OOP with Java

Yuanbin Wu
cs@ecnu

OOP with Java

- 通知
  - Project 4 提交时间 4 月 27 日晚 9 点

- 复习：对象的创建
  - new 操作符 + 构造函数
  - 构造函数
    - 一些特殊的方法
    - 函数名与类名一样
    - 没有返回值
  - new 操作符
    - 返回一个引用，指向调用构造函数所创建的对象
    - 类似于 malloc

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

    MyType() { System.out.println("HI"); };

    public static void main(String [ ]args) {
        MyType m = new MyType();
    }
}
```

- 复习
  - 函数重载：函数 = 函数名 + 参数列表

```
public class Printer {
    void print(int x) {
        System.out.println(“print an integer: ” + x);
    }
    void print(MyType m) {
        System.out.println(“print a MyType: ” + m.get());
    }
}
```

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

    MyType(double x) { set(x); };
    MyType() { System.out.println(“mytype create”); };

    public static void main(String [ ]args) {
        MyType m = new MyType();
        MyType n = new MyType(2.0);
    }
}
```

- 复习
  - this 关键字
    - 在类的非静态方法中，返回调用该方法的对象的引用

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

  - 销毁对象
    - 垃圾回收 不等于 销毁对象
    - 仅回收 new 创建的内存
    - 是否回收，何时回收由 Java 虚拟机控制 .

OOP with Java

- Java 包
- 访问控制
- 封装

Java 包

- 代码的组织方式
  - 表达式
    - a+b, x = 1,…
  - 语句
    - int a = 1, b = 2;
  - 控制结构
    - If-else, loops
  - 方法 (函数)
  - 类
  - 程序库 (多个类) ← “ 包 ”

Java 包

- 包 (package)
  - 由多个类组成
  - 这些类共享同一个名字空间 (namespace)

Java 包

- 例子 1:java.util
  - 包的名字 : java.util
  - 包含 java 提供的常用工具
  - LinkedList, Date, Random 为 java.util 包中的三个类
    - LinkedList.java, Date.java, Random.java

```
java.util.LinkedList
java.util.Date
java.util.Random
….
```

Java 包

- 例子 2: java.io
  - 包的名字 : java.io
  - 包含一些 java 的 io 操作
  - FileInputStream, FileOutputStream, FileReader, FileWriter 为 java.io 中的四个类

```
java.util.FileInputStream
java.util.FileOutputStream
java.util.FileReader
java.util.FileWriter
….
```

Java 包

- 使用包
  - 使用包中的类 (如 java.util.Random)
  - 直接使用

```
public class DirectAccess{
    public static void main(String []argv){
        java.util.Random r = new java.util.Random();
        System.out.println(r.nextInt());
    }
}
```

Java 包

- 使用包
  - import 语句

```
import java.util.Random;

public class ImportAccess{
    public static void main(String []argv){
        Random r = new Random();
        System.out.println(r.nextInt());
    }
}
import java.util.LinkedList;
import java.util.Date;
import java.util.Random;

import java.util.*;
```

Java 包

- 创建包
  - package 语句
    - .java 文件首行
    - 指定当前 java 文件中的类属于哪一个包
  - 包的结构
    - 包的结构与文件目录结构一致
  - Let' try

Java 包

```
package mypackage;
public class MyType {
    int i;
    double d;
    char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
    public MyType(int i1, double d1, char c1){
        i = i1; d = d1; c = c1;
    }
    public static void main(String []args){
        MyType m = new MyType(1, 1.0, 'a');
        System.out.println(m.get());
    }
`
javac mypackage/MyType.java
java mypackage/MyType
java mypackage.MyType
```

Java 包

- 一个 java 包会含有不止一个 java 文件
- 如何组织这些文件？
  - 方案 1: 所有 java 文件放在同一目录中
  - 方案 2: 将 java 文件归类，放入不同的目录中
- Java 包的结构
  - java.util.*
  - 等于 java 文件目录结构

