Virtual Method
(run-time polymorphism)

- Derived class may need to provide customized implementation for inherited method (ex: Display method) this behavior called *Override*

```
public class Human
{…
  public void Dispaly()
  {
   Console.WriteLine($" { Name}/t{ Age}");
  }
}
```

```
public class Employee:Human
{
  public void Dispaly()
  {
   Console.WriteLine($"{ Name}   /t { Age} /t { ID} /t {Salary}");
  }
}
```

Virtual Method

- This scenario could be achieved by mark base class member as ***virtual*** and child class member as ***override***

```
public class Human
{…
  public virtual void Dispaly()
  {
   Console.WriteLine($" { Name}/t{ Age}");
  }
}
```

```
public class Employee:Human
{
  public override void Dispaly()
  {
   Console.WriteLine($"{ Name}   /t { Age} /t { ID} /t {Salary}");
  }
}
```

Virtual Method

- Run-time polymorphism achieved by using a reference of base class type with object to child class

```
Human h= new Employee{Age=40};
h.Display(); // call Employee method not Human method
```

- Both virtual and override methods must have the same signature (name + parameter)
  - Demo without virtual & override
- ***virtual*** modifier used with methods ad properties

Virtual Method

- Why virtual??
  - Demo Human , Employee Display method
  - Code in notes

Virtual Method

- ***new*** modifier
    - Derived class may need to hide inherited method this behavior called method hiding
    - This scenario could be achieved use new modifier
        - Ex: inherit class from external API and hide some members
    - Both old and new methods must have the same signature (name + parameter)
    - ***new*** modifier used with (const and static) fields, method and properties

Object class

- *Object* class is the parent Data type for all Data type in .NET directly or indirectly
- If a class has no parent it is implicitly inherited from *Object* class

Method	Description
public virtual bool Equals (object o)	if reference type check reference equality if value type check value(if different type return false even value is equal)
public Type GetType()	object type not reference type
public virtual string ToString()	Return a string (default return type as a string)
public virtual void Finalize()	implemented through destructor
public static bool ReferenceEquals (object a , object b)	check reference equality

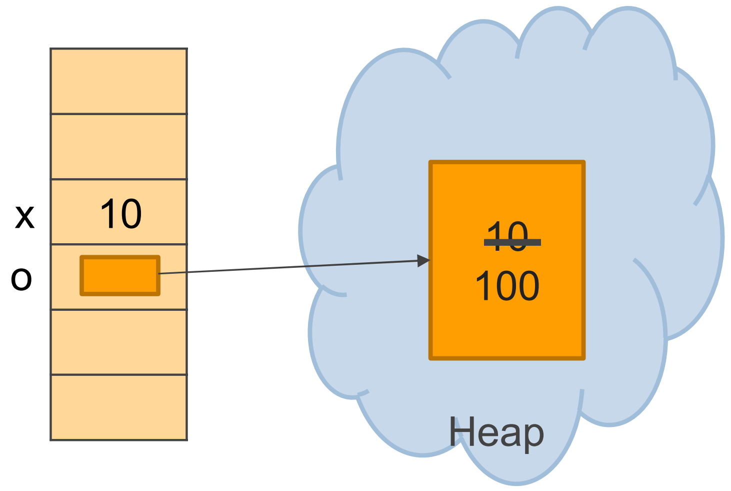
Object class

- Boxing
  - Boxing is the process of storing a value type inside an object

```
int x = 10;
object o = x; // boxing
```

