JAVA Sum

变量/数据结构

变量

基本变量

> a.八大基本数据类型:
> 整数类型：byte，short，int，long
> 浮点数类型：float，double
> 字符类型：char
> 布尔类型：boolean

引用变量

1. String

创建指针指向对应的字符串，并且之后的操作都是围绕对这个指针的改变。每次对字符串的增删都可导致创建新的对象存储，然后让指针去指。
并且字符串和int等基本数字类型可以转换。
在java中，“”双引号表示String，但是‘’单引号表示char

遇到转译序列，参考如下：

表 4-5　转义序列

转义序列	名称	Unicode 码	十进制值
\b	退格键	\u0008	8
\t	Tab 键	\u0009	9
\n	换行符	\u000A	10
\f	换页符	\u000C	12
\r	回车符	\u000D	13
\\	反斜杠	\u005C	92
\"	双引号	\u0022	34

2. StringBuffer

```
StringBuffer sb="";
for (int i=0;i<100;i++){
        sb.append(i);
}
String s=sb.toString()
```

StringBuffer是动态的，可以支持修改但不生成新的对象。

3. Array

类似于python的列表，但是只能是同一种的数据类型并且确定长度后不能修改。

```
int[] array_ob=new int[100];//the number must be int
```

4. 自己构建的类

对象的数据和方法可以运用点操作符，通过对象的引用变量进行访问。

```
Circle objectRefVar = new Circle()
```

数据结构

Container,容器

> 容器是一个java所编写的程序，管理对象的生命周期和对象之间的关系

1. ArrayList

```
ArrayList<String> sites = new ArrayList<String>();
```

动态存储objects（区别于array），并且items在里面是遵守顺序的。

2. Iterator

java中的Iterator是一种接口，它定义了一种方法，允许容器（list，set等）的用户遍历容器的元素。

```
Iterator<String> it = notes.iterator();
```

迭代方法：
`it.hasNext()` 容器中是否还存在下一个元素
`it.next()` 输出容器中的下一个元素

3. MAP

包含treemap和hashmap，map类似于python的字典保存键值对。

```
HashMap <String,int> a = new HashMap <String, int>;
```

4. Set

集合

```
Hash<E> a = new Hash<E>();
```

程序逻辑

运算符

逻辑运算符

下表列出了逻辑运算符的基本运算，假设布尔变量A为真，变量B为假

操作符	描述	例子
&&	称为逻辑与运算符。当且仅当两个操作数都为真，条件才为真。	（A && B）为假。
||	称为逻辑或操作符。如果任何两个操作数任何一个为真，条件为真。	（A || B）为真。
!	称为逻辑非运算符。用来反转操作数的逻辑状态。如果条件为true，则逻辑非运算符将得到false。	！（A && B）为真。

关系运算

operator	description
= =	Equal
!=	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

自增运算符

```
int a = 1
a++
++a
```

a++,先调用a输出，再进行加1；
++a，进行加1操作后，再输出

IF 语句

```
if (a ) {.
}else if (){
}else { }
```

While loop

简单loop

```
While () {. }
```

do-while loop

```
do {
[loop body];
}while(condition);
```

Unfold loop

可以通过合并逻辑实现精简的loop，可以叫为这个

For loop

```
for (int i=0;i<100;i++){
          [p1].    [p2]  [p3]
}
```

[p1]=the initiation of the for loop
[p2]=the judgement condition
[p3]=the process after a loop

