Answer the discussion questions in writing.

1. What is the difference between deep copy and shallow copy?

Deep copy is every reference type has an associated clone method so that the actual contents of the of the objects and fields are copied.  
Shallow copy is when a clone method simply copies references to a location in memory.

1. What is the value of a reference after you declare and initialize it?

When you declare a reference, the actual value held by the reference is the address of an object in memory. For example, when a method instantiates the WrappedInt class with ‘wi’, ‘wi.Number’ is initialized to 0 by the compiler-generated default constructor.

1. How do you declare a value type?

State whether the value is public or private; state the type; state the variable name. For example:

public int number;

1. How do you declare a reference type?

First create a new instance of a class and invoke its default constructor. Then you can declare a reference type. For example:  
Circle a = new Circle()  
a.Number;

1. Does C# allow you to assign NULL to a value type?

No, but you can declare it as a nullable value type. For example:

int? i = null.

1. Can you assign a nullable value type to a non-nullable variable of the same type? Why or why not?

No, because the nullable variable might contain null and the non-nullable variable cannot contain null. But you can assign a non-nullable variable to a nullable value type. For example:  
  
int? i = null;   
int j = 99;   
i = 100; // legal  
i = j; // legal  
j = i; // illegal

1. What is the difference between the stack and the heap?

Stack: all value types are created on the stack, and all references to objects are on the stack. When you call a method, the memory required for its parameters and its local variables is always acquired from the stack.

Heap: All reference types (objects) are created on the heap. When you create an object by using the ‘new’ keyword, the memory required to build the object is always acquired from the heap. The same object can be referenced from several places by using reference variables.

1. What does it mean when we say that all classes are specialized types?

All classes are specialized types because they inherit their properties from System.Object and you can use System.Object or just ‘object’ to create a variable that can refer to any reference type

1. What does ref do?

Ref tells the compiler to generate code that passes a reference to the actual argument rather than a copy of the argument. Anything you do to the parameter you also do to the original argument.

1. What does out do?

‘Out’ is similar to ref in that anything you do to the parameter you also do to the original argument. However, with ‘ref’ you cannot pass uninitialized values as an argument to a method, whereas with ‘out’ you can pass uninitialized values because the method must assign a value to it before it finishes or returns.

1. Describe boxing and unboxing in your own words.

Boxing occurs automatically when an object tries to reference a value type; a copy of the value type is created on the heap so that the object does not reference the stack. For example:

int i = 42; // allocates memory on the stack   
object o = i; // the runtime allocates a piece of memory from the heap, copies the value of integer i to this piece of memory, and then refers the object o to this copy.

Unboxing is when an object refers to the same value type as another value type variable, and the compiler extracts the value form the boxed object and copies it to the value type variable. For example:

int i = 42;  
object o = i; // boxing  
int j = (int) o; // compiler checks if object o is an int, if so it unboxes 42 and copies it to int j.

1. What does cast do?

Cast checks whether converting an item of one type to another is safe before actually making the copy. If safe, the first item is unboxed and copied to the second item.