Java中的包 package.

定义包 package 必须放在 source code 的第一行：

package com.imooc.music.MyClass/Music 类

package com.imooc.movie.MyClass/Music

系统中的包：java.功能.class

java.lang...
java.util...
java.io...

包的引用 import com.imooc.music.MyClass/Music

访问修饰符 —— 范围

	本类	同包(一个路径下)	子类	其它
private	√			
default	√	√		
protected	√	√	√	
public	√	√	√	√

this —— 当前对象

this.attribute

this.method

Inner class：更好的封装，隐藏在外部类中，不允许同一个包中的其它类访问

- Inner class 可以直接访问 Outer class 中所有 data，包括 private 数据

分为① 成员内部类

② 静态 —— static

方法 ——

匿名 ——

如果 Inner Outer 中出现了重名变量或方法

Inner 默认访问自己的

如要访问外部类，用 this.

外部类.对象名 = 外部类对象.new 内部类()

4. 外部 class 不能直接使用 Inner 中的成员和方法.

要先创建内部类对象，通过其访问方法

① 成员 Inner class.

```
public class Outer{
    // Outer class's private attribute
    private String name = "java outer";
    // Outer class's member attribute
    int age = 20;
    public class Inner{
        String name = "inner java";
        public void show(){
            ---- // 可以直接访问 Outer class
        }
    }
    public static void main(String[] args){
        Outer o = new Outer();
        Inner in = o.new Inner();
    }
```

Inner 访问外部类的成员用 Outer.this.age/name

③. 静态内部类：a. 不依赖于外部类的非静态成员

可以直接 new 外部类().成员的方式访问外部类的……

b. 如果外部类的static与Inner类的name相同

就近原则：静态成员 一般指外部类……

不相互直接 静态成员访问

c. 可以在静态创建，不需要先创建外部类 Inner class name = new Inner();

④ 方法内部类：内部类定义在外部类的方法中

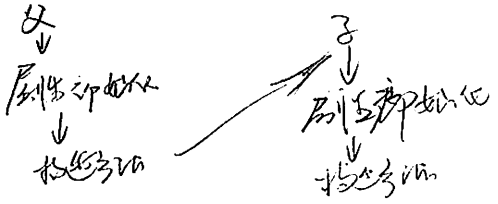
此时只在该方法中使用 不使用该方法的时候 static修饰

继承：子类拥有父类的所有Attribute+methods 除了private修饰的

class 子 extends 父{ }

子类对父类的父类不满意 可以重写方法（会优先调用子类）

注：返回值类型/方法名/参数类型个数均要相同



final —不可修改 { 修饰class：不允许被继承
—— method：不允许被重写
—— attribute：必须有初始化值，不能修改 //常量

super：在对象内部，代表父类对象 super.age;
super.eat();

子类构造时调用父类的构造（super 隐式）
（或显式写一行 super()）
子类 (String name)

如果父类中没有无参的构造方法，子类必须显式super()，否则报错

Object 类：无 extends 就是所有类的父类

Object method：1) toString()：返回对象的 hashcode（地址值）字面量为 toString() 表示的对象的属性

Source 中自动生成 ← @Override
public String toString() {
return "Dog[age=" + age + "]";
}

(2) equals() 方法：比较的是对象的引用是否指向同一块内存地址

Dog dog = new Dog()

以上比较引用

生活中的判等 < 同一个 / 同样的 > circumstances

创建的 objects 即使 attributes 一致 也是不同的 引用 因为不同的存储空间

↓ Override.

Source 中自动生成

多态

1) 引用多态 { 父类的引用可以指向本类对象 Animal obj1 = new Animal();
父类的引用可以指向子类的对象 Animal obj2 = new Dog();
Dog obj3 = new Animal() ✗

2) 方法多态 { 创建本类对象时，调用本类方法 obj1.eat();
创建子类对象时，调用为子类重写或继承的方法 obj2.eat();

父类引用不能调用子类特有的方法，子类调用

引用类型转换：{ ① 向上（自动/隐式）小 → 大（自动）无风险

② 向下（强制类型转换）：大 → 小 有风险（编译通过）

③ instanceof 运算符：来判断引用对象的类型，避免转换的异常发生

① Dog dog = new Dog();
Animal animal = dog; //①
Dog dog2 = (Dog) animal; //② 强制
Cat cat = (Cat) animal; // ② 转换，编译时 Cat 类型；运行时 Dog 类型

③ 为了避免问题，最好使用 instanceof 关键字（向下转换时）

if (animal instanceof Cat) {
Cat cat1 = (Cat) animal;
} else { ...

抽象类 Abstract.

