1 abstract放在关键词class前面来标示。一个抽象类中的抽象方法不能用{}，即不能够实现，只能够声明。如果一个类里包含有一个或多个抽象方法，那么该类必须指定为抽象类，在这里，类的声明受到了方法声明的限制。abstract不能够修饰构造器。并且不能和final连用。

如：

```
abstract class Test{
 abstract void growl(){...}
}
```

显示：编译出错。不能有{..}

如：

```
class Test{
 abstract void growl();
}
```

显示：编译出错。class前要加abstract

如：

```
abstract class Test{
 void growl(){...}
}
```

显示：编译正常。

2 当一个类继承抽象类时，它要么实现所有的抽象方法，要么就声明为abstract类。此种类型有不少关于implements接口实现的题目。

如：

```
abstract class Test
{
 abstract void amethod();
 static int i;
}
public class Mine extends Test
{
 public static void main(String argv[])
 {
  int[] ar = new int[5];
  for(i=0; i<ar.length; i++)
   System.out.println(ar[i]);
 }
}
```

显示：编译错误。必须实现abstract void amethod()方法。

3 用final关键字作为类修饰符的类，均不能被子类重载（子类化），它们的方法可以被子类调用，不能被继承，用extends继承会编译出错。（注意这里是类被修饰final）

如：

```
final class t extends test12
{}
```

```
class e extends t
{}
```
显示：编译错误

类中有final关键字的方法的类可以被子类继承，只是final方法不能够被重载，可以被调用。（注意这里是方法被修饰final）
如：
```
class Test
{
 public final void amethod()
 {
  System.out.println("amethod");
 }
}
public class Fin extends Test
{
 public static void main(String argv[])
 {
  Test b = new Test();
  b.amethod();
 }
}
```
显示：amethod　编译正常
如：
```
class Test
{
 public static void main(String[] args)
 {
  CellPhone cell = new CellPhone();
  cell.emergency();
 }
}
class Phone
{
 final void dial911(){}
}
class CellPhone extends Phone
{
 void emergency()
 {
  dial911();
 }
}
```

显示：编译运行正常

3 private,(default),protected,public它们作用的对象是类，方法，数据。
private保证对象不被子类用到，只允许类本身的访问。
default只能被同一包中的类访问。
protected只允许子类来访问，但子类可以位于不同的包中。
public可以无限制被访问。
如：

```
class Person
{
 private int a;
}
public class Teacher extends Person
{
 pulbic static void main(String [] args)
 {
  int i;
  Person p = new Person();
  i = p.a;
 }
}
```

显示：编译错误 private修饰的，只能够在本身类里面被用到。
private, (default)lprotected,public都可以用来修饰构造器。

在java中与原文件名相同的类名的访问权限一定是public或default。java中规定一个非内部即顶层类的访问修饰符只能使用public（默认应也可），不能把private和protect应用于顶层类。
如：

```
protected class Fred
{
 private int x = 0;
 private Fred (int xval)
 {
  x = xval;
 }
}
```

显示：编译错误
4 来自于一个类的实例可以相互访问
如：

```
class Test
{
 public static void main(String args[])
 {
  AClass ref1 = new AClass(5);
```

```
  AClass ref2 = new AClass(10);
  ref1.getAndShow(ref2);
  System.out.println(ref1.add(ref2));
 }
}
class AClass
{
 private int x;
 AClass(int x)
 {
  this.x = x;
 }
 void getAndShow(AClass ref)
 {
  System.out.print(ref.x );
 }
 int add(AClass ref)
 {
  return ref.x + x;
 }
}
```

显示：10　15

5 static静态方法只能访问被声明为静态的类属性(类属性指在类中声明的变量)，而不能直接访问非静态的类属性。静态方法要访问非静态方法只能通过创建一个类实例后，对其访问。

如：

```
class Test
{
 private int m;
 static int n;
 public static void fun(){..}
}
```

fun()可以访问n，不能访问m

如：

```
public class Text
{
 public static void main(String arguments[])
 {
  amethod(arguments);
 }
 public void amethod(String[] arguments)
 {
```

```
  System.out.println(arguments);
  System.out.println(arguments[1]);
 }
}
```
显示：编译错误。不能直接调用amethod，必须建立对象再使用。

在静态方法中，不能够使用this。
如：
```
class test implements Runnable
{
 public static void main(String[] args)
 {
  Thread t = new Thread(this);  //(this)出错。没有创建对象，无法确知。
  t.start();
 }
 public void run()
 {
  System.out.println("hi");
 }
}
```
显示：编译错误
如：
```
public class Test
{
 static int a = 100;
 public static void main(String [] args)
 {
  System.out.println(this.a);
 }
}
```
显示：编译错误

如：
```
class test13
{
 String s = "Hello";

 public static void main(String args[])
 {
  test13 h = new test13();
  h.methodA(s);  //参数s有问题，应为h.s;而methodA前的h.可以去掉
  System.out.println(h.s);
  System.out.println(h.s.replace('H','e'));
```

```
 }
 public static void methodA(String s) //没有作用
 {
  s.replace('l','e');  //String replace(char oldChar, char newChar)
              //Returns a new string resulting from replacing all occurrences
        //of oldChar in this string with newChar.
        //因此没有作用，关于参数传递的问题

  s+="World!!!";  //这个是没有作用的，关于参数的传递问题
 }
}
```

显示：编译错误

静态方法的调用是在编译期决定的，和类中的变量有点相似。

如：

```
class MyTest
{
 public void myTest()
 {
  System.out.println("Printing myTest in MyTest class");
 }
 public static void myStat()
 {
  System.out.println("Printing myStat in MyTest class");
 }
}
public class Test extends MyTest
{
 public void myTest()
 {
  System.out.println("Printing myTest in Test class");
 }
 public static void myStat()
 {
  System.out.println("Printing myStat in Test class");
 }

 public static void main(String args[])
 {
  MyTest mt = new Test();
  mt.myTest();
  mt.myStat();
 }
```

}
显示：Printing myTest in Test class　　Print myStat in MyTest class

静态static静待块代码不是一个方法，仅执行一次，首次装载类时被执行。并且先于构造器执行。
如：
```
public class test15
{
 static
 {
  System.out.println("Hi there");
 }
 public test15()
 {
  System.out.println("Hello");
 }
 public static void main(String args[])
 {
  new test15();
  new test15();
 }
}
```
显示：Hi there　Hello　Hello

6 本地方法的声明(所谓本地方法，就是采用非java语言编写的依赖本地平台的方法体。本地方法在java中只能访问，而不能写)，用native. 一个声明为本地的方法只需在方法名前加上修饰符native即可。native修饰符只能用于方法，不能用于类和属性。本地方法如同一个抽象的方法，并没有实际的方法体，因此不能定义方法体。
native的位置：public static native void test(参数);
如：
```
public native void test(){}
public void native test()
public native test(){}
```
错误。
因此native和abstract一样，不能够用来修饰构造器。
比如：
```
class a{
abstract a();
}
```
编译错误
比如：
class a{native a();}编译错误