

复旦大学计算机科学技术学院



编程方法与技术

5.1. 上次课复习

周扬帆

2021-2022第一学期

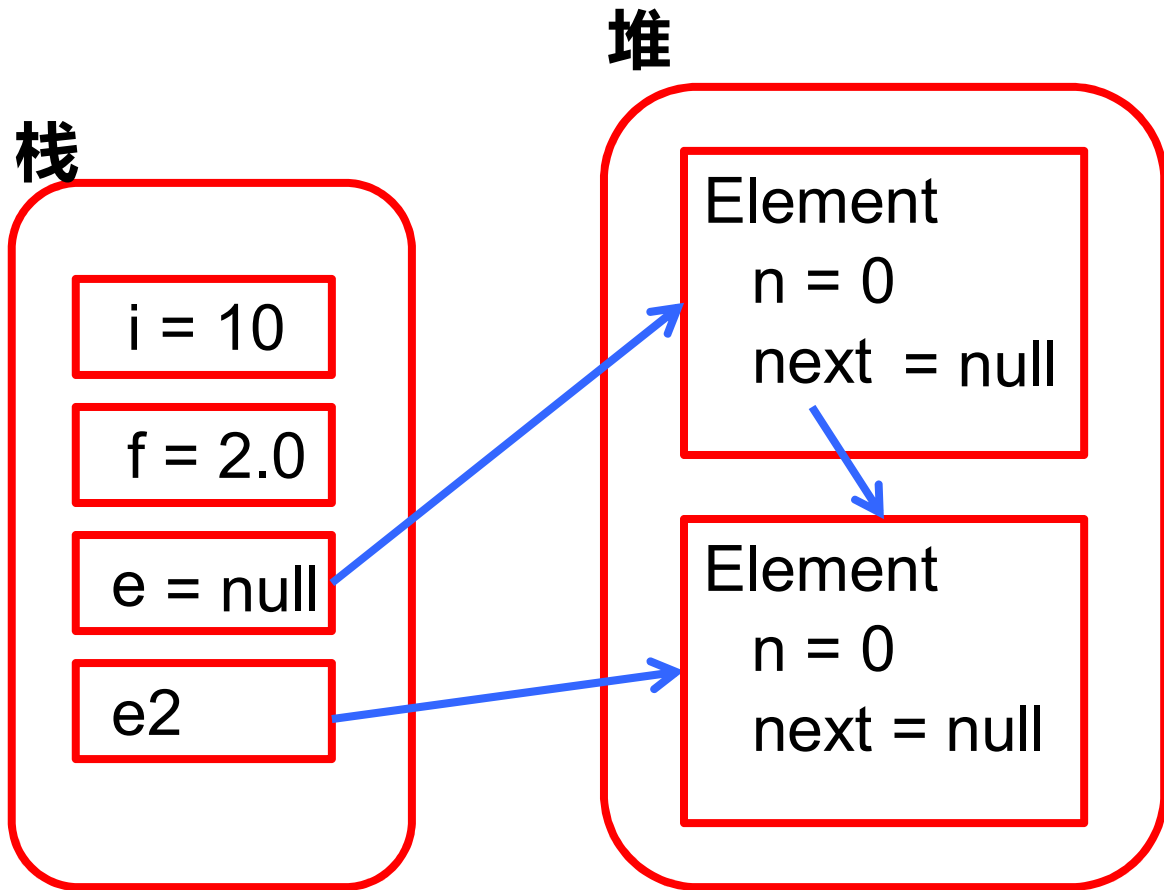
Java的类成员数据

- 类的成员数据有两类：基本数据类型，对象的引用
- 作用域修饰
 - public: 全局可访问
 - private: 仅自己可访问
 - 无: 包(package)内可访问
 - protected: 子类可访问
- **final**: 必须在分配内存空间的时候赋值
- **static**: 全局，所有类的对象共享一份，在对象创建前分配内存空间
- `static final int N = 5;`

Java的内存管理

```
void example() {  
    int i = 10;  
    float f = 2.0;  
    Element e;  
    e = new Element();  
    Element e2 =  
        new Element();  
    e.setNext(e2);  
}
```

```
class Element  
{  
    private int n;  
    private Element next;  
    public void setNext(Element nextElement) {  
        next = nextElement;  
    }  
}
```



