Java 基础（7）：深入解析 Java 的四中访问权限

引言

Java 中的访问权限理解起来不难，但完全掌握却不容易，特别是 4 种访问权限并不是任何时候都可以使用。下面整理一下，在什么情况下，有哪些访问权限可以允许选择。

一、访问权限简介

访问权限控制： 指的是本类及本类内部的成员（成员变量、成员方法、内部类）对其他类的可见性，即这些内容是否允许其他类访问。

Java 中一共有四种访问权限控制，其权限控制的大小情况是这样的：**public > protected > default(包访问权限) > private** ,具体的权限控制看下面表格，列所指定的类是否有权限允许访问行的权限控制下的内容：

访问权限	本类	本包的类	子类	非子类的外包类
public	是	是	是	是
protected	是	是	是	否
default	是	是	否	否
private	是	否	否	否

- **public：** 所修饰的类、变量、方法，在内外包均具有访问权限；
- **protected：** 这种权限是为继承而设计的，protected 所修饰的成员，对所有子类是可访问的，但只对同包的类是可访问的，对外包的非子类是不可以访问；
- **包访问权限（default）：** 只对同包的类具有访问的权限，外包的所有类都不能访问；
- **private：** 私有的权限，只对本类的方法可以使用；

注意： 要区分开 protected 权限、包访问权限，正确使用它们；

- 当某个成员能被所有的子类继承，但不能被外包的非子类访问，就是用 protected；
- 当某个成员的访问权限只对同包的类开放，包括不能让外包的类继承这个成员，就用包访问权限；

使用访问权限控制的原因：

1) 使用户不要碰触那些他们不该碰触的部分；
2) 类库设计者可以更改类的内部工作的方式，而不会担心这样会对用户产生重大影响；

二、访问权限控制的使用场景

访问权限使用的场景可以总结为下面的五种场景，分别对访问权限的使用有不同的限制：

1. 外部类的访问控制

外部类（外部接口） 是相对于内部类（也称为嵌套类）、内部接口而言的。**外部类的访问控制只能是这两种：public 、default 。**

```
//public 访问呢权限的外部类，所有类都可以使用这个类
public class OuterClass {
}
//default 权限的外部接口，所有类、接口均可以使用此接口
interface OuterInterface{
}
```

2. 类里面的成员的访问控制

类里面的成员分为三类 ： **成员变量、成员方法、成员内部类（内部接口）。**
类里面的成员的访问控制可以是四种，也就是可以使用所有的访问控制权限。

```
public class OuterClass {
    public int aa; //可以被所有的类访问
    protected boolean bb; //可以被所有子类以及本包的类使用
    void cc() { //default 访问权限，能在本包范围内使用
        System.out.println("包访问权限");
    }
    //private 权限的内部类，即这是私有的内部类，只能在本类使用
    private class InnerClass{
    }
}
```

注意：

这里的类里面的成员 是指类的全局成员，并没有包括局部的成员（局部变量、局部内部类，没有局部内部接口）。或者说，局部成员是没有访问权限控制的，因为局部成员只在其所在的作用域内起作用，不可能被其他类访问到。

```
public void count(){
    //局部成员变量
```

```
    public int amount;//编译无法通过，不能用 public 修饰
    int money;//编译通过
    //局部嵌套接口
    class customer{//编译通过
    }
}
```

上面的两种场景几乎可以适应所有的情况，但有一些情况比较特殊，还做了有些额外访问权限的要求

3. 抽象方法的访问权限

普通方法是可以使用四种访问权限的，**但抽象方法是有一个限制：不能用 private 来修饰，也即抽象方法不能是私有的，否则，子类就无法继承实现抽象方法。**

4. 接口成员的访问权限

接口由于其的特殊性，所有成员的访问权限都规定得死死的，下面是接口成员的访问权限：

- 变量： public static final
- 抽象方法： public abstract
- 静态方法： public static，JDK1.8 后才支持
- 内部类、内部接口 ： public static

也因为所有的一切都默认强制规定好了，所以我们在用的时候，并不一定需要完整写出所有的修饰符，编译器会帮我们完成的，也就是，可以少写修饰符，但不能写错修饰符。

```
public interface Interface_Test {
    public int aa = 6; //少写了  static final
    int bb = 5; //
    //嵌套接口，可以不写 public static
     interface cc{
    }
}
```

5. 构造器的访问权限

构造器的访问权限可以是以上四种权限中的任意一种：

1) 采用 private：一般是不允许直接构造这个类的对象，再结合工厂方法（static 方法），实现单例模式。注意：所有子类都不能继承它。
2) 采用包访问控制：比较少用，这个类的对象只能在本包中使用，但是如果这个类有 static 成员，那么这个类还是可以在外包使用；（也许可以用于该类的外包单例模式）。

注意：外包的类不能继承这个类；

3) 采用 protected ：就是为了能让所有子类继承这个类，但是外包的非子类不能访问这个类；

4) 采用 public ：对于内外包的所有类都是可访问的；

注意： 构造方法有点特殊，因为子类的构造器初始化时，都要调用父类的构造器，所以**一旦父类构造器不能被访问，那么子类的构造器调用失败，意味子类继承父类失败！**

1) 下列哪种说法是正确的（ D）

A． 实例方法可直接调用超类的实例方法

B． 实例方法可直接调用超类的类方法

C． 实例方法可直接调用其他类的实例方法

D． 实例方法可直接调用本类的类方法

2) 如何利用 ServletContext 和 ServletConfig 对象获得初始化参数

```
String psw = config.getInitParameter("psw");
ServletContext ss = config.getServletContext();
String ppp = ss.getInitParameter("name");
```

3) 写一段 Jdbc 连接 Oracle 的程序,并实现数据查询。

创建一个连接数据库的工具类

```
  import java.sql.*;
   public class DbUtil {
   public static Connection getConnection(){
   String driver = "";
   String url = "";
   String name = "scot";
   String psw = "123";
   Connection conn = null;
try {
   Class.forName(driver);
   conn = DriverManager.getConnection(url,name,psw);
} catch (ClassNotFoundException e) {
   e.printStackTrace();
} catch (SQLException e) {
   e.printStackTrace();
}
   return conn;
}
}

import java.sql.*;
public class SearchInfo {
```

```
   public void searchInfo(int id){
   Connection conn = null;
   PreparedStatement pstat = null;
   ResultSet res = null;
   String sql = "select * from users where id=?";
   conn = DbUtil.getConnection();
   try {
      pstat = conn.prepareStatement(sql);
      pstat.setInt(1, id);
      res = pstat.executeQuery();
      while(res.next()){
          String name = res.getString("name");
      }
  } catch (SQLException e) {
      e.printStackTrace();
  }
  }
}
```