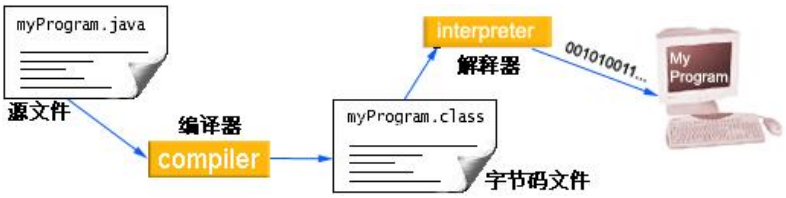
JAVA 知识点整理

1. Java 的工作原理

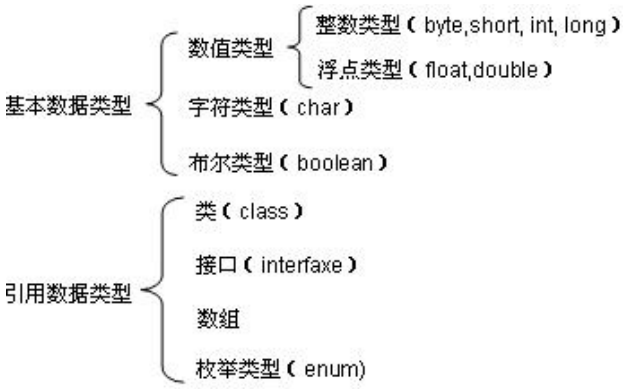


2. Java 采用 Unicode 编码
3. Java 应用程序的结构

```
package java1_3_1.exam;  //打包语句
import java.util.Date;  //导入语句
public class Welcome  //定义一个类，名为 Welcome
{
    public static void main(String args[])//main 是类的主方法
    {
        System.out.println(new Date());  //控制台显示日期
        System.out.println("欢迎进入 Java 世界！");  //控制台输出字符串
    }
}
```

- package 语句（打包语句）是程序的第一条语句，它是可选的。一个源程序最多只能有一个打包语句。它指明编译后的字节码文件（.class）存放的位置。
- import 语句(导入语句)用于导入所需的其他类，可以有多个，但必须放在 package 之后，class 之前。
- 接下来是类的定义。Java 程序都是以类的方式组织的，class 关键字用于定义类，每个类都有类名，花括号括起的部分为类体。
- package、import、class 三者有次序。
- Java 程序由类构成，每一个类的类体中可以包含多个成员方法。
- class 前面的 public 表明这个类是公有的，这种类的源文件必须和类名同名。Java 源文件都保存在.java 文件中，编译后的文件（字节码文件）存放在.class 文件中。一个源文件中可以包含多个类，但只能有一个是 public 类型。
- main()方法是一个特殊的方法，它是程序执行的入口。main 方法说明的格式是特定的：public static void main(String args[])。一个应用程序只有一个类包含 main()方法，它是程序的主类。
- System.out.println()方法用于在标准输出设备（屏幕）上输出数据。
- Java 程序大小写是敏感的。语句的分割用分号。“//”为行注释。

4. 标识符合法判断，注意不能以数字开头
   - Java 语言规定标识符由字母、下划线、$和数字组成
   - 标识符应以字母、下划线、美元符$开头，不能以数字开头。
   - 标识符区别大小写，标识符长度不限，但是实际命名不宜过长。
   - 标识符不能与关键字同名。
5. Java 命名规范：
   - 包：由小写字母组成。如：com.sun.eng
   - 类：由一个或几个单词组成，每个单词的第一个字母大写。类名一般使用完整单词，避免缩写词（除非该缩写词被更广泛使用，像 URL，HTML）。
   - 接口：与类相同，可以使用形容词词缀，比如 Runnable，Comparable。
   - 方法：除第一个字母小写外，和类、接口的命名规则一样。
   - 全局变量：除第一个字母小写外，和类、接口的命名规则一样。比如：personInfo
   - 局部变量：命名与全局变量相同，可以使用简写，比如 i，j，temp，maxNumber。
   - 常量：由一个或多个被下划线分开的大写单词组成，比如：PAGE_SIZE。
6. Java 数据类型的划分



   - 整数类型
     - 整数有三种表示形式：
       - ✧ 十进制整数：如 123，-456，0。
       - ✧ 十六进制整数：以 0x 或 0X 开头，如 0x123，-0X12。
       - ✧ 八进制整数：以 0 开头，如 012，-027。
     - 如果要表示长整型数，可以在数字的后面加上 L 或 l，如 125L。
   - 浮点类型
     - 浮点类型的数据有如下表示形式：
       - ✧ 十进制数形式：由数字和小数点组成，如：0.123，1.23，123.0。
       - ✧ 科学计数法形式：如 123e3 或 123E3。
     - 如果表示 float 型的数据要在数字后加 f 或 F，如 1.23f；表示 double 型的数据在数字后面加 d 或 D，如 2.3d，带小数点的数默认就是双精度浮点型，d