Java数据的存储

□ 方法中的局部数据

- 基本数据类型
- 类的引用

□ 类的成员数据

■ static数据

- 所有类的对象共享一份
- 在类加载进JVM时 (类实例化之前)分配空间
 - `Student a = new Student();`

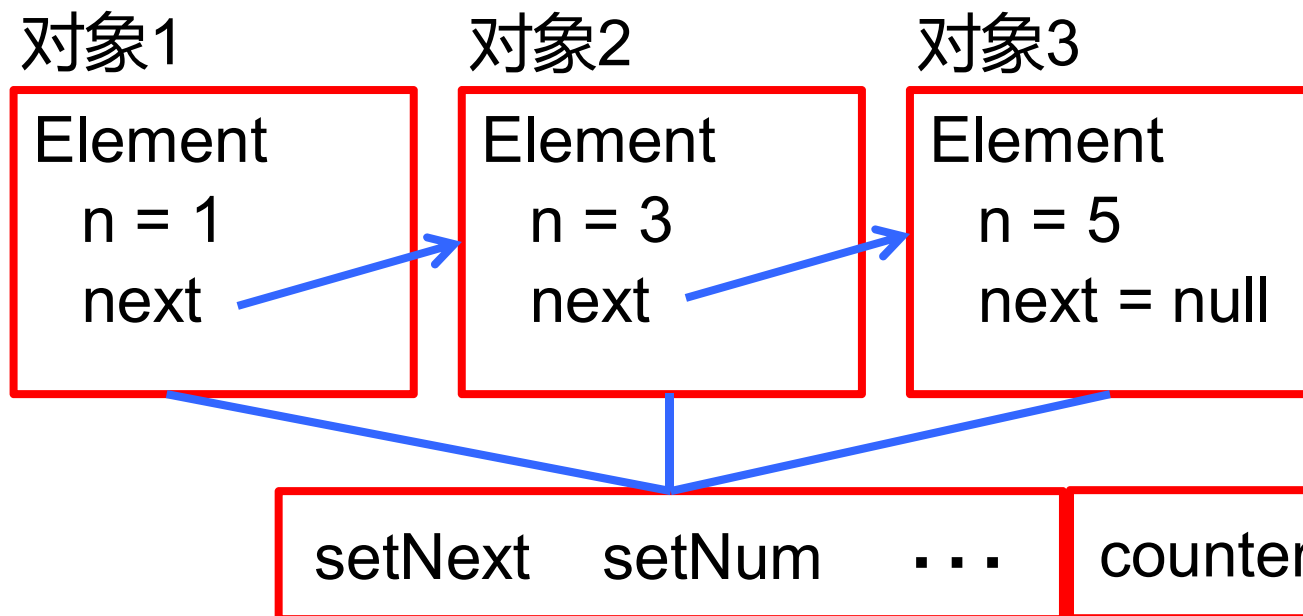
■ 普通数据

static变量

□ 类的static数据

- 节约空间，只存一份
- 访问快速方便，不需创建对象
- 方便对象共享

```
class Element
{
    private static int counter = 0;
    private int n;
    private Element next;
    Element() {
        counter ++;
    }
}
```



字符串类

□ String

- 字串的存储: `private final char [] value`
- Java把常量字符串视为String对象
 - `String a = "Java";`
- Java特别地为String重载了+操作符
 - `String a = a + "Java";`
- 常量池, 堆, `intern()`

□ StringBuffer/StringBuilder

□ StringTokenizer/String.split

- 正则表达式

JavaScript的字符串

□ 引擎(Runtime)决定, 基本一样

□ V8

- 常量字符串一定分配在data space数据区空间

```
var a = "hello";  
// a is Internalized  
  
var b = a;  
// b is Internalized  
same pointer, assignment  
  
var c = a + "world"; // #world is Internalized  
// c is not Internalized  
  
function subject() {  
    return "world"; // #world is Internalized  
};  
var d = a + subject();  
// d is not Internalized  
same pointer, internalized
```

- 字符串被设置为对象属性名时会被尝试改造为常量化版本

→ var obj = {}; obj[c] = 1;

//obj.helloworld === 1

//Object.keys(obj)[0] === "helloworld"

复旦大学计算机科学技术学院



编程方法与技术

5.2. 方法的重载

周扬帆

2021-2022第一学期

String的例子

□ String的多种构造方法

```
byte [] ascii={97,98,99,100,101};
```

```
char [] chars = {'a', 'b', 'c', 'd', 'e'};
```

```
String a    = new String(String b); //拷贝字符串b
```

```
            = new String(ascii);   //ascii码数组
```

```
            = new String(ascii, 1, 2);
```

```
            //从ascii索引为1的元素开始，长度为2
```

```
            = new String(chars); //字符数组
```

```
            = new String(chars, 1, 3);
```

```
            //从chars索引为1的元素开始，长度为3
```

类里方法的重载

□ Java支持同一个类里方法的重载

- 方法名字一样
- signature (参数表) 不一样
- 返回值可以不一样

例: String的取子串方法

String subString(int beginIndex);

参数是一个整数

String subString(int beginIndex, int endIndex);

参数是两个整数

□ 为什么?

为什么需要实现方法的重载

□ 原因1：提高可读性、方便

- 功能类似的操作，用同样的命名简洁、方便理解

■ 例子1

- findStudentBySID(int SID) + findStudentByName(String name)
- findStudent(int SID) + findStudent(String name)

■ 例子2

```
char a[] = {'a', ' ', 's', 't', 'r'};
```

```
int b[] = {1, 2};
```

```
System.out.println("This is " + a);
```

```
System.out.println(a);
```

```
System.out.println(b);
```

This is [C@4554617c

This is a str

This is [I@74a14482

为什么需要实现方法的重载

□ 原因：方便多种构造方法的实现

■ 构造方法必须有同样的命名规则

→ 编译器才能找到构造方法，并在类实例化的时候调用

■ 命名规则？

■ 为了实现多种构造，必须允许构造方法的多种样式

```
char [] chars = {'a', 'b', 'c', 'd', 'e'};
```

```
String a = new String(String b); //拷贝字符串b
```

```
        = new String(chars);    //字符数组
```

类里方法的重载

□ 同一个类里方法的重载

- 代码放置于不同的内存区域
- 把方法名转换成和参数有关的名字
 - 编译时完成
- 运行时便可找到实际对应的方法地址
- 问题：能否仅仅通过返回值不一样，实现重载？

```
int average(int a, int b) {  
    return (a+b) /2;  
}  
  
double average(int a, int b) {  
    return (double)(a+b)/2.0;  
}
```

类里方法的重载: 总结

□ 重载目的

- 简洁命名、方便理解
- 方便一个类实现多种构造方法

□ 重载方法

- 一个类里两个方法的名字可以一样
- 参数表不一样
- 返回值可以一样或者不一样
- 调用时, 根据参数不同, 找对应的方法
- **不能**仅靠返回值不一样 (参数表一样) 重载

思考

- **思考重载的实现原理**
- **思考Java与C++重载的异同**
- **思考重载在实际应用情景中的意义与使用场合**

复旦大学计算机科学技术学院



编程方法与技术

5.3. 类的继承

周扬帆

2021-2022第一学期

StudentRecord例子的回顾

□ 需求

■ 数据记录

→ 名字

→ 学号

→ GPA

`char [] name`

`int studentID;`

`double gpa;`

■ 针对数据的操作

→ 设定名字、学号、GPA

→ 比较GPA

→ 比较学号

→ 输出数据

`setName, setSID, setGPA`

`compareGPA`

`compareSID`

`printRecord`

StudentRecord例子的回顾

```
class Student {
```

```
    char name[] = new char[10];  
    double gpa;  
    int studentID;
```

