Lecture1 信息系统、JVM、编程基础

1. 纲要

- 信息系统中的计算
- 使用Java (JVM)
- 编程的基本知识：内置类型、数组、循环和基本 I/O

2. 信息系统 Information System

> 阅读第二本教材的 Chapter1 部分

计算 Computing 发生在**信息系统 information system** 中

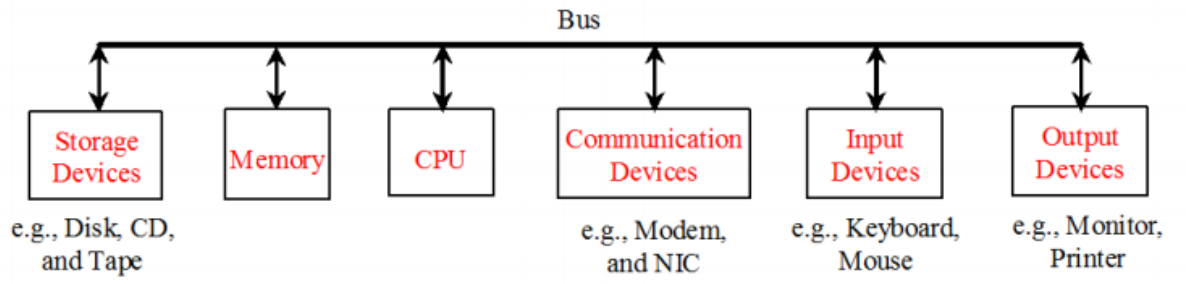
信息系统的部分有

- 用户、业务流程、软件、硬件、数据、网络等...

Data 与 Information 的区别

- Information = bits of context, a presentation, useful, some data in format
- Data = Data Type / ADT(Abstract Data Type)

硬件



计算机由称为硬件的各种设备组成

- 键盘，屏幕，鼠标，磁盘，内存， DVD、CD-ROM 和中央处理器 (CPU)

软件

在计算机上运行的带有**数据 data 和文件 documents** 的**程序 programs** 被称为软件

- 程序的运行需要数据和文件做支撑，否则单独的程序没有用

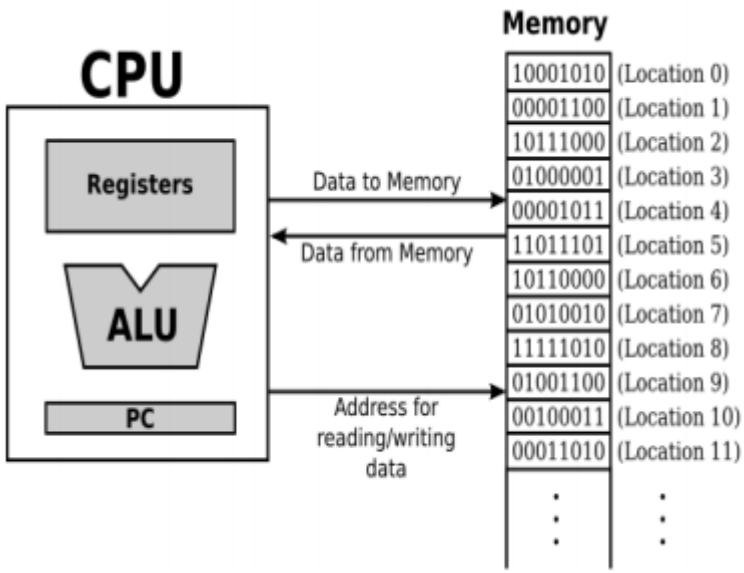
程序 Program

指定如何执行计算的指令序列
A sequence of instructions that specifies how to perform a computation.

指令或者语句会执行

- **输入 input**：获得数据
- **输出 output**：输出数据
- **数学 math**：执行数学操作
- **测试 testing**：检查条件，运行语句
- **重复 repetition**：重复执行某些行为

3. 计算机工作的逻辑



- 主存储器保存机器语言程序和以二进制编码的数据
- CPU 按顺序从内存中取出指令并执行
- 每条指令都是非常小的任务
- 该程序必须是完整的和明确的，因为 CPU 执行的一切都完全按照写的

取指-执行周期 Fetch Execute Cycle

取指-执行周期是计算机操作处理指令的基本周期

在这个周期中，计算机从内存中接收到一个程序指令，然后它建立并执行指定的操作

- **取指 Fetch**：从计算机的内存中获取下一个程序命令
- **解码 Decode**：解码程序告诉计算机做什么
- **执行 Execute**：执行要求的行动
- **存储 Store**：将结果保存到寄存器或内存中

