，Java解释器得到字节码后，把它转换为目标平台的机器语言，使之能够在不同的平台运行

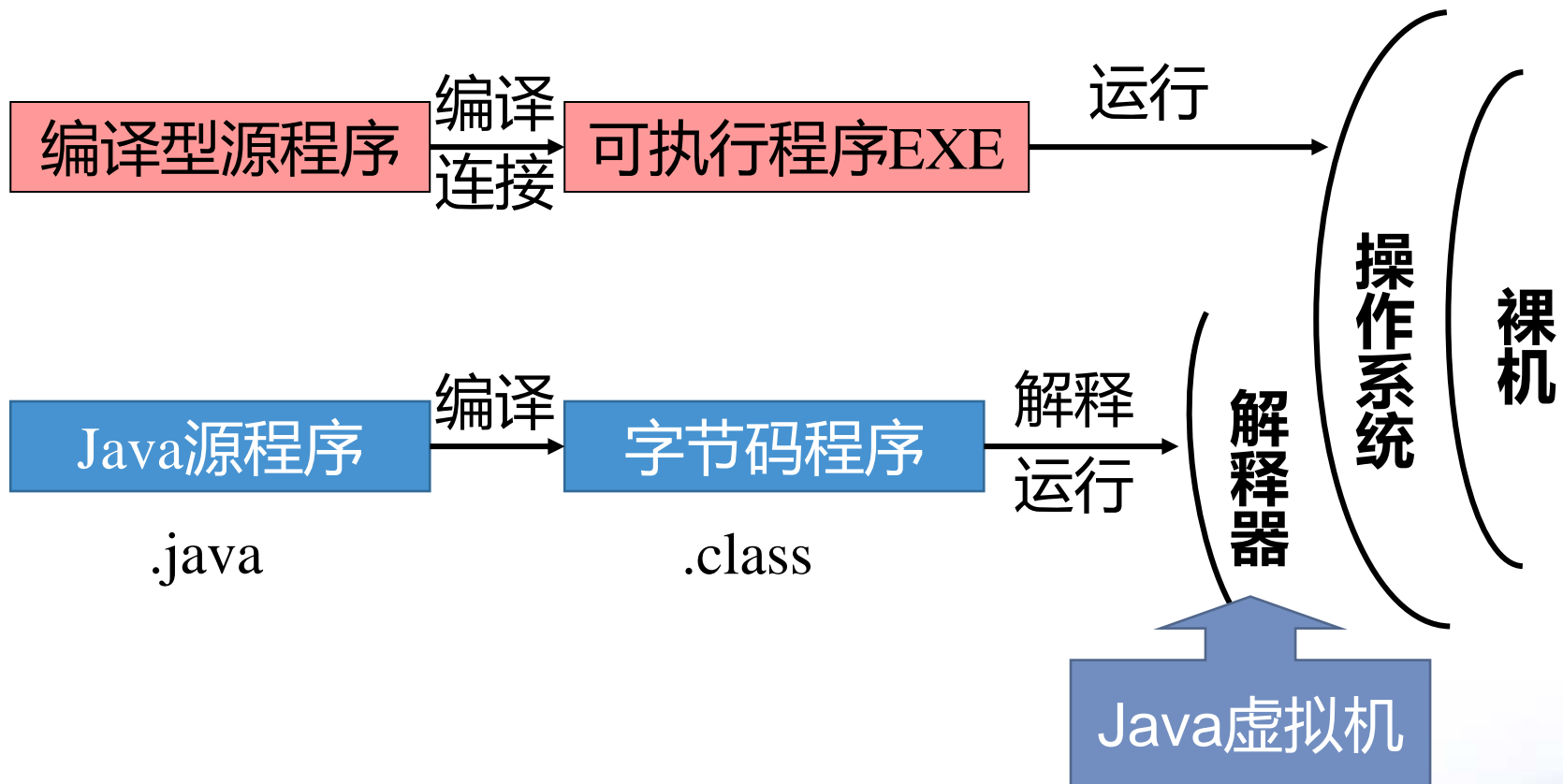
## ● 健壮性

垃圾回收、异常机制



## 部分特点 cont.

*Java是解释型的高级语言*





# Java程序开发、运行的基本过程

## 1. 利用编辑工具编写Java源程序

源代码文件：类名.java

## 2. 利用编译器将源程序编译成字节码

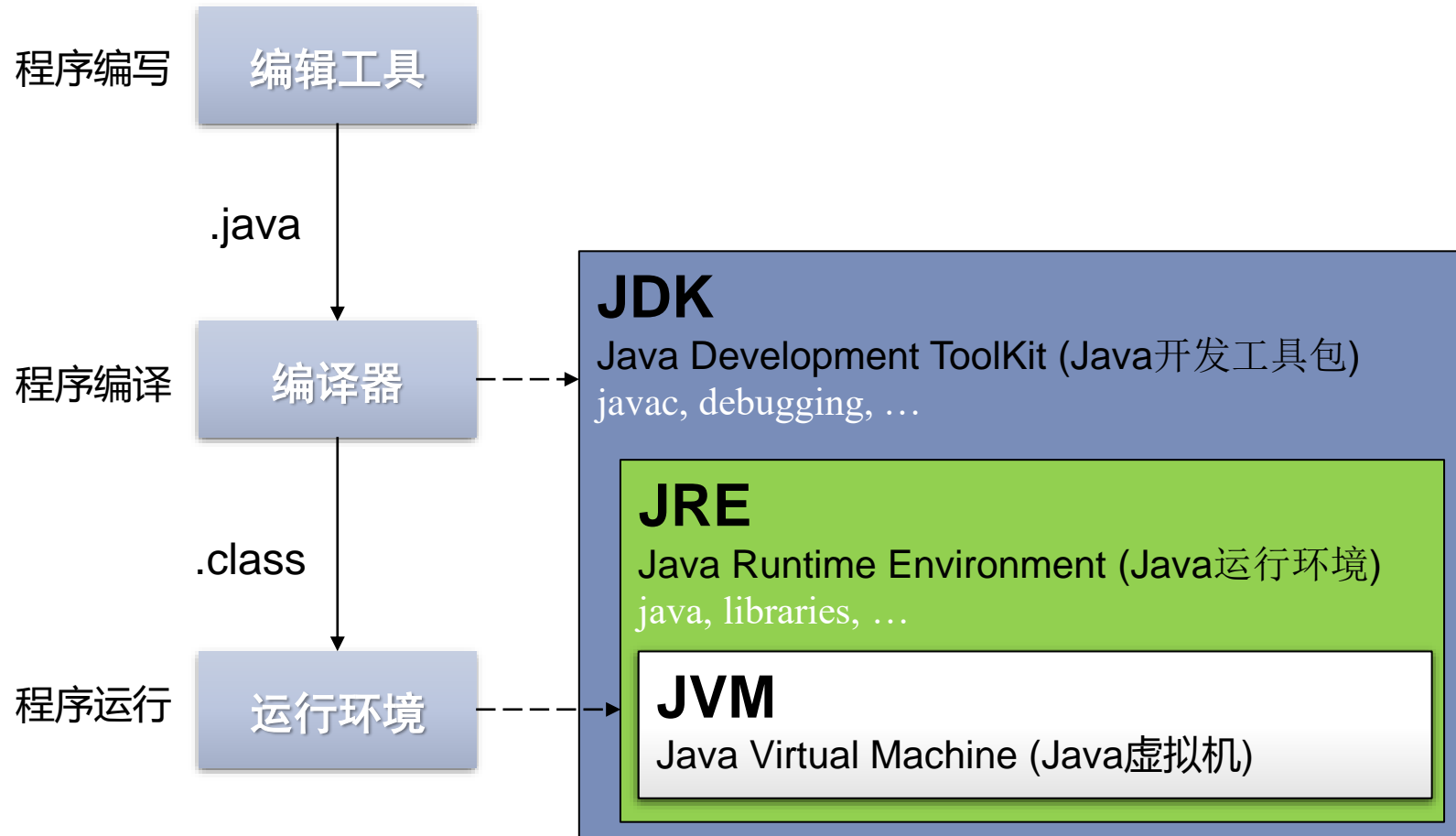
字节码文件：类名.class

## 3. 利用Java虚拟机运行

载入、代码校验、执行



# Java程序开发、运行的基本过程







# 安装并配置JDK

## ● 下载JDK

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

## ● 配置环境变量

### **JAVA\_HOME**

JDK安装路径

### **PATH**

;%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin

### **CLASSPATH**

.;%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar



# 第一个Java应用程序

import语句

class

主类名称

{

public static void main(String[ ] args)

{

方法体

}

}

示例





# 剖析Java程序代码

- 类名
- 主方法
- 语句
- 保留字
- 块
- 注释
- ...



