```
class Complex
{
    int Real, Imag;
            //Default Constructor
    public Complex()
    {       Real = Imag = 0;    }
            //Parametrized Constructor
    public Complex(int m)
    {       Real = Imag = m;   }
    public Complex(int r, int i)
    {       Real = r;  Imag = i; }
    public void SetR(int r)
    {       Real = r;  }
    public void SetI(int i)
    {       Imag = i;}
    public int GetR()
    {       return Real;}
    public int GetI()
    {       return Imag;}
    public ~Complex()
    {Console.WriteLine("Object Removed");}
}

class Test
{
    public static void Main()
    {
                                cpl1.Complex();
            Complex cpl1 = new Complex();
                                cpl2.Complex(4, 8);
            Complex cpl2 = new Complex(4, 8);
    }
}
```

Creation of Object:

1) Declaration
2) Creation a) Allocate (new)
             b) Initialize (constructor)

Constructor:

is a special kind of method, called implicitly when creating an object

*) has the same name of the class
*) has no return type, even void
*) public

Removing Object from Memory:

De-initialize (Destructor)
De-Allocate (Garbage Collector)

Destructor:

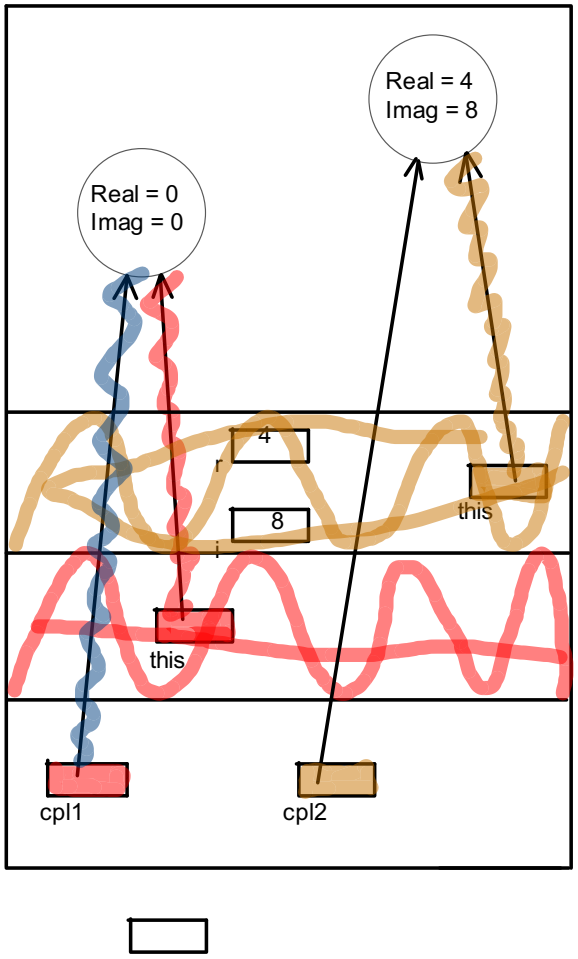
is a special kind of method, called implicitly when removing an object from memory

