目 录

1 复用类

Java 中复用类的两种方式：

- 在新的类中产生现有类的对象，这种方法称为组合。
- 按照现有类的类型来创建新类，这种方法称为继承。

1.1 类的组合

组合技术很直观，只要将对象引用置于新类中即可。例子如下：

```
class WaterSource
{
    private String s;
    WaterSource()
    {
        System.out.println("WaterSource");
        s = "Constructed";
    }
}
```

1.2 类的继承

Java 中继承的语法和 C++ 类似，不过 Java 中使用关键字 extends 声明。如果继承基类，新类就会得到基类中所有非私有的域和成员函数。例子如下：

```
class Cleanser
{
    private String s = "Cleanser";
    public void append(String a)
    {
        s += a;
    }
    public static void main(String[] args)
    {
        Cleanser x = new Cleanser();
        x.append(" hello world");
    }
}

public class Detergent extends Cleanser
{
    public static void main(String[] args)
    {
        Detergent x = new Detergent();
        x.append(" hello world");
        Cleanser.main(args);
    }
```

```
23      }
```

1.2.1 基类的初始化

如果没有特别声明，将调用基类默认的构造器或者无参数构造器。如果想调用一个带参数的基类构造器，就必须使用 super 显式地调用基类构造器。例子如下：

```
1   class Game
2   {
3       Game(int i)
4       {
5           System.out.println("Hello World");
6       }
7   }
8
9   public class Chess extends Game
10  {
11      Chess()
12      {
13          super(1);
14          System.out.println("Chess constructor");
15      }
16      public static void main(String[] args)
17      {
18          Chess c = new Chess();
19      }
20  }
```

1.2.2 名称屏蔽

与 C++ 不同的是，Java 中导出类如果重载基类中的函数，并不会屏蔽其在基类中该函数的任何版本。例子如下：

```
1   class Homer
2   {
3       char doh(char c)
4       {
5           return c;
6       }
7       float doh(float c)
8       {
9           return c;
10      }
11  }
12
13  class Bart extends Homer
14  {
15      String doh(String s)
16      {
17          return s;
```

```
        }
    }
```

需要注意的是，因为这个语法特点，Java 中其实是没有名称屏蔽的。那么当我们要覆写基类中的一个函数时，很可能将其重载而非覆写。为了防止这个错误的发生，Java 提供了 @Override 注解相应的函数。如果这个函数是重载而非覆写时，编译器就会产生错误：

```
class Lisa extends Homer
{
    @Override
    String doh(String s)
    {
        return s; // 将产生错误
    }
}
```

1.2.3 向上转型

继承技术最重要的不是为新的类提供函数，而是用于表现新类和基类之间的关系。新类是现有类的一种类型。例子如下：

```
class Instrument
{
    public void play(){}
    static void tune(Instrument i)
    {
        i.play();
    }
}

public class Wind extends Instrument
{
    public static void main(String[] args)
    {
        Wind flute = new Wind();
        Instrument.tune(flute);
        // tune函数接受的是Instrument对象
        // 这里它也可以接受Wind对象
        // 因为Wind是Instrument的一种类型
    }
}
```

将导出类引用转换为基类引用的动作称为向上转型。在实现上看，导出类是基类的一个超集。在向上转型的过程中，导出类引用转换为基类引用，并且只保留基类拥有的方法。

导出类无法继承 private 函数。即使在导出类中以相同的名称声明一个函数，也不会覆盖基类中相应的 private 函数，而是生成了一个新的函数。当向上转型时，这个函数将会被丢弃。

1.2.4 继承技术的用途

相对于组合技术，继承技术不常用。只有需要从新类向基类进行向上转型，继承才是必要的。

1.3 final 关键字

final 关键字可以修饰数据、函数和类。

1.3.1 final 数据

Java 中使用 final 告知一块数据是恒定不变的，相当于 C 中的 const 关键字。需要知道的是，Java 中常量必须是基本数据类型。

一个既是 static 又是 final 的域只占据一段不能改变的存储空间。

当用 final 修饰对象引用时，这个引用将恒定不变。也就是说，引用一旦被初始化指向一个对象，就无法再把它改为指向另一个对象，而被引用的对象本身是可以被修改的。Java 没有提供使任何对象恒定不变的途径。

Java 允许生成空白 final。也就是这个域被 final 修饰但又没有赋初值。final 域在使用前必须被初始化。

在函数参数列表中将参数指明为 final，那么在函数中就无法修改参数引用所指向的对象。

1.3.2 final 函数

使用 final 函数的原因如下:

- 将函数锁定。以防任何继承类修改它的实现。
- 追求效率。当一个函数指明为 final，编译器就将该函数的所有调用都转为内嵌调用。这和 C++ 的 inline 关键字的作用一样。

类中 private 方法都隐式地指定为 final。

1.3.3 final 类

当将某个类的整体定义为 final，那么这个类就无法被继承。final 类中的域不一定是 final 的。