Java 包

- 包结构与目录结构一致
  - 将 java 文件放入不同的子目录中
  - Let's try

```
restaurant/
    - people/
        - Cook.class
        - Waiter.class
    - tools/
        - Fork.class
        - Table.class
```

```
import restaurant.people.Cook;
import restaurant.tools.Fork;
import restaurant.tools.*;
// import restaurant.*;
```

Java 包

- 使用包
  - classpath
    - javac, java 的参数 (-cp), CLASSPATH 环境变量
    - 指定使用包的位置
  - Let’s try

Java 包

- 类共享同一个名字空间 (namespace)
  - 同一个包中，类的名字不能相同
  - 不同包中，类的名字可以相同
- 例如

包结构 = 目录结构

```
restaurant/
    - people/
        - Cook.class
        - Waiter.class
        - A.class
    - tools/
        - Fork.class
        - Table.class
        - A.class
```

```
import restaurant.people.A;
import restaurant.tools.A;
```

Java 包

- jar 包
  - “ 打包” 包 (packed package)
  - 将包 (目录) 变成文件
    - 方便发布 , 使用
      - c: create
      - f: output to file

```
jar cf restaurant.jar restaurant
```

- 复习
  - Java 包
    - 创建包 : package 语句，包结构与目录结构一致
    - 使用包 : import

```
restaurant/
    - people/
        - Cook.class
        - Waiter.class
    - tools/
        - Fork.class
        - Table.class
```

```
import restaurant.people.Cook;
import restaurant.tools.Fork;
import restaurant.tools.*;
import restaurant.*;
```

访问控制

- 访问控制
  - 类：数据 + 方法
  - 公开的数据和方法
    - 提供服务
    - 所有用户都可以使用
    - 需要保持稳定，不经常变化
  - 隐藏的数据和方法
    - 细节的，辅助性的方法和数据
    - 不向用户公开
    - 易变

访问控制

- 例子

```
public class Refrigerator{
      Refrigerator() { … }
      open() { … }
      close() { … }
      put() { … }
      get() { … }
      engineStart() {…}
      engineSleep() {…}
      lightOn() { … }
      lightOff() { … }
}
```

访问控制

- 访问控制
  - 修饰类的数据 / 方法
  - 是否能被访问？ 能被哪些用户访问
- package access
  - 类 A 的数据 / 方法能够被同一包中的类 B 访问，不能被其他类访问
- public
  - 类 A 的数据 / 方法能够被任意一个类 B 访问
- private
  - 类 A 的数据 / 方法仅能被 A.java 中的类访问，其他任意类 B 不能访问
- protected
  - 类 A 的数据 / 方法仅能被类 A 的子类 B 访问，其他类不能访问

访问控制

- package access
  - 一个类的成员被标识为 package access
    - 同一个包中的类可以访问
    - 其他包中的类不能访问
  - 没有标识符

访问控制

- package access
  - 例子

```
package mypackage;
public class MyType {
    int i;
    double d;
    char c;
    void set(double x) { d = x;}
    double get() { return d; }
    MyType(int i1, double d1, char c1){
        i = i1; d = d1; c = c1;
    }
}
```

```
package mypackage;
public class AnotherType {
    public static void main(String [ ]args){
        MyType m = new MyType(1, 1.0, 'a');
        int i = m.i;
        double d = m.d;
        char c = m.c;
        m.set(2);
        System.out.println(m.get());
    }
}
```

访问控制

- package access

默认包 (Default Package)

```
//package mypackage;
public class MyType {
    int i;
    double d;
    char c;
    void set(double x) { d = x;}
    double get() { return d; }
    MyType(int i1, double d1, char c1){
        i = i1; d = d1; c = c1;
    }
}
```

```
//package mypackage;
public class AnotherType {
    public static void main(String [ ]args){
        MyType m = new MyType(1, 1.0, 'a');
        int i = m.i;
        double d = m.d;
        char c = m.c;
        m.set(2);
        System.out.println(m.get());
    }
}
```

访问控制

- package access
  - 默认包 (default package)
  - 如果没有 package 语句 , java 默认当前目录中的 java 文件属于同一个包

