Java

0. 概述

平台无关：Java运行环境（JRE）由Java虚拟机、类库以及一些核心文件组成

Java SE：Java标准平台，提供标准的JDK

开发步骤：源文件 -> 字节码 -> 解释器执行字节码

主类必须和文件名一致，包含 `main` 函数，不一定是public类型

```
javac hello.java
java A
```

1. 基本数据类型

Unicode字符集：65536个字符集

1.1 基本数据类型：

逻辑：boolean,

整数：byte(1), short(2), int(4), long

```
byte: -128~127
```

字符：char(2)　0~65535

浮点数：float(常量后面有f)（32）, double（64）

```
long number = 10L;
```

类型的转换运算：**byte, short, char, int, long, float, double**，支持从左到右自动转换，否则报错（或者必须强制转换）

```
float x = 1 // int->float自动转换
```

1.2 输入输出：

输入

```
import java.util.Scanner;

Scanner reader = new Scanner(System.in);
int x = reader.nextInt();
double y = reader.nextDouble();
```

输出

`System.out.print()` 和 `System.out.printIn()` 类似，区别在于换行，可使用 `+` 将变量、字符串并置输出

`System.out.printf()` 用法与C语言类似

1.3 数组

声明：不允许在声明的 `[]` 中指定数组元素的个数，声明方式为 `int [] a;`

```
//与C语言不同
int size = 30;
int [] a = new int[size]
// 或者初始化
int [] a = {1,2,3};
```

赋值：`a = new int[4]`，此时数组元素会赋上一个默认的值

二维数组：

可以这样初始化：`int [][] a = {{1,2,3},{4,5,6}};`

```
int [][] x = new int [4][3];
//遍历
for(int [] items: x){
    for(int item: items)
        System.out.print(item+" ");
    System.out.print("\n");
}

int [][] b = new int[3][]; //也可
b[0] = new int[3];
b[1] = new int[12];
b[2] = new int[1]
```

length：`a.length()`

数组的引用：当对数组进行赋值时，`a = b`，系统自动释放原来a的空间

2. 运算符

instanceof: obj intanceof class，判断对象是否右边类的实例

```
class Card(){
    int n;
}

public class hhh{
    public static void main(String [] args){
        Card c = new Card();
        System.out.println(c instance of Card);
        // 子类对象 intanceof 父类 = true;
    }
}
```

```
import java.util.Scanner;

int sum = 0;
// int x;
Scanner reader = new Scanner(System.in);
while(reader.hasNextInt()){
    sum += reader.nextInt();
}
```

3. 类与对象

new的结果是一个16进制数，成为对象的引用

没有析构方法

局部变量没有默认值

成员变量的赋值可作为初值赋予，但是不能这样：

```
class A {
   int a;
   float b;
   a = 12;        //非法，这是赋值语句（语句不是变量的声明，只能出现在方法体中）
   b = 12.56f;    //非法
}
```

3.1 参数传递

值传递：基本数据结构的参数传递都是值传递，不会影响原来的变量

引用传递：数组、对象、接口为引用传递

可变参数：

```
public void func(int ... x){ // 可变参数必须写在参数表的最后
    x[0]++;
    for(int item: x) // 同样适用于可变参数
        System.out.print(item+" ");
    System.out.print("\n");
}
```

3.2 对象的组合

```
// ???
class A{
    int x;
}
class B{
    int y;
    A a; //刚刚创建的时候还是一个空对象，具体使用要在后面进行赋值，具体关系见书
}
```

3.3 实例成员和类成员

类成员变量共享存储空间

```
class Animal{
    int x; // 实例变量
    static int number; // 类变量，可以通过类名直接访问
    int max(int a, int b){} // 实例方法，两种变量都可以操作
    static int min(int a, int b){} // 类方法，只能操作类变量，可以直接类名调用
}
```

典型的类方法有Array和Math

```
int [] sum = {0,1,3,-9,2};
Arrays.sort(sum);
for(int item: sum)
    System.out.print(item + " ");
```