# 类名

- 每个Java程序至少有一个类，类名首字母大写

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```





# 主方法

- 主方法是程序运行的“入口”

主方法不是类的必要组成，但如果要运行一个类，则该类需要包含一个主方法

除了参数名称，主方法的格式是固定的

```
// This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```



# 语句

- 一条语句表示一个或一组行为
- 语句以 “; ” 结束

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



# 保留字

- 保留字，也称关键字，是对编译器有特定含义的单词

例如：当编译器看到单词class，就会明白class后面的单词就是这个类的

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



# 块

## ● 一对大括号 { }

类块

方法块

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```





# 注释

- 行注释

行注释是在单行上最前面加两个斜杠 (//) 表示注释某行

- 段注释

段注释是在一行或多行中用 /\*与 \*/ 括住某一段表示注释某段

- javadoc注释

javadoc注释是以 /\*\* 开始, 以 \*/结束

```
// This program prints Welcome to Java!  
  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



# 一些特殊符号

```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```



# 程序编写风格

- 适当的注释
- 命名习惯
- 适当的缩进和空白
- 块的风格

There are two types of people:

```
if (Condition) {  
    Statement  
    /* ....  
    */  
}
```

```
if (Condition)  
{  
    Statement  
    /* ....  
    */  
}
```



# 程序错误

- 语法错误

由编译器检测

- 运行错误

导致程序不能正常运行

- 逻辑错误

产生不正确的结果



# 调试 (debug)

- 程序的逻辑错误被称为bug，查找和更正错误的过程被称为debug
- 调试的一般方法是使用各种方法来逐步缩小bug所在范围

你可以手工跟踪程序（即通过读程序找错误）

也可以插入打印语句以显示变量的值或程序的执行流程

对于大型的、复杂的程序，最有效的调试方法是使用调试工具





# Java程序语言基础（二）

## 基本数据类型、运算符、流程控制

# 数据类型

Name	Range	Storage Size
byte	$-2^7$ (-128) to $2^7-1$ (127)	8-bit signed
short	$-2^{15}$ (-32768) to $2^{15}-1$ (32767)	16-bit signed
int	$-2^{31}$ (-2147483648) to $2^{31}-1$ (2147483647)	32-bit signed
long	$-2^{63}$ to $2^{63}-1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754



# 数值运算符

Name	Meaning	Example	Result
+	Addition	$34 + 1$	35
-	Subtraction	$34.0 - 0.1$	33.9
*	Multiplication	$300 * 30$	9000
/	Division	$1.0 / 2.0$	0.5
%	Remainder	$20 \% 3$	2



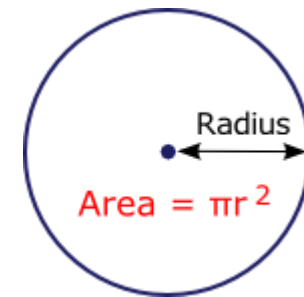
