ITEM 14 :

Date:

Be careful about copying behaviour in resource managing classes.

Since auto-ptr & shared-ptr are generally inappropriate as resource handlers, we may need to create your own resource managing classes.

eg > class lock which acquire mutex

```
Class Lock {
  public :
    explicit lock (Mutex * pm): mutexPtr(pm)
        { lock (mutexptr); }
  ~Lock() { unlock (mutexPtr) ; }
  private
      Mutex  *mutexPtr ;
};

     Lock m1 (&m);   // lock m
     lock m2 (m1) ;  // what should happen
                       here? copy m1 to m2?
```

① Prohibit copying.
  by using uncopyable as the parent class.

Date:..............................

```
class Lock : private Uncopyable .
  { public:

  }
```

② Reference count the underlying resource.

To hold onto a resource until last object using it has been destroyed.

If Lock wanted to employ reference counting it should change the type of mutexPtr from Mutex * to

tr1 :: shared-ptr < Mutex >

To avoid delete of mutex before unlocking tr1 :: shared-ptr allows specification of a deleter - a function / function object to be called when reference call goes to zero.

Date:.........................

③ Copying underlying resource.

Copying a resource managing object performs a deep copy.

When a string object is copied, a copy is made of both the pointer & the memory it points to.

④ Transfer ownership of underlying resource

~~Request~~ Resource Acquisition is Initializatn (RAII)

To make sure only one RAII object refers to a raw resource & that when the RAII object is copied, ownership of resource is transferred from copied object to the copying object!

So unless compiler generated version of copy ctor & copy assignment operator will do what you want, you will need to write them yourself.