

# OOP with Java

Yuanbin Wu  
cs@ecnu

# OOP with Java

- 通知：
  - Project 2 提交时间：10 月 8 日晚 9 点

# • 复习：Java 类型

## – 基本类型

- boolean, char, 封装 (wrappers)

## – 类 (class)

### 定义

```
class MyType {  
    int i;  
    double d;           数据 (Fields)  
    char c;  
    void set(double x);  
    double get();  
}                       方法 (Methods)
```

### 创建对象

```
MyType a = new MyType();
```

### 使用对象

```
int b = a.i;  
a.set();  
a.get();
```

- More on String, Integer class

## – 数组

### 初始化

```
int [ ]a = {1, 2, 3,4, 5};  
MyType [ ]a = new MyType[3];  
MyType [ ]a = new MyType[ ] {new MyType(), new MyType(), new MyType()};
```

### 数组作为对象

```
int i = a.length; int t = a[3];
```

- 复习

- 引用 (Reference)

- 对象的名字
    - 受限指针

创建一个引用

引用间赋值

创建对象  
返回对象的引用

```
int [ ] a = new int[ ]{1,2,3};  
int [ ] b;  
b = a;  
b[0] = 4;  
System.out.println(a[0]);
```

- 不可变类型 (Immutable type)

- 一旦创建就不能改变

```
String s = "Hello World";  
System.out.println(s.toUpperCase());  
System.out.println(s);
```

# OOP with Java

- Java 操作符
- Java 控制结构
- 静态方法
- 库与模块化编程

# OOP with Java

- Java 操作符
- Java 控制结构
- 静态方法
- 库与模块化编程

# Java 操作符

- Java 操作符
  - 赋值操作
  - 算术操作
  - 自增自减操作
  - 关系操作
  - 逻辑操作
  - **if-else** 操作
  - 位操作
  - **String** 连接操作
  - 强制转换操作
  - **sizeof**
  - 优先级

# Java 操作符

- 操作符 (operator)
  - a + b
  - a != b
  - a && b
- 表达式 (expression)
  - 常量, 变量, 函数, 操作符按照 "语法" 组成的 "语句"
- 表达式的值 (value of expression)
  - 编程语言计算表达式后返回的值
  - 表达式 → 函数
    - 操作符: 函数名
    - 操作数: 参数
    - 表达式的值: 返回值



# Java 操作符

- 赋值操作
  - `a = b;`
  - `a = b = c;`
- 表达式的值
  - 赋值号 (=) 左边表达式的值

# Java 操作符

- 算术操作
  - “+” 加法 (addition):  $a+1$
  - “-” 减法 (subtraction):  $a-1$
  - “\*” 乘法 (multiplication):  $a*2$
  - “/” 除法 (division):
    - $7/8$
  - “%” 取模 (modulus)
    - $5.0\%2.6$
- 与赋值结合
  - $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $\%=$

# Java 操作符

- 正负操作符 “+”, “-”
  - $x = -a; x = a * -b$
  - “+”: 将 byte, short, char 转换为 int
- 自增自减操作
  - $a++; a--;$
  - $++a; --a;$

# Java 操作符

- 关系操作
  - “==”, “!=”, “>”, “<”, “>=”, “<=”
  - 表达式的值为 `boolean`
    - `1==1 : true`
    - `2 > 3: false`

# Java 操作符

- 判断是否相等：`==`
  - Let's try
- `==`
  - 对基本类型：比较它们的值
  - 对类的对象：比较引用

# Java 操作符

- `.equals()` 方法
  - 默认：比较引用（与直接使用 `==` 相同）
  - 可根据需求重写 `equals()`
  - 例子：`String`, `MyType`

```
String s = new String("hello");
String t = new String("hello");
String r = new String("hallo");
boolean b1 = (s == t);
boolean b2 = (s == r);
```

```
String s1 = "hello";
String s2 = "hello";
boolean b3 = (s1==s2);
```

```
public class MyType
{
    ...
    boolean equals(Object o){
        MyType m = (MyType)o;
        return m.i==i;
    }
    ...
}
```

```
MyType m = new MyType();
MyType n = new MyType();
boolean b = m.equals(n);
```

# Java 操作符

- `.equals()` 方法
- 哪里来的？
  - 继承
  - 所有类都默认是 `Object` 类的子类
  - `Object` 类包含 `.equals(Object o)` 方法

