Chp7 三个修饰符

参考答案

1. 输出结果
    300
    200
    300
    400

2. EG 静态方法中不能访问非静态成员，即不能访问非静态变量和调用非静态函数。

3. 输出结果
    1
    2
    3
    用这种方式可以用来统计总共创建了多少个对象。

4. 输出结果为
    In Static
    MyClass()
    20
    MyClass(int)
    10

5. 输出结果为
    m1 in Super
    m2 in Sub
    m1 in Sub
    m2 in Sub
    注意，静态方法没有多态。

6. ADE
B 选项错误，非静态方法中可以调用静态方法
C 选项错误，静态方法可以被覆盖，注意：静态方法只能被静态方法覆盖，并且没有多态
F 选项错误，静态方法中不能访问 this

7. C
这个程序选择了在构造方法时对 final 属性赋值。在原代码中，如果创建对象时调用了无参构造方法，在整个创建对象的过程中都没有为 final 属性赋值，这样会造成编译错误。
为了保证创建对象时，无论调用哪一个构造方法，final 属性都会被正确赋值，要求必须在

每个重载的构造函数中，都加上对 final 属性赋值的语句。

8. A
//1, //2 均正确，因为在 printValue 方法中，没有修改 final 的形参。
//3, //4 均正确，因为在 changeValue 中，修改的是形参的值，而没有涉及到实参。因此不过实参是否是 final 的，都不影响形参能否被修改。

9. C
final 修饰引用类型，表示的是引用指向的对象不能改变。
在本题中，final 修饰 mv 引用。mv = new MyValue(); 表示让 mv 引用指向一个 MyValue 对象。这样就对 mv 引用进行了一次赋值，之后，mv 引用就不能被改变。要注意的是，所谓 mv 引用不能改变，指的是 mv 引用中保存的地址不能改变，也就是说，mv 引用不能指向别的对象。
然而，mv 所指向的对象，其属性是可以修改的。
因此，对于本题来说，

```
final MyValue mv = new MyValue();
mv.value = 100;
//1
System.out.println(mv.value);
```

在//1 处写上 mv.value = 200 可以编译通过，因为这修改了 mv 引用所指向对象的属性，而不是让 mv 指向别的对象。
如果在//1 处写上 mv = new MyValue()则不能编译通过，因为这修改了 mv 引用的值，让 mv 引用指向了不同的对象。

10. 可以编译通过，输出结果为
m1() in Super
m1(int) in Sub
m1(double) in Sub
注意，父类有 m1 方法，并且是 final 的，子类也有 m1 方法，但是子类的 m1 方法和父类的 m1 方法不构成方法覆盖。

11. BC B 选项：抽象方法不能有方法体；C 选项：子类的覆盖方法访问权限修饰符相同或更宽

12. ABCD

13. DF
A 错误，abstract 不能与 final 连用，abstract 方法必须被子类覆盖，而 final 方法不能被覆盖，矛盾。
B 错误，应该写成 public final void ...
C 错误，abstract 不能与 static 连用。abstract 方法被子类覆盖之后，会多态的进行调用，而 static 方法没有多态，矛盾。
E 错误，abstract 不能与 private 连用。private 方法不能被继承，因此子类就无法覆盖父类的 private 方法。而 abstract 方法要求一定要被子类覆盖，矛盾。

特别的，abstract 方法的访问修饰符也不能是(default)，abstract 修饰方法时，只能与访问修饰符 public 或 protected 连用。

14. 参考 Shape.java

15. 参考 TestRole.java。注意：抽象类可以有构造方法。

16. 参考 MyClass.java

17. 参考 TestAccount.java

18. 输出结果

In ClassA Static
ClassA()
In Static MyClass
In ClassB Static
In ClassC Static
ClassB()
ClassC()
MyClass()
ClassB()
ClassC()
MyClass()

