The .NET Framework's garbage collector manages the allocation and release of memory for your application. Each time you create a new object, the common language runtime allocates memory for the object from the managed heap. As long as address space is available in the managed heap, the runtime continues to allocate space for new objects. However, memory is not infinite. Eventually the garbage collector must perform a collection in order to free some memory. The garbage collector's optimizing engine determines the best time to perform a collection, based upon the allocations being made. When the garbage collector performs a collection, it checks for objects in the managed heap that are no longer being used by the application and performs the necessary operations to reclaim their memory.

Mark and Sweep Algorithm

Marking Phase

When a garbage collection starts, it looks at a set of references called the ‘GC roots’. These are memory locations that are designated to be always reachable for some reason, and which contain references to objects created by the program. It marks these objects as ‘live’ and then looks at any objects that they reference; it marks these as being ‘live’ too. It continues in this manner, iterating through all of the objects it knows are ‘live’. It marks anything that they reference as also being used until it can find no further objects.

Sweep Phase

Once all of these live objects are known, any remaining objects can be discarded and the space re-used for new objects. .NET compacts memory so that there are no gaps (effectively squashing the discarded objects out of existence) – this means that free memory is always located at the end of a heap and makes allocating new objects very fast.

Finalization(Destructor is implemented)

**Finalization**

When an object of a class is created in the heap that implements a finalize(destructor) method, a pointer to the object is stored in the finalization queue.  The garbage collector periodically scans this finalization queue to get the pointers.  When it identifies one, it removes the same from the finalization queue and adds the pointer to the reachable queue.  Then the finalize method is called on the object and the reachable queue is emptied.

Generation

A generational garbage collector collects the short-lived objects more frequently than the longer lived ones.  Short-lived objects are stored in the first generation, generation 0.  The longer-lived objects are pushed into the higher generations, 1 or 2.  The garbage collector works more frequently in the lower generations than in the higher ones.

When an object is first created, it is put into generation 0.  When the generation 0 is filled up, the garbage collector is invoked. The objects that survive the garbage collection in the first generation are promoted onto the next higher generation, generation 1.  The objects that survive garbage collection in generation 1 are promoted onto the next and the highest generation, generation 2.  This algorithm works efficiently for garbage collection of objects, as it is fast.  Note that generation 2 is the highest generation that is supported by the garbage collector.

1. Objects NOT implementing Finalize methods, there Memory is reclaimed immediately,unless of course, they are not reacheable by  
   application code anymore
2. Objects implementing Finalize Method, The Concept/Implementation of Application Roots, Finalization Queue, Freacheable Queue comes before they can be reclaimed.
3. Any object is considered garbage if it is NOT reacheable by Application Code

Classes/Objects A, B, D, G, H do NOT implement Finalize Method and C, E, F, I, J implement Finalize Method.

When an application creates a new object, the new operator allocates the memory from the heap. ***If the object's type contains a Finalize method, then a pointer to the object is placed on the finalization queue***.  
  
*therefore pointers to objects C, E, F, I, J gets added to finalization queue.*