# Java 操作符

- 逻辑操作
  - “&&” 与运算 (and)
  - “||” 或运算 (or)
  - “!” 非运算 (not)
- 表达式的值为 `boolean`
  - `if (a == 1 || b = 1) { ... }`
- 短路 (short-circuiting)
  - `if(1 != 1 && a++ == 2)`
  - `if(1 != 1 || a++ == 2)`



# Java 操作符

- **if-else** 三目操作
  - `a == b ? 1 : 0`

# Java 操作符

- 位操作
  - “&” 位与操作 (bitwise and)
  - “|” 位或操作 (bitwise or)
  - “~” 位否操作 (bitwise not)
  - “^” 位异或操作 (bitwise exclusive or), XOR
- 与赋值结合
  - &=, |=, ^=

# Java 操作符

- 移位操作 (shift)
  - 带符号移位
    - `>>`, `<<`
  - 不带符号移位
    - `>>>`
  - `char`, `byte`, `short` 先转为 `int`
- 与赋值结合
  - `>>=`, `<<=`, `>>>=`

# Java 操作符

- String 连接操作

```
String s = "Hello";  
String r = "World";  
String t = s + r;
```

```
String u = s;  
s += t;  
System.out.println(t);  
System.out.println(s);  
System.out.println(u);
```

```
int x = 1;  
int y = 2;  
System.out.println("a" + x + y);  
System.out.println('a' + x);
```

- ToString()

- 基本类型
- 类

- Object 类方法
- 默认输出类名称, Hash code

# Java 操作符

- 强制转换操作
  - 基本类型
    - `int a = (int)1.0f;`
    - 自动转换：当转换是安全的（例如 `int` 转为 `double`）
    - 显式转换：当转换将损失精度（例如 `double` 转 `int`）
    - `boolean` 类型 不能强制转换
  - 类
    - 一般不允许强制转换 ...
    - 特殊情况：继承中父类与子类之间

# Java 操作符

- 副作用 (side effects)
  - 操作符的运算是否对外界有影响
- 有副作用
  - `a = b`, `a++`, `a--`
- 无副作用
  - `a+b`, `a == b`, `a >= b`
- 判断是否有副作用：
  - 将表达式替换为表达式的值是否影响程序的执行

# Java 操作符

- sizeof
  - Java 没有 sizeof 操作符
    - 不需要知道类型的大小

# Java 操作符

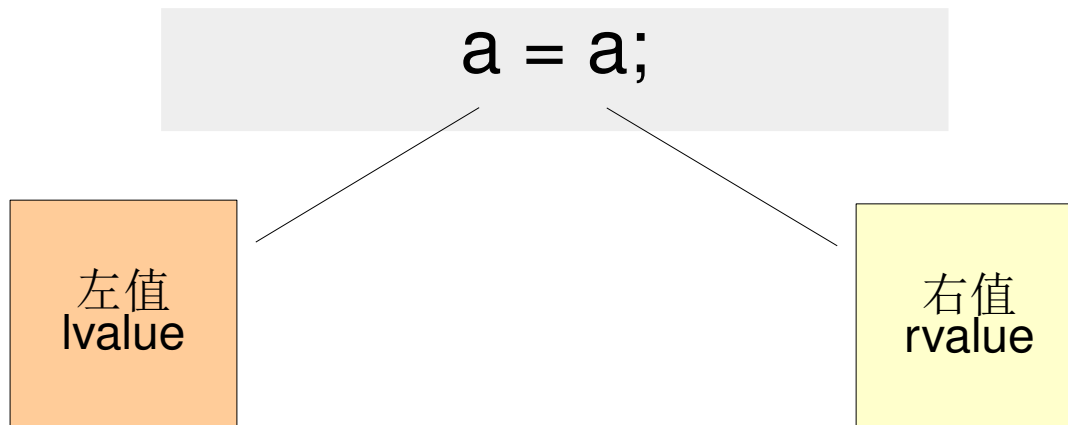
关于赋值操作：

```
int a = 1;  
int b = 2;  
  
a = b;
```

```
a = a;
```

左值  
lvalue

右值  
rvalue





- 左值 (lvalue)
  - 具有存储地址的表达式
- 右值 (rvalue)
  - 没有存储地址的表达式
  - 不能出现在赋值操作符左边
- 可修改左值 (modifiable lvalue)
  - 并非所有左值都能出现在赋值操作符左边
  - 不可修改左值: `const`, 数组, 包含 `const` 成员的 `struct/union`
- 赋值操作
  - 将 “=” 右边表达式的值放入 “=” 左边表达式的地址中