可以省略。

- 布尔类型：布尔型数据只有两个值：true 和 false，在内存中占用 4 个字节。

7. 数据类型转换
   - 自动类型转换：基本数据类型间的优先关系（从低到高）如下：byte -> short -> char -> int -> long -> float -> double
   - 强制类型转换，例如：

     ```
     int i = 12;
     byte b = (byte)i; //把 int 型变量 i 强制转换为 byte 型
     ```

   - 其他类型转换
     - 字符串转换为数字

       例如：Integer.parseInt(String s)//转换为整型的数值
     - 数字转换为字符串
       - Byte.toString(byte b)：字节型转换为字符串。
       - 也可以使用字符串类的 valueOf 方法：String.valueOf(各种类型的数值变量)。
       - 还可以用空字符串连接数字，将数字转换为字符串。如：""+25。
8. 数组定义与引用，初始化、下标越界；遍历
   - 一维数组
     - 定义：类型 数组名[]; 或类型 []数组名;其中，类型可以为 Java 中任意的数据类型，包括基本类型和引用类型。例如：

       ```
       int intArray[];  //定义个整型数组
       String stringArray[];  //定义一个字符串数组
       ```

     - 引用：数组名[索引] 索引为数组的下标，它可以为整型的常数或表达式，下标从 0 开始。每个数组都有一个属性 length 指明它的长度，例如：

       intArray.length 指明数组 intArray 的长度。
     - 初始化
       - 直接赋值创建

         定义数组时就为数组元素赋值，数组的大小是由所赋值的个数决定。

         ```
         int intArray[] = {1,2,3,4};
         String stringArray[] = {"abc", "How", "you"};
         ```

       - 用 new 创建

         ```
         int a[];   //先定义
         a = new int[2];  //再创建
         a[0] = 4;
         a[1] = 7;
         ```

```
String []s = new String[2];  //定义和创建一起完成
s[0] = new String("Good");
s[1] = new String("bye");
```

- 遍历

- 多维数组（以二维数组为例）
  - 定义：类型 数据名[ ][ ]; 或类型 [ ][ ]数据名;
  - 引用：数组名[索引 1][索引 2] 例如： a[1][0];
  - 初始化
    - ✧ 直接赋值创建

      ```
      int a[][] = {{1,2},{2,3},{3,4,5}};
      ```

      提示 Java 语言中，由于把二维数组看作是数组的数组，数组空间不是连续分配的，所以不要求二维数组每一维的大小相同。
    - ✧ 用 new 创建

      ```
      int a[][] = new int[2][3]; //直接为每一维分配空间
      String s[][] = new String[2][]; //仅为第一维分配空间
      s[0] = new String[2]; //为第二维的第一个单元分配引用空间
      s[1] = new String[1]; //为第二维的第二个单元分配引用空间
      s[0][0] = new String("Good");
      s[0][1] = new String("Luck");
      s[1][0] = new String("You");
      ```
  - 遍历
- Java 在定义数组时，[]可以放在数组名前面，也可以放在数组名后面。数组在定义时不能指定大小。

9. 整除、求与操作：% /
10. 字符串基本操作
    - 字符串的创建
      - 创建一个空的字符串
        - ✧ String s1 = new String(); //第一种创建空字符串方法
        - ✧ String s2 = ""; //第二种创建空字符串方法
      - 直接赋值创建字符串：String s3 = "China";
      - 使用原有的 String 对象创建字符串

        ```
        String s4 = "abc";
        String s5 = new String(s4);
        ```
      - 使用数组创建字符串

        ```
        char ch1[]={'a','b','c','d','e'};
        ```

String s6 = new String(ch1); //创建字符串对象“abcde”

String s7 = new String(ch1,0,3); //创建字符串对象“abc” //从数组下标为0开始，创建有3个字符的字符串

- 字符串的基本操作
  - 字符串连接
    - 通过contact()连接

      String s1=”abc”;

      String s2=”de”;

      String s3=s1.contact(s2); //结果为abcde
    - 使用“+”链接,可以链接字符串和其他类型的数据,在连接时自动把其他类型的数据转换为字符串

