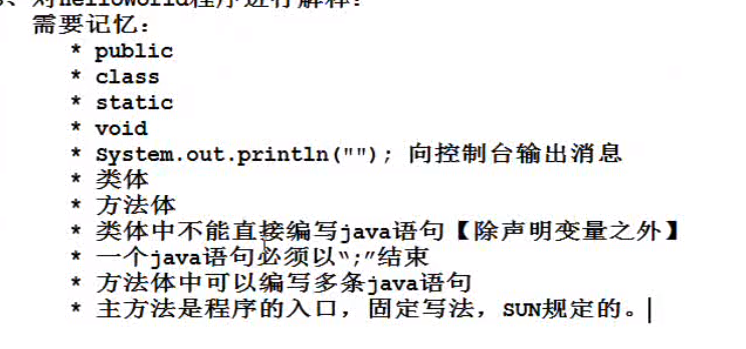
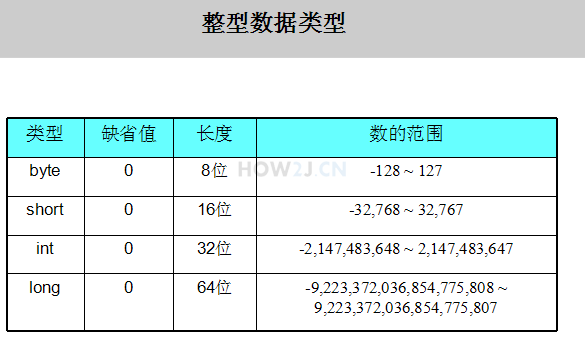
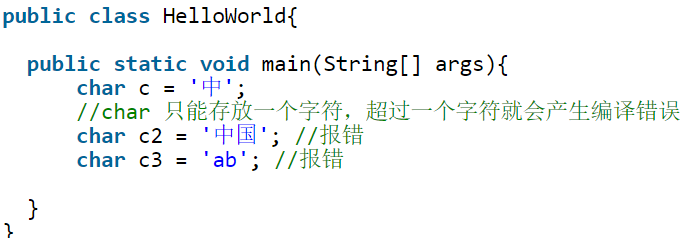
h 





**--类型转换**

**public** **class** HelloWorld {

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

**char** c = 'A';

**short** s = 80;

        //虽然short和char都是16位的，长度是一样的

        //但是彼此之间，依然需要进行强制转换

        c = (**char**) s;

    }

}



**--静态块和普通块的执行顺序**

**public** **class** HelloWy {

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

      System.out.println("执行普通块"); //后执行

  }

**static**{

      System.out.println("执行静态块"); //先执行

  }

}

**--快捷操作**

批量修改 alt+shift+r

快速输入主方法 [main]+alt+/

输入System.out.println [syso]+alt+/

快速输入for [for]+alt+/

代码格式化 ctrl+shift+f

快速助手 ctrl+shift+o

快速注释 ctrl+shift+c

转换大/小写 ctrl+shift+x / ctrl+shift+y

i++; ****先取值，再运算****   
++i; ****先运算，再取值****

**public** **class** HelloWorld {

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

**int** i = 5;

        System.out.println(i++); //输出5

        System.out.println(i);   //输出6

**int** j = 5;

        System.out.println(++j); //输出6

        System.out.println(j);   //输出6

    }

}

例题：

int i = 1;

boolean b = !(i++ == 3) ^ (i++ ==2) && (i++==3);

System.out.println(b);

System.out.println(i);

数组

public class HelloWorld {

public static void main(String[] args) {

//写法一： 分配空间同时赋值

int[] a = new int[]{100,102,444,836,3236};

//写法二： 省略了new int[],效果一样

int[] b = {100,102,444,836,3236}

//写法三：同时分配空间，和指定内容

//在这个例子里，长度是3，内容是5个，产生矛盾了

//所以如果指定了数组的内容，就不能同时设置数组的长度

int[] c = new int[3]{100,102,444,836,3236};

}

}

**//增强型for循环遍历**

**for** (**int** each : values) {

    System.out.println(each);

}

**//复制数组**

System.arraycopy(src, srcPos, dest, destPos, length)

src: 源数组  
srcPos: 从源数组复制数据的起始位置  
dest: 目标数组  
destPos: 复制到目标数组的起始位置  
length: 复制的长度

**//转字符串Arrays.toString()**

**import** java.util.Arrays;

**public** **class** HelloWorld {

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

**int** a[] = **new** **int**[] { 18, 62, 68, 82, 65, 9 };

        String content = Arrays.toString(a);