定义数据

```
    void setName(char [] studentName) {  
        name = new char [10];  
        for(int j = 0; j < name.length && j < studentName.length; j ++) {  
            //只取前10个  
            name [j] = studentName [j];  
        }  
    }
```

定义针对数据的操作方法

```
    boolean compareSID(Student s2) {  
        return studentID > s2.studentID;  
    }  
    boolean compareGPA(Student s2) {  
        return gpa > s2.gpa;  
    }  
    void printRecord() {  
        System.out.println("Student: "+ String.valueOf(name)  
            + ", ID = " studentID + ", GPA = " + gpa);  
    }  
}
```

StudentRecord例子的回顾

构造方法

```
public class StudentRecord {  
    Student students[];  
    StudentRecord() {  
        ...  
        students = new Student[n];  
        for(int i = 0; i < n; i++) {  
            students[i] = new Student ();  
            students[i].setGPA(...);  
            students[i].setName(...);  
            String name = ...  
            char [] charName = name.toCharArray();  
            students[i]. setName(charName);  
        }  
    }  
    void printStudents() { /*print the results*/ ... }  
    public static void main (String[] args) {  
        StudentRecord studentRecord = new StudentRecord();  
        studentRecord.printStudents();  
    }  
}
```

存储Student数据引用的数组

StudentRecord例子的回顾

```
public class StudentRecord {
```

```
    Student students[];
```

存储Student数据引用的数组

```
    StudentRecord() {
```

```
        ...
```

输出结果示例

```
Student: student0, ID = 5, GPA = 3.8441517056853165
Student: student1, ID = 4, GPA = 3.581222057848837
Student: student2, ID = 3, GPA = 2.3226028811143973
Student: student3, ID = 2, GPA = 2.895426842379352
Student: student4, ID = 1, GPA = 3.740106136451817
```

构造方法

```
}
```

```
void printStudents() { /*print the results*/ ... }
```

```
public static void main (String[] args) {
```

```
    StudentRecord studentRecord = new StudentRecord();
```

```
    studentRecord.printStudents();
```

```
}
```

新增数据的处理

□ 新需求：数据记录需要增加一个记录

- 名字、学号、GPA
- + 地址 `String address`

□ 怎么办？

- 直接方法：重新编程
- 新建一个数据结构

```
class Student2 {  
    char name[] = new char[10];  
    double gpa;  
    int studentID;  
    String address;  
}
```

新增数据的处理

- ❑ 新需求：数据记录需要增加一个记录
- ❑ 新建数据结构

```
class Student2 {  
    char name[] = new char[10];  
    double gpa;  
    int studentID;  
    String address;  
}
```

- ❑ 回顾面向对象
 - 方法是针对数据结构的操作
 - 新数据结构 – 需要为之实现对应操作的方法

新增数据的处理

- ❑ 新需求：数据记录需要增加一个记录
- ❑ 新建数据结构

```
class Student2 {  
    char name[] = new char[10];  
    double gpa;  
    int studentID;  
    String address;  
}
```

- ❑ 实现相应的方法
 - setGPA setName setSID **setAddress**
 - compareSID compareGPA
 - printRecord

新增数据的处理

□ 重新实现方法的弊端？

- 很多方法和更新之前的类的方法是一样的
 - setName, setGPA, setSID
 - compareSID, compareGPA
 - printRecord
- 实现：拷贝太麻烦、拷贝容易出错
- 维护：如有更改，需要同时做更改，容易出错
 - Debug
 - 升级