冯诺依曼体系结构

是由著名的 John von Neumann 首先提出的**冯·诺伊曼体系结构**，这是当今大多数计算机所遵循的框架

计算机的存储器 memory 包括

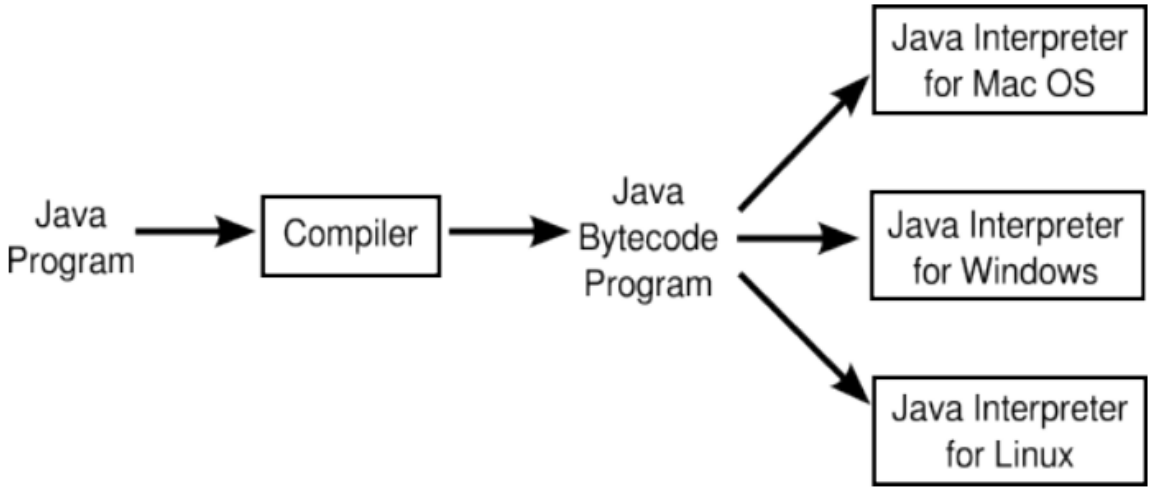
- **可执行指令 executable instructions**：代码

- **静态数据 static data**：全局的一些数据
- **栈 stack**：与函数相关的易失的数据从这里来和走
- **堆 heap**：对象在这里创建