访问控制

- public
  - 类的成员标识为 public，则所有用户都能访问该成员

```
package mypackage;
public class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
    public MyType(int i1, double d1, char c1){
        i = i1; d = d1; c = c1;
    }
}
```

```
import mypackage.MyType;
public class AnotherType {
    public static void main(String [ ]args){
        MyType m = new MyType(1, 1.0, 'a');
        int i = m.i;
        double d = m.d;
        char c = m.c;
        m.set(2);
        System.out.println(m.get());
    }
}
```

访问控制

- public

```
package mypackage;
public class MyType {
    int i;
    double d;
    char c;
    void set(double x) { d = x;}
    double get() { return d; }
    public MyType(int i1, double d1, char c1){
        i = i1; d = d1; c = c1;
    }
}
```

```
import mypackage.MyType;
public class AnotherType {
    public static void main(String [ ]args){
        MyType m = new MyType(1, 1.0, ‘a’);
        int i = m.i;
        double d = m.d;
        char c = m.c;
        m.set(2);
        System.out.println(m.get());
    }
}
```

访问控制

- private
  - 类的成员被标识为 private，则除了该类自身外，任何类不能访问改成员

访问控制

- private

```
package mypackage;
public class MyType {
    private int i;
    private double d;
    private char c;
    private void set(double x) { d = x;}
    private double get() { return d; }
    public MyType(int i1, double d1, char c1){
        i = i1; d = d1; c = c1;
    }
    public static void main(String []args){
        MyType m = new MyType(1, 1.0, 'a');
        m.i = 2;
        System.out.println(m.get());
    }
}
```

```
import mypackage.MyType;
public class AnotherType {
    public static void main(String [ ]args){
        MyType m = new MyType(1, 1.0, ‘a’);
        int i = m.i;
        double d = m.d;
        char c = m.c;
        m.set(2);
        System.out.println(m.get());
    }
}
```

Access Denied!

访问控制

- private
  - 同一类的不同对象

```
class B {
    private int i;
    public B(int k) {i = k;}
}
public class A{
    private int i;
    public A(int k) {i=k;}
    public static void main(String args[]){
        A a1 = new A(10);
        A a2 = new A(11);
        B b1 = new B(10);
        B b2 = new B(11);
        System.out.println(a1.i+a2.i);
        System.out.println(b1.i+b2.i);
    }
}
```

访问控制

- private 构造函数
  - 无法创建该类的对象

```
package mypackage;
public class MyType {
    private int i;
    private double d;
    private char c;
    private void set(double x) { d = x;}
    private double get() { return d; }
    private MyType(int i1, double d1, char c1){
        i = i1; d = d1; c = c1;
    }
    public static void main(String []args){
        MyType m = new MyType(1, 1.0, 'a');
     }
}
```

访问控制

- private 构造函数应用：该类只有一个对象

```
package mypackage;
public class MyType {
    private int i;
    private double d;
    private char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
    private MyType(int i1, double d1, char c1){
        i = i1; d = d1; c = c1;
    }
    private static MyType instance= null;
    public static MyType getInstance(){
        If (instance == null)
            Instance = new MyType(1, 1.0, ‘a’);
        return instance;
    }
}
```

