**Overload Constructors:**

* Like methods, constructors can also be overloaded.
* Doing so allows you to construct objects in a variety of ways.
* For example, consider the following program:

// Demonstrate an overloaded constructor.

using System;

class MyClass

{

public int x;

public MyClass()

{

Console.WriteLine("Inside MyClass().");

x = 0;

}

public MyClass(int i)

{

Console.WriteLine("Inside MyClass(int).");

x = i;

}

public MyClass(double d)

{

Console.WriteLine("Inside MyClass(double).");

x = (int) d;

}

public MyClass(int i, int j)

{

Console.WriteLine("Inside MyClass(int, int).");

x = i \* j;

}

}

class OverloadConsDemo

{

public static void Main()

{

MyClass t1 = new MyClass();

MyClass t2 = new MyClass(88);

MyClass t3 = new MyClass(17.23);

MyClass t4 = new MyClass(2, 4);

Console.WriteLine("t1.x: " + t1.x);

Console.WriteLine("t2.x: " + t2.x);

Console.WriteLine("t3.x: " + t3.x);

Console.WriteLine("t4.x: " + t4.x);

}

}

**The output from the program is shown here:**

Inside MyClass().

Inside MyClass(int).

Inside MyClass(double).

Inside MyClass(int, int).

t1.x: 0

t2.x: 88

t3.x: 17

t4.x: 8

* **MyClass( )** is overloaded four ways, each constructing an object differently.
* The proper constructor is called based upon the arguments specified when **new** is executed.
* By overloading a class’ constructor, you give the user of your class flexibility in the way objects are constructed.
* One of the most common reasons that constructors are overloaded is to allow one object to initialize another.
* For example, here is an enhanced version of the **Stack** class developed earlier that allows one stack to be constructed from another:

// A stack class for characters.

using System;

class Stack

{

// These members are private.

char[] stck; // holds the stack

int tos; // index of the top of the stack

// Construct an empty Stack given its size.

public Stack(int size)

{

stck = new char[size]; // allocate memory for stack

tos = 0;

}

// Construct a Stack from a stack.

public Stack(Stack ob)

{

// Allocate memory for stack.

stck = new char[ob.stck.Length];

// Copy elements to new stack.

for(int i=0; i < ob.tos; i++)

stck[i] = ob.stck[i];

// Set tos for new stack.

tos = ob.tos;

}

// Push characters onto the stack.

public void Push(char ch)

{

if(tos==stck.Length)

{

Console.WriteLine(" -- Stack is full.");

return;

}

stck[tos] = ch;

tos++;

}

// Pop a character from the stack.

public char Pop()

{

if(tos==0)

{

Console.WriteLine(" -- Stack is empty.");

return (char) 0;

}

tos--;

return stck[tos];

}

// Return true if the stack is full.

public bool IsFull()

{

return tos==stck.Length;

}

// Return true if the stack is empty.

public bool IsEmpty()

{

return tos==0;

}

// Return total capacity of the stack.

public int Capacity()

{

return stck.Length;

}

// Return number of objects currently on the stack.

public int GetNum()

{

return tos;

}

}

// Demonstrate the Stack class.

class StackDemo

{

public static void Main()

{

Stack stk1 = new Stack(10);

char ch;

int i;

// Put some characters into stk1.

Console.WriteLine("Push A through J onto stk1.");

for(i=0; !stk1.IsFull(); i++)

stk1.Push((char) ('A' + i));

// Create a copy of stck1.

Stack stk2 = new Stack(stk1);

// Display the contents of stk1.

Console.Write("Contents of stk1: ");

while( !stk1.IsEmpty() )

{

ch = stk1.Pop();

Console.Write(ch);

}

Console.WriteLine();

Console.Write("Contents of stk2: ");

while ( !stk2.IsEmpty() )

{

ch = stk2.Pop();

Console.Write(ch);

}

Console.WriteLine("\n");

}

}

**The output is shown here:**

Push A through J onto stk1.

Contents of stk1: JIHGFEDCBA

Contents of stk2: JIHGFEDCBA

* In **StackDemo**, the first stack, **stk1**, is constructed and filled with characters.
* This stack is then used to construct the second stack, **stk2**.
* This causes the following **Stack** constructor to be executed:

// Construct a Stack from a stack.

public Stack(Stack ob)

{

// Allocate memory for stack.

stck = new char[ob.stck.Length];

// Copy elements to new stack.

for(int i=0; i < ob.tos; i++)

stck[i] = ob.stck[i];

// Set tos for new stack.

tos = ob.tos;

}

* Inside this constructor, an array is allocated that is long enough to hold the elements
* contained in the stack passed in **ob**.
* Then, the contents of **ob**’s array are copied to the new array, and **tos** is set appropriately.
* After the constructor finishes, the new stack and the original stack are separate, but identical.