第一题：

不能通过编译，原因如下。

类 A 中只定义了有参的构造函数，而它的子类 B 的构造函数里面没有显示地调用 A 的有参构造，所以默认地调用 A 的无参构造函数，但由于 A 没有无参构造函数，所以会出错。修改方式如下图：

```
1
2  class A {
3   int a;
4   A(int a) {
5       this.a = a;
6   }
7  }
8
9  public class B extends A {
10  int b;
11  B(int b) {
12      super(5);
13      this.b = b;
14  }
15
16  B(int a, int b) {
17      super(a);
18      super.a = a;
19      this.b = b;
20  }
21
```

或者给 A 定义一个无参构造函数。

第二题：

不能通过编译，因为两个函数都不是抽象函数 abstract，又都没有函数体。

如果给两个函数都加上函数体的话就可以通过编译，虽然父类中的 test 函数是 final，但它是私有的，而子类是无法继承父类的私有函数，所以对于子类来说，它并不是继承重载了父类的 test，而是定义了一个自己的 test，这样是不会与父类中 test 函数的 final 属性冲突的。

第三题：

final：final 修饰的类不能被继承，修饰的变量不能被修改，赋值只能一次，在初始化的时候进行。final 修饰的方法不能被子类重载。

static：静态修饰符，在类中，被 static 修饰的成员变量和方法在程序编译时便分配好了空间并初始化，之后该类所有的对象共用一个 static 变量，并且 static 方法只能调用 static 变量。

abstract：一个成员方法定义为 abstract 后就不能写函数体，并且一个类中如果存在一个抽象的成员方法，那么该类也必须定义为抽象类。一个子类继承了有抽象方法的父类，那么这个子类就必须把抽象方法补全。并且抽象的类是不能够直接 new 出对象的，但是可以定义一个抽象类的引用指向已经实现的子类的对象。

private：私有的成员变量或方法只能在类里面被调用，并且不能被子类继承。

protected：被 protected 修饰的成员变量或方法只能在类里面被使用，但是可以被子类继承。

public：被 public 修饰的成员变量或者方法是公开的，可以在任何地方被使用。每个 java 文件最多只能有一个 public 类，main 方法得写在 public 类里，并且文件名必须与 public 类名相同。
default：当没有修饰时即为默认的，默认的成员变量和方法只能在同一个包中被调用，一个默认的类也是只能在同一个包中被调用。
class：class 表示类，当一个变量被 class 修饰的时候，说明这个变量是类变量。
enum：enum 表示枚举，可以看成是一种特殊的类，一样可以继承父类，一样可以继承接口，一样可以设置有成员变量，但是枚举实例的定义必须放在最前面，而且不能 new。
extends：当一个子类要继承一个父类时，就用 extends 关键字。java 中一个类只能继承一个父类。
interface：接口的标志。接口中的成员变量只能是 static 并且是 final 的，方法也只能是 static。一个类可以继承多个接口。接口可以声明引用类型的变量，但不能用于创建实例对象，这点跟抽象函数类似。

第四题：
外部类的访问权限只能是 public 或者 default，是因为外部类是用来使用的，包中或者包外需要声明并定义这个类，如果声明为私有，那这个类就没有人能够创建它，也就没有了意义。内部类的话，如果是 public，那么外部可以直接调用，子类也可以继承，default 的话同包的其他类可以调用也能被继承，protected 的话外部不能调用但子类可以继承，private 外部不能调用子类也不能继承，但外部类本身可以调用，所以各种访问权限都有它的意义，所以内部类可以使用所有的访问修饰符。

第五题：
先吐槽一句，真的好难!!!
因为不懂的地方实在太多了，所以我只能简单说一下主要的我能看得懂的部分，而且不一定理解得对。。。。

```
     * @serial
     */
    final int signum;

    /**
     * The magnitude of this BigInteger, in <i>big-e
     * zeroth element of this array is the most-sign
     * magnitude.  The magnitude must be "minimal" i
     * int ({@code mag[0]}) must be non-zero.  This
     * ensure that there is exactly one representati
     * value.  Note that this implies that the BigIn
     * zero-length mag array.
     */
    final int[] mag;
```

上面的两个 int，signum 代表的是这个数的正负，它有三个值 0，1，-1，分别代表 0 和正负数。而 mag[]数组应该就是用来储存这个大数的，大整数的原理应该是把这个大数拆成好几

部分，每一部分分别存在 mag 数组里。
我刚开始的想法是 mag 的每一个存储大整数的一位，但好像很浪费，但是代码部分又看不太懂，然后我查了一下，网上说“每个 int 值大小范围是-2^31 至 2^31-1 即-2147483648~2147483647,因此一个 int 值最多可保存一个 10 位十进制的整数,但是为了防止超出范围(2222222222 这样的数 int 已经无法存储),保险的方式就是每个 int 保存 9 位的十进制整数”。也就是说 mag 的每一位存储 9 位数。
因为之前 c++的大整数都是用 string 去存储，所以我主要看了 public BigInteger(String val, int radix)这个构造函数，比起其他构造函数，这个相对好理解一点。

```
if (radix < Character.MIN_RADIX || radix > Character.MAX_RADIX)
    throw new NumberFormatException("Radix out of range");
if (len == 0)
    throw new NumberFormatException("Zero length BigInteger");
```