① 子类知道子类必须有哪些 method，但不知道如何具体执行实现
Abstract class 约束子类必须有xx methods，即不关心如何实现
② 从多个有相似特征类抽象出一个模板，约束有哪些特征

abstract 定义抽象 class. or 定义抽象 method (没有实现)
有 abstract 方法的类必须为 abstract 类
abstract 类中可以有普通方法.
抽象类的构造器不能直接创建. 可以定义引用变量.

```
public abstract class Telephone{
    public abstract void call();
}

public class Smartphone extends Telephone{
    @Override
    public void call(){
        ------;
    }
}

static void main(...)
    Telephone tel = new Smartphone();
```

接口：特定类中必须实现的方法
规范：约束类
⇒ interface

vs. 类：单父类
多父类的继承

[修饰符] interface 接口名 [extends 父接口1, 父接口2]{
    0~多个常量定义 (public static final → 可以)
    0~N个 abstract 方法的定义 (public abstract → 可以)
}

一个class可以实现 ≥1个接口. 用 implements
vs. 只能继承一个父类

[修饰] class 类名 extends 父类 implements 接口1, 接口2...
{
    除了实现继承的抽象方法 + 接口的方法
}

↳ 与匿名内部配合：匿名内部class → 无名字的 inner class

```
IPlayGame ip3 = new IPlayGame(){
    @Override
    public void playGame(){
        ------
    }
};
ip3.playGame();
```

```
new IPlayGame(){
    @Override
    public void playGame(){
        ...
    }
}.playGame();
```

Java中的异常

Throwable
- Error（系统错误）：无法处理
  - 内存溢出
  - VirtualMachineError
  - ThreadDeath
- Exception：编码、环境、用户操作输入出现问题
  - RuntimeException 非检查异常（JVM自动抛出，自动捕获）
    - 采用逻辑去规避 + 辅助try-catch
    - eg: 引用了一个空对象的attributes or methods
    - 数组访问越界
    - 错误的类型转换
    - 算术异常（eg. int/0）
  - Check Exception →（手动添加try catch 及处理）
    - IOException
    - SQLException

NullPointerException

```
String str = null;
System.out.println(str.length());
```

ArrayIndexOutOfBoundsException

```
int[] ary = {1,2,3};
for(int i=0; i<=3; i++){
    System.out.println(ary[i]);
}
```

ClassCastException

```
class Animal{
}
class Dog extends Animal{
}
class Cat extends Animal{
}
public class Test{
    ......
    {
        Animal a1 = new Dog();
        Animal a2 = new Cat();
        Dog d1 = (Dog)a1;
        Dog d2 = (Dog)a2;
    }
}
```

ArithmeticException

e.printStackTrace(); 只是简单打印，具体如何处理要根据程序+异常来定

善后处理的finally：去处理、释放占用的资源

```
finally{
    //最终要执行的代码
}
```

处理Exception

try-catch & try-catch-finally

```
try{
    //一些会抛出异常的方法
} catch (Exception e){
    //处理该异常的代码块
}
```

（处理该异常的代码块：发出警告；进行错误日志记录）

如try中方法确实抛出异常，则：
- 该方法终止执行；
- 程序控制权交由catch中异常处理程序处理；
- catch外面的一样执行

多个catch：多个catch后面再加一个catch(Exception)处理 ✓ 扫尾器

① 顺序问题：一定要按照先小后大（先子类后父类catch）

try中抛出很多类型的异常

```
Scanner input = new Scanner(System.in);
try{
    System.out.print("请输入第一个数:");
    int one = input.nextInt();
    int two = input.nextInt();
    System.out.println("两数相除为:" + one/two);
} catch (InputMismatchException e){
} catch (ArithmeticException e){
} catch (Exception e){ ...
```

Java-String:

int length();

无匹配返回 -1 <
int indexOf(int char); // char 第1次出现的位置
int indexOf(String str); // str — — — —

int lastIndexOf(int char); · char 最后一次
int lastIndexOf(String str);
String substring(int beginIndex); 从 beginIndex 开始到最后的 substring
String substring(int beginIndex, int endIndex); // 包括 beginIndex 字符 不包括 endIndex
String trim(); 返回除前后空格的 String
boolean equals(Object ob); 与 "==" 有区别; "==" 表示内存中首地址相同
== 严格比较 character
String toLowerCase();
String toUpperCase();
Char charAt(int index); 一获取所在 index 的字符
String[] split(String regex, int limit); 分割为子字符串,返回数组
byte[] getBytes() // 转为 byte 数组

Java 中的异常抛出

throw 语句
throws —— 声明要抛出某种类型的异常:

public void 方法名(参数) throws 异常列表{
// 调用会抛出异常的方法
or throw new Exception();
}

用 try-catch 处理
or 抛出给更上一层的去处理

自定义异常

```
public class DrunkException extends Exception{
    public DrunkException(){ }
    public DrunkException(String m){
        super(m);
    }
}
```

异常链: 将 catch 的异常包装成一个新的异常, 在新的异常中添加对原始异常的引用 再将新的异常抛出:

example.

新的异常中含有原始异常的信息:

```
public void test1() throws DrunkException{
    throw new DrunkException("......");
}
public void test2(){
    try{
        test1();
    }catch(DrunkException e){
        ① < RuntimeException newExc = new RuntimeException("...");
            newExc.initCause(e); // 引用原始异常
        throw newExc;
    }
}
```

② RuntimeException newExc = new RuntimeException(e); 直接做参

⑥

StringBuilder ： String不可变性，节约临时变量 String str = "" = new String("...");

StringBuffer vs. StringBuilder ：优先 when 创建一个内容可变的字符对象时
thread安全　　　　性能较高

修改内容，但并未创建新对象（与String区别）：
- StringBuilder append(arg)：加到末尾
- StringBuilder insert(位置, arg)
- String toString()：将 StringBuilder 变为 String
- int length()