      String str=”abc”+12;//结果为”abc12”
  - 返回长度

    String str1=”this is a string”;

    int a=str1.length();
  - 替换
    - replace(char oldChar,char newChar) 将oldChar替换为char newChar
    - replaceAll(String regex, String str)将字符串中匹配正则表达式的字符串替换成str

    String str1=”this is a string”;

    System.out.println(str1. replace(‘s’,’a’)); //结果为thia ia a atring

    System.out.println(str1. replaceAll(“this”,”that”));//结果为that is a string
  - 查找
    - indexOf(String str) 查找str在字符串中出现的位置
    - lastIndexOf(String str) 查找str在字符串中最后出现的位置

    String str1=”this is a string”;

    System.out.println(str1.indexOf(“is”)); //结果为2

    System.out.println(str1.lastIndexOf(“is”));//结果为5
  - 截取子串
    - substring(int begin) 从begin处开始取，截止到最后
    - substring(int begin,int end) 从begin处开始取，截止到end处

    String str1=”this is a string”;

    System.out.println(str1. substring(0, 4)); //结果为this

System.out.println(str1. substring(10));//结果为 string

11. 表达式合法性判断
12. 三元表达式用法及简单计算【表达式 A ？表达式 B ：表达式 C】

    先计算表达式 A 的值，当值为真时，则将表达式 B 的值作为整个表达式的值；反之则将表达式 C 的值作为整个表达式的值。
13. 运算符优先级判断

优先级	运算符
1	() [] .
2	! +(正) -(负) ~ ++ --
3	* / %
4	+(加) -(减)
5	<< >> >>>
6	< <= > >= instanceof
7	== !=
8	&(按位与)
9	^
10	|
11	&&
12	||
13	?:
14	= += -= *= /= % = &= |= ^= ~= <<= >>= >>>=

14. 注释用法，单行注释、多行注释和文档注释各自语法及用途
    - 单行注释：以“//”开头，直到行末尾。
    - 多行注释：以“/*”开头，直到“*/”结束，用来注释一行或多行。
    - 文档注释：以“/**”开头，直到“*/”结束，这是 Java 语言特有的注释方法，能被转化为 HTML 格式的帮助文档。
15. 构造方法以及作用【带参数和不带参数的构造方法应用以及子类调用父类构造方法】
    - 子类调用父类构造方法：super([参数列表]);
    - 构造方法的作用：能够初始化对象的数据，在创建对象的时候，直接给对象的数据赋值
    - 带参和不带参的构造方法（详细内容可以参考 P73 页案例 3-2）

      例如，将以下构造方法添加到类 Circle 中：

      Circle(double r) { radius=r; }

      使用下面的语句创建圆：

      myCircle=new Circle(5.0); //正确，将 myCircle.radius 赋值为 5.0

      myCircle=new Circle(); //错误，因为它使用了无参的构造方法

      一个类没有定义构造方法，系统会为其设置一个默认的无参构造方法，但是当类中已经定义了构造方法并且是有参数的，这时如果还用无参的构造方法建立对象就会出错。
    - 类的构造方法遵循以下规定
      - 构造方法与类同名。

- 构造方法没有返回类型，甚至连 void 也没有。
- 一个类可以有多个构造方法，但参数不同（个数或类型不同）。
- 如果类没有构造方法，将自动生成一个默认的无参数构造方法，并使用默认值初始化对象的属性(如，int 变量初始化为 0，boolean 变量初始化为 false)。
- 类的构造方法可以通过关键字 this 调用另一个构造方法
- 构造方法只能由 new 操作符调用

16. 类及其属性、方法 修饰符【访问范围】

➢ 成员的访问控制

权限	同一个类	同一个包	不同包的子类	不同包非子类
private	*			
friend	*	*		
protected	*	*	*	
public	*	*	*	*

➢ 实例变量、类变量和类常量

- 实例变量：没有使用 static 修饰符的数据成员是实例变量，不能被同一个类里的不同变量共享。一定要实例化。
- 类变量：使用 static 修饰符的数据成员，想让一个类的所有实例共享数据，可使用静态变量，也称为类变量。
- 类常量：一个数据成员在声明时加上关键字 final，它的值就不能再被改变，因此称为常量。如果 static 和 final 同时使用，就是类常量。
- 注意：
  - ✧ 类变量的值存储在类的公用内存，如果某个对象修改了类变量的值，同一类的所有对象都会受到影响。对于整个类来说，类变量的值只存一份。
  - ✧ 类变量一定是静态变量。