处理器（CPU）还有

- **寄存器 register**：数据和指令从存储器中取出在这里做处理

4. JAVA 虚拟机 JVM



Java 既是可编译的，又是可解释的

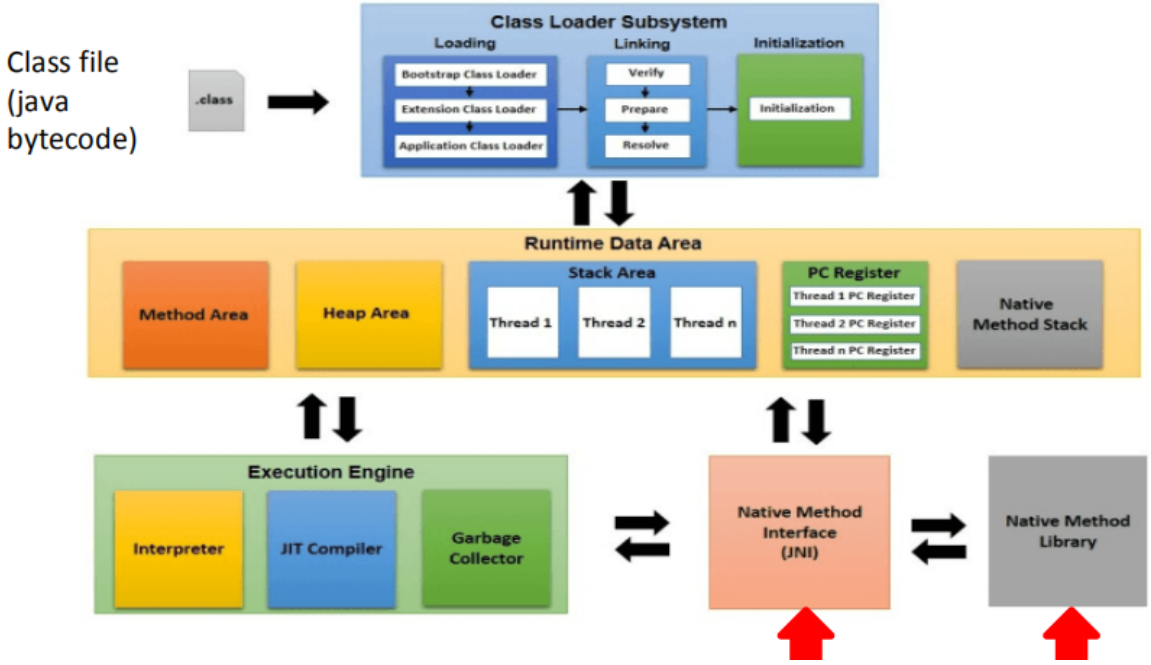
Java 虚拟机在许多物理机器上以标准方式模拟 Von Neuman 机器

JVM 在不同机器上的实现是不同的，但是它们运行的是相同的
Implementations of JVMs on different machines are different, but what they run is the same

> 什么叫做实现是不同的，运行是相同的?

- 从本质上讲，JVM 是一种理想机器类型的虚拟表示
- 虚拟化向用户隐藏了计算平台的物理特征，取而代之的是一个抽象的计算平台

JVM 是如何工作的



JVM 可以分为三个主要组件

类装载器 Class Loader

- 类装载器负责**装入.class 文件**（有时将它们组合到.jar、Java ARchive 中），这些文件包含中间代码，这些**代码不再是 Java 代码，但还不是计算机指令**
- 类装载器的任务之一是**定位代码中引用的每个类**（所有导入的类，以及在程序中引用但未定义的类，并且期望在 CLASSPATH，目录或.jar文件的列表），然后设置所有内容，以便当您调用方法时，引擎知道在哪里找到要执行的指令
- 最后，它将**初始化所有内容**，你将能够调用 main()

运行时数据区域 Runtime Data Areas

- 将在运行时中找到 Data area 所有来自 von Neumann 架构的组件，内置了对线程的支持，线程是独立且并发（同时）执行的任务
- 一台机器可以有成千上万个线程

执行引擎 Execution Engine

- 执行引擎包含一个**解释器 interpreter**，用来转换 在**.class 中找到的“在任何地方运行”的代码**会变成一些处理器可以执行的“在特定硬件上运行”的指令
- 解释代码很慢（特别是在运行循环和重复指令时），很快就在Java 中引入了 “及时编译器” 转换，然后重用转换时，相同的代码再次执行
- 由于JIT 编译器在执行代码时发现代码，所以每个优化都不能从一开始就执行，而且存在一种复杂的机制
- 最后一个组件是**垃圾收集器 garbage collector**，它标识不再使用的对象并回收它们正在使用的内存

其它虚拟机

- 在软件中模拟硬件系统并在其上安装任何操作系统（如 vmware）
- 在机器上模拟另一个不同的操作系统（例如 Docker 和 containers）

5. 编程基础

> 阅读第一本教材的 Chapter1 部分

内置数据类型

给出 b 和 c，找到函数 $x^2 + bx + c$与 x轴的交点
我们知道，二次函数的交点为 $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$

```java
import java.util.Scanner;

public class E1_Quadratic {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        double b = in.nextDouble();
        double c = in.nextDouble();

        double discriminant = b * b - 4.0 * c;
        double sqroot = Math.sqrt(discriminant);

        double root1 = (-b + sqroot) / 2.0;
        double root2 = (-b - sqroot) / 2.0;
        System.out.println(root1);
        System.out.println(root2);
    }
}
```

条件判断与循环

假设一个赌徒从某个给定的初始赌注开始，进行了一系列公平的 1 美元下注。赌徒最终总是会破产，但当我们在游戏中设置其他限制时，各种问题就会出现

例如，假设赌徒提前决定在达到某个目标后离开，赌徒赢的机会有多大？赢或输游戏需要多少赌注？在游戏过程中，赌徒的最大金额是多少？

成功的概率是赌注与目标的比率，期望下注的数目是赌注与期望收益的乘积（目标与赌注的差值）

例如，如果你去蒙特卡洛试图把500美元变成2500美元，你有一个合理的（20%）成功的机会，但你应该期待下一百万美元的赌注！如果你想把 1 美元变成 1000 美元，那么你只有 0.1% 的机会（基本不可能），如果你想把 999 美元变成 1000 美元，那么概率很大

```java
import java.util.Scanner;

public class E2_Gambler {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int stake = in.nextInt();// 初始赌注金额
        int goal = in.nextInt();// 目标挣钱额度
```

```
        int trials = in.nextInt();// 尝试的次数
        int bets = 0;// 赌博的次数
        int wins = 0;// 胜利的次数

        for (int t = 0; t < trials; t++) {
            int cash = stake;
            while (cash > 0 && cash < goal) {
                // 开始一次赌博
                bets++;
                if (Math.random() < 0.5) cash++;
                else cash--;
            }
            if (cash == goal) wins++;
        }
        System.out.println(100 * wins / trials + "% wins");
        System.out.println("Avg # bets: " + bets / trials);
    }
}
```