# 一个简单的程序示例

- 计算一个半径为特定值的圆形面积

半径 (radius) = 20

圆周率 (PI) = 3.14159

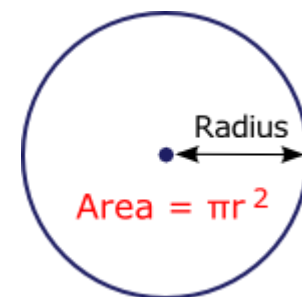
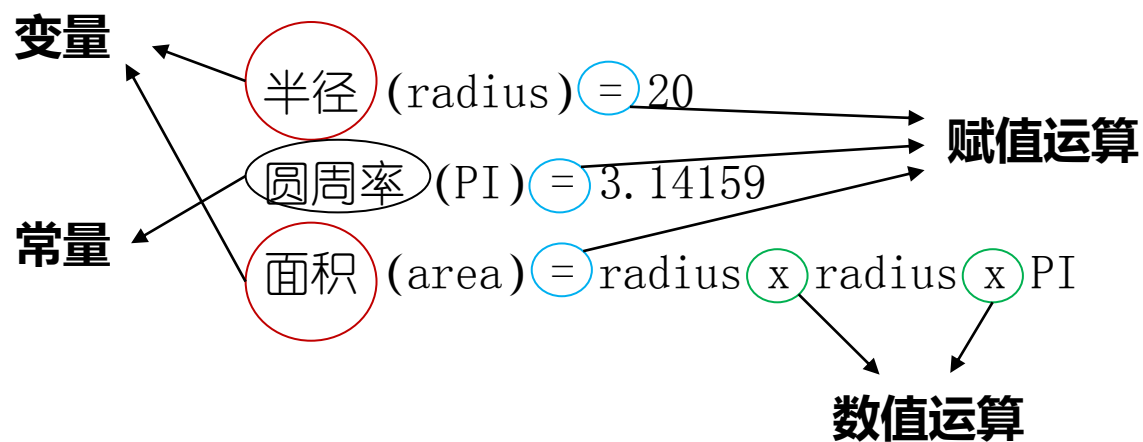
面积 (area) = radius x radius x PI





# 一个简单的程序示例

- 计算一个半径为特定值的圆形面积



示例1







# 变量、常量的声明与赋值

## ● 变量

先声明，后赋值

变量数据类型 变量名;                      比如, double radius;

变量名 = 值;                                      比如, radius = 20;

一步完成声明、赋值

变量数据类型 变量名 = 值;    比如, double radius = 20;

## ● 常量

final 常量数据类型 常量名 = 值; 比如, final double PI = 3.14159;



# 从控制台读取输入

- 创建一个Scanner类型的对象

`Scanner input = new Scanner(System.in);`

- 选用合适的方法，将输入值转换为相应的数据类型

`nextByte()`, `nextShort()`, `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`

另外，还有

`next()`，返回String(字符串值)

`nextBoolean()`，返回boolean值

## 示例2



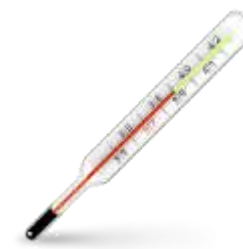


# 示例：温度转换

- 编写一个程序，将输入的摄氏温度转换成华氏温度

算数表达式

$$fahrenheit = \frac{9}{5} * celsius + 32$$



变量

`double celsius`——摄氏温度, 控制台输入

`double fahrenheit`——记录华氏温度

示例3





# 数值类型转换

- 考虑下面的语句：

`int a = 9 / 5;`                      `double a = 9 / 5;`

`int a = 9.0 / 5;`                      `double a = 9.0 / 5;`

`int a = 9.0 / 5.0;`                      `double a = 9.0 / 5.0;`



# 数值类型转换 cont.

## ● 类型拓宽

将小范围类型的变量转换为大范围类型的变量

隐式转换，程序可自动执行

`double d = 3;` （类型拓宽）

## ● 类型缩窄

将大范围类型的变量转换为小范围类型的变量

显示转换

`int i = (int)3.0;` （类型缩窄）

`int i = (int)3.9;` （截掉小数部分）

范围增加



Byte、short、int、long、float、double





# 流程控制——选择

- 在计算圆的面积时，如果半径的输入是一个负值，将出现不合理的计算结果。如何处理这个问题？



# boolean类型和运算符

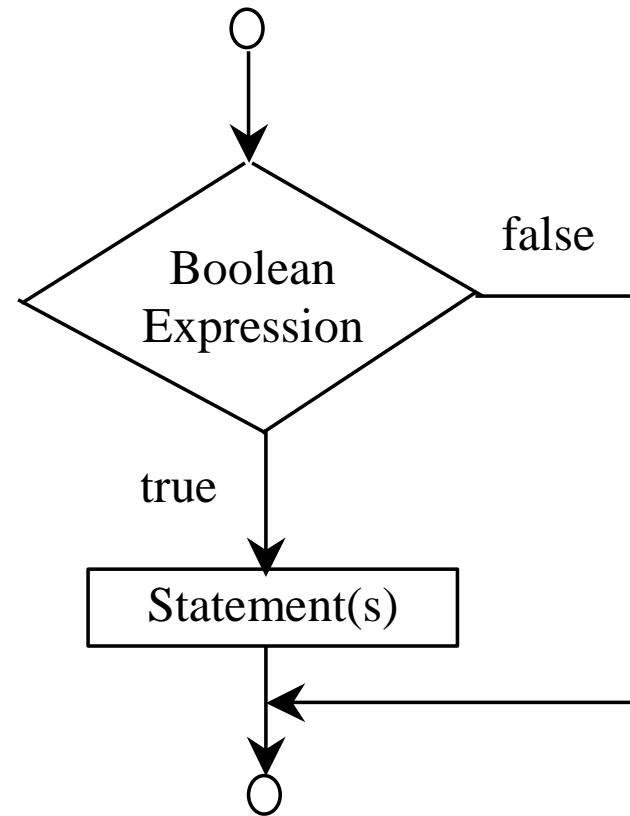
- 程序中经常需要对两个值进行比较  
例如： i 是否大于j
- Java提供了六种比较运算符（也被称作关系运算符），比较的结果是一个 Boolean值，true（真）或false（假）

运算符	名称
<	小于
<=	小于或等于
>	大于
>=	大于或等于
==	等于
!=	不等于



# 单向if语句

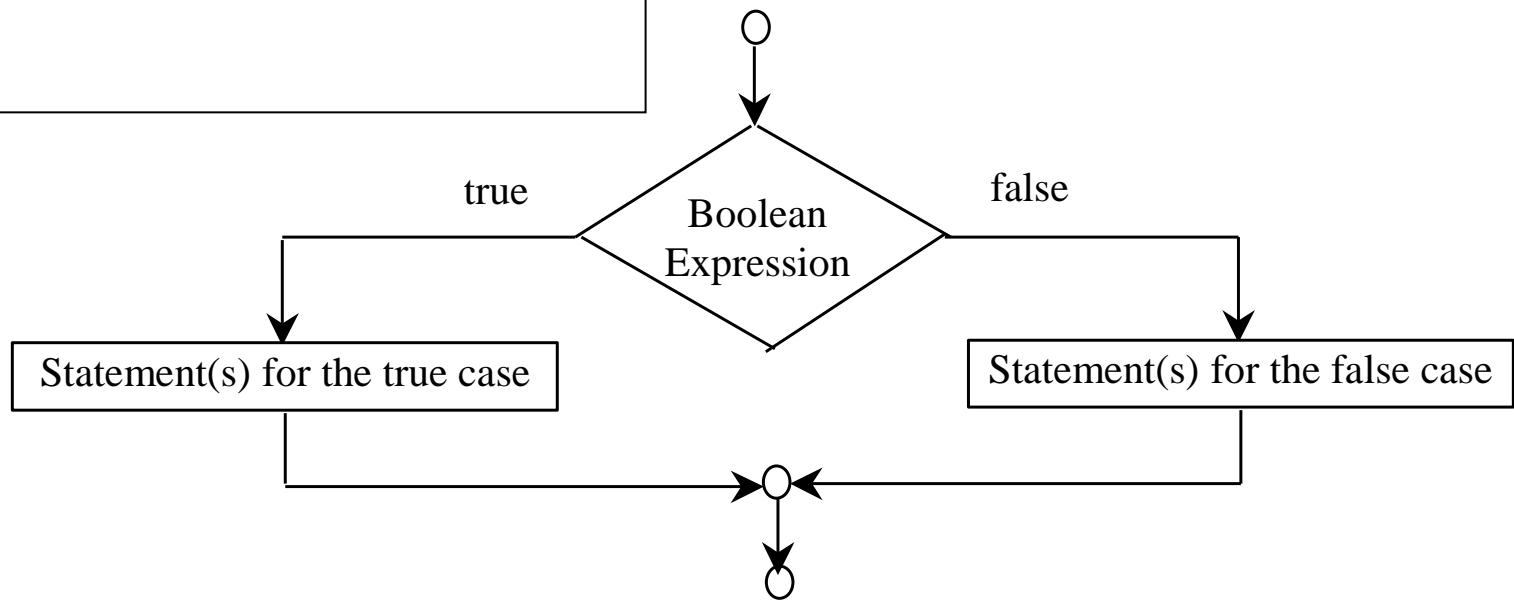
```
if (布尔表达式) {  
    语句;  
    ...  
}
```





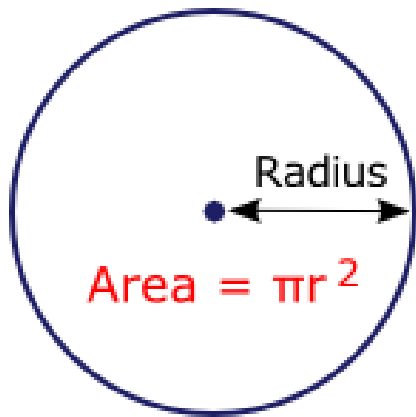
# 双向if语句

