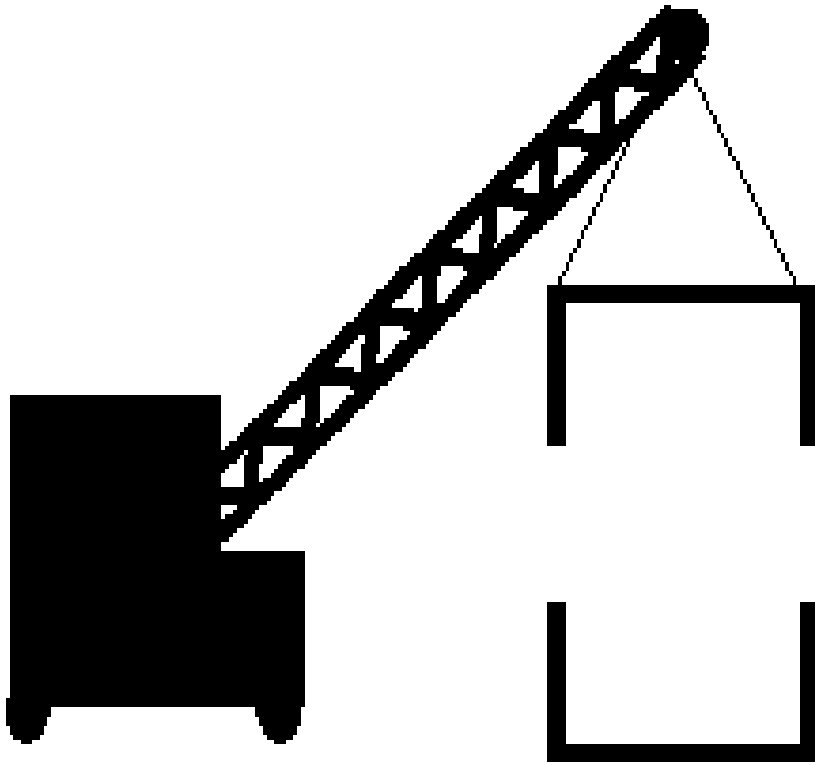
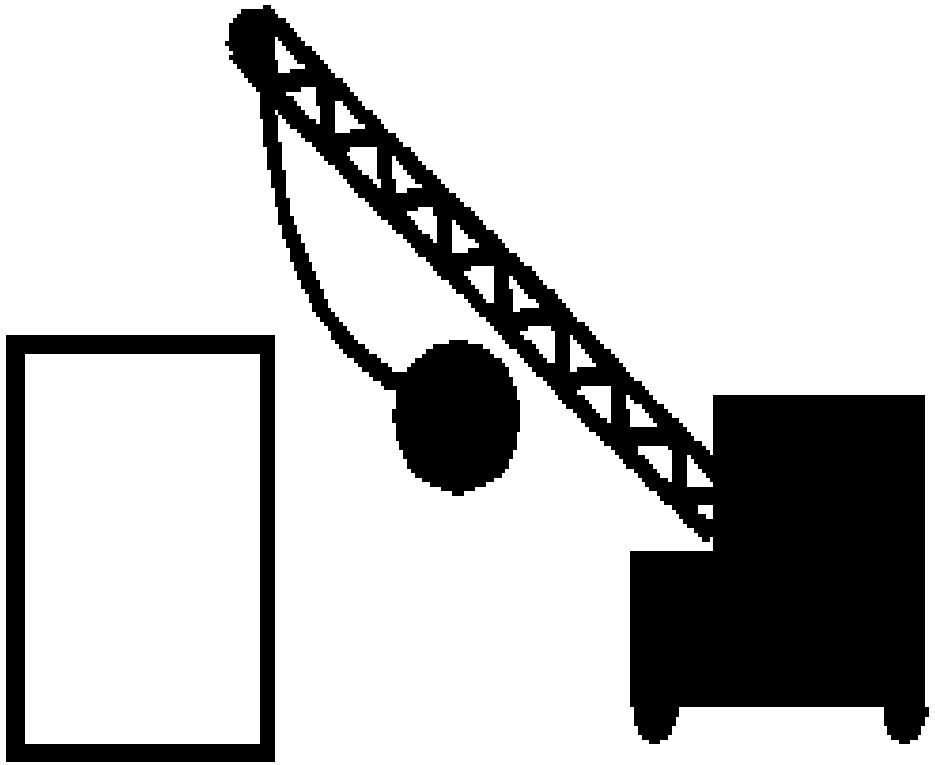
CONSTRUCTOR AND DESTRUCTOR



Constructor

```
MyClass *MyObjPtr = new MyClass();
```



Destructor

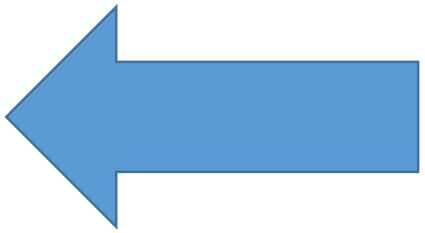
```
delete MyObjPtr;
```

Constructor is special method (Functin), used when instanting a class to initialize fields(data members)

important notes about constructor :

