

# OOP with Java

Yuanbin Wu  
cs@ecnu

# OOP with Java

- 通知
  - Project 2 提交时间 : 3 月 15 日晚 9 点
  - 作业提交格式
  - 学习使用
    - 文本编辑器
    - cmd, PowerShell (Windows), terminal(Linux, Mac)

# • 复习

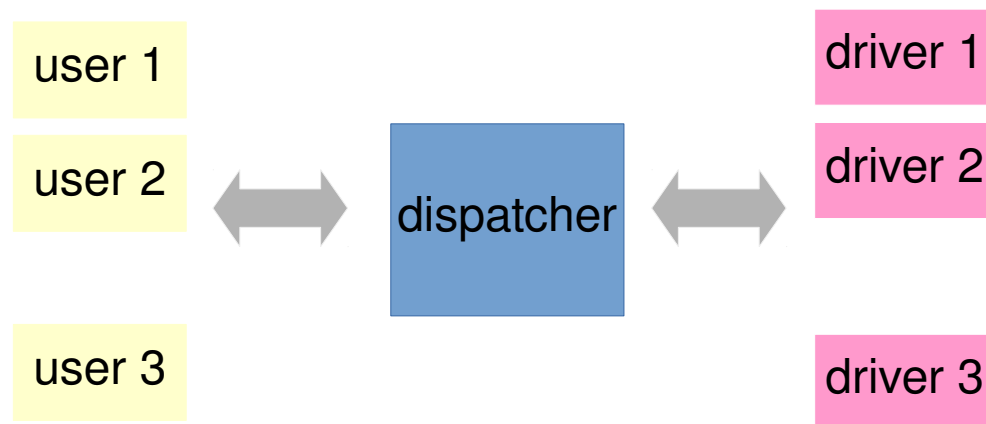
## - 面向对象编程

- 将实际问题分解成不同的对象
- 不同的对象提供不同的服务
- 对象之间可以传递消息

## - 例子

小李深夜 1 点到达虹桥机场，他打电话给出租车公司，想要订一辆车。公司调度员小马通知了司机小刘，小刘告诉小马自己可以接机，半小时后到。小马通知小李，接机司机正在路上。

对象： 用户，调度者，司机



- 复习

## 对象的基本要素：状态，行为，类型

- 状态

- 向量对象：当前的第  $i$  维是什么？
- 账户对象：有多少存款？

- 行为

- 向量对象：`set(int i, double v)` 设置第  $i$  维元素为  $v$
- 账户对象：`deposit(int num)` 存入钱款  $num$

- 类型

- 一组行为相同的对象（仅仅状态不同）
- 本节课的主题

# OOP with Java

- Java 类型
- 引用
- 不可变类型
- 对象存储位置
- 作用域

# OOP with Java

- Java 类型
- 引用
- 不可变类型
- 对象存储位置
- 作用域

# Java 类型

- Java 类型
  - 基本类型 (primitive types)
    - 频繁使用
  - 类 (class)
    - 自定义
  - 数组 (array)

# Java 类型

- 复习：类型 与 对象
  - 属于同一个类型的对象
    - 每个人的手机，手机
    - 房间中不同的灯泡，灯泡
    - `int a = 1;`
  - 类型是对象将提供的服务的描述
- 面向对象编程的基本步骤
  - 定义类型
  - 创建属于该类型的对象（实例化）
  - 使用对象的服务



# Java 类型

- Java 类型
  - 基本数据类型 (primitive types)
  - 类 (class)
  - 数组 (array)

# Java 基本类型

<b>Primitive type</b>	<b>Size</b>	<b>Minimum</b>	<b>Maximum</b>	<b>Wrapper type</b>
<b>boolean</b>	—	—	—	<b>Boolean</b>
<b>char</b>	16 bits	Unicode 0	Unicode $2^{16}-1$	<b>Character</b>
<b>byte</b>	8 bits	-128	+127	<b>Byte</b>
<b>short</b>	16 bits	$-2^{15}$	$+2^{15}-1$	<b>Short</b>
<b>int</b>	32 bits	$-2^{31}$	$+2^{31}-1$	<b>Integer</b>
<b>long</b>	64 bits	$-2^{63}$	$+2^{63}-1$	<b>Long</b>
<b>float</b>	32 bits	IEEE754	IEEE754	<b>Float</b>
<b>double</b>	64 bits	IEEE754	IEEE754	<b>Double</b>
<b>void</b>	—	—	—	<b>Void</b>

1. 跨平台
2. unsigned

# Java 基本类型

- 定义类型
- 创建对象
  - `int a = 1, double d = 1.0;`
- 使用对象
  - `int b = a * 2;`

# Java 基本类型

- **boolean 布尔型**
  - 仅取两个值 : `true`, `false`
- **逻辑表达式**
  - **C**: 非零即为真
    - `if (1)`, `if(2)`, `if(-1)`
  - **Java**
    - ~~`if(1)`, `if(2)`, `if(-1)`~~
    - `if(true)`