Java中的包装类：

基本类型	包装类
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

这些包装类提供了两大类方法：
① 将本类型与其他基本 type 进行转换的方法
② 将字符串与本类型及包装类互相转换

ex：Integer类　Constructor：
- Integer(int value)　　Integer n = new Integer(5)　→ 5
- Integer(String s)　　Integer n = new Integer("8");　→ 8

Return type	method
byte	byteValue()
double	doubleValue()
float	floatValue()
int	intValue()
long	longValue()
static int	parseInt(String s)：将String转换为int
static Integer	valueOf(String s)：将字符串转为 Integer

一、基本 type 与包装类：装箱与拆箱（可以自动完成）

ex：
```
int i = 10;
Integer x = new Integer(i); // 手动装箱
Integer y = i; // 自动装箱

Integer j = new Integer(8);
int m = j.intValue(); // 手动拆箱
int n = j; // 自动拆箱
```

二、基本类型与字符串之间的转换

基本类型 → String：（int x = 10;）
① 包装类的 toString()：String str1 = Integer.toString(x);
② String类的 ValueOf()：String str2 = String.valueOf(x);
③ 用一个空字符 加上基本类型 → 得到对应的String：String str3 = x + "";

String → 基本类型：（String s = "8";）
① 包装类的静态 parse__(String s)：int d = Integer.parseInt(s);
② 包装类的静态 valueOf(String s) 自动拆箱：int y = Integer.valueOf(s);

- Date 与 SimpleDateFormat 表示时间

* java.util.Date: 获取当前时间: Date d = new Date(); //无参表示当前时间。

System.out.println(d);

显示: 星期 月 日 Time 时区 年

Wed Jun 11 09:22:30 CST 2014

↓ 改变形式

对日期进行格式化 ← * java.text.SimpleDateFormat;

① 将日期→指定格式的文本

② 文本→日期

① format() 日期→文本格式

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String today = sdf.format(d);
```

② parse() 将文本→日期

```
String day = "2014年02月14日 10:30:25";
SimpleDateFormat df = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");
//parse()方法，将String→日期
Date date = df.parse(day);
```

⇓

会可能发生 转换异常. ParseException (java.text.ParseException)

- Calendar类 (更推荐用于日期 时间的处理)

java.util.Calendar 是 abstract class, 可 call getInstance() 获取一个Calendar对象

初始化为当前时间: Calendar c = Calendar.getInstance();

用get方法获取:

```
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH)+1; // 0表示1月
int day = c.get(Calendar.DAY_OF_MONTH);
int hour = c.get(Calendar.HOUR_OF_DAY);
int min = c.get(Calendar.MINUTE);
int sec = c.get(Calendar.SECOND);
```

转为Date对象

Date d = c.getTime();

获取当前毫秒数 → 用于计算时间

Long time = c.getTimeInMillis()

- Math: java.lang 均为静态方法 (直接调用类.方法)

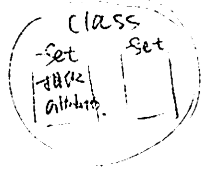
Return type	Method	
long	round()	返回四舍五入值
double	floor()	小于参数的最大整数
double	ceil()	大于参数的最小整数
double	random()	[0,1)之间随机数 (int)(Math.random()*99) → [0,99)整数

集合框架 — 容器

某个类，具有很多相似的属性 → 放在一个(容器)中，整体作为一个属性。

便于存放任意数量的具有共同属性的对象

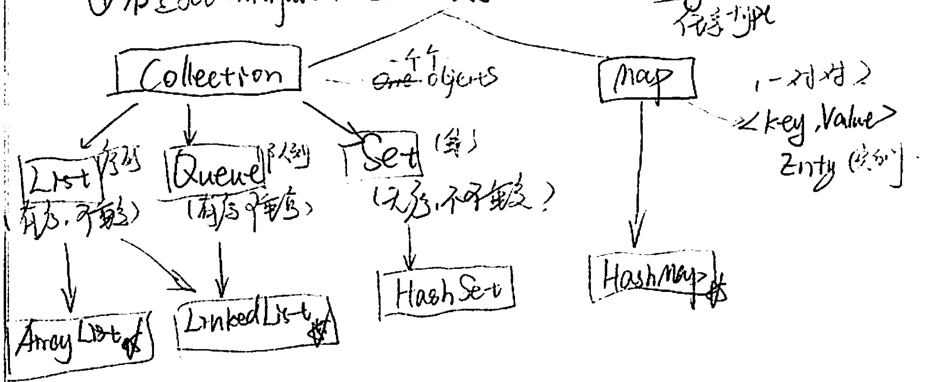
集合的作用



↓

在class内部
① 对数据进行组织
② 简单快速的搜索大数量的条目
③ 有些Set提供排列有序的元素，可以快速插入/删除
④ 有些Set interface，提供映射关系，通过key → 找到对应的唯一对象
任意类型

集合 vs. 数组
① 数组 长度固定
集合 长度可变
② {数组 — 数组下标 (int)，通过选择
{集合 — 映射，一一对应，任意type



JDK API中…
add …
remove …
…

一、Collection 接口
→ List Set Queue 的父接口
→ 定义了一些可用于 List, Set, Queue 的方法 — 增删改查

① List Interface 及实现类：ArrayList
序列：有序，可重复，可以精确控制每个元素的插入位置 或 删除某个位置的元素
ArrayList：数组序列 — 实现类，底层由数组实现

添加add　任何type的Object存到集合中都会变成Object类型，取出时需要类型转换 (ClassName)

方法一
List/ArrayList　al1 = new ArrayList();
al1.add(cr1) 元素类型，同一类型一个List中
element
(Class) al1.get(0);
index

方法二 al1.add(index, element);
插入位置，其他元素位置后移
可以访问的位置不能 <0，或 >此时List的容量
否则报错

方法三
创建一个数组
Class [] name = { new Class( ), new Class };
al1.addAll (Arrays.asList (name));
collection实例：将Array变为ArrayList

方法四：
al1.addAll (index , Arrays.asList (name));
插入数组的位置

查询
* 将元素从 List 中取出
List.size() 取长度值

① get方法
遍历每个元素：for (int i=0; i< List.size(); i++){　, 不能 [illegible]
② iterator<E> iterator() method　:
返回 iterator，通过 iterator 来遍历
Iterator ite = List.iterator();
while (ite.hasNext()){
Class cr = (Class) ite.next();
}

③ For each: for (Object obj : List){ Class cr = (Class)obj; … }

修改 List中的elements:

List.Set(int index, Class element);

删除、List中的element: remove(int index)
boolean remove(Object o) → set(index) 和 remove(obj)
boolean removeAll(Collection c)
Arrays.asList(Array)

向List中加入与其他不一样的元素却不是一个类型的元素？

泛型 → 只能添加特定类型的对象，不能加其它的
不是基本类型 → 包装类
public List<Course> names;
指定 class
new ArrayList<Course>();
以后在 add中就不能加其它class了
For each 语句 for (Course cr : courses) { ... }
不是 Object

- HashSet 无序不可重复
  - add
  - addAll
  - remove
  - removeAll
  - size()

- 遍历 Set 中的每个 element，只能用 foreach 和 iterator，不能用 get()，出来的顺序不一致（因为无序 set）
- 重复添加 无论多少次，最后也只保存了第一次添加的

HashMap - Map:

Map Interface:
- 提供了一种映射关系，其中的元素以 <key - value> 形式存在。通过 key 找到 value.
- (key - value) 以 Entry 类型的对象实例存在
- key 不可重复，value 可以
- 每个 key 只能对应一个 value，但同一个 value 可能有多个 keys 指向
- 提供了返回 key值集合 (Set keySet())，value值集合 (Collection values())，以及 Entry（键值对）Set 的方法 (entrySet() → Set<Map.Entry<K, V>>)
- Map 支持泛型: Map<K, V>

HashMap: Entry 对象是无序的

- remove: map.remove(K)
- entrySet(): map.entrySet() 返回的是一个 entry 的 set
  Set<Entry<K,V>> es = map.entrySet()

- key 值和 value 值均可为 null，But 一个 HashMap 只能有一个 key 为 null 的映射

增删改查

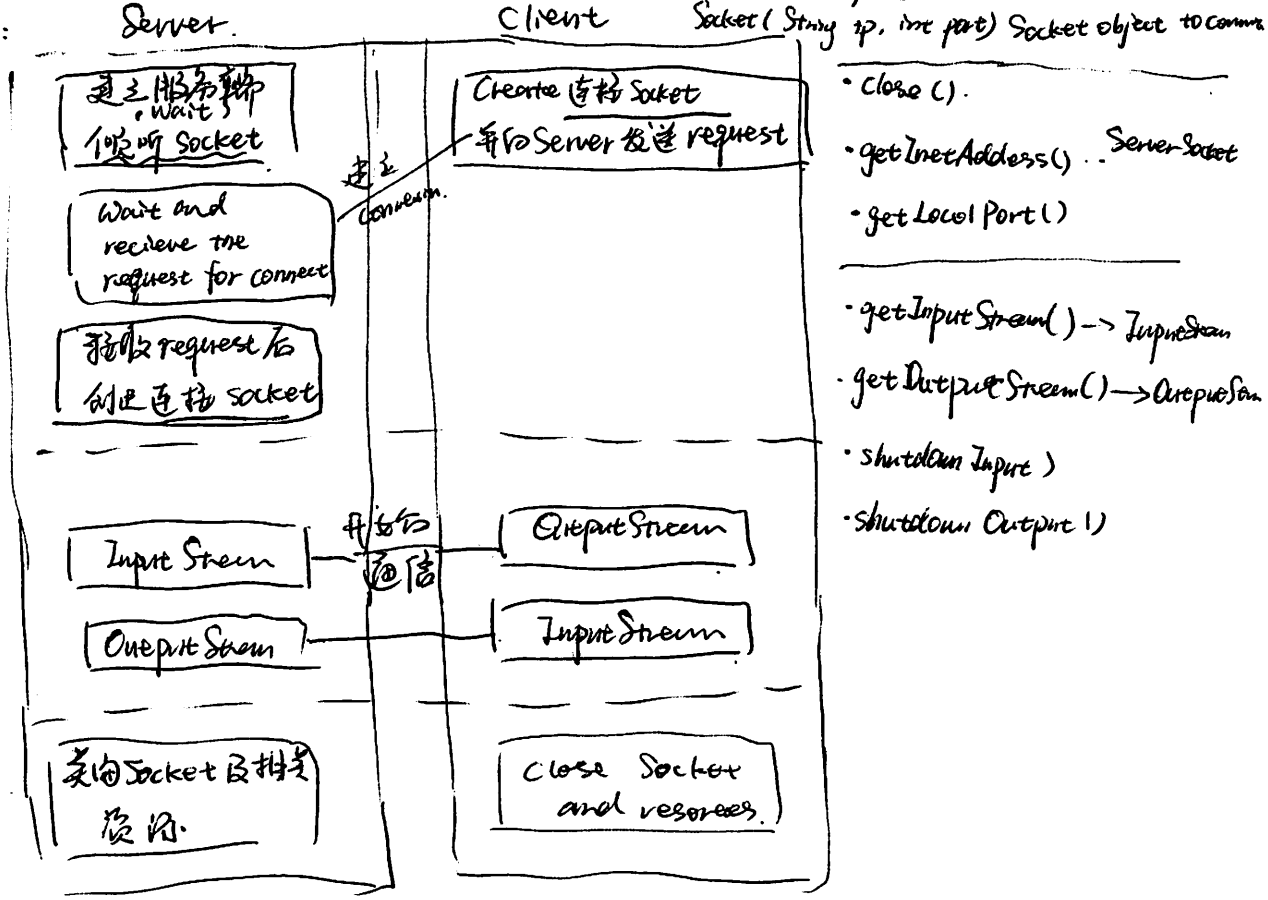
- add: map.get(key); 如果存在，返回 key 对应的 value，or 不存在返回 null
  m.put(key, value);
  Object

- print all values in the Set:
  ① Set<K> keyset = m.keySet();
  ② for (K ki : keyset){
       V vi = m.get(ki);
       if (vi != null){ .... }

Socket 通信 — TCP protocol (以字节流的方式发送data)

Java中2个class: Client — Socket — java.net. Constructor: ServerSocket(int port)
Method: accept(): 阻塞当前线程, 等待 client 连接. 一旦连接, create a
Server — ServerSocket. Constructor: Socket(InetAddress addr, int port)

Socket model:



Steps:
1. Create ServerSocket + Socket.
2. Open 连接到Socket 的 InputStream + OutputStream
3. According to TCP, write/read in Socket
4. Close Input/OutputStream and Socket

Example: 实现用户登陆.

Client ⟶ (①) Infor 发送给 Server (接收 client info, 进行响应) ⟵ 响应 Server

① 登陆 request, 包含 username + password.
② 回馈响应, 显示 info.

Server:
1. Create a ServerSocket object, 绑定监听 port
2. 通过 Accept() 监听 Client request
3. After connection, 通过 输入流, read client info
3. 通过 OutputStream 响应客户端
4. 关闭各 Stream 及 Socket.

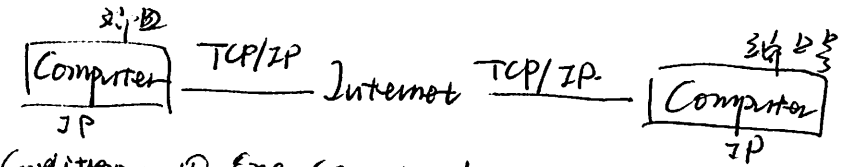
Client:
1. Create a Socket object, 绑定到 Server (IP+port)
2. After connection, 通过 OutputStream 向 Server 发送 request info 等
3. 通过 InputStream 获取 Server 的响应信息
4. 关闭 Stream + Socket.

Java Socket

1. Internet 基础
2. InetAddress Class
3. URL
4. TCP 编程
5. UDP 编程

1.

2 computers communicate with Internet.



Conditions: ① One computer has one IP address

② common language (协议) TCP/IP

③ 每台主机有指定的端口号：区分不同 apps 的通信

② TCP/IP protocol: Internet Protocol / Transmission Control protocol 以TCP, IP为基础的不同层次上多个协议的集合

TCP/IP model:

5	应用	HTTP 超文本传输协议 FTP 文件传输 SMTP: 简单邮件发送 Telnet: 远程登陆...
4	TCP/IP 传输	
3	网络	
2	数据链路	
1	physic	网卡 网线 ...

① IP address: 每台机器必须有一个唯一的标识 — 用于标识机器通信

有格式/规范: IPV4: 32 bit 二进制 198.168.0.1

③ 端口号: 区分不同的应用程序 (一台主机有多个应用程序)

↳ 1个应用有一个端口号

2. 端口号范围为 0~65535, 其中 0~1023 为系统保留.

3. IP + 端口号 ⇒ Socket, Socket是网络上运行的程序之间 双向 通信会话的端点 (TCP+UDP的基础)

4. 端口号: http: 80, ftp: 21 telnet: 23

Java中的网络支持: 针对不同层次, 4大类:

1. InetAddress: 用于标识网络上的硬件资源 (IP address)
2. URL: 统一资源定位符: 通过URL可以直接读取/写入网络上的数据
3. Sockets: 使用 TCP 协议实现 Communication 的 Socket 相关 class
4. Datagram: UDP 协议, 将数据保存在 数据报中, 发送数据报

2. InetAddress 类 — IP.

~~无Constructor~~ 不能new return type: static InetAddress 静态

~~getHost~~ getLocalHost(), getByAddress(byte[] addr), getByName(String host);

① getLocalHost();