这部分比较好理解，radix 应该是代表进制数，判断进制数是否越界还有长度是否为零并抛出错误。

```
// Check for at most one leading sign
int sign = 1;
int index1 = val.lastIndexOf('-');
int index2 = val.lastIndexOf('+');
if (index1 >= 0) {
    if (index1 != 0 || index2 >= 0) {
        throw new NumberFormatException("Illegal embedded sign character");
    }
    sign = -1;
    cursor = 1;
} else if (index2 >= 0) {
    if (index2 != 0) {
        throw new NumberFormatException("Illegal embedded sign character");
    }
    cursor = 1;
}
if (cursor == len)
    throw new NumberFormatException("Zero length BigInteger");
```

这个部分是找出那个字符串中+-号的位置，因为找不到的话是返回-1，若两者都为正代表这个字符串里面同时存在+-，抛出非法。len 是字符串的长度，而 cursor 应该是目前查找到的长度，如果一开始就跟 len 等长，就抛出大整数长度是 0 的异常。

```
// Skip leading zeros and compute number of digits in magnitude
while (cursor < len &&
       Character.digit(val.charAt(cursor), radix) == 0) {
    cursor++;
}

if (cursor == len) {
    signum = 0;
    mag = ZERO.mag;
    return;
}

numDigits = len - cursor;
signum = sign;
```

然后一直找，直到第一个不为 0，如果这个时候 cursor 等于 len，说明全部都为 0，此时把 mag 赋为 ZERO.mag（这个不太理解），然后 numDigits 应该是实际的有效数字。

```
long numBits = ((numDigits * bitsPerDigit[radix]) >>> 10) + 1;
if (numBits + 31 >= (1L << 32)) {
    reportOverflow();
}
int numWords = (int) (numBits + 31) >>> 5;
int[] magnitude = new int[numWords];
```

移位运算那部分看不太懂，好像是最后得到存储这个数所需要的位数。

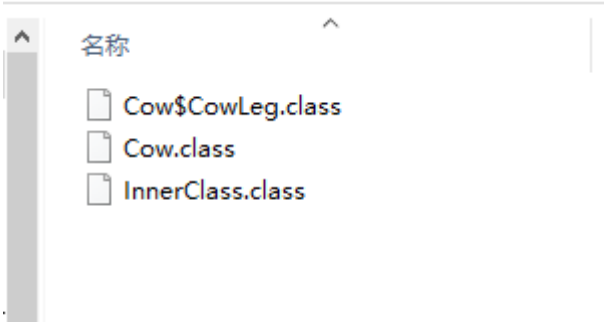
```
// Process first (potentially short) digit group
int firstGroupLen = numDigits % digitsPerInt[radix];
if (firstGroupLen == 0)
    firstGroupLen = digitsPerInt[radix];
String group = val.substring(cursor, cursor += firstGroupLen);
magnitude[numWords - 1] = Integer.parseInt(group, radix);
if (magnitude[numWords - 1] < 0)
    throw new NumberFormatException("Illegal digit");
```

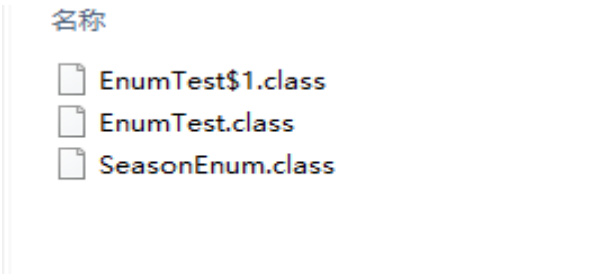
这里好像是把切割开的第一段数据放进数组里面。

```
// Process remaining digit groups
int superRadix = intRadix[radix];
int groupVal = 0;
while (cursor < len) {
    group = val.substring(cursor, cursor += digitsPerInt[radix]);
    groupVal = Integer.parseInt(group, radix);
    if (groupVal < 0)
        throw new NumberFormatException("Illegal digit");
    destructiveMulAdd(magnitude, superRadix, groupVal);
}
// Required for cases where the array was overallocated.
mag = trustedStripLeadingZeroInts(magnitude);
if (mag.length >= MAX_MAG_LENGTH) {
    checkRange();
}
```

最后这里把剩下的一个一个转换进去。

第六题：





上面两张图是我 javac 之后生成的文件，从上可以看出，命名规则应该是，外部类的命名规则跟普通类是一样的，都是“类名.class”，而内部类应该是“外部类名+$+内部类名.class”。而当这个内部类是一个匿名类的时候，以上格式的内部类名就用数字编号替代，第一个为 1，第二个为 2，以此类推。