```
if (布尔表达式) {  
    布尔表达式为真时执行的语句;  
    ...  
} else {  
    布尔表达式为假时执行的语句;  
    ...  
}
```





# 示例：输入的检查



示例4







# 示例：对账

- 对比交易所反馈的成交额与券商自己统计的成交额

交易所发给券商当天的成交总额

券商自己系统里有每一笔成交的金额

券商需要每天进行汇总、对账，确定当日记录的成交金额是否正确

示例5





# 关于浮点运算的精度问题

- 浮点数据 (double, float) 的二进制表达不精确

要把小数装入计算机，总共分几步？

第一步：转换成二进制

第二步：用二进制科学计算法表示

第三步：表示成 IEEE 754 形式

在上面的第一步和第三步都有可能丢失精度！

比如：**0.1** 到 **0.9** 的 **9** 个小数中，只有 **0.5** 可以用二进制精确的表示

关于浮点数的精度

<http://justjavac.iteye.com/blog/1725977>

<http://justjavac.iteye.com/blog/1724438>



# 逻辑运算符

## ● 如何进行多个条件的判断？

### 运算符

### 名称

!

逻辑非

&

逻辑与

|

逻辑或

^

逻辑异或

&&

短路与（第一个操作数为假则不判断第二个操作数）

||

短路或（第一个操作数为真则不判断第二个操作数）



# 逻辑运算符——真值表

a	b	!a	a&b	a b	a^b	a&&b	a  b
true	true	false	true	true	false	true	true
true	false	false	false	true	true	false	true
false	true	true	false	true	true	false	true
false	false	true	false	false	false	false	false



## 示例：判断闰年

- 根据用户输入年份，判断是否为闰年

某年可以被4整除但不能被100整除，或者能被400整除，即为闰年

$(\text{year \% } 4 == 0 \ \&\& \ \text{year \% } 100 \neq 0) \ || \ (\text{year \% } 400 == 0)$

示例6







# Switch语句

- 用于针对某表达式的不同值，进行条件判断，从多个程序语句中，选择其中一个执行

**switch** (表达式或变量) {

**case** 条件值1:

语句块1;

**break**;

**case** 条件值2:

语句块2;

**break**;

...

**default**:

语句块N;

**break**;

}

## 规则

1. 表达式的值必须是下述几种类型之一:

*int, byte, char, short*

2. *case* 子句中的条件值必须是常量，且所有 *case* 子句中的值应是不同的;

3. *default* 子句是可选项;

4. *break* 语句用来在执行完一个 *case* 分支后使程序跳出 *switch* 语句块；否则会继续执行下去。

示例7





# 流程控制——循环

- 如何打印一个字符串（例如：“Welcome to Java!”）100次？

100次

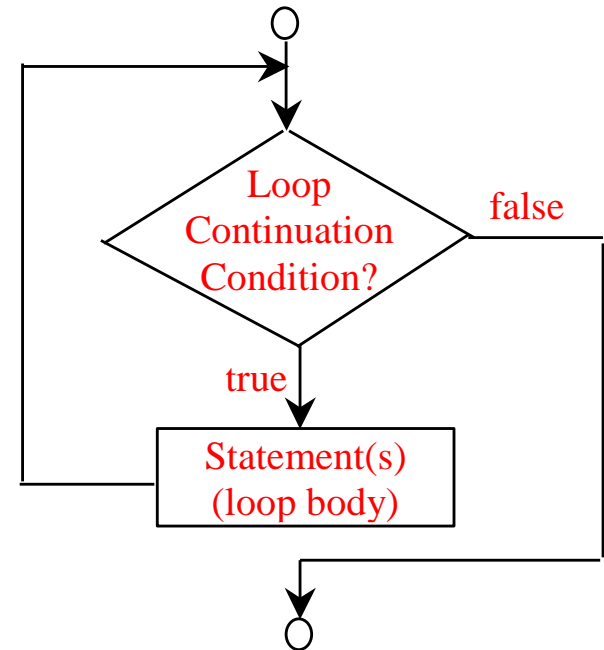
```
System.out.println("Welcome to Java!");  
System.out.println("Welcome to Java!");  
System.out.println("Welcome to Java!");  
System.out.println("Welcome to Java!");  
System.out.println("Welcome to Java!");  
System.out.println("Welcome to Java!");  
...  
...  
...  
System.out.println("Welcome to Java!");  
System.out.println("Welcome to Java!");  
System.out.println("Welcome to Java!");
```



# while循环

布尔表达式

