第2章 一切都是对象

章节目标

A 了解Java基本组成部分

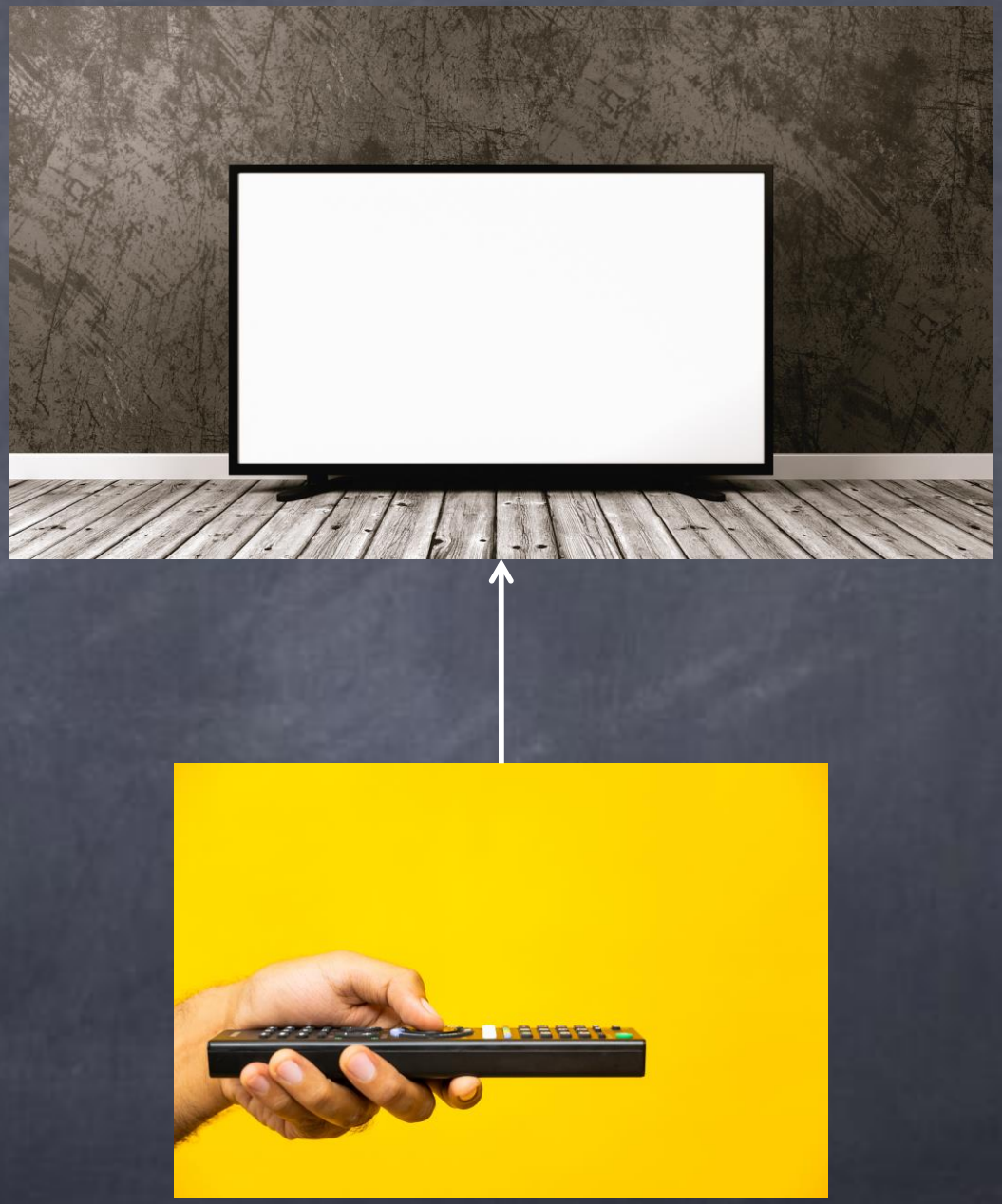
B 体会Java如何实现面向对象

C 搭建Java开发环境

2.1使用引用操纵对象

- 拥有一个引用，而不是对象
- String s;