```
InetAddress addr = InetAddress.getLocalHost();
```

直接 print addr: → 主机名/IP.

addr.getHostName() → addr.getHostAddress(); 打印数字组: Arrays.toString(bytes)

Or 返回 address 以 bytes Array 的形式: byte[] bytes = addr.getAddress();

② 根据机器名获取 InetAddress Object.

InetAddress addr2 = InetAddress.getByName("host");

③ 根据机器 IP → InetAddress Object;

InetAddress addr3 = InetAddress.getByName("IPadress"); (1.0.11...)

④ 根据 - - IP - - - getByAddress(Array bytes[]);

3. URL: Internet上某一resource的地址.

2 parts: 协议名 + : + 资源名.
http www....

java.net package中 提供了 URL class 表示URL

Constructor:
URL url = new URL("网络的 string"); ← 会有异常 (try/catch) /index.html? username=Tom #test (参数, 锚点)
URL url2 = new URL(url object, "加在后面的 Source");

Methods:
url2.getHost(); → 返回主机名
url2.getProtocol(); → protocol
url2.getPort(); → port
getPath(); → index.html
getFile(); → file name: → Path + ?后#前
getRef(); → 相对路径 → #后 锚点
getQuery() → 查询字符串 (参数)

* 未指定port号, 则使用默认 port (http → 80), 而 getPort() 返回值为 -1 (使用了default port)

通过URL读取网络上的资源.(网页内容)

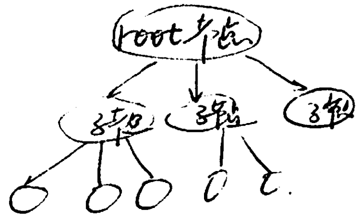
openStream(). → 获取资源的输入流 → return type: InputStream

通过InputStream → 读取/visit 网页上的资源.