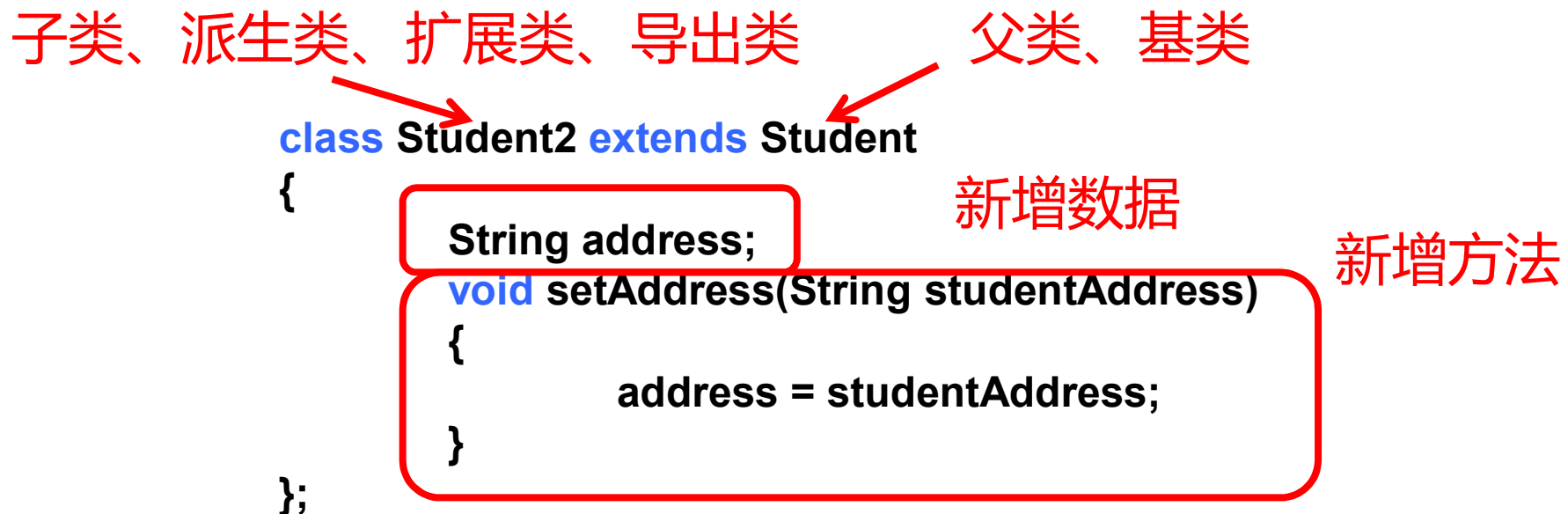
新增数据的处理

□ 解决方法: 数据结构和操作方法的复用

■ 类的继承

□ Java类的复用机制

■ 在类的定义时, 使用extends关键字, 指定复用的类



新类的用法

```
public class StudentRecord {
```

```
    Student2 students[];
```

```
    StudentRecord() {
```

```
        ...
```

```
        students = new Student2[n];
```

```
        for(int i = 0; i < n; i++) {
```

```
            students[i] = new Student2 ();
```

```
            students[i].setGPA(...);
```

```
            students[i].setName(...);
```

```
            String name = ...
```

```
            char [] charName = name.toCharArray();
```

```
            students[i]. setName(charName);
```

```
            students[i].setAddress(...);
```

```
        }
```

```
    }
```

```
    void printStudents() { /*print the results*/ ... }
```

```
    public static void main (String[] args) {
```

```
        StudentRecord studentRecord = new StudentRecord();
```

```
        studentRecord.printStudents();
```

```
    }
```

存储Student2数据引用的数组

方法可复用

调用新实现的

方法

新类的用法

```
public class StudentRecord {
```

```
    Student2 students[];
```

```
    StudentRecord() {
```

```
        ...
```

输出结果示例

```
Student: student0, ID = 5, GPA = 2.90946809697055
Student: student1, ID = 4, GPA = 1.738727380160656
Student: student2, ID = 3, GPA = 2.50848926218552
Student: student3, ID = 2, GPA = 1.5557628596425621
Student: student4, ID = 1, GPA = 2.9767440233151685
```

构造方法

存储Student2数据引用的数组

调用新实现的方法

```
    void printStudents() { /*print the results*/ ... }
```

```
    public static void main (String[] args) {
```

```
        StudentRecord studentRecord = new StudentRecord();
```

```
        studentRecord.printStudents();
```

```
    }
```

方法复用的问题

□ Student2对Student方法的复用

■ 适应新数据结构的问题

→ printStudents方法无法显示新数据address

■ 升级更改的问题

→ setName方法参数是char[]不方便，需要让它接受String参数

□ 解决办法：方法覆盖及方法重载

方法覆盖示例

□ Student2对Student方法的覆盖

```
class Student2 extends Student {  
    void printRecord() {  
        System.out.println("Student: " + String.valueOf(name)  
            + ", ID = " + studentID + ", GPA = " + gpa);  
        System.out.println("---Address: " + adress);  
    }  
    ...  
};
```

□ 覆盖方法使用示例

```
Student2 student = new Student2();  
...  
student.printRecord();
```

调用子类的方法 printRecord()

方法重载示例

□ Student2对Student方法的重载

```
class Student2 extends Student {  
    void printRecord() {  
        System.out.println("Student: " + String.valueOf(name)  
            + ", ID = " + studentID + ", GPA = " + gpa);  
        System.out.println("---Address: " + adress);  
    }  
    void setName(String studentName) {  
        char [] charName = studentName.toCharArray();  
        name = new char[10];  
        for(int j = 0; j < name.length && j < charName.length; j++){  
            //只取前10个  
            name[j] = charName [j];  
        }  
    }  
};
```

方法重载示例

□ Student2对Student方法的重载

```
class Student2 extends Student {  
    void printRecord() { //覆盖  
        ...  
    }  
    void setName(String studentName) { //重载  
        ...  
    }  
};
```

□ 重载方法使用示例

```
Student2 student = new Student2();  
String name = "YZ";  
student.setName(name);      调用子类重载方法 setName(String)  
student.setName(name.toCharArray());  
                             调用父类方法 setName(char [])
```

子类的同名变量

□ Student2对Student方法的复用

- name是char [] 型变量，不方便
- 有更新该变量类型的需求

□ 解决办法：创建同名变量

```
class Student2 extends Student {
```

```
    String name;
```

```
    void printRecord() {
```

```
        ...
```

```
    }
```

```
    void setName(String studentName) {  
        name = studentName;
```

```
    }
```