3.4 重载

多态：重载、重写（继承有关）

重载：参数的个数、参数的类型有不同存在；**注意返回类型和参数名字不参与比较**

歧义调用

3.5 this关键字

表示某个对象，可以出现在实例方法、构造方法中，**但是不能出现在类方法中**

当实例变量、类变量都在实例方法中出现时，默认格式为（方法的调用也与其类似）：

```
class A{
    int x;
    static int y;
    void f(){
        this.x = 100;
        A.y = 200; // 在没有出现名字相同的局部变量时，二者前面的也可以忽略
    }
}
```

3.6 包

包编译、运行必须在上一级目录进行

```
package top.omysycamoro.top;
```

```
cd "d:\Github\2021study\Java" ; if ($?) { java top.omysycamore.hello }
########################
cd top/omysycamore
cd ../..
javac *.java
java top.omysycamore.hello
```

3.7 import语句

系统自动引入`java.lang`包中的类

引入自定义包中的类

```
import top.omysycamore.*;
// 可以更新位置
set classpath = ...;
hello h = new hello();
```

3.8 访问权限

私有：private，在其他类中无法访问

共有：public

受保护：protected ，在同一个包类可以访问，可**以被所有的子类访问（子类可以不在同一个包内）**

友好：什么都不用修饰，只能访问**同一个包**的友好变量和方法

注意类的修饰没有protected和private

3.9 基本类型的类封装

Double, Float, Integer, Long, Byte, Short, Character

3.10 对象数组

```
Student [] s = new Student[10];
for(Student item: s)
    item = new Student(); //还要对每一个元素进行初始化
```

4. 子类和继承

```
class A extends Rational
```

只能继承一个父类

在同一个包内，继承除了private所有变量和方法；不在同一个包内，继承public和protected

对于父类的private变量，可以留出public的方法，方便子类对其进行访问

```
class A
{
    private int number;;
    public int getNumber(){
        return number;
    }
}
```

成员变量的隐藏和方法重写

重写：方法的名字、参数个数、参数的类型和父类完全相同

重写时，不能降低方法的访问权限，但是可以提高（顺序：public, protected, 友好的，private）

4.1 super关键字

```
class Sum{
    int [] x;
    Sum(int [] tmp){
        x = tmp;
    }
    int caculator(){
        int sum = 0;
        for(int item: x)
```

```
            sum += item;
        return sum;
    }
}

class Average extends Sum{
    // 在Java的继承中，要求子类的构造方法必须调用父类的构造方法，
    // 如果子类没有显示的调用父类的构造方法，就会隐含调用父类的无参构造方法
    Average(int[] tmp) {
        super(tmp); // super必须是构造方法中第一条语句
        // TODO Auto-generated constructor stub
    }
    @Override // 重写方法
    int caculator(){
        int sum = 0;
        for(int item: x)
            sum += item;
        return sum/x.length;
    }
    double getSum(){
        return super.caculator(); // super调用父类隐藏的方法
    }
}
```

使用super调用被隐藏的方法时，使用的成员变量同样是被隐藏的成员变量，或继承的成员变量

4.2 final关键字

```
final class A{
    ...
}
// final类不能被继承
// final方法不允许被子类重写
// final修饰常量，在声明时必须赋值
```

4.3 上转型对象

```
class Animal{}
class Tiger extends Animal{}
...
Animal a;
Tiger b = new Tiger();
a = b; // 称为对象a是对象b的上转型对象，“老虎是动物”
```

