1.理解多態性：可以理解為一個事物的多種形態。

2.何為多態性：

對象的多態性：父類的引用指向子類的對象（或子類的對象賦給父類的引用）

舉例：

person p = new man();

object obj = new Date();//每一個類都是object類的子類

3.多態的使用：虛擬方法調用

有了對象的多態性以後，我們在編譯期，只能調用父類中聲明的方法，  
但在運行期，我們實際執行的是子類重寫父類的方法。

總結：編譯，看左邊；運行，看右邊。

4.多態性的使用前提： 1類的繼承關係2方法的重寫

5. 多態性的應用舉例：

舉例一：

**public** **void** func(Animal animal){//Animal animal = new Dog();

animal.eat();

animal.shout();

舉例二：

**public** **void** method(Object obj){

}

舉例三：

**class** Driver{

**public** **void** doData(Connection conn){//conn = new MySQlConnection(); / conn = new OracleConnection();

//規範的步驟去操作數據

// conn.method1();

// conn.method2();

// conn.method3();

}

}

6. 多態性使用的注意點：

對象的多態性，只適用於方法，不適用於屬性（編譯和運行都看左邊）

※總結:

1.若子類重寫了父類方法，就意味著子類裡定義的方法徹底覆蓋了父類裡的同名方法，

系統將不可能把父類裡的方法轉移到子類中：編譯看左邊，運行看右邊

2.對於實例變量則不存在這樣的現象，即使子類裡定義了與父類完全相同的實例變量，

這個實例變量依然不可能覆蓋父類中定義的實例變量：編譯運行都看左邊

（當看見使用多態｛子類對象賦給父類引用｝時，

編譯時，屬性、方法都看左邊（父類），

運行時，屬性也看父類，方法看右邊（子類））

7. 關於向上轉型與向下轉型：//day13

7.1 向上轉型：多態

7.2 向下轉型：

7.2.1：為什麼使用向下轉型：

有了對象的多態性以後，內存中實際上是加載了子類特有的屬性和方法的，但是由於變量聲明為父類類型，導致編譯時，只能調用父類中聲明的屬性和方法。子類特有的屬性和方法不能調用。

如何才能調用子類特有的屬性和方法？向下轉型。

7.2.2：如何實現向下轉型：

使用強制類型轉換符:(強轉成的類型)

7.2.3：使用時的注意點：

1. 使用強轉時，可能出現ClassCastException的異常。

2. 為了避免在向下轉型時出現ClassCastException的異常，我們在向下轉型之前，先進行instanceof的判斷，一旦返回true，就進行向下轉型。如果返回false，不進行向下轉型。

7.2.4：instanceof的使用：

1. a instanceof A:判斷對象a是否是類A的實例。如果是，返回true；如果不是，返回false。

2. 如果 a instanceof A返回true,則 a instanceof B也返回true.其中，類B是類A的父類。

3. 要求a所屬的類與類A必須是子類和父類的關係，否則編譯錯誤。

7.2.5：圖示：



向下轉型注意：

*Collection<T> values = map.values();  
return (List<T>) values;//運行時會報錯*

//因為*map.values()*返回的為Collection,不能直接向下強轉為其子類(*List*),

//要向下轉型需:原本返回的是list然後賦給父類Collection,才能再向下轉成list

8. 面試題：

8.1 談談你對多態性的理解?

8.1.1.實現代碼的通用性。

8.1.2.Object類中定義的public boolean equals(Object obj){ }

JDBC:使用java程序操作(獲取數據庫連接、CRUD)數據庫(MySQL、Oracle、DB2、SQL Server)

8.1.3.抽象類、接口的使用肯定體現了多態性。(抽象類、接口不能實例化:不能造對象。)

8.2 多態是編譯時行為還是運行時行為?