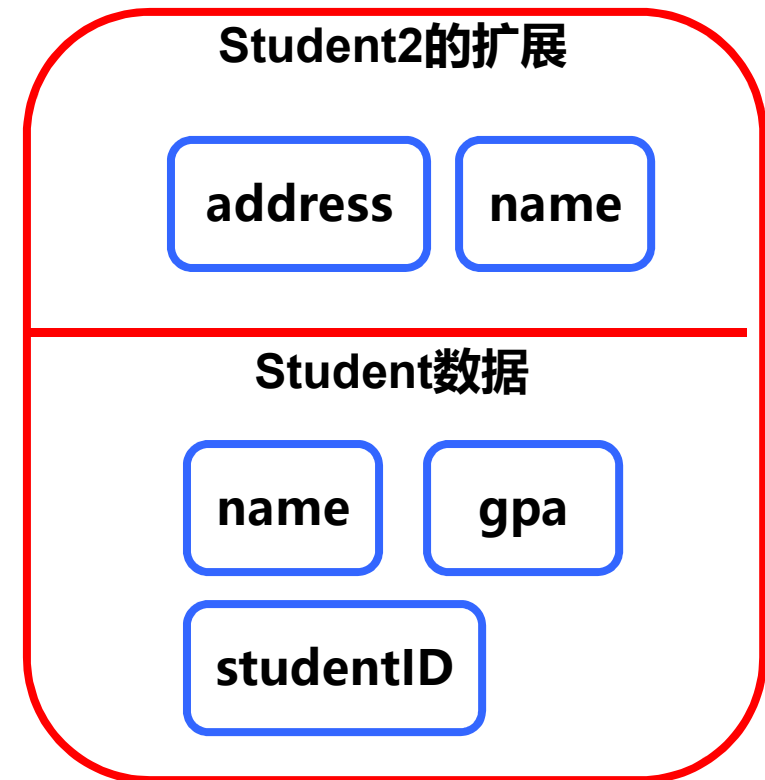
```
};
```


子类的同名变量

- 子类同名变量能否覆盖父类对应的变量
 - 定义了student2.name, 对于student2而言, 其父类的name变量还存在吗?
 - 必须存在
 - 父类有方法, 可能需要使用这个变量
 - setName(char []) 方法
 - 子类也可能使用这个变量
- 因此
 - 子类存在两个同名变量 name

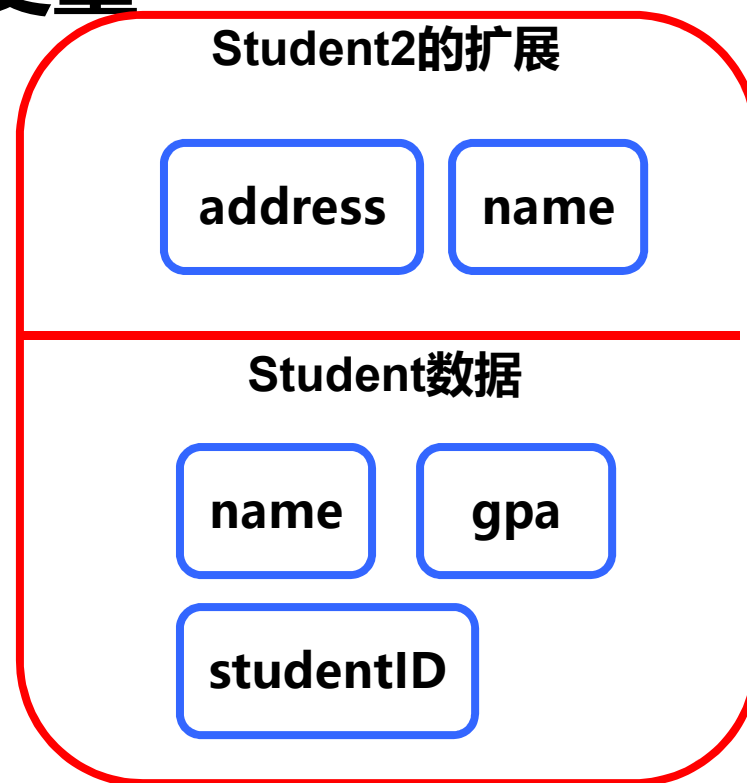
子类的同名变量

- 子类和父类的同名变量共存
- 如何定位：就近原则
 - 类中出现变量a
 - 寻找自己定义的a
 - 找不到 → 父类中寻找
 - 找不到 → 父类的父类寻找
 - ...



子类的同名变量

- ❑ 子类和父类的同名变量如何区分
- ❑ 如果需要用到父类的同名变量
 - 使用`super`关键字
- ❑ `super`关键字
 - 父类实例的引用
 - `super.name`
- ❑ 问题: 子类的静态方法能否访问父类的变量?
 - 怎么访问?



同名变量示例

```
class Student2 extends Student {
```

```
...
```

```
String name;
```

```
void printRecord() {
```

```
...
```

```
}
```

```
void setName(String studentName) {
```

```
name = studentName;
```

```
char [] charName = studentName.toCharArray();
```

```
super.name = new char[10];
```

```
for(int j = 0; j < name.length && j < charName.length; j++){
```

```
//只取前10个
```

```
super.name [j] = charName [j];
```

```
}
```

```
}
```

```
};
```

为什么常
有需要同
时去改变
父类的同
名变量



方法重载中的复用

```
class Student2 extends Student {  
    void printRecord() {
```

```
        System.out.println("Student: " + String.valueOf(name)  
        + ", ID = " + studentID + ", GPA = " + gpa);  
        System.out.println("---Address: " + adress);  
    }
```

```
    void setName(String studentName) {  
        name = studentName;
```

```
        char [] charName = studentName.toCharArray();  
        super.name = new char[10];  
        for(int j = 0; j < name.length && j < charName.length; j++){  
            //只取前10个  
            super.name [j] = charName [j];  
        }  
    }
```

和父类相关方
法的实现一样

```
};
```

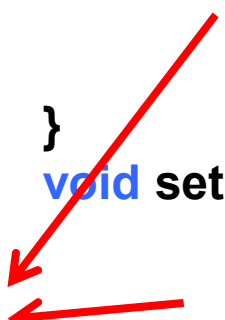
□ 更好的实现: **复用**

■ 如何复用: 借助 **super** 关键字

方法重载中的复用

```
class Student2 extends Student {  
    void printRecord() {  
        super.printRecord();  
        System.out.println("---Address: " + adress);  
    }  
    void setName(String studentName) {  
        name = studentName;  
        super.setName(studentName.toCharArray());  
    }  
};
```