# Java 基本类型

- char
  - C: 实际为整数
    - signed char: 有符号, 至少包含 [-127, 127]
    - unsigned char: 无符号, 至少包含 [0, 255]
  - Java: 16bit

# Java 基本类型

- 基本类型的封装 (Wrapper)
  - int: 基本类型
  - Integer: 类
    - 提供更多与整数相关的功能

# Java 基本类型

- More on Java char (16 bit)
  - ASCII
  - Unicode
    - 新的字符编码方式，目前已使用超过 128,000 个（是否能用 16bit 存储？）
  - Utf-8:
    - 一种 Unicode 的变长实现方式，以 8bit 为单位增长，
    - 与 ASCII 兼容
  - Utf-16:
    - 一种 Unicode 的变长实现方式，一个或两个 16bit.
    - 前 16bit 能代表常用的 Unicode (basic multilingual plane)
  - char:
    - 16 位
    - 仅能表示 Unicode 中的 basic multilingual plane
  - char array, String
    - Utf-16
    - 每个 unicode code point 可能有两个 16 bit (称为 unicode code unit).

字符数组 ≠ 字符对象组成的数组

# Java 类

- 数据成员默认初始值

Primitive type	Default
boolean	false
char	'\u0000' (null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d



# Java 类型

- Java 类型
  - 基本类型 (primitive types)
  - 类 (class)
  - 数组 (array)

# Java 类

- 定义 Java 类

```
class MyType {
```

```
    int i;  
    double d;  
    char c;
```

数据 (Fields)

```
    void set(double x);  
    double get();
```

方法 (Methods)

```
}
```

# Java 类

- 创建类的对象

malloc

构造函数

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

- 访问数据成员，使用对象的方法

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

- Let's try

# Java 类

- C 语言
  - struct
  - typedef

```
typedef MyType MyType;
```

```
struct MyType {  
    int i;  
    double d;  
    char c;  
    void set(double x);  
    double get();  
}
```

```
MyType m;  
m.i = 1; m.d = 1.0; m.c = 'a';
```

```
MyType* n = (MyType*) malloc(sizeof(MyType));  
n->i = 1; n->d = 1.0; n->c = 'a';  
free(n);
```

# Java 类

- 创建类的对象

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

- 使用对象的方法

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

- ? 销毁对象
  - Java 自动销毁

# Java 类

- 对比基本类型与类
  - 基本类型 : `int`
  - 类 : `Integer`
- `String`

# Java 类型

- Java 类型
  - 基本类型 (primitive types)
  - 类 (class)
  - 数组 (array)

# 数组

- 顺序存储对象
  - 数组类型

```
int a[ ];  
int [ ]a;
```

```
MyType m[ ];  
MyType [ ]m;
```

不指定长度



# 数组

- 初始化
  - 静态初始化

```
int [ ]a = {1, 2, 3, 4, 5};
```

# 数组

- 初始化
  - 动态初始化 1

```
int [ ]a = new int[5];
```

```
MyType [ ]m = new MyType[3];
```

# 数组

- 初始化
  - 动态初始化 2

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

```
MyType [ ]a = new MyType[ ] {  
    new MyType(),  
    new MyType(),  
    new MyType()  
};
```

# 数组

- 数组元素的默认值
  - 基本类型
    - 默认值
  - 类
    - Null

# 数组

- 二维数组
  - 定义

```
int a[ ][ ];  
int [ ][ ]a;
```

```
MyType m[ ][ ];  
MyType [ ][ ]m;
```

# 数组

- 二维数组
  - 静态初始化

```
int [ ][ ] a= { {1,2,3}, {4, 5, 6} };
```

# 数组

- 二维数组
  - 动态初始化 1

```
int [ ][ ] a= new int[2][3];
```

```
MyType [ ][ ] m= new MyType[2][2];
```

# 数组

- 二维数组
  - 动态初始化 2

```
int [ ][ ]a = new int[ ][ ]{ {1,2,3}, {4,5,6} };  
  
MyType [ ][ ]m = new MyType[ ][ ]{  
    {new MyType(), new MyType()},  
    {new MyType(), new MyType()}  
}
```



# 数组

- 多维数组

```
int [ ][ ][ ] a = new int [2][3][3];  
MyType [ ][ ][ ] m = new int[6][6][6];
```

# 数组

- 不规则数组 (ragged array)

```
int [ ][ ] a= { {1, 2}, {3, 4, 5}, {7, 8, 9, 10}};
```

# 数组

- 数组

- 数组可以视为一个”类”
- 数据成员：`length`
  - 数组的长度 (Let's try)

```
int [ ][ ] a= { {1, 2}, {3, 4, 5}, {7, 8, 9, 10}};  
int i = a.length;
```

