1.子类继承父类的变量，代码块和构造器，但并不意味着父类完全初始化之后才进行子类的初始化。

静态变量和静态初始化块是依照他们在类中的定义顺序进行初始化的。

2.静态类型和动态类型

任何一个引用变量都有两个类型：一个静态类型，一个动态类型

A a = new B();

A是静态类型，B是动态类型

3.静态绑定

所有依赖于静态类型来将某方法和该方法所在的类关联起来的动作都是静态绑定。因为静态绑定在程序运行前发生，所有又叫前期绑定。

**private**：不能被继承，则不能通过子类对象调用，而只能通过类本身的对象进行调用，所以可以说private方法和方法所属的类绑定；

**final**：final方法虽然可以被继承，但是不能被重写（覆盖），虽然子类对象可以调用，但是调用的都是父类中的final方法（因此可以看出当类中的方法声明为final的时候，一是为了防止方法被覆盖，而是为了有效关闭java的动态绑定）；

**static**：static方法可以被子类继承，但是不能被子类重写（覆盖），但是可以被子类隐藏。（这里意思是说如果父类里有一个static方法，它的子类里如果没有对应的方法，那么当子类对象调用这个方法时就会使用父类中的方法。而如果子类中定义了相同的方法，则会调用子类的中定义的方法。唯一的不同就是，当子类对象上转型为父类对象时，不论子类中有没有定义这个静态方法，该对象都会使用父类中的静态方法。因此这里说静态方法可以被隐藏而不能被覆盖。这与子类隐藏父类中的成员变量是一样的。隐藏和覆盖的区别在于，子类对象转换成父类对象后，能够访问父类被隐藏的变量和方法，而不能访问父类被覆盖的方法）。

4.动态绑定

所有依赖于动态类型来将某方法和该方法所在的类关联起来的动作都是动态绑定。因为动态绑定是在程序运行时，通过RTTI实现，所以又叫后期绑定。

调用的方法依赖于隐式参数的实际类型，并且在运行时实现动态绑定。动态绑定的过程分为以下几个环节：

（1）编译器查看对象的声明类型和方法名；

（2）编译器查看调用方法时提供的参数类型。例如x.f(“hello”)，编译器将会挑选f(String)，而不是f(int)，由于存在类型转换（int转换为double），所以可能会更复杂。如果编译器没找到参数类型匹配的方法，或者发现有多个方法与之匹配，就会报告一个错误。

至此，编译器获得了需要调用的方法名字和参数类型。

（3）采用动态绑定调用方法的时候，一定调用与所引用对象的实际类型最合适的类的方法。如果x的实际类型是D，它是C类的子类，如果D定义了一个方法f(String)，就直接调用它，否则将在D类的超类中寻找f(String)，以此类推。

5.执行流程

静态代码块>构造代码块>构造方法

静态的内容是随着类的加载而加载，即静态代码块的内容会优先执行

子类的所有构造方法默认都会访问父类的无参数构造方法