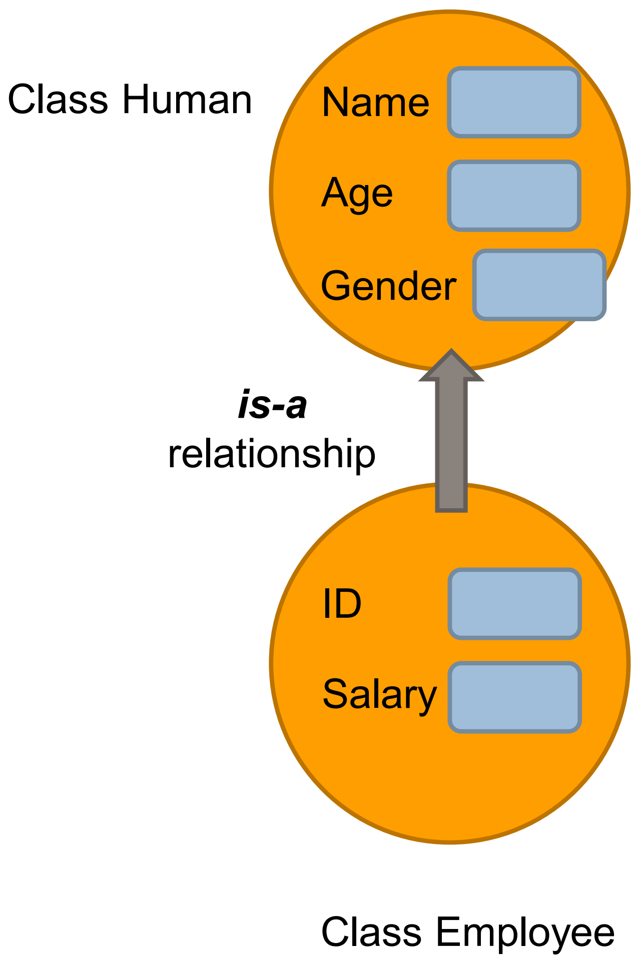
- Unboxing
  - Opposite of boxing

```
int x = 10;
object o = x; // boxing
o = 100;
int y =(int) o; // unboxing
```



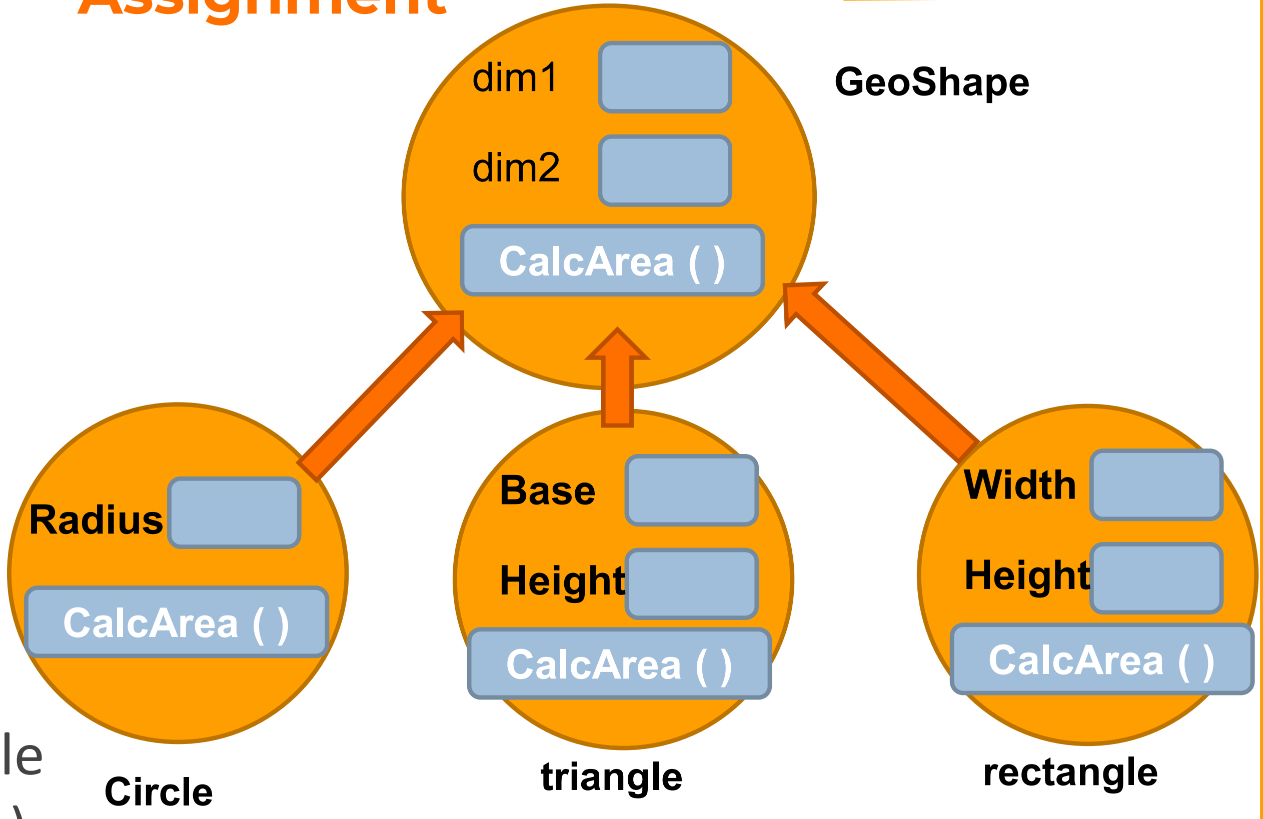
Assignment

- Modify menu Program by
- Design class Human (Age , Name, Gender) and modify employee class to inherit from it
- Override Tostring( ) method in both classes