- 方法成员：`[i]`
  - `a[i]` 返回数组的第 `i` 个元素

# 总结

- Java 类型
  - 基本类型
    - int, char, float, double, boolean, long, short
  - 类
    - `class MyType {int I; double d; double get(); }`
  - 数组

# OOP with Java

- Java 类型
- 引用
- 不可变类型
- 对象存储位置
- 作用域

# 引用

- 以下程序的运行结果？

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

# 引用

- 引用 (Reference, 类型的引用)
  - 对象的名字
  - 同一个对象可以有不同的名字

创建一个应用

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]);
```

# 引用

- 引用 (Reference)

```
MyType m = new MyType();  
MyType n;  
n = m;  
n.set(1.0);  
System.out.println(m.get());
```



# 引用

- 指针？

Java

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

C

```
int *a = (int *)malloc(sizeof(int) * 3);  
a[0] = 1;a[1] = 2; a[2] = 3;  
int *b = a;  
b[0] = 4;  
printf(“%d\n”, a[0]);
```

# 引用

- Java 标准并没有指定 引用 应该如何实现
- 绝大多数 Java 内部使用指针实现引用

创建一个引用  
指针

引用间赋值  
指针

创建对象  
返回对象的引用  
指针

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

# 引用

- 引用与指针的关系
  - 引用是受限的指针
    - 不允许指针运算 : `int a[] = {1, 2, 3}; a++;`
    - 不能强制转换
- 引用
  - 类的引用，数组的引用
  - 基本类型的引用
    - `int a = 1;`
    - `a` 为普通变量，并非 '指针'

# 引用

- 引用的初始化
  - `MyType m = new MyType();`
  - `MyType m = null;`
- 默认初始化
  - 类定义中的引用被默认初始化为 `null`

```
Class A {  
    MyType m;  
}
```

- 数组中元素的引用被默认初始化为 `null`

```
MyType [ ]m = new MyType[3];  
for (int i= 0; i < 3; ++i)  
    System.out.println(m[i]);
```

# OOP with Java

- Java 类型
- 引用
- 不可变类型
- 对象存储位置
- 作用域

# 不可变类型

- 不可变类型 (immutable)
  - 类型的对象一旦创建就不能被改变
  - 例子 String 类, Integer 类, Float 类 ...

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

- 可变类型 (mutable)
  - 例子 MyType, 数组

```
MyType m = new MyType();  
System.out.println(s.get());  
m.set(1.0);  
System.out.println(s.get());
```

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

# 不可变类型

- 不可变类型的优点
  - 简单
  - 易用
  - 安全

# OOP with Java

- Java 类型
- 引用
- 不可变类型
- 对象存储位置
- 作用域



# 对象存储位置

- 对象在内存中的位置
  - C 语言

```
void func()
{
    int x = 0;
    ...
}
```

栈内存（局部变量）：  
1. 自动分配 / 销毁  
2. 由编译器完成  
3. 又称“自动内存”

```
void func()
{
    int *ptr = (int*)malloc(sizeof(int));
    ...
}
```

堆内存：  
由程序员负责分配 / 销毁

当函数返回时？

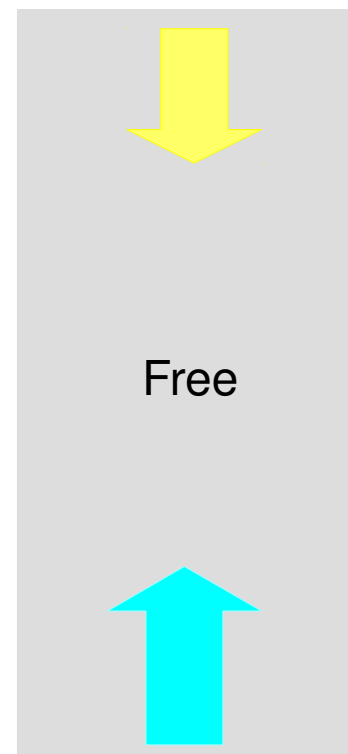
# 对象存储位置

- 对象在内存中的位置

00000000

Code

Heap



Free

Stack

FFFFFFFF

# 对象存储位置

- Java

- 基本类型

- 栈内存

- 类

- 堆内存

- `new`  $\approx$  `malloc`

- 不用显式回收

- Java 虚拟机自动回收无效的内存

- 垃圾回收 (Garbage Collection)

# OOP with Java

- Java 类型
- 引用
- 不可变类型
- 对象存储位置
- 作用域

# 作用域

- 作用域
  - 大括号: {}

```
int x = 12;  
{  
    int q = 96;  
}
```

```
{  
    String s = "1234";  
}
```

# 总结

- Java 类型
  - 基本类型, 类, 数组
- 引用
  - 受限的指针
- 不可变类型
  - 一旦创建对象则不能改变
- 对象存储位置
  - 栈, 堆
- 作用域
  - {}