**Memory Allocation– Part 1**

**Slide 1**

In the first part of this presentation on memory allocation we will discuss the approach taken by various languages with regard to whether data is allocated on the stack or in the heap.

**Slide 2**

Different languages have different approaches.

The traditional approach to memory allocation is that used by C, C++, Pascal and Ada. In those languages all variables are declared on the stack, only pointers point to the heap.

Newer languages take a different approach. In Java, primitives and object references are allocated on the stack, but objects and arrays are allocated on the heap.

C# makes a distinction between value types and references type. Primitives and objects of a struct type are value types and arrays and objects of class types are reference types. Value types are allocated on the stack, whereas reference types are allocated on the heap.

**Slide 3**

Next we consider an important distinction that must be considered whenever data is allocated on the heap.

Whenever pointers or references are used, it becomes necessary to distinguish between shallow and deep copies and comparisons.

Shallow copies and comparisons only involve the pointer or reference. A shallow copy creates what is referred to as an alias, two names that refer to the same data.

In contrast, deep copies and comparisons involve the object pointed to.

**Slide 4**

To illustrate these differences we will examine an example in Java, that both shallow and deep copies an object. Then we will perform shallow comparisons between the original object and the copy.

This first statement declares reference p to a Point object and initializes that reference to an object containing 0, 0.

Notice that the reference is allocated on the stack and the object that it points to on the heap.

This next statement declares another reference to a Point object, we have named q.

Assignments are shallow copies, which means that only the reference is copied. So now we have two references that point to the same object.

In this statement we declare a third reference r, and assign it to the result of invoking the clone method on the original object p.

The clone method makes a deep copy, which creates an identical object on the heap, which r now points to.

Next we will perform two comparisons using the relational operator == that performs a shallow comparison.

In the first case, we compare p and q. Because a shallow comparison compares the references and the two references refer to the same object, the result will be true.

In the next case we compare p and r. Recall that r is a deep copy of p, and so it points to an identical but different object.

In spite of the fact that the objects are identical, the references are different, so the result is false. A common mistake when comparing Java strings is to use a shallow comparison, which will give the wrong result when two identical objects are compared.

**Slide 5**

We will again both shallow and deep copy an object. Then this time we will perform deep comparisons between the original object and the copy.

So we begin with the same three initial statements. The first declares p and initializes it. The second declares q and assigns it to p, which is a shallow copy, and the third declares r and assigns it to result of cloning p, which produces a deep copy.

This time we will use the equals method that performs a deep comparison and compare p with q.

In this case both the objects and references are the same, so the comparison succeeds, and true is displayed.

In the next case, we again use the equals method but compare p and r.

Although the references are different, the objects are the same, so this comparison, unlike when we used a shallow comparison, will also succeed, so again another true will be displayed.