```
// 返回字节输入流
InputStream is = url.openStream(); ← 会有异常 try/catch
// 字节输入流 → 字符输入流
InputStreamReader isr = new InputStreamReader(is, "utf-8"); // "gbk" 识别的字符集
// 为字符输入流添加缓冲
BufferedReader br = new BufferedReader(isr);
// 读取数据 line by line
String data = br.readLine();
while (data != null){
    data = br.readLine();
}
// close all:
br.close();
isr.close();
is.close();
```

一. XML : ___.xml

1. 存储结构：树形结构 — 倒长的树形.



<??> 中是版本信息、编码

← <?xml version="1.0" encoding="UTF-8"?> (encoding: attribute)

<bookstore>

<book id="1"> → 也可以有子节点

<name>121</name>

<price>...</price>

</book>

</bookstore>

二. Why XML?

use same <XML>

1. 程序之间的通信 (different apps)
2. 不同platform之间的通信 (different OSs)
3. 不同平台的数据共享 (网站 <-> 手机)

三. Java Program中如何获取XML文件的内容? ⟶ 解析 (obtain) { Node Name, Node Value, Attributes < name, value.

Dertn -

4 Methods: DOM, SAX, DOM4J, JDOM (额外下载包)

在Java中保存XML的结构(层级关系)

① DOM方式解析XML

· 准备工作:

. Create a DocumentBuilderFactory Object

→ DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

· Create a DocumentBuilder Object.

⇒ dbf.newDocumentBuilder();

```
try {
    DocumentBuilder db = dbf.newDocumentBuilder();
} catch (- - - - e) {

};
```

· Use DocumentBuilder's parse Method. parse(String); (重载 overload; String: 绝对/相对路径 url)

又会有try-catch异常. 选择抛出

Return type

⇒ Document document = db.parse(url); ("....xml")

Document: org.w3c.dom

· DOM 解析 Attributes name + value.

- 打印 Attributes Name + Value of Each Node.

// obtain all nodes use: Document. getElementsByTagName("Node Name");
return type: NodeList

⇒ NodeList bookList = document.getElementsByTagName("book");

// Go through all nodes in NodeList: return list的长

for (int i=0; i < booklist.getLength(); i++) {
从0开始

// use NodeList.item(index) method to obtain each node
return type: Node

Node book = booklist.item(i);

// Go through every attributes of the Node.

// Node.getAttributes() return type: NamedNodeMap

NamedNodeMap attrs = book.getAttributes();

// go through every attribute to obtain attribute name + value.

for (int j; j < attrs.getLength(); j++){

Node attr = attrs.item(j);

String name = attr.getNodeName();

String value = attr.getNodeValue();

}

NodeList
↓
Node
↓
NamedNodeMap
A map for all attributes
↓
Node — attribute

Do NOT know how many/attributes short in the Node

↓ if know the only one attribute name

强制转换 Node → Element.

Element book2 = (Element) bookList.item(i);

String attrValue = book2.getAttribute("id");

Node 类型
Type
Element

3. Named Constructor

二. 打印 Node 的 subNodes

Array & Strings

1. Hash Tables

```
public HashMap<Integer, Student> buildMap(Student[], students) {
    HashMap<Integer, Student> map = new HashMap<Integer, Student>();
    for (Student s: students) map.put(s.getId(), s);
    return map;
}
```

2. ArrayList (Dynamically Resizing Array); providing O(1) access.

typical implementation: when a vector is full, the array doubles its size. Each doubling takes O(n) time, but happens so rarely that its amortized time is still O(1).