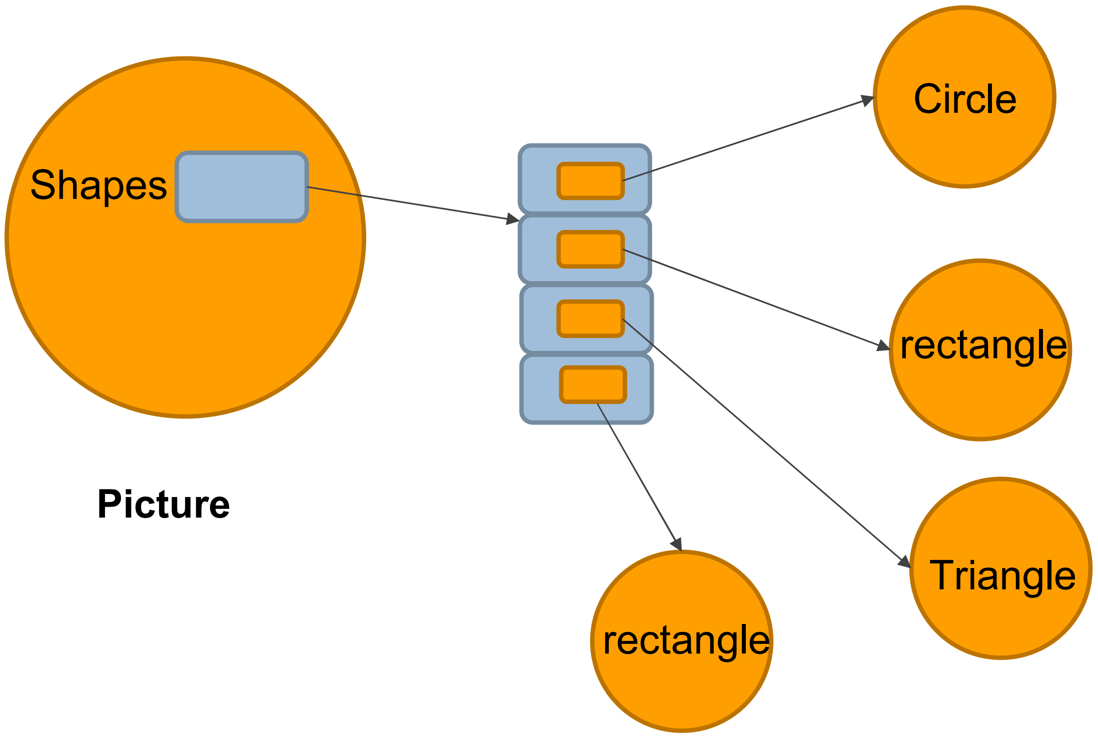
Assignment

- Design GeoShape , rectangle , triangle , circle classes
- And calculate areas individual then through Array of Geoshape type each element refer to different object (circle , Triangle , Rectangle)



Assignment

- Design a class **Picture** that encapsulate number of shapes (circle ,rectangle ,triangle)then calculate sum of their areas



Inheritance II: Abstract class and Interface

abstract class

- Is a class not intended to be instantiated , used for design Only , Used to define common member to its ***concrete*** subclasses
  - Ex: GeoShape class
- The major characteristic of abstract class that it contain at least one **abstract member** (method or property)

```
abstract class Geoshape
{
        protected int dim1, dim2;
    …
        public abstract float CalcArea();
}
```

abstract member

- Abstract member is a method or property that has no Implementation ,it can exist ***only*** in abstract class.
  - Ex: converting CalcArea () into abstract method since it does not has a logical meaning to return 0

```
public abstract float CalcArea();
```

- Inheriting from abstract class **enforce** subclasses to override (implement) abstract members
- Abstract members can not be private nor static
- Abstract method implicitly virtual method

