1. **What is the difference between a managed resource and an unmanaged resource?**

A managed resource is something managed by the CLR. Unmanaged resources are not managed by the CLR and need to be dropped for disposal.

1. **When is memory for an object (reference type) allocated? When is the memory deallocated?**

Memory for an object is allocated when "new" is used. When you create the new object a chunk of raw memory is carved out of the heap. The memory is deallocated once the object is out of scope. You can do this manually with a destructor.

1. **What is a destructor?**

A destructor is a method that the CLR calls after references to an object have disappeared. It is a special method of a class that is used to destroy the object or instance of a class when it is no longer needed. It clears out the heap.

1. **What is the difference in syntax between a constructor and a destructor?**

The tilde (~) followed by the name of the class is the symbol for a destructor. Also the constructor is in the beginning when an instance of a class is created versus the destructor which comes at the end when the object or instance is no longer needed. Destructors apply only to reference types; you cannot declare a destructor in a value type. You cannot specify an access modifier (such as public) for a destructor. You never call the destructor in your own code; part of the CLR called the garbage collector does this for you. Destructors cannot accept parameters and cannot list access types.

1. **Give some examples of scarce resources. Why would you want to manage scarce resources?**

Scarce resources may include memory, database connections, bandwidth, and connection to a printer. These resources are accessed by many programs, and possibly other users. They must be managed appropriately because they are too valuable to be sitting around waiting for a machine to release them.

1. **What is exception-safe disposal?**

Exception-safe disposal refers to ensuring disposal occurs even if an exception is thrown in the middle of a process. A finally block can be a solution to this.

1. **How do you think that the using statement works for resource management? Give an informal, English language, explanation of how it works.**

Using statements define scope, just like namespaces or classes. They ensure that resources are freed when the block runs and the resources drop out of scope.

1. **What ill effects could result from attempting to dispose of a resource more than once?**

Disposing of the resources held by an object more than once might or might not be disastrous, but it is definitely not good practice. You don't always know when the garbage collector is going to run on its own and if you call a dispose method and they run simultaneously you can lose data and application performance. The recommended approach to resolving this problem is to add a private Boolean field to the class to indicate whether the Dispose method has already been invoked, and then examine this field in the Dispose method.

1. **We will look at threads later in the term. For now, what is your understanding of how threads interact with resource management? A good guess is a sufficient answer to this question.**

Threads are paths executing code that can run concurrently while you work on an application. A resource could be accessed by multiple threads depending on how the compiler determines which processes execute most efficiently. The garbage collector is one of them. Disposal of a resource while multiple threads are accessing it could result in serious errors.

1. **Why does the book recommend not attempting to force the garbage collector? Are their any exceptions to this recommendation?**

The GC.Collect method starts the garbage collector, but the process runs asynchronously—the GC.Collect method does not wait for garbage collection to be complete before it returns, so you still don’t know whether your objects have been destroyed.