C 语言!

```
int a = 1;    // a is a lvalue, 1 is a rvalue
a = 2;       // OK, since a has an address
1 = a;       // ERROR, 1 doesn't have an address

int const b = 2;    // b is a non-modifiable lvalue
int c[3] = {1, 2, 3}; // Array is non-modifiable lvalue

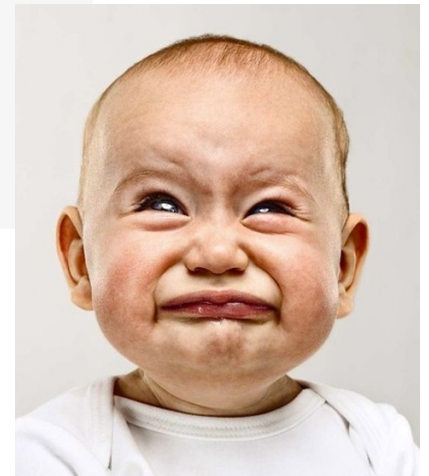
b = 3;         // ERROR, non-modifiable lvalue
int d[3] = {4, 5, 6};
c = d;         // ERROR, non-modifiable lvalue
```

- 左值 与 右值的转换
  - 算术 / 关系 / 逻辑操作：取操作数的右值，返回一个右值
  - 取地址操作 “&”：取左值表达式的地址，返回一个右值
  - 取值操作 “\*”：
    - 仅对指针
    - 取指针的值，返回一个左值

C 语言！

```
int a = 1;      // a is a lvalue, 1 is a rvalue
int b = a + 1; // a+1 is a rvalue
(a+1) = 2;     // ERROR, (a+1) is a rvalue
```

```
int *p = &a;
*p = 3;
```



# OOP with Java

- Java 操作符
- Java 控制结构
- 静态方法
- 库与模块化编程

# Java 控制结构

- Java 控制结构
  - 条件
  - 循环
  - 跳转

# Java 控制结构

- 条件
  - If else

```
if (boolean expression) {  
    statements;  
}  
else {  
    statements;  
}
```

# Java 控制结构

- 循环
  - while, do-while, for

```
while (boolean expression) {  
    statements;  
}
```

```
do {  
    statements;  
}while (boolean expression);
```

```
for(initialization; boolean expression; step){  
    statements;  
}
```

# Java 控制结构

- 循环
  - foreach

```
int []a = {1, 2, 3, 4, 5};  
for (int i : a)  
    System.out.println(i);
```

# Java 控制结构

- 跳转
  - return, break, continue
  - switch
    - break
    - default



# OOP with Java

- Java 操作符
- Java 控制结构
- 静态方法
- 库与模块化编程

# 静态方法

- 操作符
- 控制语句

函数？

# 静态方法

- MyType

- 定义类型
  - 数据
  - 方法
- 创建对象
  - new
- 调用对象的方法
  - m.set(), m.get()

```
public class MyType {
    int i;
    double d;
    char c;
    void set(double x) {
        d = x;
    }
    double get() {
        return d;
    }
    public static void main(String [ ]args) {
        MyType m = new MyType();
        MyType n = new MyType();
        m.set(1);
        n.set(2);
    }
}
```

**问题：**

函数 get(), set() 与 main() 的区别？

# 静态方法

```
public class MyType {  
    int i;  
    double d;  
    char c;  
    void set(double x) {  
        d = x;  
    }  
    double get() {  
        return d;  
    }  
    public static void main(String [ ]args) {  
        System.out.println("Hello");  
    }  
}
```

# 静态方法

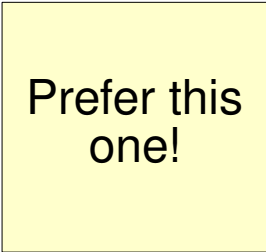
- 静态方法 (**static methods**)
  - 不用创建对象既可被调用的方法
  - 在定义时加 **static** 关键字
  - 也称为：类方法 (**class methods**)

# 静态方法

- 例子：

```
public class StaticTest {  
    static void display() {  
        System.out.println("Hello");  
    }  
  
    public static void main(String [ ]args) {  
        display();  
        StaticTest.display();  
    }  
}
```

Prefer this  
one!