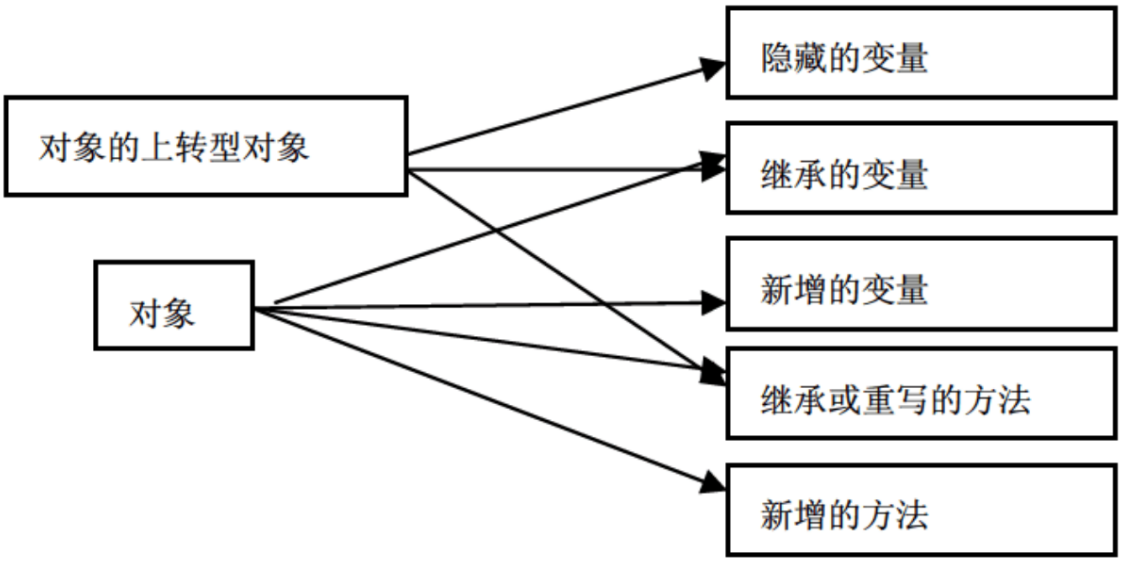

图 5.9　上转型对象示意图

没有新增的变量和方法，但是仍然保存重写的方法

```
// 但是可以再次将上转型对象转化为子类对象
// 子类对象又具有子类所有的属性和方法
// 如果子类重写了父类的静态方法，那么子类对象的上转型对象不能调用子类重写的静态方法，只能调用父类的静态方法
Tiger c = (Tiger)a;
```

4.4 多态

abstract关键词：抽象类和抽象方法

子类必须对父类的抽象方法进行重写

不能用final和abstract同时修饰方法类，static也不能，即必须是实例方法

抽象类不能用new创建该对象，但是可以用new成为其子类的上转型对象，由此可以使用子类重写的方法

```
Animal a = new Tiger();
a = new Lion();
a.cry();
// 新的上转型对象
a = new Dog();
a.cry();
```

```
// 设计思路
public abstract class Geometry
{
    public abstract double getArea();
}
// Pillar
public class Pillar
```

```
{
    Geometry bottom;
    double height;
    Pillar(Geometry b, double h){
        g = b;
        height = h;
    }
    public double getVolume(){
        return bottom.getArea()*height;
    }
}
// 下面子类各自实现即可
```

4.5 补遗

4.5.1 静态块

当类的字节码进入内存时，类的静态块会立刻被执行。

```
class AAA{
    static {
        System.out.println("Hello in AAA");
    }
}
public class test{
    static {
        System.out.println("I'm the first");
    }
    public static void main (String [] args){
        AAA a = new AAA();
        System.out.println("I am studying (static) block");
    }
}
```

5. 接口与实现

接口体中只有抽象方法，所有**常量**（不能有变量）的权限都是public、static，所有**方法**的权限都是public、abstract（允许省略）

5.1 接口实现

```
class A implements Printable, Addable
```

必须重写接口中的方法：**必须加public修饰**

如果没有重写，该类必须为抽象类，**抽象类能够重写接口方法**

接口前面的修饰（public，private）与类和权限访问类似

接口同样能通过extends继承

5.2 接口回调

```
Com com;  // 接口变量
Implecom ipc; // 实现接口的类变量
com = ipc // 接口回调，接口变量可以调用类实现的接口方法
```

同上，`Com com = new ipc()`这种实现方法类似于上转型对象可以使用子类重写的方法

接口回调的实现过程中体现**多态**

5.3 接口参数

```
interface SpeakHello {
    void speakHello();
}
class Chinese implements SpeakHello {
   public void speakHello() {
      System.out.println("中国人习惯问候语：你好,吃饭了吗？ ");
   }
}
class English implements SpeakHello {
    public void speakHello() {
      System.out.println("英国人习惯问候语:你好,天气不错 ");
    }
}
class KindHello {
   public void lookHello(SpeakHello hello) {     //接口类型参数
      hello.speakHello();                        //接口回调
   }
}
public class Example6 5 {
   public static void main(String args[]) {
      KindHello kindHello=new KindHello();
      kindHello.lookHello(new Chinese());
      kindHello.lookHello(new English());
   }
}
```