数组

是一种内存的表达

```
int[] a = new int[7];
```

- 当使用关键字 new 创建数组时，Java 会在内存中保留足够的空间来存储指定数量的元素
- 这个过程称为**内存分配 memory allocation**
- 程序中使用的所有变量都需要同样的处理过程（但是对于基本类型的变量，不需要使用关键字new，因为 Java 知道要分配多少内存）
- 必须访问数组的任何元素之前创建数组，如果不能遵守此规则，则在编译时将得到一个未初始化的变量错误

爱拉托逊斯筛法 Sieve of Eratosthenes

一种制作质数表的算法，按顺序写出从 2 到希望包含在表格中的最大数字 n 的整数

- 划掉所有能被 2 整除的数 > 2（每一秒数），求剩下的最小数 > 2，它是 3
- 划掉所有能被 3 整除的数 > 3（每三个数），求剩下的最小数 > 3，它是 5
- ...

$\pi(n)$是指小于或等于n的质数的数目，例如，$\pi(17)=7$，前7个质数是2、3、5、7、11、13和 17

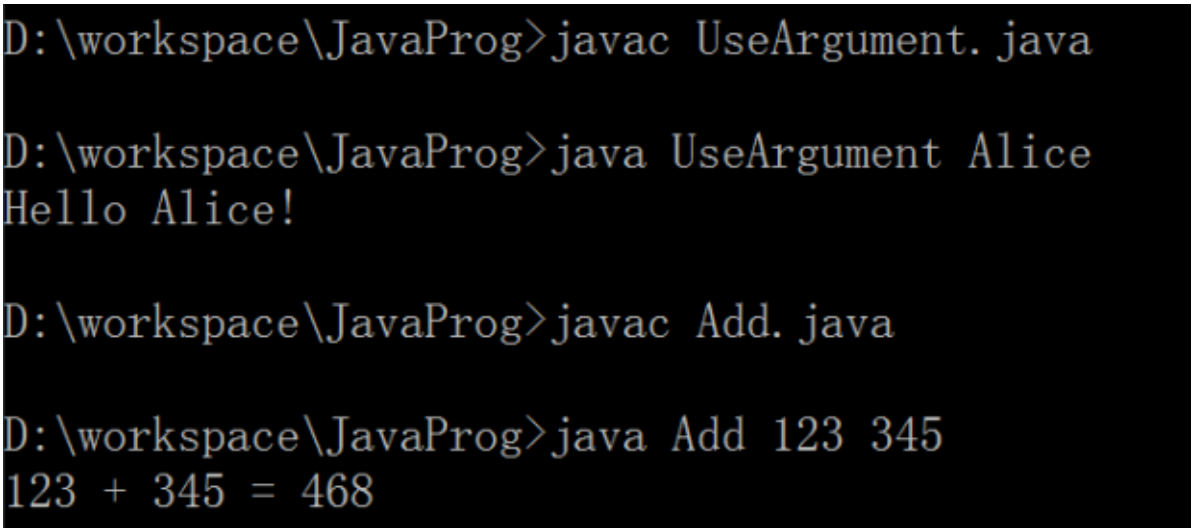
```
import java.util.Scanner;

public class E3_PrimeSieve {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        int n = in.nextInt();
```

```java
        boolean[] isPrime = new boolean[n + 1];
        // 初始情况假设所有整数都是质数
        for(int i = 2; i <= n; i++){
            isPrime[i] = true;
        }

        for(int factor = 2; factor * factor <= n; factor++){
            // 如果一个数是质数，则它的倍数都不是质数
            if(isPrime[factor]){
                for(int j = factor; factor * j <= n; j++){
                    isPrime[factor * j] = false;
                }
            }
        }

        int primes = 0;
        for(int i = 2; i <= n; i++){
            if(isPrime[i])primes++;
        }

        System.out.println("The number of primes <= " + n + " is " + primes);
    }
}
```