```
while (循环继续条件) {  
    //循环体  
    循环继续条件为真时执行的语句;  
    ...  
}
```

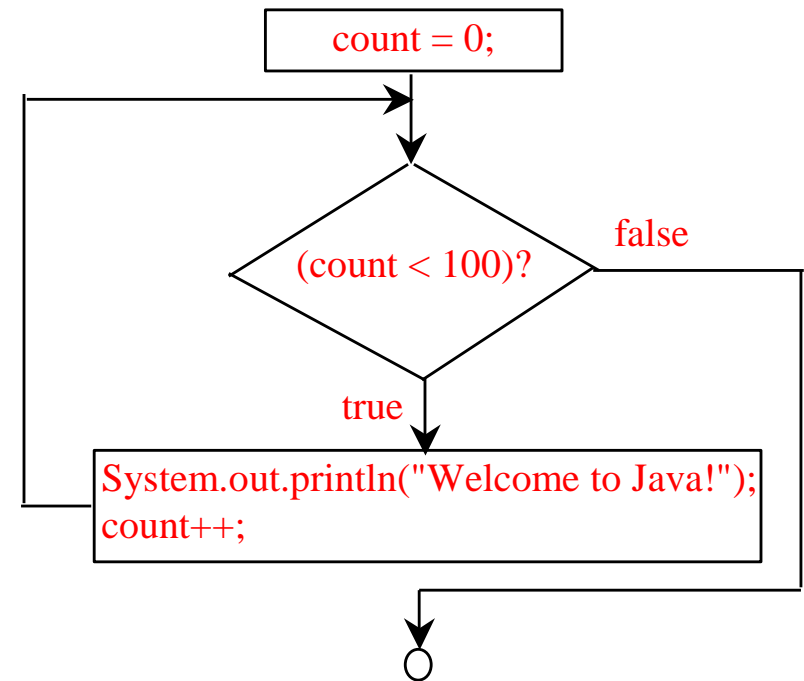




# while循环 cont.