调用相关的父类方法

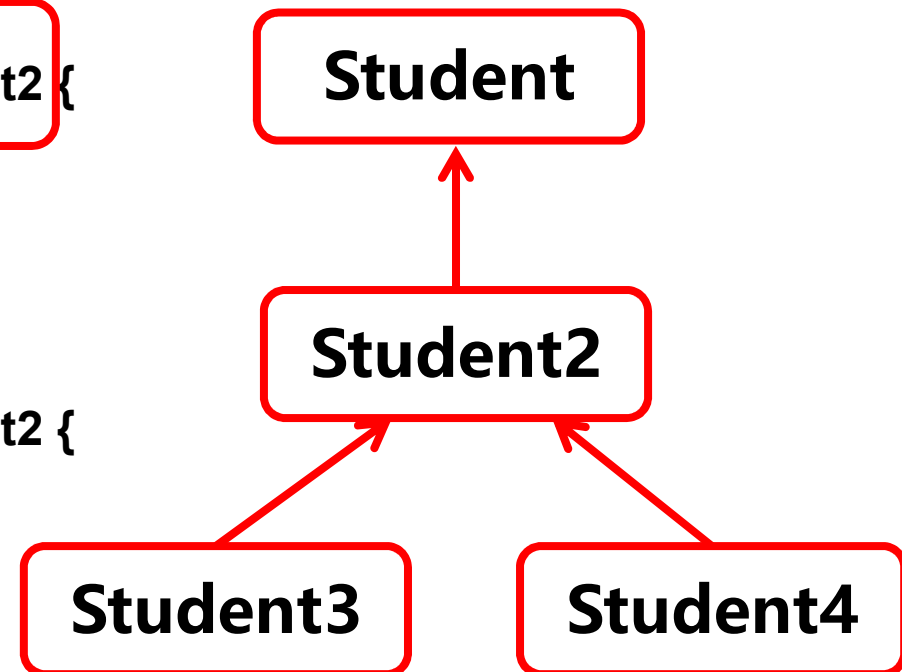


允许多次继承

```
class Student2 extends Student {  
    String name;  
    String address;  
    ...  
};
```

```
class Student3 extends Student2 {  
    String email;  
    ...  
};
```

```
class Student4 extends Student2 {  
    float scores[];  
    ...  
};
```



父类和子类的构造

□ 构造方法：类被实例化的时候会被执行

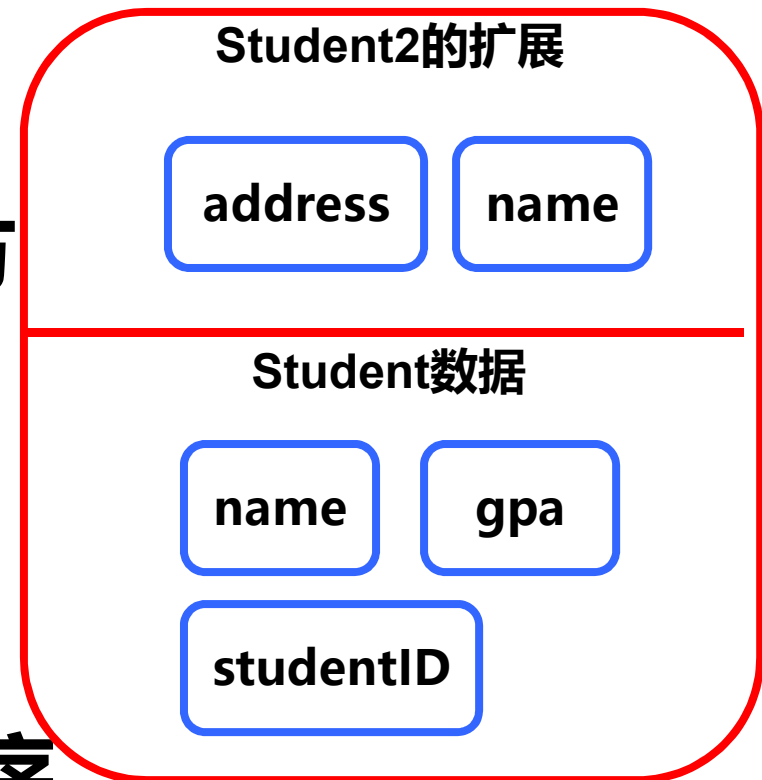
```
class Student2 extends Student {  
    ...  
};  
Student2 student = new Student2();
```

□ 问题1: Student的构造方法被执行吗？

- 实例化Student2的时候，同样需要分配空间给父类对象
→ 父类的构造函数会被执行

□ 问题2：构造函数执行顺序

- 父类 – 子类



父类和子类的构造

□ 如果父类有多个构造函数

```
class Student {  
    Student() {  
        ...  
    };  
    Student(String name) {  
        ...  
    };  
}  
class Student2 extends Student {  
    Student2() {  
        ...  
    };  
}
```

□ 用super来表示父类构造函数，显式调用

- super()或super("abc") 调用不同构造函数
- 必须为子类构造函数第一条语句

方法和变量的访问控制

- ❑ 类有些方法和变量，不对外公开
- ❑ 访问控制关键字 (access modifier)
 - public: 全局
 - private: 仅自己
 - 缺省: 包(package)内
 - **protected**
 - 子类可访问

方法的访问控制

- ❑ 类有些公有方法，不允许覆盖

- ❑ 加上**final**关键字

```
class Student2 extends Student {  
    final void setName(String studentName) {  
        name = studentName;  
        super.setName(studentName.toCharArray());  
    }  
};  
class Student3 extends Student2 {  
    void setName(String studentName) { //报错  
        ...  
    }  
}
```

- ❑ 原因

- 保护没必要覆盖的方法，避免错误

方法的访问控制

- ❑ 类的方法加上**final**关键字，不允许覆盖
- ❑ 原因：保护没必要覆盖的方法，避免错误
 - 为什么不采用private修饰？
- ❑ 注意：依然允许重载

```
class Student2 extends Student {  
    final void setName(String studentName) {  
        name = studentName;  
        super.setName(studentName.toCharArray());  
    }  
};  
class Student3 extends Student2 {  
    void setName(char [] studentName) { //允许  
        ...  
    }  
}
```

类的继承控制

- 有些类，不允许被继承

- 加上**final**关键字

```
final class Student2 extends Student {  
    void setName(String studentName) {  
        name = studentName;  
        super. setName(studentName.toCharArray());  
    }  
};  
class Student3 extends Student2 { //报错  
    void setName(String studentName) {  
        ...  
    }  
}
```