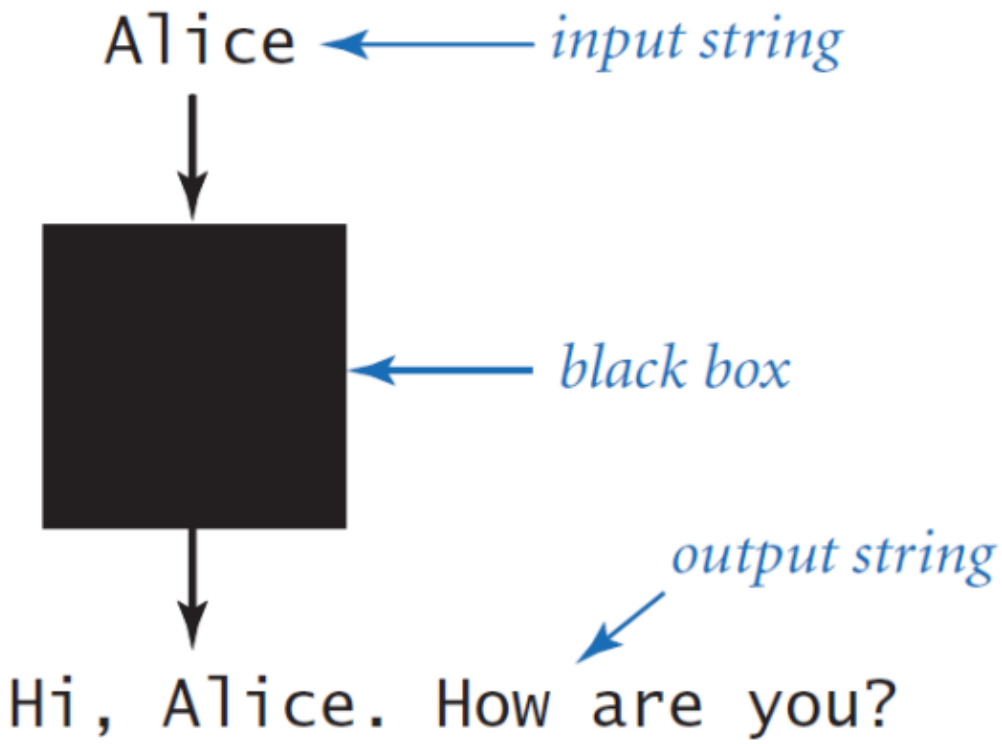
I / O - 命令行参数

使用下面的 cmd 命令来执行 Java 文件

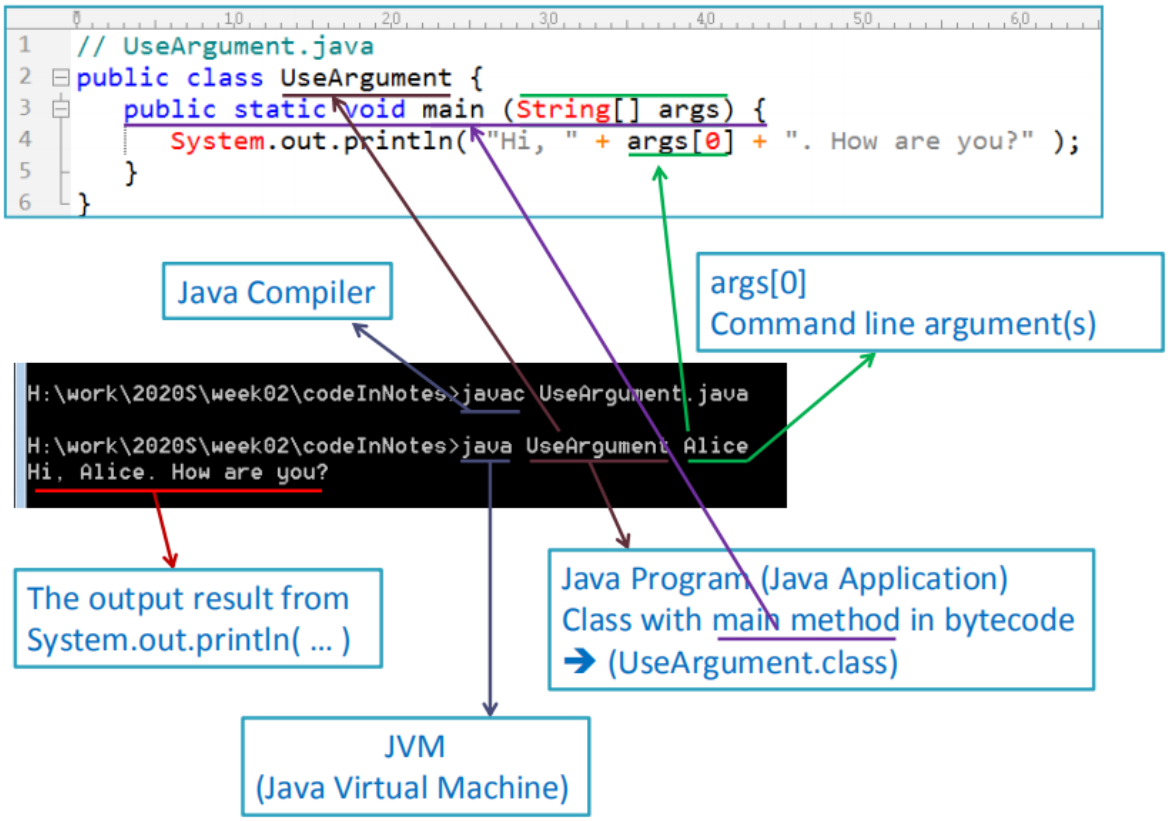
```
// 编译
javac [类名].java
// 运行
java [类名] [参数1] [参数2] ...
```