```
public ArrayList<String> merge(String[] words, String[] more) {
    ArrayList<String> sentence = new ArrayList<String>();
    for (String w : words) sentence.add(w);
    for (String w : more) sentence.add(w);
    return sentence;
}
```

3. StringBuffer / StringBuilder

Q: What's the running time of this code?

```
public String makeSentence(String[] words) {
    StringBuffer sentence = new StringBuffer();
    for (String w : words) sentence.append(w);
    return sentence.toString();
}
```

⇒ O(n²). n is the number of letters in sentence. Each time append a string to the sentence, you create a copy of sentence and run through all letters in sentence to copy them over, loop at least n times ⇒ O(n²)

StringBuilder can avoid this.

Test
- Extreme cases: 0, negative, null, maximums, etc
- User errors: user passes in null or negative number
- General cases: Test normal case

Five algorithm approaches: "mixed and matched"

1. Examplify: write out specific examples → figure out general rule.

Ex: Given a time, calculate the angle between the hour and minute hands.

Approach: an example: 3:27

Minute angle: $\frac{27}{60} \times 360°$ → $\frac{m}{60} \times 360°$ from 12 o'clock

Hour angle: $\frac{3}{12} \times 360° + \frac{m}{60} \times 360° \times \frac{1}{12}$ → $\frac{(h \% 12)}{12} \times 360 + \frac{m}{60} \times \frac{360°}{12} = (h \% 12) \times 30 + \frac{m}{2}$

Angle between: (hour angle − minute angle) % 360

2. Pattern Matching: consider what problem the algorithm is similar to → figure out if you can modify the solution to develop an algorithm. for this problem

Ex: A sorted array has been rotated so that the elements might apear 3 4 5 6 7 1 2 ...
How to find the minimum element?

Approach: Similar problems: ① find the minimum element in array
② Find a particular element in an array (binary search)

⇒ sorted but rotated: → Reset = minimum →

3 4 5 6 7 1 2

1 6 7 1 2 3 4

Binary search: first vs. middle: in the range ↗, so reset point after middle.
→ binary if Left < Right, not include reset
if Left > Right. Contains reset point

3. Simplify & Generalize: change a constraint (data type, length, size ...) to simplify the problem. → have an algorithm for the "simplified problem"

Ex: A ransom note can be formed by cutting words of a magazine to form a new sentence.
How would you figure out if a ransom note (string) can be formed from a given magazine (string)?

Simplification: words → characters.

Approach: creating an array and counting characters (每个to letter)
First count the number of times each characters in the ransom note. then go to the magazine.

⇒ generalize: create a hash table, each word maps to the number of times the word appens

4. Base case and build: (recursive.) first for a base case (ex: one element)
then try to solve it for elements one and two.
Assuming have the answer for one, then try to solve it ....

Ex: Design an algorithm to print all permutations of a string.
For simplicity, assume all characters are unique.

Test String: abcdefg

case "a" → {a}

case "ab" → {ab, ba}

case "abc" → ?

If we had answer to P("ab"), the to P{"abc"}
additional letter is "c", add "c" at every possible point.

merge ('c', ab) → cab, acb, abc

merge ('c', ba) → cba, bca, bac

Algorithm: Use a recursive algorithm.
Generate all permutations of a string by "chopping off" the last character and generating all permutations of s[1,... n-1]. Then insert s[n] into every location of the string.

5. Data Structure brainstorm: Run through a list of data structures and try to apply each one.

Ex: Numbers are randomly generated and stored into an (expanding) array.
How would you keep track of the median?

⇒ Linked List? Not — tend not to do well with accessing and sorting numbers

⇒ Array? Maybe — sorted in array → expensive.

⇒ Binary tree? do fairly well with ordering.
in fact, if is perfectly balancing, the top might be the median.
But, if there's an even number of elements, the median is actually the average of the middle two elements, not at the top.

⇒ Heap? good at basic ordering and keeping track of max and mins.

1.1 Implement an algorithm to determine if a string has all unique characters.

每一个和剩下的对比. 有一样就不是.

My solution:

```
String str.;
boolean = true;
for ( int i=0; i < str.length() ; i++ ) {
        val = str.charAt (i);
        for ( int j= i+1; j < str.length(); j++){
                if ( str.charAt(j) == val ) //
                        bool = false
                        break;
        }
}
return bool;
```

$O(n^2)$ ⇒ Not good. No space complexity

Better Solution: Simplicity: assume char set is ASCII. (or need to increase storage size)

```
public static boolean isUniqueChars2 ( String str){
    boolean [] char_set = new boolean [256];   → 每个default false, 值对应的位置上.
    for (int i=0 ; i< str.length (); i++) {    若找一次设值当为true.
        int val = str.charAt(i);
        if ( char_set [val] ) return false; // if 该值设过 返回false.
        char_set [val] = true; // 第一次出现设值, 对应位置设为 True.
    }
    return true;
```

⇒ $O(n)$ where n is the length of the String. (Space / time)

⇒ Best: Reduce space usage a little bit by using a bit vector. (Assume all lower case) 'a'→'z'

②

```
public static boolean isUniqueChars( String str) {
    int checker =0;
    for ( int i=0; i < str.length; i++ ){
        int val = str.charAt(i) - 'a';
        if (( checker & (1 << val)) >0 ) retne false;
        cheeker |= ( 1 << val);
    }
    retm true
}.
```

Q: How to define Null (true/false)? Can destroy the input String.

1.2. Write code to reverse a C-style String ("abcd" 5 characters, including null) → use a null terminator ('\0', ASCII code 0)

0 1 2 3 4 5

My solution:

① (In the code, `characters.length-1` is written above `str.length()-1`, which is boxed and partly crossed out.)