Interface

- Interface like abstract class it contain only abstract members, it **can't** contain implementation **nor** member fields.
  - No *abstract* modifier is used
- Interface defines a contract any class implements(inherit) that contract must provides an Implementation of the members defined in the interface
- Interface members has not access modifier (since they must be public)

Abstract property
Not
Auto-implement property

```
interface Imyinter
{
   int prop { set; get; }
   void mymethod();
}
```

Interface

- A class can implements more than interface
- Interface support inheritance
  - Ex: : ***IQueryable*** :***IEnumerable***
- Interface support loose coupling (Example in notes)
- A type, regardless of whether it is a reference type or a value type, can implement any number of interfaces.

Implement interface

```
interface Imyinter
{
    int prop { set; get; }
    void mymethod();
}
```

- **Implicitly**
  - Through class reference
  - Through interface reference

```
class myclass : Imyinter
    {
        void mymethod()
        {…..}
    }
```

- **Explicitly**
  - No access modifier
  - Through interface reference only
  - Used in case of multiple implementation

```
class myclass : Imyinter
    {
        void Imyinter.mymethod()
        {…..}
    }
```

Why interface

- Capturing similarities among unrelated classes without artificially forcing a class relationship.
- Ex: PrintData for Employee ,Point

Assignment

- In Menu Program
- Add Sort button
  - Sort Array of Employee
    - Using Array.Sort(array) (hard coding)
      - Implmenting **IComparable** interface by Employee class
    - Using Array. Sort(array, ***IComparer***)
      - By implementing the way of sorting in classes that implements ***Icomparer*** Interface

Association, Aggregation and Composition

Association

- Association relationship is referred to as "***uses a***" relationship where a class uses another class to perform some operation.
- Association happens between the classes where one class ***provides a service*** to another class or the class delegates some kinds of behaviors to another class.
- both classes can exist independently where nobody is the owner of another.
  - Ex: driver and car
  - Ex: student and Teacher

Association

```
public class Driver
{
    public string? Name{set;get;}
    public void Drive(Car c)
    {
        c.Move();
    }
}
```

```
public class Car
{
    public string? Model{set;get;}
    public string? Color{set;get;}
    public void Move()
    {
  Console.WriteLine("Car is moving!!");
    }
}
```

```
Car car =new Car{Model="BMW",Color="Black"};
Driver driver =new Driver{Name="Mohamed"};
driver1.Drive(car );
```

Aggregation

- Aggregation is referred to as "***has a***" relationship where a class Contains object(s) of another class(es)
- both classes can exist independently
  - Ex: picture and Circle or line

```
Public class Picture
{
    Line line;
    public Picture(Line l)
    {
       line=l;
    }
}
```

```
Public class Line
{
    Point Start;
    Point End;
    public Line()
    {
       Start=new Point();
       End=new Point();
    }
}
```