- 拥有引用和对象
- String s = “asdf” ;



2.2创建对象

- 必须自己创建所有对象
- String s = new String(“abcd”);

- 数据存储到什么地方
- 特例:基本类型

数据存储在哪里

寄存器（Registers）

最快的存储区域，位于 CPU 内部
没有直接控制权

栈内存（Stack）

存在于常规内存 RAM
处理器的直接支持
存放基本类型与引用

堆内存（Heap）

通用的内存池（也在 RAM 区域）
所有 Java 对象都存在于其中
对象生存时长未知
new 关键字
自动垃圾回收

常量存储

直接放在程序代码中
永远不会改变

非 RAM 存储

完全存在于程序之外
序列化对象、持久化对象

特例:基本类型

基本类型	大小	最小值	最大值	包装类型
boolean	—	—	—	Boolean
char	16 bits	Unicode 0	Unicode $2^{16}-1$	Character
byte	8 bits	-128	+127	Byte
short	16 bits	-2^{15}	$+2^{15}-1$	Short
int	32 bits	-2^{31}	$+2^{31}-1$	Integer
long	64 bits	-2^{63}	$+2^{63}-1$	Long
float	32 bits	IEEE754	IEEE754	Float
double	64 bits	IEEE754	IEEE754	Double
void	—	—	—	Void