```
public static String reverse (String str){
    if (str.isEmpty() || str.length()==1) return str;
    char[] characters = str.toCharArray(); // 将string vector变成 characters array   (the last one is null)
    int i = 0;
    int j = str.length()-1;   // characters.length-1;
    char temp;
    // swap i, j , i++, j--, i<j
    while (i<j){
        temp = characters [i];
        characters [i] = characters [j];
        characters [j] = temp;
        i++;
        j--;
    }
    String s = new String (characters);
    return s;
}
```

② Recursively

Solution from Internet.

```
public static String reverse2 (String str){
    if (str.isEmpty() || str.length()==1)
        return str;
    // iteractive
    int len = str.length();
    return str.charAt(len-1)+
        reverse2 (str.substring (0,len));
```

Thoughts: in Java, no null character at the end of a string.

Solution: (pure C)

```
void reverse (char * str){
    char * end = str;
    char temp;
    if (str){  // to handle null string
        while (*end){  // find the end character
            ++end;
        }
        --end;  // last meaningful element
        while (str < end){  // terminal condition
        // str and end meets in the middle
        temp = *str;  // normal swap subroutine
        *str++ = *end;  //str advance one step
        *end-- = temp;  // end back one step
        }
}
```

Q: Can I use Java?
Should I also mention the difference in String between C and Java
or just emphasize the algorithm is nearly the same?

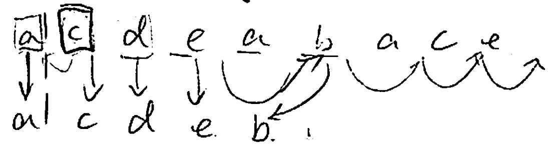
1.3. Design an algorithm and write code to remove duplicate characters in a string without using any additional buffer. (No extra copy of the array)

what does this mean? Can use an additional array of constant size?

$O(N^2)$

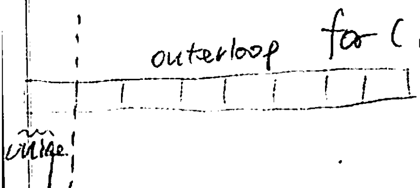
① Algorithm — No (large) Additional Memory: { ① For each char, check if it's a duplicate of already found chars ② If it is, skip it to next, and update non duplicate one.

ex:

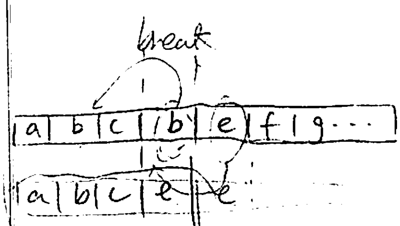


② With additional Memory of Constant Size.
↓
类似 1.1.
用 boolean[] hit = new boolean[256]
对256个字符初始化为false.
对 str[i] 中的每一个.
hit[str[i]] = false
则加到 Unique array 中
再设为 true;

```
public static void removeDuplicate( char[] str ){
    if ( str.length < 2 || str == null ) return;
    char[] chars = str.toCharArray(); × 存多余 memory.
    int unique = 1;   // number of unique chars in the array
    // start from the 2nd char to the end of array.
    // the first char is unique one;
    for ( int i=1 ; i< str.length ; i++) {
        int j;            ↙ ++j means 在 loop 时 +1
        for ( j=0 ; j< tail ; ++j ) {   // for every char in the outer loop
            if ( str[i] == str[j] )  // check if that char is already seen
                break;   // 若在 out loop 中发现了有出现在 unique 中的 char
                         则 break 出 inner loop.
        }
        //. j==tail 说明在 unique array 里没有找到有重复的, 则将这个 outerloop 中的 char
                         加入到 unique array 里)
        if ( j == tail ) {
            str[tail] = str[i];
            ++tail;  // tail 长度加 1
        }
    }
    ☆ str[tail] = 0; // mark 此处是结尾.   ×
                ↓ 不会把前面空的去掉
    for ( ; tail < len; tail++){ str[tail] = 0;}
}
```



[0, tail) 中均为 unique chars.



OR. StringBuffer str.

```
str.charAt(i) == str.charAt(j);
str.setCharAt( tail , str.charAt(i));
☆ str.setLength(tail); // 在 [0,tail) 中是 unique 的; 在 [tail, input.length) 中有 duplicates
```

1.4 Write a method to decide if two strings are anagrams or not. char 顺序不同. 个数一致.

2 . methods
- sort() both. compare from start to end → $O(n\log n)$
- hash map. $O(n)$ time count the number of each character. $O(1)$ space

① Import java.util.Arrays;

```
public boolean isAnagram(String s1, String s2){
    char[] a1 = s1.toCharArray();
    char[] a2 = s2.toCharArray();
    Arrays.sort(a1);
    Arrays.sort(a2);
    if (Array.toString(a1).equals(Arrays.toString(a2))) return true;
    return false.
```

② public static boolean anagram(String s, String t){