- 原因和例子？

- 保护没必要重载的类
- 比如各种JDK提供的类库，如String

类与类的关系

```
class NewElement extends Element {  
    private Student student;  
    public NewElement getNext() {  
        return next;  
    }  
}
```

□ **继承** `class NewElement extends Element`

- “是一种” (is a)关系
- NewElement数据是一种Element数据

□ **声明引用** `private Student student;`

- “包含” (has a)关系
- NewElement包含Student数据

类与类的关系

```
class NewElement extends Element {  
    private Student student;  
    public NewElement getNext() {  
        return next;  
    }  
}
```

□ 构造顺序

■ 先父类，后子类

■ 类内部的构造顺序

→ static 类的引用 = new ..., 先构造

→ 类的引用 = new ..., 次之

如果声明时没有加 “= new ...”，则仅声明，不构造

→ 再执行构造函数

面向对象语言的多重继承

□ 多重继承

- 一个类继承两个父类

- 逻辑上合理

 - Professor类可以继承Researcher类和Lecturer类

 - 既是上课的老师也是科研人员

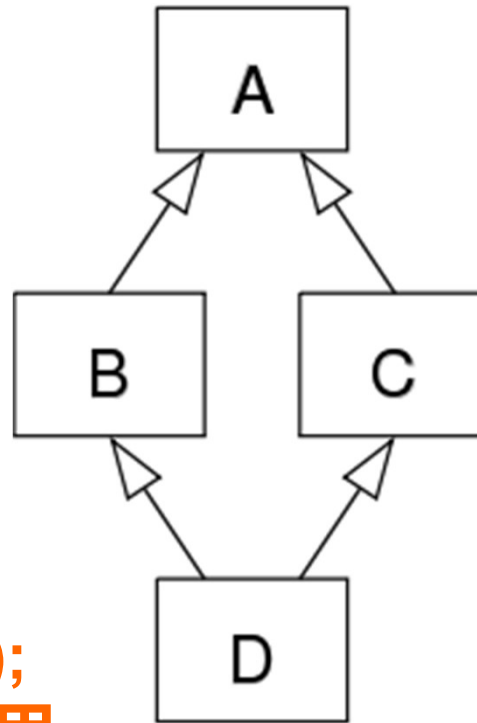
□ Java不允许多重继承(<1.8)

- 为了语言的简洁，避免实现的复杂化

面向对象语言的多重继承

❑ 多重继承的钻石问题(The diamond problem)