# 静态方法

```
public class StaticTest {  
    double d;  
    static void display() {  
        System.out.println("Hello");  
    }  
  
    public static void main(String [ ]args) {  
        display();  
        StaticTest.display();  
        StaticTest s = new StaticTest();  
        s.display();  
    }  
}
```

**问题：替换为以下语句会发生什么？**  
System.out.println(d);

# 静态方法

- 静态方法不依赖于类的实例化 ( 创建对象 )
- 不能使用需要实例化后才分配空间的变量 / 函数



# 静态方法

- 静态数据 (static data)
  - 类似于静态方法，不依赖于类的实例化
  - 也称为类数据 (class data)

```
public class StaticTest {  
    static int i = 1;  
    static void display() {  
        System.out.println("Hello");  
    }  
  
    public static void main(String [ ]args) {  
        display();  
        StaticTest.display();  
        int a = StaticTest.i;  
    }  
}
```

# 静态方法

- 静态数据类型

```
public class StaticTest {  
    double d;  
    static int i = 1;  
    static void display() {  
        System.out.println("Hello");  
    }  
  
    public static void main(String [ ]args) {  
        display();  
        StaticTest.display();  
        StaticTest s = new StaticTest();  
        System.out.println(s.i);  
        System.out.println(StaticTest.i)  
    }  
}
```

# 静态方法

- 静态数据
  - 在类的不同对象中共享
  - Let's try

```
public class StaticTest {  
    double d;  
    static int i = 1;  
  
    public static void main(String [ ]args) {  
        StaticTest s = new StaticTest();  
        StaticTest t = new StaticTest();  
        t.d = 0.1;  
        System.out.println("object data:" + t.d + " " + s.d);  
        StaticTest.i = 5;  
        System.out.println("class data:" + StaticTest.i + " " + s.i + " " + t.i);  
    }  
}
```

# 静态方法

- 例子：
  - Math.sqrt()
  - Integer.parseInt()
  - Integer.MAX\_VALUE
  - main()
    - MyType.java
    - public static void main()

```
public class MyType {
    int i;
    double d;
    char c;
    void set(double x) {
        d = x;
    }
    double get() {
        return d;
    }
    public static void main(String [ ]args) {
        MyType m = new MyType();
        MyType n = new MyType();
        m.set(1);
        n.set(2);
    }
}
```

# 静态方法

- 参数传递：传值 (pass by value)
  - 基本类型
  - 类 / 数组
    - 传入“引用的值”

```
public class ArgumentTest {
    static void arrayAddOne(int b[ ]) {
        for (int i = 0; i < b.length; ++i)
            b[i]++;
    }
    static void intAddOne(int y) {
        y++;
    }
    public static void main(String [ ]args) {
        int x = 0;
        ArgumentTest.intAddOne(x);
        int [ ]a = {1, 2, 3, 4, 5};
        ArgumentTest.arrayAddOne(a);
    }
}
```

# OOP with Java

- Java 操作符
- Java 控制结构
- 静态方法
- 库与模块化编程

# 库与模块化编程

- 我们已经可以做些什么？
  - 运算符，表达式
  - 控制结构
  - 函数（静态）
  - 定义类型，使用对象
- 将程序放入多个 `.java` 文件

# 库与模块化编程

- 使用其他 .java 文件中的程序

```
public class MyType {  
    int i;  
    double d;  
    char c;  
    void set(double x) {  
        d = x;  
    }  
    double get() {  
        return d;  
    }  
    public static void main(String [ ]args) {  
        MyType m = new MyType();  
        MyType n = new MyType();  
        m.set(1);  
        n.set(2);  
    }  
}
```

MyType.java

```
public class MyTest {  
  
    public static void main(String [ ]args) {  
        MyType k = new MyType();  
        k.set(3);  
        System.out.println(k.get());  
    }  
}
```

MyTest.java

- 编译
  - 将 MyType.java MyTest.java 放在同一目录下
  - javac MyTest.java

## 问题：

执行 java MyTest 和 java MyType 不同？  
多个 main 函数？



# 库与模块化编程

- **public** 关键字

```
public class MyType {  
    int i;  
    double d;  
    char c;  
    void set(double x) {  
        d = x;  
    }  
    double get() {  
        return d;  
    }  
}
```

```
public class MyType {  
    int i;  
    double d;  
    char c;  
    public void set(double x) {  
        d = x;  
    }  
    public double get() {  
        return d;  
    }  
}
```

**问题：**

1. set, get 加上 public 是否有影响？
2. 去掉第一行中的 public 是否有影响？

# 库与模块化编程

- 库 (Library)
  - 一组功能相关的类，为其他用户提供服务
- 用户程序 (Client)
  - 使用库的程序
- 例子：
  - Integer, Math, MyType
  - stdio.h

# 库与模块化编程

- 为什么使用不同的 `.java` 文件
- 模块化编程
  - 将任务分解成为简单，更容易管理的子任务
- 优点：
  - 简单
  - 易于 `debug`
  - 代码重用
  - 易维护