Garbage Collection

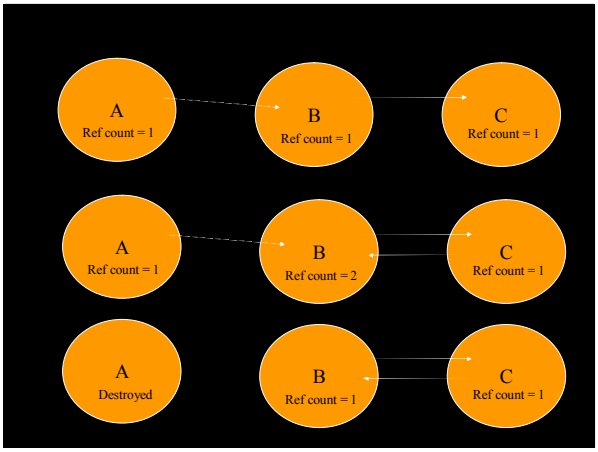
Correct Way to use Objects

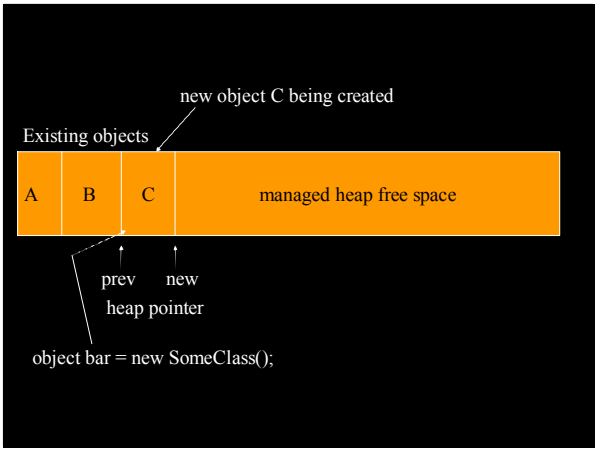
```
public void AFunction()
{
  //create a new object, allocating memory from heap

  SomeClass *p = new SomeClass();

  //use object for something

  WorkWith (p);
  //delete the object, returning object's memory to the heap

  delete p;
}
```

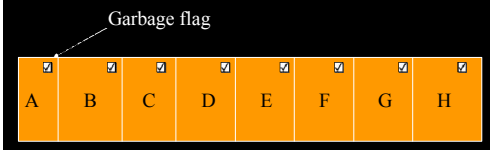
Incorrect way to use Objects

```
public void AFunc()
 {
  //create a new object, allocating memory from program heap

  SomeClass *p = new SomeClass();

  //use the object to do something

  WorkWith (p);

  // Bug: Failure to delete object means the object's memory
  // never goes back to the heap. The pointer goes out of scope
  // so we can never release the block. The memory has been
  // leaked away.
}
```

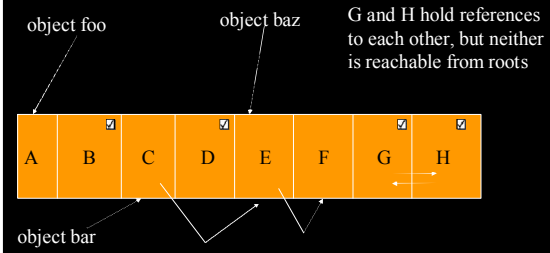
```
SomeClass *p2;
public void AFunc()
{
  // create new object from program heap
  SomeClass *p = new SomeClass();
  // make a copy of the reference
  p2 = p;
  // use object to do something
  WorkWith (pFoo);
  // be a good programmer and release memory
  delete p;
}
public void AnotherFunc()
{ // incorrectly use object reference that another part of
  // program has deleted.
  WorkWith (p2);  // pray for a crash
}
```



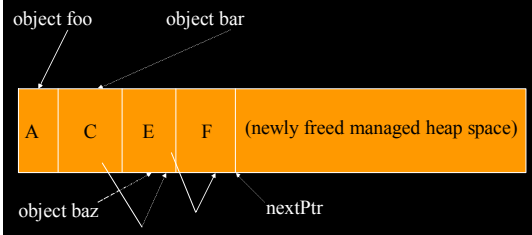





```
System.GC.Collect();  // triggers a collection

System.GC.Collect();
System.GC.WaitForPendingFinalizers();
```



Garbage collector follows program roots, clears garbage flag on reachable objects



Garbage collector compacts heap, fixes up remaining program references to each object's new location in heap

```
// Override System.Object.Finalize

public MyClass
{
  protected void override Finalize()
  {
    //clean up code
    base.Finalize();
  }
}
```

Short hand for writing a finalizer

```
public MyClass
{
   ~MyClass()
    {
      // clean up code
      // call to base.Finalize() is automatic
    }
}
```

Finalizer Warnings

1) You never know when a finalizer will run.

2) You never know the order in which a set of objects' finalizers will be called.

3) Objects with finalizers are automatically promoted To the next garbage collection generation.