注：该题较难

解释：

该题需要创建两个 MyClass 对象。

一般而言，创建对象的过程如下：

1) 分配空间
2) 递归构造父类对象
3) 初始化本类属性
4) 调用本类构造方法

如果某个类是 JVM 运行中第一次遇到，则会进行类加载的动作。类加载会初始化该类的静态属性，并执行该类的静态初始化代码块。

因此，本题中创建的两个 MyClass 对象，依次会进行如下步骤：

1) 加载 MyClass 类，并初始化其静态属性
2) 为 MyClass 分配空间
3) 递归构造 MyClass 的父类对象。
4) 初始化 MyClass 属性
5) 调用 MyClass 的构造方法。

至此，第一个对象创建完成。之后创建第二个对象时，由于类加载已经完成，因此跳过类加载的步骤，即：

6) 为 MyClass 分配空间
7) 递归构造 MyClass 的父类对象
8) 初始化 MyClass 属性

9) 调用 MyClass 的构造方法。

经过 9 个步骤，两个对象创建完毕。

其中，在第 1 步时，类加载时会初始化其静态属性，之后会执行静态初始化代码块，因此对第 1 步进行细分：

1.1 初始化 ca 属性

1.2 执行 MyClass 的静态初始化代码块。

在 1.1 执行时，初始化 ca 属性会创建 ClassA 对象。由于这是第一次在程序中用到 ClassA 对象，因此会执行对 ClassA 对象的类加载。即：1.1 步可以细分为以下步骤：

1.1.1 加载 ClassA 类

1.1.2 创建 ClassA 对象

在初始化 MyClass 属性时，需要创建 ClassC 对象。而程序执行到第 4 步时是第一次遇到 ClassC 类型的对象，因此会执行 ClassC 的类加载。因此，对第 4 步和第 8 步进行细化：

第 4 步：初始化 MyClass 属性：

4.1 加载 ClassC

4.2 为 ClassC 分配空间

4.3 递归构造 ClassC 的父类对象

4.4 初始化 ClassC 属性

4.5 调用 ClassC 的构造方法

第 8 步，初始化 MyClass 属性：

8.1 为 ClassC 分配空间

8.2 递归构造 ClassC 的父类对象

8.3 初始化 ClassC 属性

8.4 调用 ClassC 的构造方法

对于 4.1 而言，为了创建 ClassC 对象，必须获取 ClassC 类的信息。而获得 ClassC 类完整信息的前提，是获得 ClassB 类的信息。由于是第一次遇到 ClassB 和 ClassC，因此会先加载 ClassB，之后加载 ClassC。细分之后，4.1 分为以下两步：

4.1.1 加载 ClassB

4.1.2 加载 ClassC

完整列出所有步骤如下：

1.1.1 加载 ClassA 类	→ 输出 In ClassA Static
1.1.2 创建 ClassA 对象	→ 输出 ClassA()
1.2 执行 MyClass 的静态初始化代码块	→ 输出 In Static MyClass
2 分配 MyClass 的空间	→ 无输出
3 递归构造 MyClass 的父类对象	→ 无输出
4.1.1 加载 ClassB	→ 输出 In ClassB Static
4.1.2 加载 ClassC	→ 输出 In ClassC Static
4.2 分配 ClassC 的空间	→ 无输出
4.3 构造 ClassC 的父类对象(ClassB)	→ 输出 ClassB()
4.4 初始化 ClassC 属性	→ 无输出
4.5 调用 ClassC 的构造方法	→ 输出 ClassC()
5 调用 MyClass 的构造方法	→ 输出 MyClass()
6 为 MyClass 分配空间	→ 无输出

7 递归构造 MyClass 的父类对象 → 无输出
8.1 为 ClassC 分配空间 → 无输出
8.2 递归构造 ClassC 的父类对象 → ClassB()
8.3 初始化 ClassC 的属性 → 无输出
8.4 调用 ClassC 的构造方法 → ClassC()
9 调用 MyClass 的构造方法 → MyClass()