相当于

```
SpeakHello hello = new Chinese(); //接口回调
hello.speakHello(); // 可以调用类实现接口的方法
```

7. 内部类和异常类

7.1 内部类

外嵌类的成员变量和方法在内部类中仍然有效

外部类可以声明内部类的实例

可以给内部类添加static关键字，成为static内部类

内部类不能声明类变量和类方法

```
class CowFarm
{
    static String cname;
    Cow cow;
    CowFarm(int a, int b, int c, String s) {
        cow = new Cow(a, b, c);
        cname = s;
    }
    void farmSpeak(){
        cow.speak();
    }
    class Cow{
        int height;
        int weight;
        int price;
        Cow(int a, int b, int c){
            height = a;
            weight = b;
            price = c;
        }
        void speak(){
            System.out.println("height = " + height + "\nweight = " + weight
+ "\nprice = " + price);
        }
    }
}
public class test
{
    public static void main(String[] args) {
        CowFarm c = new CowFarm(300, 200, 400, "Betty");
        c.farmSpeak();
    }
}
```

7.2 匿名类

匿名类可以继承也可以重写父类方法

使用匿名类，必须在某个类中直接用匿名类创建对象，即匿名类必须是内部类

匿名类可以访问外嵌类中的成员变量和方法，匿名类的类体不可以声明static成员变量和static方法

```
class Polygon {
   public void display() {
      System.out.println("在 Polygon 类内部");
   }
}

class AnonymousDemo {
   public void createClass() {
      // 创建的匿名类继承了 Polygon 类
      Polygon p1 = new Polygon() {
          // 可以在内部对父类的方法进行重写
         public void display() {
            System.out.println("在匿名类内部。");
         }
      };
```

```
        p1.display();
    }
}

class Main {
    public static void main(String[] args) {
        AnonymousDemo an = new AnonymousDemo();
        an.createClass();
    }
}
```

函数可以使用匿名类作为参数

```
class People{
    void speak(){
        System.out.println("People");
    }
}

public class test{
    public static void sayhello(People p){
        System.out.println("hhh");
        p.speak();
    }
    public static void main(String[] args) {
        sayhello(new People(){void speak(){System.out.println("already
changed");}});
    }
}
```

匿名接口类似

7.3 异常类

```
try{
    // 可能包含异常的语句
}
catch(ExceptionSubClass1 e){
    e.getMessage(); // 输出异常信息
    e.printStackTrace();
    e.toString();
}
catch(ExceptionSubClass2 e){
    ...
}
finally{
    // 无论异常，都会被执行
}
```

自定义异常类：

```java
// 看完
package LoginException;
public class MyException extends Exception{
    private String message;
    public MyException(String m){
        super(m);
        message = m;
    }
}
```

```java
package LoginException;

public class Login {
    String usrname;
    String password;
    public Login(String name, String psword){
        usrname = name;
        password = psword;
    }
    public void isCorrect(String psword) throws MyException{
        if(!psword.equals(password))
            throw new MyException("password incorrect");
    }
}
```

```java
package LoginException;

public class Main {
    public static void main(String[] args) {
        Login log = new Login("hhh", "123");
        try{
            log.isCorrect("1234");
        }catch(MyException me){
            System.out.println("Log failed: " + me.getMessage());
        }
    }
}
```

7.4 断言

```java
assert number>=0: "xxxxx"; // 错误提示语句
// 值为true继续执行，否则立即停止
```

```java
Scanner reader = new Scanner(System.in);
int sum = 0;
int cnt = 0;
while(reader.hasNextInt()){
    int x = reader.nextInt();
    assert x<=100 && x>=0: "The grade is not correct";
    sum += x;
    cnt++;
    if(cnt==5)
        break;
}
System.out.println("sum = " + sum + "  ave = " + sum*1.0/cnt);
```

```
PS D:\Github\2021study\Java> javac test.java
PS D:\Github\2021study\Java> java -ea test
1
101
Exception in thread "main" java.lang.AssertionError: The grade is not correct
        at test.main(test.java:57)
```

```java
java -ea filename #"启用断言"
```