A有void foo()方法



D d = new D();
d.foo()怎么调用

思考

- 思考理解继承(面向对象的关键思想之一)
- 思考回忆你了解的面向对象语言（如C++）的继承机制，与Java比较

复旦大学计算机科学技术学院



JAVA语言

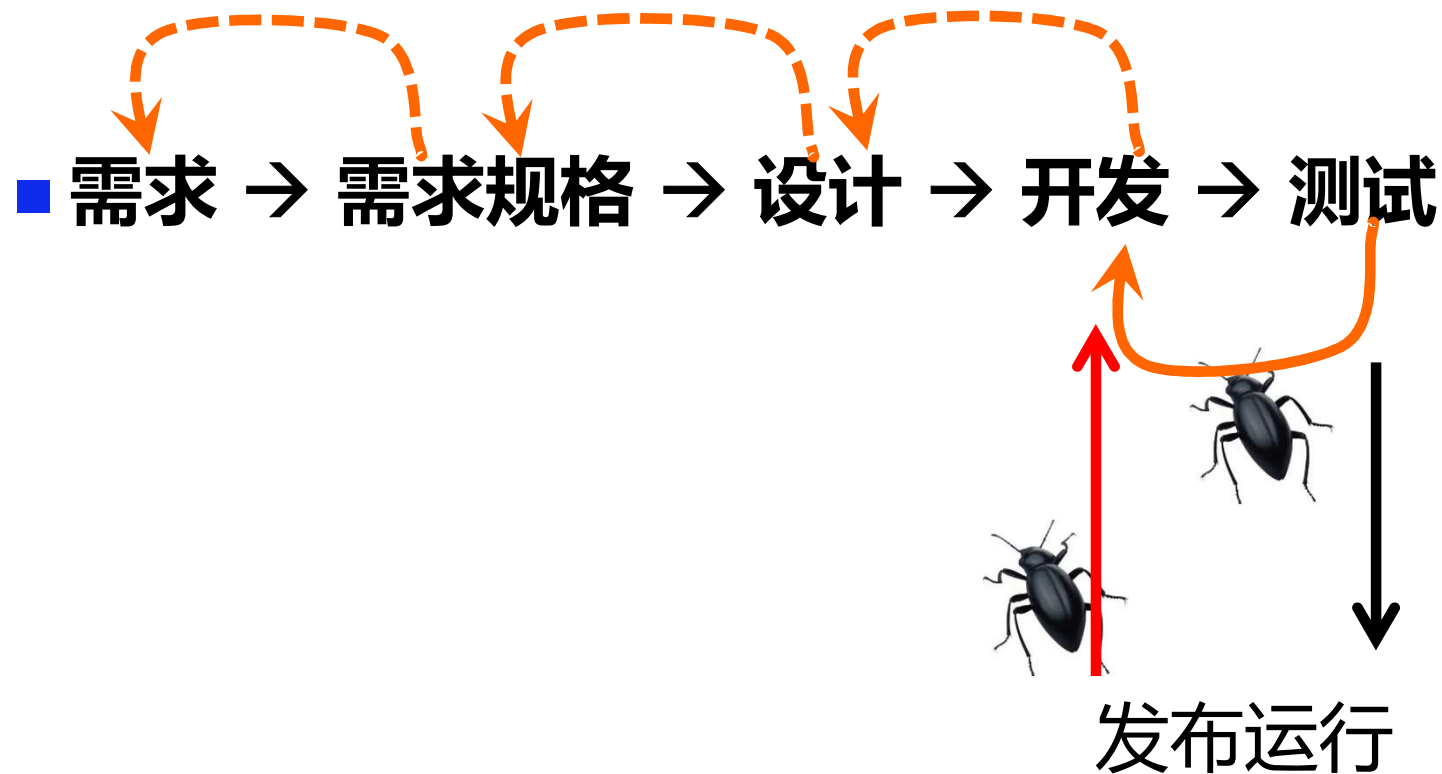
19. Debug

周扬帆

2020-2021第二学期

关于Bug那些事

□ 人无完人，工业软件几乎一定会有缺陷



什么是Bug

- 软件/系统 行为不符合**设计预期**
 - 开发人员的错误
- 有时候甚至开发时都不知道设计预期是什么
 - 软件使用时出现不符合用户预期的行为
 - 软件使用时出现没想到的、有不良影响的行为
 - 也经常叫bug



什么是Bug

□ Bug大致的表现形式

- 功能出错（死机、结果错等等）
- 性能故障（慢）
- 安全漏洞
- 资源滥用
- ...

Bug例子:瑞穗证券乌龙指

□ 本来客户委托

淘股网

J-Com股票大特卖

促销 ¥61万元

件 (库存1件)

立即购买

加入购物车

Bug例子: 瑞穗证券乌龙指

□ 手残君的操作

淘股网

J-Com股票大特卖

促销 ¥1元

看这里

— 1 + 件 (库存61万件)

立即购买

加入购物车

Bug例子:瑞穗证券乌龙指

□ 交易所跌停处理

淘股网

J-Com股票大特卖

促销 ¥57万元

— 1 + 件 (库存61万件)

立即购买

加入购物车

Bug例子：瑞穗证券乌龙指

- **问题：手残君在开盘前发现了**
 - 好彩能撤销！
 - Bug：撤销不了
- **结果：白菜价卖股票，损失400亿元 (27亿人民币)**
- **瑞穗证券：富士通开发的软件的Bug应该负责！！！！**

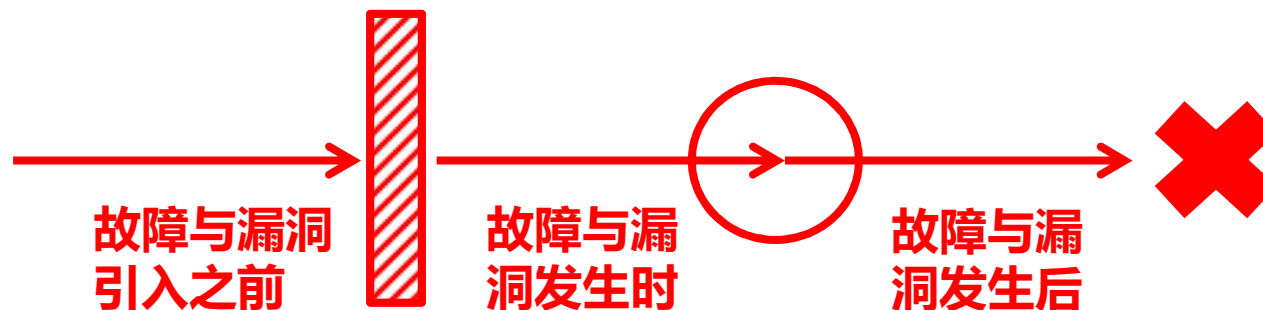
Bug例子：瑞穗证券乌龙指

- ❑ 瑞穗证券：白菜价卖股票，损失400亿元 (27亿人民币)
 - 富士通开发的软件的Bug应该负责!!!
- ❑ 程序的bug算是“重大过失”吗？
 - 难说！系统不能保证没有bug（人类做不到）
 - 结论：Bug的损失不能都归罪开发商
 - 微软（最大的bug制造商）：好彩！
 - 需要证明开发者确实尽力了
 - 各种过程文档很重要啊！ 打官司用 😊

软件工程

- **花最少的钱，做出最好的系统**
- **对付bug的方法**
 - **好的设计、开发方法**
 - 容错、代码审查等
 - **好的测试方法**
 - **好的侦错、除错方法**

软件工程对待故障



- **防: 防止软件故障的发生**
 - 软件采取的主动(proactive)性的预防策略
- **容: 降低故障对软件/系统的破坏**
 - 软件采取的被动(reactive)性的防御策略
- **除: 排除已经发生的故障**
 - 定位其所在, 以协助人工干预或采用自动化手段, 更正软件

思考

- ❑ **思考、了解软件测试的方式（除了课程之前提及的单元测试），思考他们为什么有效**
- ❑ **总结你平时debug的方式方法，思考有没有更好的方法**
- ❑ **了解单步执行与断点操作，思考其原理和实现**

复旦大学计算机科学技术学院



JAVA语言

20. 课堂练习

周扬帆

2020-2021第二学期

课堂练习：找Bug

- **new project**
 - 找到示例程序LinkedList的bugs
 - 修改，直到能通过TA给出的测试用例
- **关键1：设计测试用例**
- **关键2：出错之后，重现错误，采用断点/单步跟踪debug**

课堂练习：找Bug-代码解读

链表的元素

```
class Element
{
    private int n;
    private Element next;
    public void setNum(int num) {
        n = num;
    }
    public void setNext(Element nextElement) {
        next = nextElement;
    }
    public int getNum() {
        return n;
    }
    public Element getNext() {
        return next;
    }
}
```

课堂练习：找Bug-代码解读

链表

```
class LinkedList
{
    Element first = null;
    Element last = null;
    public void add(Element e) {
        if(last == null) {
            first = new Element();
            first.setNum(e.getNum());
            last = first;
        }
        else {
            Element newLast = new Element();
            newLast.setNum(e.getNum());
            last.setNext(newLast);
            last = newLast;
        }
    }
}
```