如何使用

```
char c = 'x';
Character ch = new Character(c);
```

一般使用方法

```
Character ch = new Character('x');
```

步骤合并

```
Character ch = 'x';
```

自动装箱

```
char c = ch;
```

自动拆箱

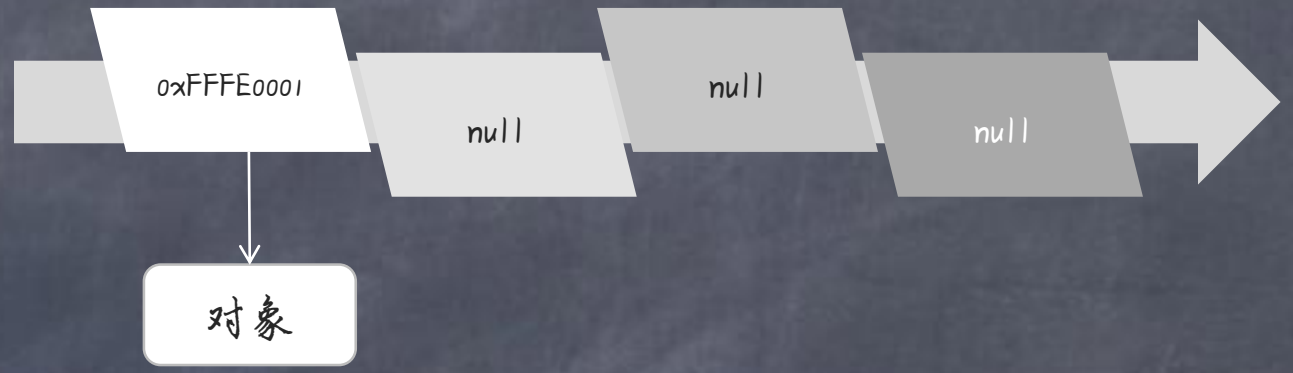
```
BigInteger、BigDecimal
```

高精度数字

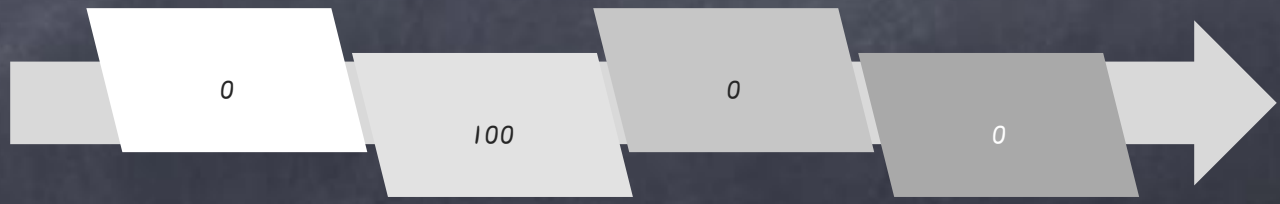
Java中的数组

- C/C++
  - 内存块
  - 容易溢出
- Java
  - 对象
  - 运行时有安全检查
  - 是引用的数组

对象数组



基本类型数组



2.3永远不需要销毁对象

- 作用域
  - 由花括号决定
  - 只可用于作用域结束前
  - 变量不“隐藏”

```
{
    int x = 12;
    {
        int x = 96; // Illegal
    }
}
```

```
{
    int x = 12;
    // 仅 x 变量可用
    {
        int q = 96;
        // x 和 q 变量皆可用
    }
    // 仅 x 变量可用
    // 变量 q 不在作用域内
}
```

- 对象的作用域
  - 引用s活在作用域中
  - 只要需要,对象可以一直活着
  - 自动垃圾回收

```
{
    String s = new String("a string");
}
// 作用域终点
```

2.4创建新的类型

- class ATypeName {
- // 这里是类的内部
- }

- ATypeName a = new ATypeName();

2.4.1字段和方法

- DataOnly类型
- 对象出生后,字段有初始值
- 通过小数点访问字段

```
class DataOnly {
    int i;
    double d;
    boolean b;
}

DataOnly data = new DataOnly();

data.i = 47;
data.d = 1.1;
data.b = false;
```

字段默认值

- 基本类型作为类字段时，有默认值
- 基本类型做临时变量是，没有默认值
- 对象引用作为类字段时，其默认值为null

基本类型	初始值
boolean	false
char	\u0000 (null)
byte	(byte) 0
short	(short) 0
int	0
long	0L
float	0.0f
double	0.0d

2.5方法、参数和返回值

- 方法:名称、参数列表、返回值、方法体
- 方法签名:方法名+参数列表
- 区别方法签名的标准:
  - 方法名、参数的数量、类型、顺序
- 调用方法:用小数点调用

```
[返回类型] [方法名](/*参数列表*/){
    // 方法体
}

[对象引用].[方法名](参数1, 参数2, 参数3);

int x = a.f();//发送消息给对象
```

2.5.1参数列表

- 参数传递的是对象引用或数值
- 需要按照方法签名的要求来调用
- 可以返回,也可以不返回

```
int storage(String s) {
    return s.length() * 2;
}
int length = obj.storage( "abc" );
```

```
boolean flag() {
    return true;
}
double naturalLogBase() {
    return 2.718;
}
void nothing() {
     return;
}
void nothing2() {
}
```

2.6构建一个Java程序

- 2.6.1名字可见性
  - 命名空间(包)
  - 采用域名翻转的方式
  - 全部小写,用小数点隔开,有对应文件夹
  - 文件在包中
  - package cn.edu.nankai.cc
- 2.6.2使用其他类
  - 导入类所在的包
  - import java.util.ArrayList;
  - import java.util.*;

2.6.3 static关键字

- 共享字段
  - 只想为特定字段(也称为属性、域)分配一个共享存储空间,而不去考虑究竟要创建多少对象,甚至根本就不创建对象。

- 共享方法
  - 创建一个与此类的任何对象无关的方法。也就是说,即使没有创建对象,也能调用该方法。

共享字段

```
class StaticTest {
    static int i = 47;
}
StaticTest st1 = new StaticTest();
StaticTest st2 = new StaticTest();
st1.i++;//48
st2.i++;//49
StaticTest.i++;//50
```

共享方法

```
class Incrementable {
    static void increment() {
      StaticTest.i++;
    }
}
Incrementable sf = new Incrementable();
sf.increment();//31
Incrementable.increment();//32
```

2.7解读Java程序

```
// objects/HelloDate.java
import java.util.*;

public class HelloDate {
    public static void main(String[] args) {
        System.out.print("Hello, it's: ");
        System.out.println(new Date());
    }
}
```

- java.lang包是默认导入的
- 其它包需要明确导入
- System类
- out静态字段,PrintStream类型
- println,out对象的动态方法
- Date对象自动转换成字符串
- Date对象会被自动垃圾回收