8. 常用实用类

8.1 String类

```java
// 下面两者具有不同的引用
String s = new String("hello");
String t = new String("hello");
// 通过String对象创建
String Tom = new String(Jim);
// 通过字符数组创建
char [] a = {'1', '2', '3', '4', '5'};
String sa = new String(a, 1, 3); // <-> String sa = new String("234")
//下面两者有相同的引用
String s, t;
s = 'hello';
t = 'hello';
```

字符串的并置:

```java
String st = s + t;
```

关于常量池的内容，未看完。

常用方法

```java
//
String a;
int lena = a.length();
//
String Tom = "hello";
String Bob = "hello";
Tom.equals(Bob); //返回boolean
//
```

```
a.startsWith("sss");
a.endsWith("sss"); // 返回boolean
//
a.compareTo("xxx");
//
a.contains("xxx");
//
a.indexOf('xxx', num);
//
a.substring(int startrpoint, int endpoint); //[start, end)
//
a.trim();
```

字符串和基本数据的相互转化

```
String s = "876";
int x = Integer.parseInt(s);
//
String str = String.valueOf(123.12)
```

对象的字符串表示

```
// public String toString()方法
// 返回：类名@对象引用的字符串表示
```

8.2 StringBuffer类

```
StringBuffer s = new StringBuffer("hello");
s.append(" ok");
s.charAt(0); // 获得指定位置字符串
s.charAt(0, 'i'); // 替换字符串
s.insert(0, "hhh");
s.reverse();
s.delete(0,2);
s.replace(0,1,"h");
```

8.3 Data类与Calendar类

```
Date now = new Date(); // 获取当前时间
Date d = new Date(1000); // 公元后（北京是1970年01月01日8:00:01）1000ms
```

```
Calendar ca = Calendar.getInstance();
// 设置
ca.set(new Date());
// 取得
ca.get(Calendar.MONTH);
```

8.4 格式化

```
Date now = new Date();
String s1 = String.format("%ty-%tm-%td", now, now, now);
String s2 = String.format("%tF", now);
```

```
int x = 0;
String s = String.format("hhh %d", x);
```

9. 组件及事件处理

容器类、组件类

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;


class calFrame extends JFrame{
    Box boxv, boxv1, boxv2, boxh1, boxh2;
    JLabel lb1, lb2, lb3;
    JTextField tf1, tf2, tf3;
    JButton bt;
    MyListener lis;

    public calFrame(){
        setBounds(100,100,310,260);
        setLayout(new FlowLayout());
        init();
        setVisible(true);
        setTitle("MyWin");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    void init(){
        boxv1 = Box.createVerticalBox();
        lb1 = new JLabel("number 1: ");
        lb2 = new JLabel("number 2: ");
        lb3 = new JLabel("result: ");
        boxv1.add(lb1);
        boxv1.add(lb2);
        boxv1.add(lb3);

        boxv2 = Box.createVerticalBox();
        tf1 = new JTextField(10);
        tf2 = new JTextField(10);
        tf3 = new JTextField(10);
        boxv2.add(tf1);
        boxv2.add(tf2);
        boxv2.add(tf3);

        boxh1 = Box.createHorizontalBox();
        boxh1.add(boxv1);
        boxh1.add(boxv2);

```

```
        boxh2 = Box.createHorizontalBox();
        bt = new JButton("Mutiply");
        boxh2.add(bt);

        boxv = Box.createVerticalBox();
        boxv.add(boxh1);
        boxv.add(boxh2);

        add(boxv);

        lis = new MyListener();
        bt.addActionListener(lis);
    }

    class MyListener implements ActionListener{
        public void actionPerformed(ActionEvent e){
            double num1 = Double.parseDouble(tf1.getText());
            double num2 = Double.parseDouble(tf2.getText());
            tf3.setText(String.valueOf(num1*num2));
        }
    }
}

public class note{
    public static void main(String [] args){
        calFrame win = new calFrame();
    }
}
```

10. 输入输出流

10.1 对象流

```
import java.io.*;
try{
    //从文件读入对象
    File file = new File("out/file.txt");
    FileInputStream fileIn = new FileInputStream(file);
    ObjectInputStream objectIn = new ObjectInputStream(fileIn);
    Player [] playArry = (Player[])objectIn.readObject();
    objectIn.close();
}catch(IOException event){
    System.out.println("3 "+event);
}catch(ClassNotFoundException event){
    System.out.println("2 "+event);
}
```