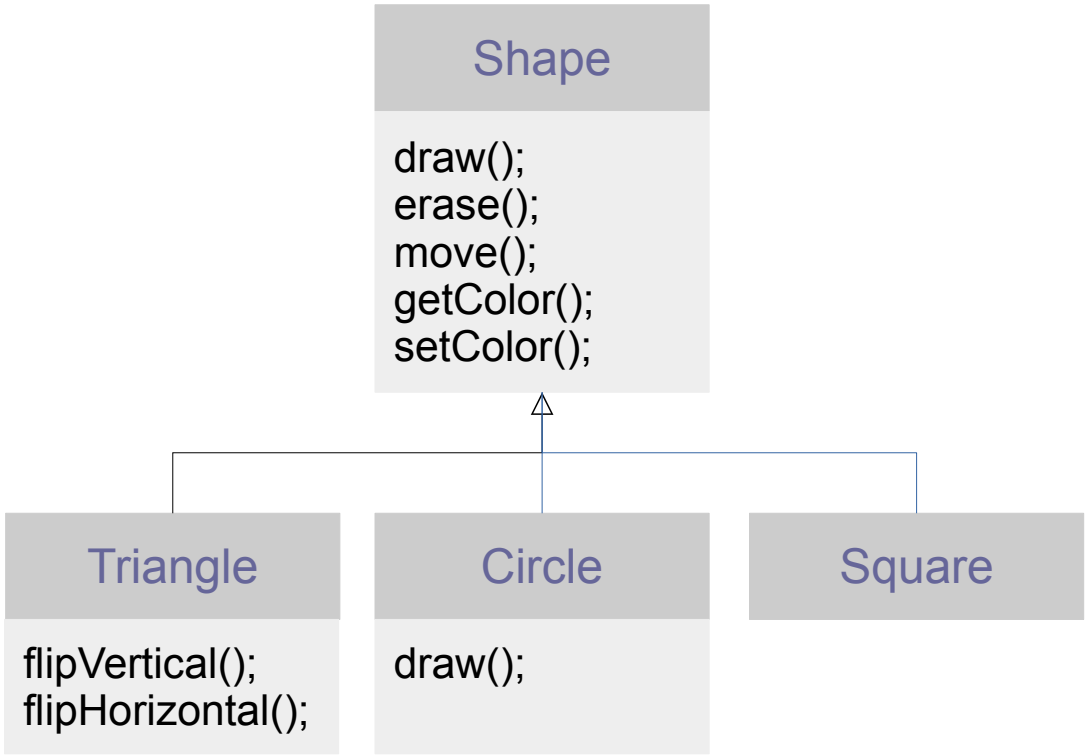
```
import mypackage.MyType;
public class AnotherType {
    public static void main(String [ ]args){
        MyType m = MyType.getInstance();
        m.set(2);
        MyType n = MyType.getInstance();
        System.out.println(n.get());
    }
}
```

单件模式 (Singleton)
Design pattern

避免使用单件模式
Design anti-pattern

访问控制

- protected
  - 继承




```
class Shape {
    private int width;
    private int hieght;
    protected int data;
    public void draw() {…}
    public void earse() {…}
    public void move() {…}
    ….
}

class Triangle extend Shape{
    public void flipVertical() { data }
    public void flipHorizontal() { data }
}
```

访问控制

- 类成员的访问控制
  - package access, private, public protected
- 类的访问控制
  - package access, public

访问控制

- public class
  - 每个 .java 文件包含一个 public class
  - 该 class 的名字等于 .java 文件名
- package access class
  - 每个 .java 文件中除去 public class 外，其他的 class 为 package access

访问控制

```
package mypackage;
public class MyType {
    int i;
    double d;
    char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
    public MyType(int i1, double d1, char c1){
        i = i1; d = d1; c = c1;
    }
}

class PAClass{
    public PAClass() {
        System.out.println(“PA class”);
    }
}
```

```
package mypackage;
public class AnotherType {
    public static void main(String [ ]args){
        MyType m = new MyType(1, 1.0, 'a');
        PAType p = new PAType();
    }
}
```

访问控制

- MyType Revisited

```
package mypackage;
public class MyType {
    private int i;
    private double d;
    private char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
    public MyType(int i1, double d1, char c1){
        i = i1; d = d1; c = c1;
    }
    public static void main(String []args){
        MyType m = new MyType(1, 1.0, 'a');
        System.out.println(m.get());
    }
}
```

封装

- 封装 (Encapsulation)
  - 访问控制的设计
  - Public
    - 提供服务，稳定
    - 修改 public 成员将影响用户程序
  - Private
    - 提供辅助，易变
    - 修改 private 成员是安全的

Separate things that change from things that stay the same.

封装

- 封装 (Encapsulation)
  - 在满足需求的情况下，接口尽量简单
  - 在可能的情况下尽量使用 private

封装

- C 语言的封装
  - Header file
  - static and extern