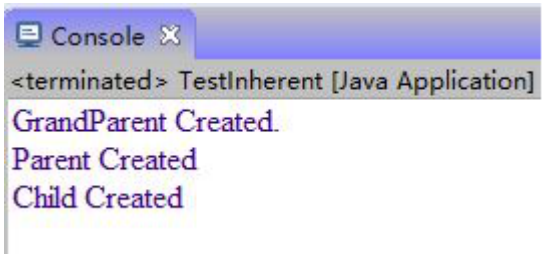
一、运行 TestInherits.java 示例，观察输出，注意总结父类与子类之间构造方法的调用关系修改Parent构造方法的代码，显式调用GrandParent的另一个构造函数，注意这句调用代码是否是第一句，影响重大！

```
class Grandparent {

    public Grandparent() {
        System.out.println("GrandParent Created.");
    }

    public Grandparent(String string) {
        System.out.println("GrandParent Created.String:" + string);
    }
}

class Parent extends Grandparent {

    public Parent() {
        //super("Hello.Grandparent.");
        System.out.println("Parent Created");
       // super("Hello.Grandparent.");
    }
}

class Child extends Parent {

    public Child() {
        System.out.println("Child Created");
    }
}

public class TestInherent {

    public static void main(String args[]) {
        Child c = new Child();
    }
}
```

结果：



总结：通过 super 调用基类构造方法，必须是子类构造方法中的第一个语句。子类的构造方法在运行之前，必须调用父类的构造方法。因为继承是在已有类的基础上，添加新的变量与方法，从而产生一个新的类，子类是必须建立在父类的基础上才能继承，不能反过来。

二、不可变类的实例：Address.java

源程序：

```
public final class Address
{
    private final String detail;
    private final String postCode;

    //在构造方法里初始化两个实例属性
    public Address()
    {
        this.detail = "";
        this.postCode = "";

    }
    public Address(String detail , String postCode)
    {
        this.detail = detail;
        this.postCode = postCode;
    }
    //仅为两个实例属性提供getter方法
    public String getDetail()
    {
         return this.detail;
    }

    public String getPostCode()
    {
         return this.postCode;
    }
    //重写equals方法，判断两个对象是否相等。
    public boolean equals(Object obj)
```

```
    {
        if (obj instanceof Address)
        {
            Address ad = (Address)obj;
            if (this.getDetail().equals(ad.getDetail()) && this.getPostCode().equals(ad.getPostCode()))
            {
                return true;
            }
        }
        return false;
    }
    public int hashCode()
    {
        return detail.hashCode() + postCode.hashCode();
    }
}
```

结果：无结果

总结：不可变的“类”有何用？（1）可以方便和安全地用于多线程环境中，（2）访问它们可以不用加锁，因而能提供较高的性能。

三、参看ExplorationJDKSource.java示例

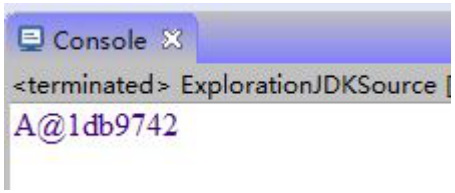
此示例中定义了一个类A，它没有任何成员：

class A { }

源程序：

```
public class ExplorationJDKSource {
    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println(new A());
    }
}
class A{}
```

结果：



总结：前面示例中，main方法实际上调用的是：public void println(Object x)，这一方法内部调用了String类的valueOf方法。valueOf方法内部又调用Object.toString方法：

public String toString() {

return getClass().getName() +"@" +

Integer.toHexString(hashCode());

}

hashCode方法是本地方法，由JVM设计者实现：

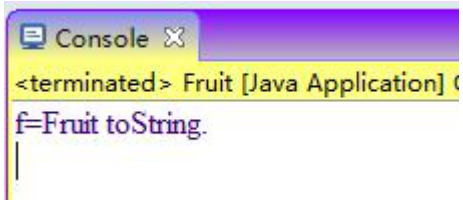
public native int hashCode();

四、神奇的加号

源程序：

```
public class Fruit
{
    public String toString()
    {
        return "Fruit toString.";
    }
    public static void main(String args[])
    {
        Fruit f=new Fruit();
        System.out.println("f="+f);
    //    System.out.println("f="+f.toString());
    }
}
```

结果：



总结：一个字串和一个对象“相加”，得到的结果为字符串，这是因为Fruit类覆盖了Object类的toString方法。 在“+”运算中,当任何一个对象与一个String对象，连接时，会隐式地调用其toString()方法，默认情况下，此方法返回“类名 @ + hashCode”。为了返回有意义的信息，子类可以重写toString()方法。方法覆盖要求子类与父类的方法一模一样，否则就是方法重载（overload）。

五、请自行编写代码测试以下特性（动手动脑）：在子类中，若要调用父类中被覆盖的方法，可以使用super关键字。

源程序：

```
class Grandparent{
    public Grandparent(){//方法的重载
        System.out.println("Grandparent Created.");
    }
    public Grandparent(String string){
        System.out.println("GrandParent Created.String:"+string);
    }
}
class Parent extends Grandparent{
    public Parent(){
        super("Hello.Grandparent");
        System.out.println("Parent Created");
        //super("Hello.Grandparent");
    }
}
class Child extends Parent{
    public Child(){
        System.out.println("Child Created");
    }
}
public class TestInherent {
public static void main(String args[]){
    Child c=new Child();//构造方法
    }
}
```

结果：

总结：在Parent子类中，若要调用父类Grandparent中被覆盖的方法Grandparent()，使用super关键字后，结果如上图。