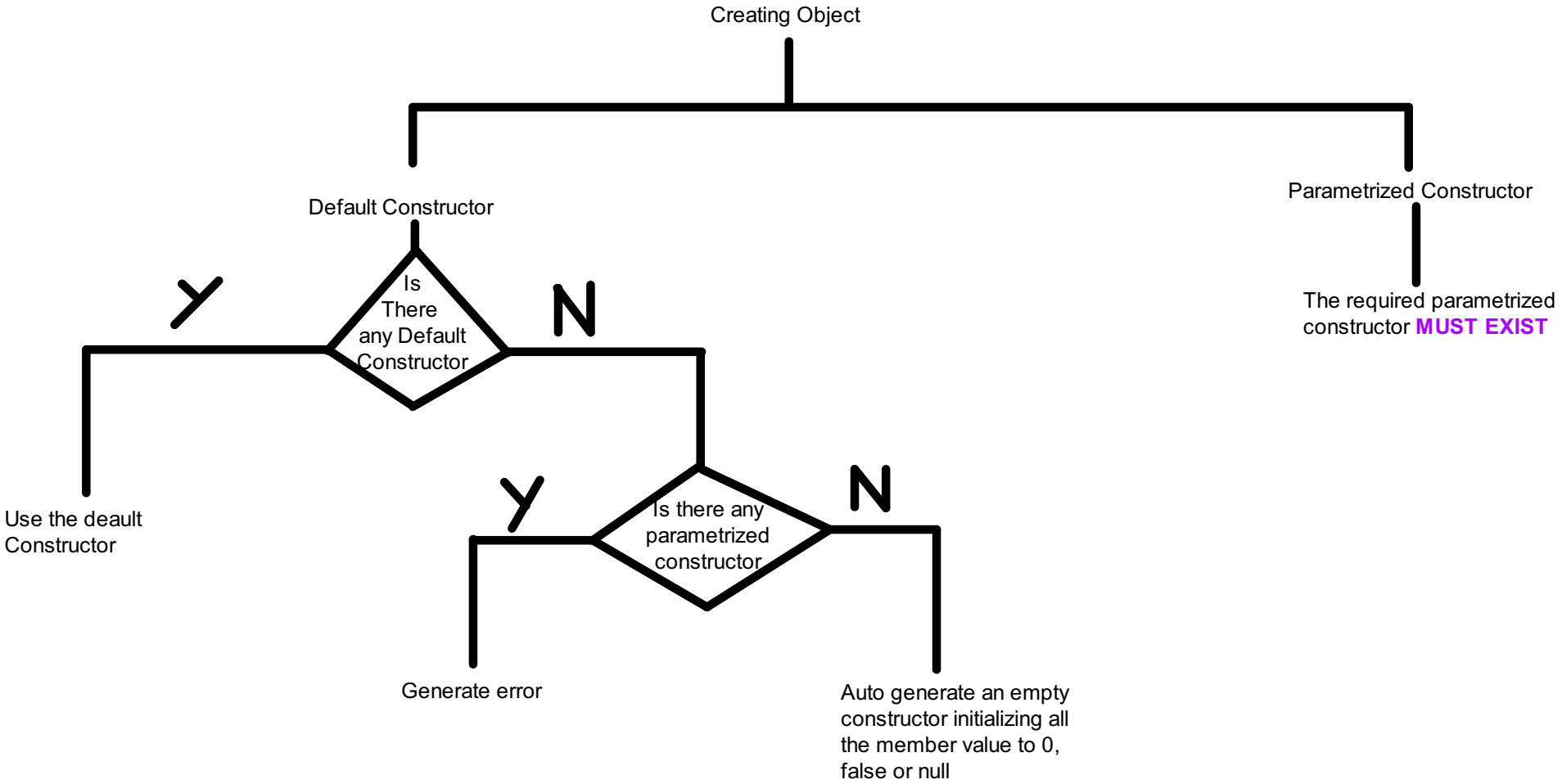
*) has the same name of the class preceeding ~
*) has no return type even void
*) has no parameter list (only 1 destructor)
*) public

Method Signature: Method_Name + Parameter_List

Method Overloading:

More than one method, having the same name and different in parameter (type / number / both) "Different in signature"




Creating Object
Default Constructor
Parametrized Constructor
The required parametrized constructor MUST EXIST
Is There any Default Constructor
Y
N
Use the deault Constructor
Y
Is there any parametrized constructor
N
Generate error
Auto generate an empty constructor initializing all the member value to 0, false or null

Inheritance (Is Relation)

```
class Parent
{
        protected int x, y;
        public Parent()
        {x = y = 0;}
        public Parent(int m)
        {x = y = m;}          3       4
        public Parent(int m, int n)
        {x = m;     y = n;}
        public int GetX()
        { return x;}
        public int GetY()
        {return y;}
        public void SetX(int m)
        {x = m;}
        public void SetY(int n)
        {y = n;}
        public int Sum()
        {return (x + y);}
}

class Child : Parent
{
        int a;
        public Child()
        {a = 0;}              3      4        5              3  4
        public Child(int l, int m, int n) : base(l, m)
        { a = n;}
        public int GetA()
        {return a;}
        public void SetA(int m)
        {a = m;}
        public int Product()
        {return (x * y * a);}
}
```

```
class Test
{
        public static void Main()
        {
                Child obj;
                obj = new Child(3, 4, 5);
                Console.WriteLine(obj.Product());
        }
}
```

protected:
no one can access protected member except member of the same class and the child classes