System的其它用法

```
// objects/ShowProperties.java
public class ShowProperties {
    public static void main(String[] args) {
        System.getProperties().list(System.out);
        System.out.println(System.getProperty("user.name"));
        System.out.println(System.getProperty("java.library.path"));
    }
}
```

编译和运行

- 安装JDK
- PATH C:\program files\java\jdk1.7.0\bin
- JAVA_HOME C:\program files\java\jdk1.7.0

- javac HelloDate.java
- java HelloDate

2.8 注释和嵌入式文档

- 单行注释
- //这是一个单行注释
- 多行注释
- /* 这是多行注释 */
- 注释文档
- /** 这是注释文档 */

- javadoc HelloDate.java

2.8.2注释文档语法

```
//:object/Documentation1.java
/** A class comment */
public class Documentation1 {
    /** A field comment */
    public int i;
    /** A method comment */
    public void f(){}
}
```

- 注释文档应用于类、字段和方法
- 一般用在public和protected字段或方法前
- 用于private字段或方法，会被自动忽略，必须用-private标记才能看到

常用标签示例

```
/** 一个普通类
  @author flow
  @version 1.0
*/
public class Documentation1 {
    /** 一个啥也不干的方法
        @param age 岁数,也叫年龄,要求在1~100之间,不是也没关系
        @return 啥也不返回
    */
    public void f(int age){}
}
```

2.9编码风格

```
class AllTheColorsOfTheRainbow {
  int anIntegerRepresentingColors;
  void changeTheHueOfTheColor(int newHue) {......}
    // ...
}
```

- 驼峰命名法
- 类的首字母大写
- 如果类名由多个单词构成，则每个单词的首字母都应大写
- 不使用下划线分隔
- 方法，字段(成员变量)和对象引用名都采用驼峰命名的方式，但是首字母不大写

习题

完成所有课后练习

练习1：(2) 创建一个类，它包含一个**int**域和一个**char**域，它们都没有被初始化，将它们的值打印出来，以验证Java执行了默认初始化。

练习2：(1) 参照本章的**HelloDate.java**这个例子，创建一个“Hello，World”程序，该程序只要输出这句话即可。你所编写的类里只需一个方法（即“main”方法，在程序启动时被执行）。记住要把它设为**static**形式，并指定参数列表—即使根本不会用到这个列表。用**javac**进行编译，再用**java**运行它。如果你使用的是不同于JDK的开发环境，请了解如何在你的环境中进行编译和运行。

练习3：(1) 找出含有**ATypeName**的代码段，将其改写成完整的程序，然后编译、运行。

练习4：(1) 将**DataOnly**代码段改写成一个程序，然后编译、运行。

练习5：(1) 修改前一个练习，将**DataOnly**中的数据在**main()**方法中赋值并打印出来。

练习6：(2) 编写一个程序，让它含有本章所定义的**storage()**方法的代码段，并调用之。

练习7：(1) 将**Incrementable**的代码段改写成一个完整的可运行程序。

练习8：(3) 编写一个程序，展示无论你创建了某个特定类的多少个对象，这个类中的某个特定的static域只有一个实例。

练习9：(2) 编写一个程序，展示自动包装功能对所有的基本类型和包装器类型都起作用。

练习10：(2) 编写一个程序，打印出从命令行获得的三个参数。为此，需要确定命令行数组中String的下标。

练习11：(1) 将**AllTheColorsOfTheRainbow**这个示例改写成一个程序，然后编译、运行。

练习12：(2) 找出**HelloDate.java**的第二版本，也就是那个简单注释文档的示例。对该文件执行**javadoc**，然后通过Web浏览器观看运行结果。

练习13：(1) 通过**Javadoc**运行**Documentation1.java**，**Documentation2.java**和**Documentation3.java**，然后通过Web浏览器验证所产生的文档。

练习14：(1) 在前一个练习的文档中加入各项的HTML列表。

练习15：(1) 使用练习2的程序，加入注释文档。用javadoc提取此注释文档，并产生一个HTML文件，然后通过Web浏览器查看结果。

练习16：(1) 找到第5章中的**Overloading.java**示例，并为它加入javadoc文档。然后用**javadoc**提取此注释文档，并产生一个HTML文件，最后，通过Web浏览器查看结果。