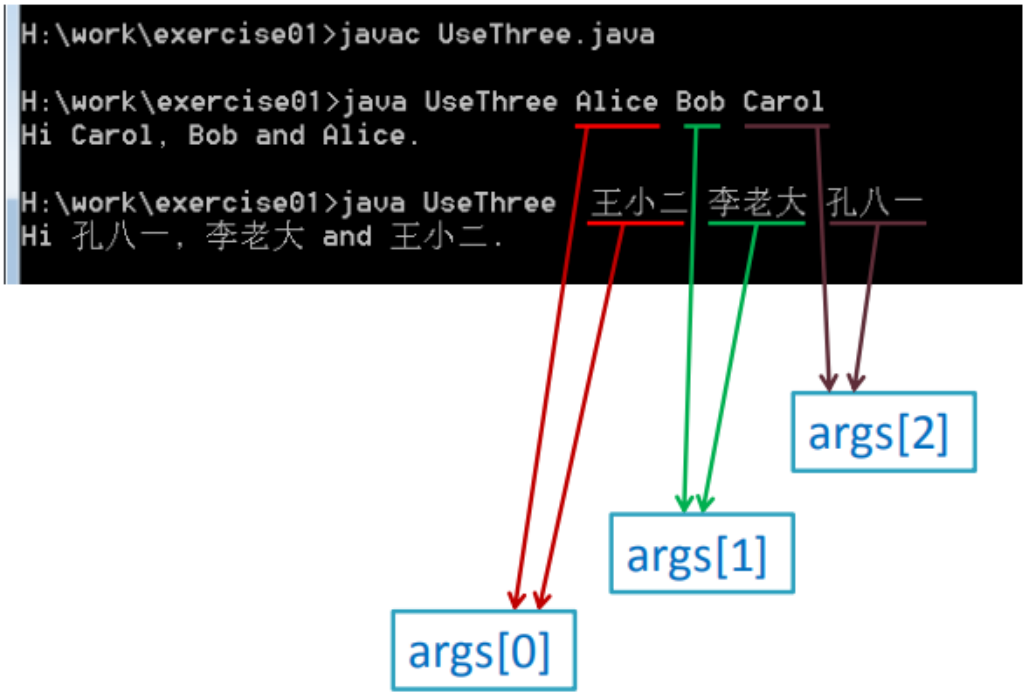


效果像一个黑盒子


Alice
input string
black box
output string
Hi, Alice. How are you?

剖析命令行程序


// UseArgument.java
public class UseArgument {
public static void main (String[] args) {
System.out.println("Hi, " + args[0] + ". How are you?");
}
}
Java Compiler
args[0]
Command line argument(s)
H:\work\2020S\week02\codeInNotes>javac UseArgument.java
H:\work\2020S\week02\codeInNotes>java UseArgument Alice
Hi, Alice. How are you?
The output result from
System.out.println(...)
Java Program (Java Application)
Class with main method in bytecode
➔ (UseArgument.class)
JVM
(Java Virtual Machine)


H:\work\exercise01>javac UseThree.java
H:\work\exercise01>java UseThree Alice Bob Carol
Hi Carol, Bob and Alice.
H:\work\exercise01>java UseThree 王小二 李老大 孔八一
Hi 孔八一, 李老大 and 王小二.
args[2]
args[1]
args[0]