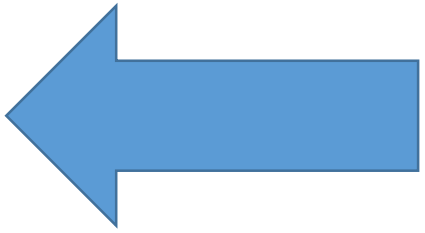
- A constructor has the same name as the class.
- Has no data type - A constructor can never return anything, which is why you don't have to define a return type for it.
- If no constructor defined then the CLR(Common Language Runtime) will provide an implicit constructor which is known as a *Default Constructor*.
- Constructors can be overloaded. - class can have any number of constructors and they vary with the number of arguments that are passed
- We don't use references or pointers on constructors because their addresses cannot be taken.

```
public string bike()
{
}
```



A normal method is defined like this:

```
public      bike()
{
}
```



A simple constructor(without parameters) can be defined like this:

example of constructor:

```
public class bike
{
        private int mileage;
        private string color;

        public bike()
        {

        }

        public bike(int mil, string col)
        {
                mileage = mil;
                color = col;
        }

        public void DisplayBikeData()
        {
                Console.WriteLine("Bike's Mileage is " + mileage + " and color is " + color);
        }
}
```

Class

Attributes

//constructor without parameter

//constructor with two parameters "mil" and "col"

```
using System;
namespace Tutlane
{
    class User
    {
        public string name, location;
        // Default Constructor
        public User()
        {
            name = "Ali";
            location = "Baghdad";
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            User user = new User();
            Console.WriteLine(user.name);
            Console.WriteLine(user.location);
            Console.ReadLine();
        }
    }
}
```

```
Ali
Baghdad
```

Output

constructor method has called automatically and initialized the parameter values after creating an instance of our class.

```
using System;
namespace Tutlane
{
    class User
    {
        public string name, location;
        public User(string a, string b)
        {
            name = a;
            location = b;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            User user = new User("Ali", "Baghdad");
            Console.WriteLine(user.name);
            Console.WriteLine(user.location);
            Console.ReadLine();
        }
    }
}
```

```
// Parameterized
Constructor
```

```
// constructor called
once the instance of c
lass created
```

```
Ali
Baghdad
```

Output

Destructors:

garbage cleanup is automatic system, (framework will free the objects that are no longer in use)

BUT there may be times where we need to do some manual cleanup. In this case we can use Destructor, which is used to destroy the objects that we no longer want to use (free or cleanup resources used by the object)

A destructor method called once an object is disposed.

```
public class Bike
{
    public Bike()
    {
        //Constructor
    }

    ~Bike()
    {
        //Destructor
    }
}
```

Once the class object is instantiated, *Constructor* will be called

and when object is collected by the garbage collector, *Destructor* method will be called.

The purpose of the destructor method is to remove unused objects and resources.
Destructors are not called directly in the source code but during garbage collection.
A destructor is invoked at an undetermined moment. More precisely a programmer can't control its execution; rather it is called by the Finalize () method.
Like a constructor, the destructor has the same name as the class except a destructor is prefixed with a tilde (~).

There are some limitations of destructors as in the following;

- Destructors are parameterless.
- A Destructor can't be overloaded.
- Destructors are not inherited.

CLASS CONSTANTS

C# enables to create class constants.

These constants do not belong to a concrete object. They belong to the class.

constants are written in uppercase letters.

We have a **Math** class with a **PI** constant.

```
public const double PI = 3.14159265359;
```

The **const** keyword is used to define a constant.

The **public** keyword makes it accessible outside the body of the class.

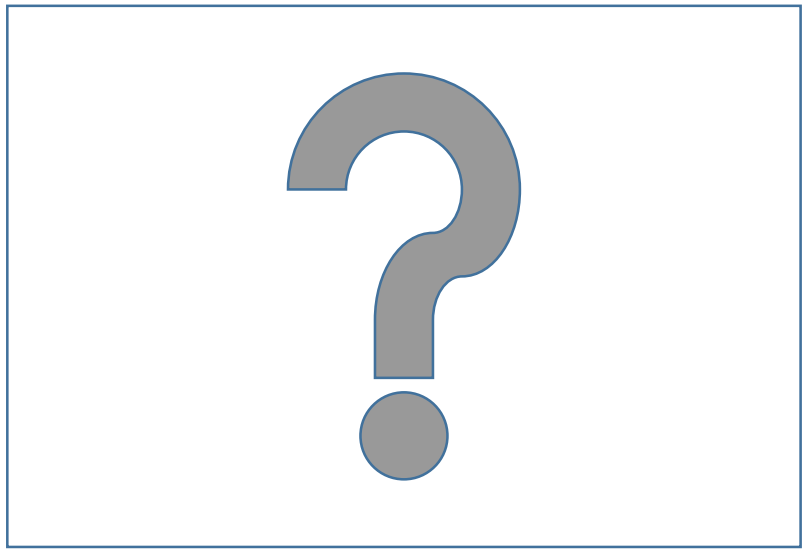
Program.cs

```
using System;

namespace ClassConstants
{
    class Math
    {
        public const double PI = 3.14159265359;
    }

    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(Math.PI);
        }
    }
}
```

```
3.14159265359
```



QUESTION

Google Classroom :

3hzp6bz