        System.out.println(content);

    }

}

**import** java.util.Arrays;

**//排序Arrays.sort()**

**public** **class** HelloWorld {

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

**int** a[] = **new** **int**[] { 18, 62, 68, 82, 65, 9 };

        System.out.println("排序之前 :");

        System.out.println(Arrays.toString(a));

        Arrays.sort(a);

        System.out.println("排序之后:");

        System.out.println(Arrays.toString(a));

    }

}

**//搜索Arrays.binarySearch()**

**import** java.util.Arrays;

**public** **class** HelloWorld {

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

**int** a[] = **new** **int**[] { 18, 62, 68, 82, 65, 9 };

        Arrays.sort(a);

        System.out.println(Arrays.toString(a));

        //使用binarySearch之前，必须先使用sort进行排序

        System.out.println("数字 62出现的位置:"+Arrays.binarySearch(a, 62));

    }

}

**//判断是否相同Arrays.equals(a,b)**

**import** java.util.Arrays;

**public** **class** HelloWorld {

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

**int** a[] = **new** **int**[] { 18, 62, 68, 82, 65, 9 };

**int** b[] = **new** **int**[] { 18, 62, 68, 82, 65, 8 };

        System.out.println(Arrays.equals(a, b));

    }

}

**//填充Arrays.fill(a,填充物)**

**import** java.util.Arrays;

**public** **class** HelloWorld {

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

**int** a[] = **new** **int**[10];

        Arrays.fill(a, 5);

        System.out.println(Arrays.toString(a));

    }

}

**//多个参数**

public class ADHero extends Hero {

public void attack() {

System.out.println(name + " 进行了一次攻击 ，但是不确定打中谁了");

}

// 可变数量的参数

public void attack(Hero... heros) {

for (int i = 0; i < heros.length; i++) {

System.out.println(name + " 攻击了 " + heros[i].name);

}

}

public static void main(String[] args) {

ADHero bh = new ADHero();

bh.name = "赏金猎人";

Hero h1 = new Hero();

h1.name = "盖伦";

Hero h2 = new Hero();

h2.name = "提莫";

bh.attack(h1);

bh.attack(h1, h2);

}

}

**//-----**



**//单例——饿汉式**

package charactor;

public class GiantDragon {

//私有化构造方法使得该类无法在外部通过new 进行实例化

private GiantDragon(){

}//必须

//准备一个类属性，指向一个实例化对象。 因为是类属性，所以只有一个

private static GiantDragon instance = new GiantDragon();//前者有分配空间，后者没有

//public static 方法，提供给调用者获取12行定义的对象

public static GiantDragon getInstance(){ //关键、直接调用单例

return instance;

}

}

**//单例——懒汉式**

package charactor;

public class GiantDragon {

//私有化构造方法使得该类无法在外部通过new 进行实例化

private GiantDragon(){

}//必须

//准备一个类属性，用于指向一个实例化对象，但是暂时指向null

private static GiantDragon instance;

//public static 方法，返回实例对象

public static GiantDragon getInstance(){

//第一次访问的时候，发现instance没有指向任何对象，这时实例化一个对象

if(null==instance){

instance = new GiantDragon();

}

//返回 instance指向的对象

return instance;

}

}

**//单例模式三要素**

1. 构造方法私有化  
   2. 静态属性指向实例  
   3. public static的 getInstance方法，返回第二步的静态属性

**//匿名类**

**public** **abstract** **class** Hero {

    String name; //姓名

**float** hp; //血量

**float** armor; //护甲

**int** moveSpeed; //移动速度

**public** **abstract** **void** attack();

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

        ADHero adh=**new** ADHero();

        //通过打印adh，可以看到adh这个对象属于ADHero类

        adh.attack();

        System.out.println(adh);

        Hero h = **new** Hero(){

            //当场实现attack方法

**public** **void** attack() {

                System.out.println("新的进攻手段");

            }

        };

        h.attack();

        //通过打印h，可以看到h这个对象属于Hero$1这么一个系统自动分配的类名

        System.out.println(h);

    }

}

**//本地类**

**public** **abstract** **class** Hero {

    String name; //姓名

**float** hp; //血量

**float** armor; //护甲

**int** moveSpeed; //移动速度

**public** **abstract** **void** attack();

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

        //与匿名类的区别在于，本地类有了自定义的类名

**class** SomeHero **extends** Hero{

**public** **void** attack() {

                System.out.println( name+ " 新的进攻手段");

            }

        }

        SomeHero h  =**new** SomeHero();

        h.name ="地卜师";

        h.attack();

    }

}