```
int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java!");
    count++;
}
```

示例8.1





# 自增和自减运算符

运算符	名称	说明
<u>++var</u>	前置自增	变量var的值加1，使用var增加后的新值
<u>--var</u>	前置自减	变量var的值减1，使用var减少后的新值
<u>var++</u>	后置自增	变量var的值加1，使用var原来的值
<u>var--</u>	后置自减	变量var的值减1，使用var原来的值

- 若运算符是在变量的前面，则该变量自增1或自减1，然后返回的是变量的新值
- 若运算符是在变量的后面，则该变量自增1或自减1，然后返回的是变量的旧值



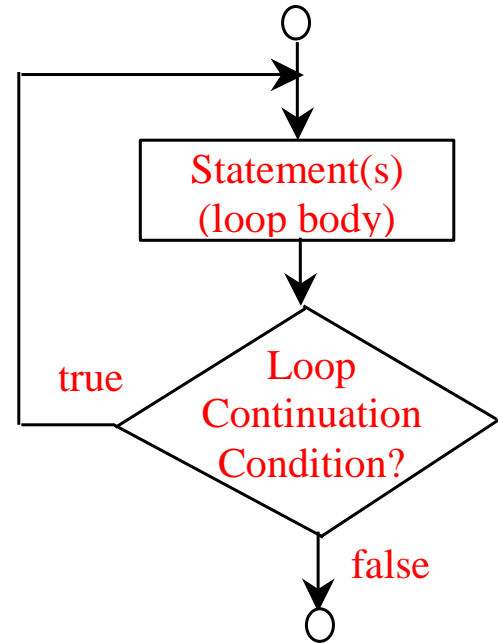


# do-while循环

```
do {  
    //循环体  
    语句;  
    ...  
} while (循环继续条件);
```

```
int count = 0;
```

```
do {  
    System.out.println("Welcome to Java!");  
    count++;  
} while (count < 100);
```



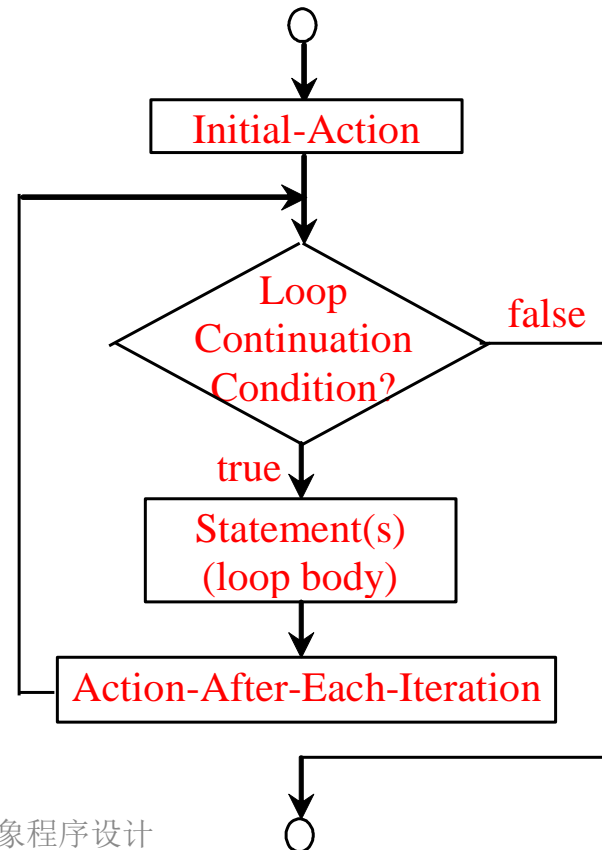
示例8.2





# for循环

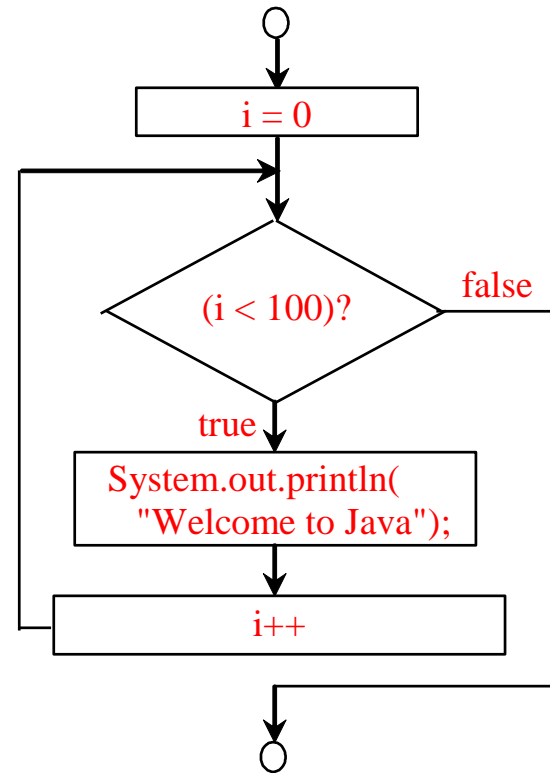
```
for (初始操作; 循环继续条件; 每次迭代后的操作) {  
    //循环体  
    语句;  
    ...  
}
```



# for循环 cont.