OPP面向对象编程

类

类Class定义同一类型的模版
类的封装思想encapsulation：
对象的值不应该能被直接修改或者访问，而是要通过method来实现

对象（object）

```
public class Puppy{
        public Puppy(){
        }
```

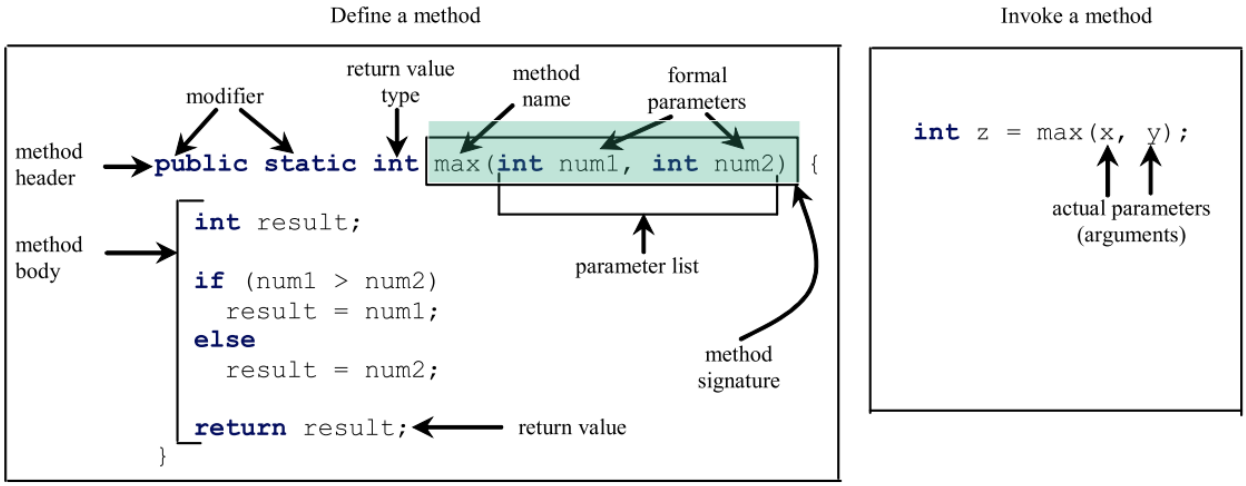
```
        public Puppy(String name){ // 这个构造器仅有一个参数: name
        }
        }

puppy dog = new puppy("dog")
```

在类中，创建对象和他的构造方法（ constructor）构造方法必须和类同名。**对象是类的实例，创建实例的过程叫做实例化。**

一个对象拥有： 1. 状态attribute，由数据域radius和当前的值来决定
2. 行为，由调用对象的方法决定
如果对象的数据域没有初始化的时候赋值，那么数据域会是null。

方法（method）

一般方法的构建方式：



形式参数和实参

former parameter： 不是实际的一个值，定义了这个参数的一些特点

```
char calculate(int num1)
```

这个的num1就是形式参数

real parameter：实际传入的值

`calculate(num)`

方法参数的传递

按值传递（passing value）

当传递基本数据类型给方法时，仅仅是传递了一个副本。方法中对传入的value的改变并不会改变参数原始值。

按对象传递（passing object）

当传递一个对象给方法，传递的是对象的引用，这时如果改变了对象的状态，那么原始对象的状态也会发生改变。

方法重写Overloading

同名的方法，但是不一样的参数。这是允许的，java编译器也会自动根据调用方法时候传入的参数，进行判断。

- Ambiguous Invocation：

> Ambiguous Invocation是指在一个程序中，调用一个函数或方法时，存在多个同名函数或方法，但参数列表略有不同。编译器无法自动判断应该调用哪个函数或方法，导致程序中的歧义性问题。这种情况通常称为“二义性调用”。

抽象类，Abstract class

抽象类只包含了申明的函数，但是不能定义或者生成实例

```
public abstract class a{

}
```

抽象类中存在抽象方法，abstarct method：

```
public abstract double computePay();
```

抽象方法不存在body，因此不能被直接用来调用，必须经过子类override

> 抽象类不一定含有抽象方法，但是含有抽象方法的类一定是抽象类（如果不是接口）。

接口Interface

```
public interface InterfaceName extends
BaseInterfaces;
```

特征：

1. 所有的方法必须是抽象方法
2. 没有构造器
3. 所有的方法都是public，并且所有的data members都是`public static final`

静态变量+方法

静态变量

如果想让一个类的所有实例共享数据，就要使用静态变量。当对其中一个实例的这个静态变量进行修改，就会对全体的静态变量进行修改。

```
static int a = 1;
```