4) You never know for certain that a finalizer will run.

```
public class Class1 : SomeBase, IDisposable
{
 private bool bDisposed = false;  // have we been disposed?
 public void Dispose()
 {
   // do whatever we need to do to release resources
   // if base class has a Dispose(), call it

   base.Dispose();

   // set our own internal flag to know we have been disposed
   bDisposed = true;

   // mark our object as no longer needing finalization
   System.GC.SuppressFinalize(this);
 }
}
```

```
public class Class1 : SomeBase, IDisposable
{
 private bool bDisposed;
 ~Class1()   // finalizer calls Dispose(false)
 {  Dispose(false) ;  // call actual cleanup method }

 // a public method so can be called by a client of this object
 public void Dispose() {
   // just call common cleanup method
   Dispose(true);
 }

 // this public method may be called instead of Dispose()
 public void Close()
 {  Dispose(true);  // call actual cleanup method }
```

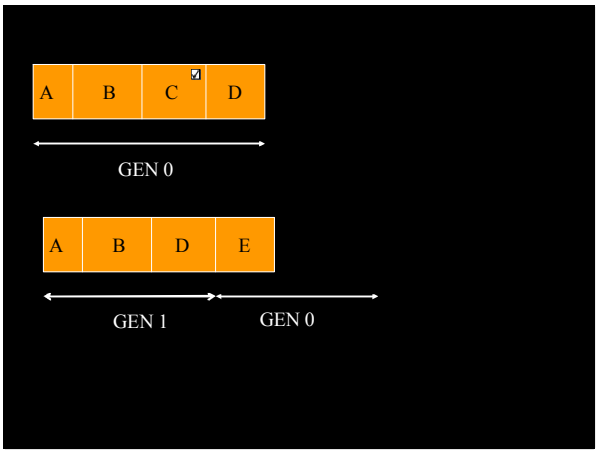
```
private void Dispose(bool disposing)
{ // Common method that does the cleanup, called from either
  // Dispose() or Close()
  // Synchronize threads calling Dispose/Close simultaneously

   lock (this)
   {
    if (disposing)
    {  // object is being explicitly Disposed/Closed, not finalized
       System.GC.SuppressFinalize(this);  // don't need it to run
       // clean up code
       bDisposed = true;  // we have been disposed
    }
   }
}
```

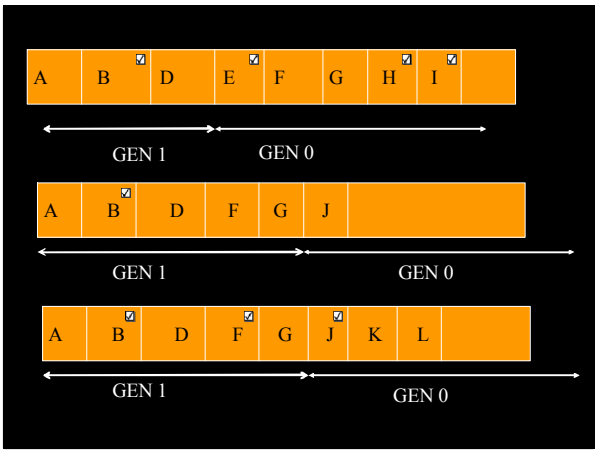
```
// create object that supports IDisposable
object o = new SomeClass();

// perform some operation on the object
try
{
  DoSomethingWith (o);
}
// finally block ensures we call Dispose, even if an exception
finally
{
   o.Dispose();
}
```

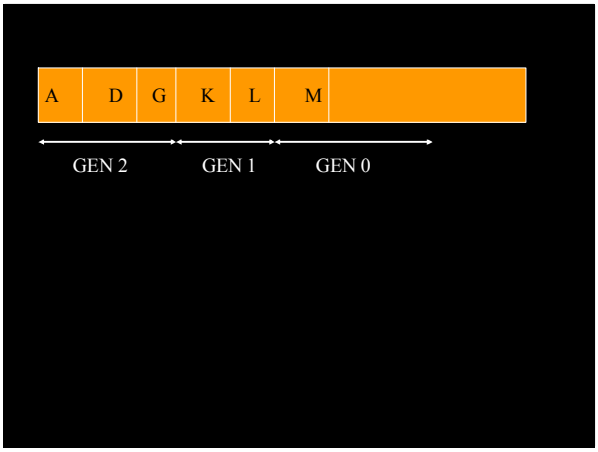
```
//create object that supports IDisposable, but use using

using (object o = new SomeClass())
{
  DoSomethingWith (o);
}

 Compiler will insert the

 try {}
 finally{}

 where Dispose() will be called
```

Generations

- Generation 0 – newest objects
- Generation 1 – older objects
- Generation 2 – oldest objects


A
B
C
D
GEN 0
A
B
D
E
GEN 1
GEN 0


A
B
D
E
F
G
H
I
GEN 1
GEN 0
A
B
D
F
G
J
GEN 1
GEN 0
A
B
D
F
G
J
K
L
GEN 1
GEN 0


A
D
G
K
L
M
GEN 2
GEN 1
GEN 0