➢ 全局变量和局部变量

- 全局变量：实例变量和类变量（类变量一定是静态变量）
- 局部变量：在方法内部说明的变量
- 注意：
  - ✧ 全局变量的作用域是所有实例方法，局部变量的作用域从它的说明开始延续到包含它的块尾。
  - ✧ 全局变量不赋值有默认初值，但局部变量没有默认初值的，使用时必须赋初值。
  - ✧ 全局变量只能声明一次，但在一个方法互不嵌套的块内，可以多次声明同一个局部变量。
  - ✧ 局部变量与全局变量同名时，局部变量优先，同名的全局变量被隐藏。
  - ✧ 在构造方法中可使用 this 调用其他构造方法。例如，圆的默认构造方法

中使用 this 调用有参的构造方法。

- 实例方法（非静态方法）和类方法（静态方法）（判断题常考）
  - 实例方法：没有使用 static 修饰符的方法为实例方法，实例方法必须通过对象来调用，不可以通过类名调用。
  - 类方法：在定义的时候加上 static 修饰符，就是类方法。类方法可以通过类名调用，也可以通过对象调用。
  - 类方法只能操作类变量，实例方法既可以操作实例变量也可以操作类变量。
  - 静态方法可以调用非静态方法，但是非静态方法不能调用静态方法

17. 修饰符混用：
    - abstract 不能与 final 并列修饰同一个类【正确】
    - abstract 类中不可以有 private 的成员【正确】
    - abstract 方法必须在 abstract 类中【错误】
    - static 方法中能处理非 static 的属性【正确】
18. 抽象方法、抽象类
    - 抽象类：Java 语言中，用 abstract 关键字来修饰一个类时，这个类称为抽象类。

      抽象类的定义格式如下：

      [修饰符] abstract class 类名

      { 类体 }

      注意：
      - 抽象类可以包含抽象方法，也可以不包含抽象方法。但是包含抽象方法的类必须定义成抽象类。（判断题常考）
      - 抽象类不能被实例化，抽象类可以被继承，不能被定义成 final 类。
      - 继承抽象类的类必须实现抽象类的抽象方法，否则，也必须定义成抽象类。
      - 一个类实现某个接口，但没有实现该接口的所有方法，这个类必须定义成抽象类。
    - 抽象方法：用 abstract 关键字来修饰一个方法时，这个方法称为抽象方法。

      [修饰符] **abstract** 返回值类型 方法名([参数列表]);

      注意：抽象方法只有声明，没有实现。
19. 方法的覆盖：子类中的某个方法与父类的某个方法说明（指名称、参数和返回值类型）一样。在覆盖的情况下，子类将使用自己的方法。关于覆盖，需要注意以下几点：
    - 一个方法将方法的父类实现替换为自己的实现，其说明必须和父类方法说明相同，但返回类型可以按照某种特定的方式变化。如果返回类型是引用类型，则覆盖方法的返回类型可以声明为父类方法声明的返回类型的子类型；如果返回类型是基本类型，则覆盖方法的返回类型必须和父类方法的返回类型相同。
    - 覆盖方法有自己的访问修饰符，但只限于提供同样或更多的访问权限。

- 覆盖方法的 throws 子句可以和父类方法有所不同，它列出的每一个异常类型都应该和超类中的异常类型相同，或者是父类异常类型的子类型。
- 不能用子类的静态方法覆盖父类中的实例方法。
- 带关键字 final 的方法不能被覆盖。
- 抽象方法必须在子类中被覆盖，否则子类也必须是抽象的。

20. 接口的继承的特点：可以多继承，例如：

```
interface IChineseWelcome {
    String CHINESE_MSG = "你好，欢迎你!";  //定义常量
    void sayChinese();
}
interface IEnglishWelcome {
    String ENGLISH_MSG = "Hello,Welcome!";  //定义常量
    void sayEnglish();
}
interface IWelcome extends IChineseWelcome, IEnglishWelcome {  //继承接口
    String ENGLISH_AND_CHINESE_MSG = "Hello,Welcome! 你好，欢迎你!";
    void sayChineseAndEnglish();
}
```

- 定义接口的注意事项
  - 只包含常量和抽象方法，不能包含变量和具体的方法
  - 常量都是 public static final 类型，方法都是 public abstract 类型
- 接口的使用：在类的声明中用 implements 子句来表示它所实现的接口。实现某接口的类，必须实现接口中定义的所有方法，否则需定义成抽象类。在类体中可以使用接口中定义的常量。一个类可以实现多个接口，在 implements 子句中用逗号分开。