静态方法

静态方法是指无需创建实例，就可以调用的函数。调用时，类名.方法名（）。当然对象名.方法名（）也是可以的。

```
static int num;
static int getNum() {
        return num;
}
```

NB. 非静态方法既可以访问静态数据成员又可以访问非静态数据成员，但是，静态方法只能访问静态数据成员。

常量

类中的常量被所有的对象所共享，因此常量应该申明为final static

```
final static int i = 1;
```

> 关于`final`修饰符：当final修饰对象时，对象引用位置不能再改变，但是可以修改其内部的属性

继承Inheritance

Advantages of inheritance

- Avoiding code duplication
- Code reuse
- Easier maintenance
- Extendibility

一般类继承

本质是对某一批类的进一步抽象。只存在单继承，被继承类被称为父类（superclass），继承类被称为子类（subclass）。子类获得父类的参数，方法。同时子类拥有自己的内容。

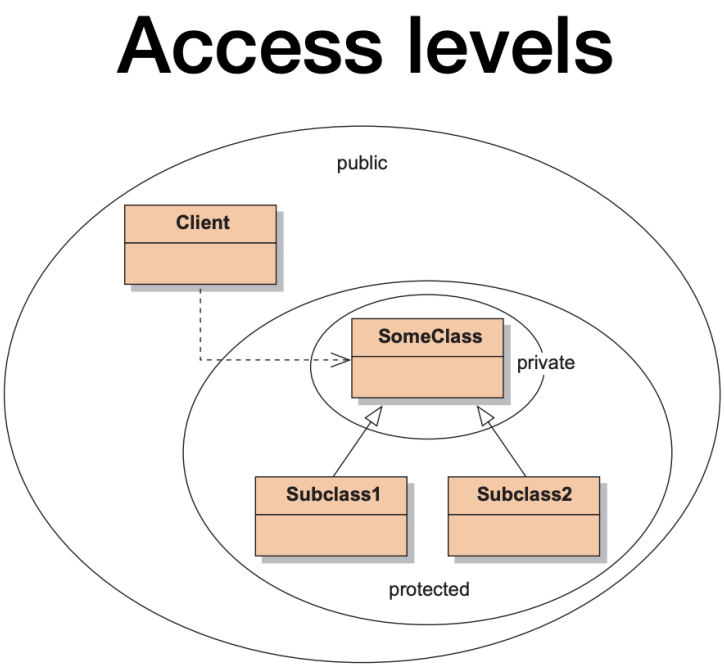
关键字：`extends`

> 子类的构造器和父类的构造器是有区别的；子类必须在构造自己前，先构造一个父类的构造器。即调用super（）

```
public class Animal {
    private String name;

    public Animal(String name) {
        this.name = name;
    }
}

public class Dog extends Animal {
    private String breed;

    public Dog(String name, String breed) {
        super(name); // 调用父类构造器
        this.breed = breed;
    }
}
```

如果父类中的仅有的构造器是有参数的，子类的构造器必须依照这个规定，调用super。
如果父类存在不需要参数的构造器，显式写出super（）或者不写都可以（编译器自动加入）。

继承的修饰符有：private，protected，public，关系大概如下



Abstract Class继承

继承方法和一般类继承一样，但是如果子类是一般类，就必须把父类中的所有的抽象方法全部重写。

Interface 继承

```
class ClassName implements interfaces;
```

`implements` 就是为了让其他的class能够访问这个接口，类似于继承。
接口也可以继承另一个接口；一个类可以implement多个接口

> difference between interface and abstract class

1. 抽象类允许具体方法体；接口必须全是抽象方法
2. 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是 **public static final** 类型的。
3. 一个类只能继承一个抽象类，而一个类却可以实现多个接口。