```
int i;  
for (i = 0; i < 100; i++) {  
    System.out.println("Welcome to Java!");  
}
```

示例8.3





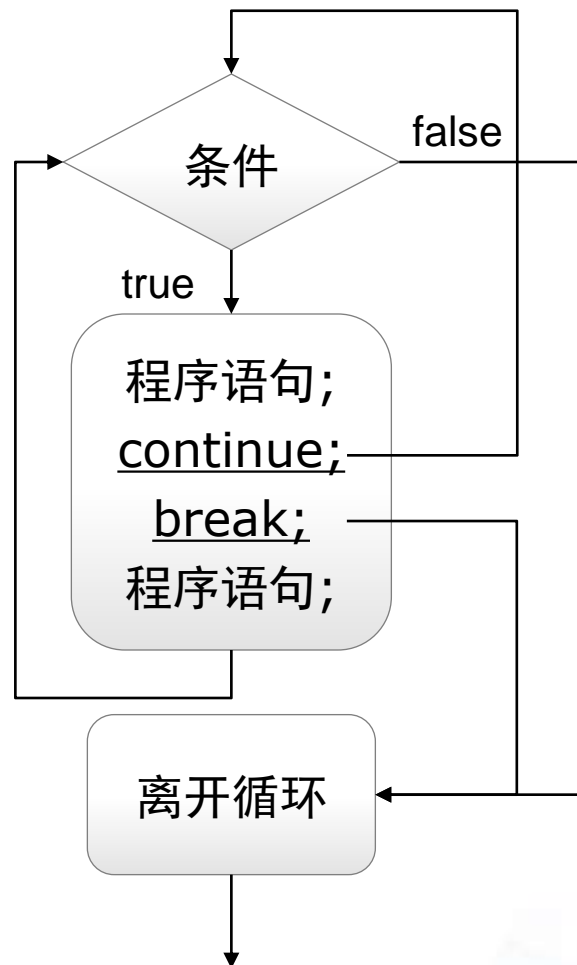
# 跳转语句——break和continue

- continue语句

用于循环结构内，终止当前这一轮循环

- break语句

用于循环结构内，跳出循环





# 使用哪种循环语句？

- 使用自己觉得最自然、最舒适的那种循环语句

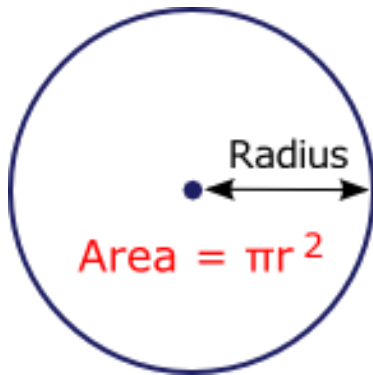
*如果重复的次数是已知的，采用for循环*

*如果重复的次数是未知的，采用while循环*

*如果在检查继续条件前必须执行循环体，采用do-while循环替代while循环*



# 示例：next round? retype?



示例9







## 示例：乘法表（嵌套循环）

Multiplication Table										
	1	2	3	4	5	6	7	8	9	
1	1	2	3	4	5	6	7	8	9	
2	2	4	6	8	10	12	14	16	18	
3	3	6	9	12	15	18	21	24	27	
4	4	8	12	16	20	24	28	32	36	
5	5	10	15	20	25	30	35	40	45	
6	6	12	18	24	30	36	42	48	54	
7	7	14	21	28	35	42	49	56	63	
8	8	16	24	32	40	48	56	64	72	
9	9	18	27	36	45	54	63	72	81	

示例10





# 课后练习

## 震爷手气壮

设计一个简单的微信群“拼手气”红包程序。要求可以根据输入的总金额、红包数量，输出每个红包的随机金额，其中，金额以“元”为单位，保留两位小数点。





# 课后作业

## ● 安装Java环境

# 安装JDK、Eclipse/IntelliJ IDEA

**Eclipse**, [www.eclipse.org/downloads](http://www.eclipse.org/downloads)

**IntelliJ IDEA**, [www.jetbrains.com/idea/download](http://www.jetbrains.com/idea/download)



## ● 注册个人代码托管账户

# gitee或github, 创建repository

# 将repository地址发邮件提交给助教

作业要求: **周四上午9点前提交**



# 总结

- 对象技术导论（一）  
认识软件系统的复杂性
- Java程序语言基础
  - (1) 入门
  - (2) 基本数据类型、运算符、流程控制



中国科学院大学  
University of Chinese Academy of Sciences

# Thank You!

- 面向对象程序设计
- Object-Oriented Programming