21. 接口与抽象类的区别
- 抽象类可提供某些方法的实现，而接口的方法都是抽象的
- 抽象类可以包含变量，而接口中不能包含变量，可以包含常量
- 抽象类中的成员可以有多种权限，而接口中的成员只能是 public
- 抽象类中增加一个具体的方法,则子类都具有此具体方法,而接口中增加一个方法，则子类必须实现此方法
- 子类最多能继承一个抽象类，而接口可以继承多个接口，一个类也可以实现多个接口
- 抽象类和它的子类之间应该是一般和特殊的关系,而接口仅仅是它的子类应该实现的一组规则，无关的类也可以实现同一接口

22. 类的继承【子类访问父类变量和方法问题、变量隐藏】特点
    - 类的继承（关键字：extends）特点
        - 如果类 B 是类 A 的子类，则类 B 继承了类 A 的变量和方法。在子类 B 中，包括了两部分内容：从父类 A 中继承下来的变量和方法，自己新增加的变量和方法。
        - 在 Java 中类只支持单一继承，不支持多重继承，接口可祢补这方面的一些缺陷。
        - 继承是可传递的。如果 C 从 B 派生，而 B 从 A 派生，那么 C 就会既继承在 B 中声明的成员，又继承在 A 中声明的成员。
        - 派生类可扩展它的直接基类，添加新的成员，但不能移除父类中定义的成员。
        - 除构造方法外，其他非私有成员都可以被继承。私有数据成员虽然不能被继承，但在派生类中可以通过公有方法间接访问。
        - 派生类可以通过声明具有相同说明的新成员来隐藏那个被继承的成员。但隐藏继承成员并不移除该成员，它只是使被隐藏的成员在派生类中不可直接访问。
    - 子类对父类成员的继承
        - 父类中用 public 修饰的公有成员，子类可继承，子类内部可以直接使用，外界也可以通过子类使用。
        - 父类中用 private 修饰的私有成员，子类不能继承，子类内部不能直接使用，外界也不能通过子类对象使用。但是，如果父类提供公有方法（如属性方法），在子类内部可以间接使用。
        - 父类中用 protected 修饰的保护成员，子类可以继承，子类内部可以直接使用，外界仅限于同一包类中的类可以使用。
23. 多态性：多态性是指允许不同类的对象对同一消息作出不同的响应。
24. 子类调用父类相同名字的变量和方法。
    - super 关键字：关键字 this 指类的实例自己，而关键字 super 指父类。
        - 调用父类的构造方法：super（参数名）；
            - 父类的构造方法不传给子类，他们只能用关键字 super 在子类的构造方法中调用
            - super 语句必须是第一条语句
            - 如果没有显式的使用 super 调用父类构造方法，就总是调用父类的默认构造方法
        - 调用父类的方法：super.method();
            - 子类中的某个方法与父类的某个方法说明（名称、参数和返回值类型）一样，在子类中将使用自己的方法，这时如果还想使用父类的方法，就需要使用 super

25. 创建对象数量问题：
    - 【1】String s1="bc"; String s2="bc"; 创建了一个对象
    - 【2】String s1="bc"; String s2=new String("bc"); 创建了两个对象
26. 方法返回类型及方法的定义
27. final 修饰的特点：
    - final 修饰的量是常量，在程序的运行过程中不改变。例如：final int NUM = 100;
    - 只读
    - final 修饰的类是最终类，当一个类定义后，不需要再进行扩展，可以定义成最终类。最终类的声明方法是在类名前加上 final 修饰符。
    - 修饰符 abstract 和 final 不能同时使用
28. 静态变量和非静态变量
    - 静态变量：通过类名直接访问
    - 非静态变量：实例变量，一定要实例化
29. abstract 和 final 修饰符
30. 编程题【类、抽象类、接口的定义及使用，类的继承，多看例题】
    - 类 P73 案例 3-2
    - 类的继承 P90 案例 3-6
    - 接口的定义和使用 P96

    例题：编写一个 student 类，里面有 name,id,sex,age 等属性。写方法获得 name,id,sex,age 等。写一个 toString(方法)

```
package exam;
public class Student {
   int id;
   int age;
   String name;
   boolean sex;

   //无参构造函数
   public Student() {
       super();
    }
   //有参构造函数
   public Student(int id, int age, String name, boolean sex) {
       super();
       this.id = id;
```

```
        this.age = age;
        this.name = name;
        this.sex = sex;
    }
    public int getId() {
        return id;
    }
    public int getAge() {
        return age;
    }
    public String getName() {
        return name;
    }
    public boolean isSex() {
        return sex;
    }

    public String toString() {
        return("姓名："+getName()+"学号："+getId()+"年龄："+getAge()+"性别
"+isSex());
    }

}
```