4. 继承了接口的类必须实现接口里面的所有方法；否则该继承的类必须定义为抽象类

多态

一个子类继承了父类之后，他创造的对象即使属于子类的，也是属于父类的。
体现为，父类的引用变量可以指向子类对象。

static type

```
superclass variable = new superclass();
```

dynamic type

定义时候的类型和实际运行的类型不一样

```
superclass variable = new subclass();
```

覆写Overriding

子类父类都定义同一个函数，同样的参数，但是子类可以修改这个函数，调用的时候，如果对象是子类 就直接调用子类的方法。

如果子类对象想要调用父类的被覆写的方法，可以用super.method()

Upcast

子类对象被用作父类对象

```
superclass variable = new subclass();
```

这个其实就是一个比较好的例子，不需要面对子类类型，使用父类的操作就可以完成任务。variable是父类的变量，但是指向的是实例

化的子类对象。
没有特别的修饰符，直接调用方法将会是子类的方法。

Downcast

可以将父类对象转成子类对象。
必须使用强制转换的形式

```
subtype down = (subtype) superclass
```

健壮性

Error

通常是指java运行环境或者虚拟机中的问题，通过代码通常不能修复

Exception

exception 是程序在运行过程中出现的问题，可以被开发者捕捉和处理。
Exception包含两大类：检查型异常和非检查型异常（checked/unchecked），检查型异常必须显式的处理：

1. throws关键字
2. try-catch板块

Throw

抛出异常，如果代码运行到某种情况下就不能继续执行，这个时候就可以使用throw抛出异常。

```
public void checkNUm(int num){
if (num<0){
        throw new IllegalArgumentException("Number
```

```
must be positive")
}
}
```

Throws

Throws用在方法后，声明使用指定的方法可能抛出的异常

```
public void withdraw(double a) throws xxxException
```

Try-catch statement

```
try
{
    System.out.println (Integer.parseInt(numString));
}
catch (NumberFormatException exception) {
    System.out.println ("Caught an exception.");
}
finally
{
    System.out.println ("Done.");
}
```

异常的继承

当子类继承一个有异常处理块的父类时，当覆盖一个函数的时候，子类不能声明抛出比父类版本更多的异常，并且不能抛出父类的异常的子类异常。因为子类对象可能被当作父类对象看待。
子类的构造函数中，必须声明父类可能抛出的全部异常，并且可以加上新的异常

流Stream

File writer

```
FileOutputStream out = new FileOutputStream("a.dat")
```

流过滤器

```
DataOutputStream out = new DataOutputStream(
                new BufferOutputStream(
                        new
FileOutputStream("a.dat"))
);
```

用以读写二进制方式表达的基本数据类型的数据。

文件流

利用reader和writer

- Writer

```
PrintWriter out = new PrintWriter(
   new BufferedWriter(
      new OutputStreamWriter(
         new FileOutputStream("output.txt")
      )
   )
);
```

- Reader

```
LineNumberReader lin = new LineNumberReader(
   new BufferedReader(
      new InputStreamReader(
         new FileInputStream("output.txt")
      )
   )
);
try (lin) {
  while ((line = lin.readLine()) != null) {
              System.out.println(line);
  }
} catch (IOException e) {
  e.printStackTrace();
}
```

Format

- For PrintWriter
  - public PrintStream format(String format, Object... args)
  - public PrintStream printf(String format, Object... args)
- For String
  - public static String.format(String format, Object... args)

`%[flags][width][.precision]conversion` 这里的flag是一些对后面的操作的事先说明

Flag	Meaning
-	Left justify
+	The result will always include a sign
(space)	The result will include a leading space for positive values
0	The result will be zero-padded
,	include locale-specific grouping separators
(	enclose negative numbers in parentheses

细节知识点

命名细节（驼峰命名法）：

1. 大小写敏感
2. 类名：首字母大写，驼峰
3. 方法名：首字母小写，驼峰

Java程序运行方法

Java source code --> Byte code file --> 3. JVM（java virtual machine） --> OS

IPO moel

I : input
P: process
O: output

所有的class都是继承自Object class

分离Business和presentation：business处理IO data，presentation只负责将结果输出为人类可读的形式