Composition

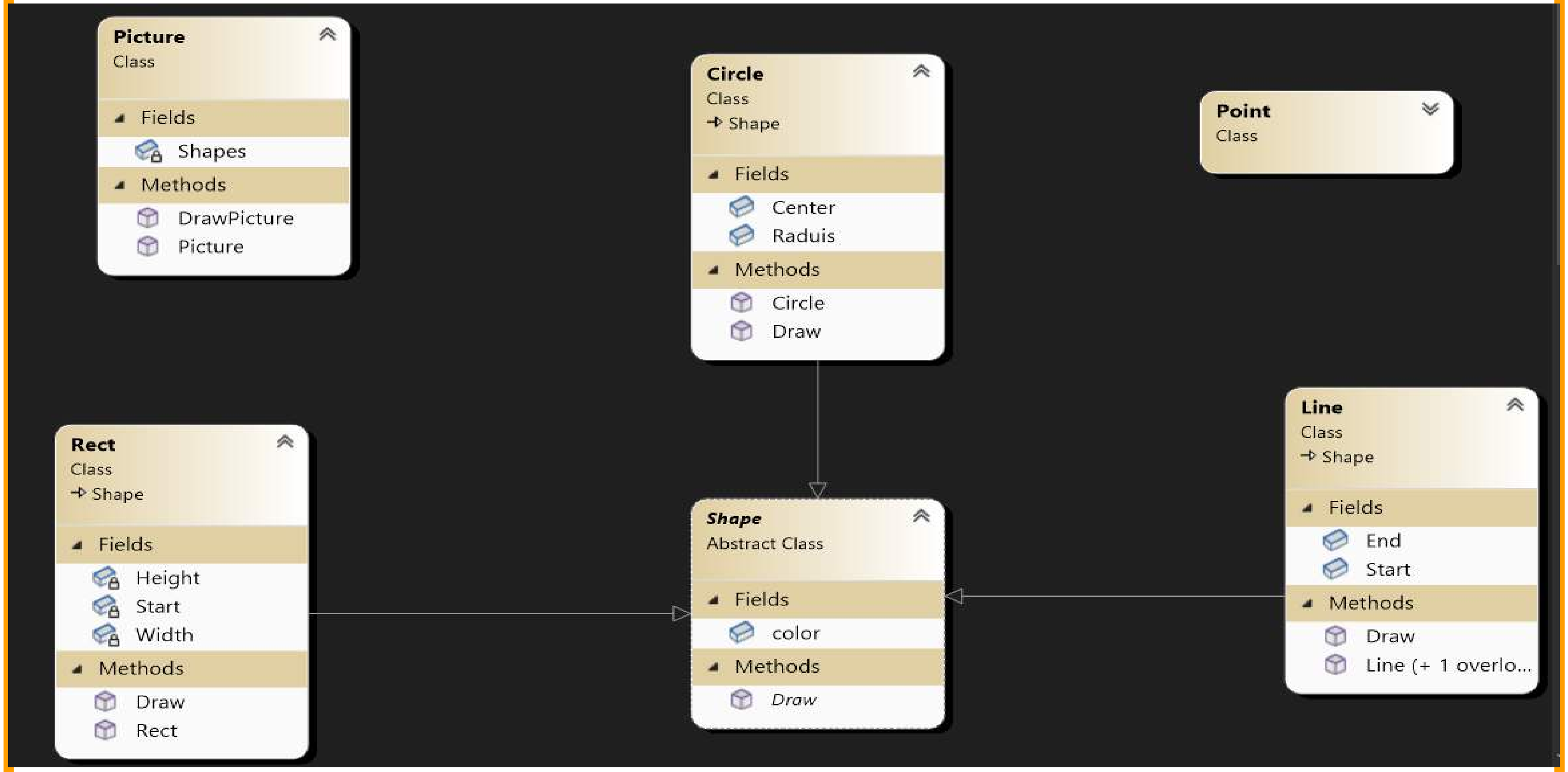
- Composition is referred to as "***has a***" relationship where a class Contains another class(es)'s object
- The container called Owner
- The contained called child
- The owner responsible of create child
- The child can not exist without owner
  - Human body => arm ,leg ,head
  - Line => point
  - Employee => address
  - House => room
- Composition considered a strong case of Aggregation

Composition

```
Public class Line
{
    Point Start;
    Point End;
    public Line()
    {
       Start=new Point();
       End=new Point();
    }
}
```

```
Public class Point
{
  public int x;
  public int y;
  public Point()
    {
       x=y=10;
    }
}
```

Assignment


Picture
Class
Fields
Shapes
Methods
DrawPicture
Picture
Circle
Class
Shape
Fields
Center
Raduis
Methods
Circle
Draw
Point
Class
Rect
Class
Shape
Fields
Height
Start
Width
Methods
Draw
Rect
Shape
Abstract Class
Fields
color
Methods
Draw
Line
Class
Shape
Fields
End
Start
Methods
Draw
Line (+ 1 overlo...

Assignment

- Modify console application Settings

```
<PropertyGroup>
        <TargetFramework>net8.0-windows</TargetFramework>
        <Nullable>enable</Nullable>
        <UseWindowsForms>true</UseWindowsForms>
        <ImplicitUsings>enable</ImplicitUsings>
</PropertyGroup>
```

Assignment

- Using the given class library draw a picture contains number of circles , lines and rectangles
- Static class DrawingClass

```
void StartDraw(int width,int height,string Title= "Graphics")
```

```
void DrawRectangle(System.Drawing.Color color,int x,int y,int width,int height,bool s)
```

```
void DrawCrcle(System.Drawing.Color color, int x, int y, int rad,bool s)
```

```
void DrawLine(System.Drawing.Color color, int x1, int y1, int x2, int y2,bool s)
```

```
void EndDraw()
```