Abstract Class, Interface, Array of Objects

Dr Athraa Juhi Jani

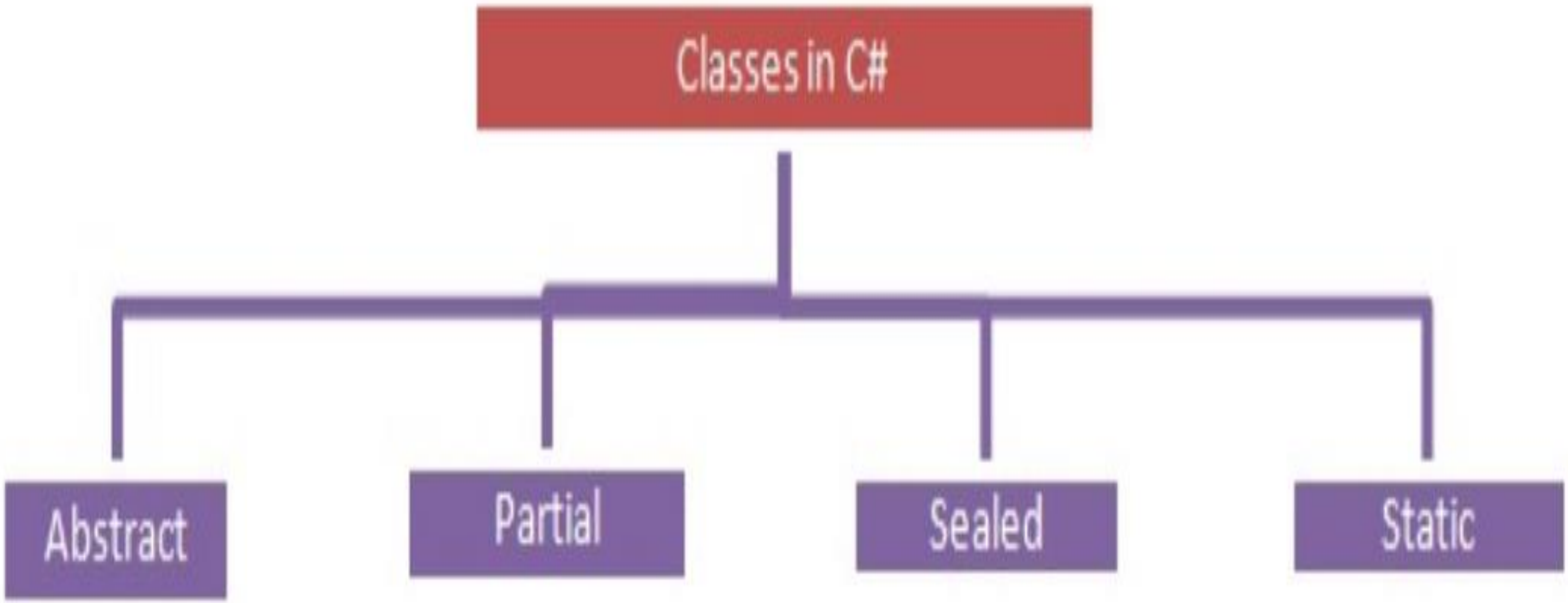
Class

- Classes are the user defined data types that represent the **state** and **behaviour** of an object.

  - **State** represents the properties of the object

  - **Behaviour** is the action that objects can perform.

The following are types of classes in C#:



Abstract Class

- Abstract classes are declared using the **abstract** keyword.

- We **cannot** create an object of an abstract class.

- If you want to use it then it must be **inherited** in a subclass.

Abstract Class

- An Abstract class contains both abstract and non-abstract methods.

- The methods inside the abstract class can either have an implementation or no implementation.

- An Abstract class has only one subclass.

Abstract Class

- Methods inside the abstract class cannot be private.
- If there is at least one method abstract in a class then the class must be abstract.

How to make a class to be abstract?

Here is an example:

```
public abstract class Shape {
    private String color;

    public Shape(){}

    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public abstract double getArea();
    public abstract double getPerimeter();
}
```

How to make a class to be abstract?

- And then in subclass, the method that mark with `abstract` keyword, it will automatically request to be override without any excuse.

```
public class Circle extends Shape{
    private double radius
    public Circle(){}
    public Circle(double radius){
        this.radius = radius;
    }
    @Override
    public double getArea(){
        return radius*radius*Math.PI;
    }
    @Override
    public double getPerimeter(){
        return 2*radius*Math.PI;
    }
}
```

How to use abstract class?

- You can use an abstract class by inheriting it using **extends** keyword.

```
public class Circle extends Shape {

}
```

- Abstract class can also be a type.

```
Shape sh;//Shape is a type of sh variable
```

- Because abstract class can also be a type, we can use polymorphism as well.

```
Shape sh = new Circle();

sh.getArea();
```

How to use abstract class?

- You CANNOT create instances of abstract classes using the **new** operator.

```
Shape shape = new Shape();// Compile Error
```

- We can make an abstract class by not making any method abstract also. There is no any error.

```
 public abstract class Shape {
    public String getColor(){
          return “”;
    }
 }
```

Importance of abstract class

- Abstract class is always a superclass. It means when you make an abstract class, you have to think that the class must be a superclass later.
- Abstract class is the way to guarantee that its closed subclasses MUST override abstract methods.
- The only reason that we have to make abstract class is because of polymorphism.
- It makes no sense if we make abstract class, but we don’t use any polymorphism.

Abstract Method

- Abstract methods, similar to methods within an interface, are **declared without any implementation**.

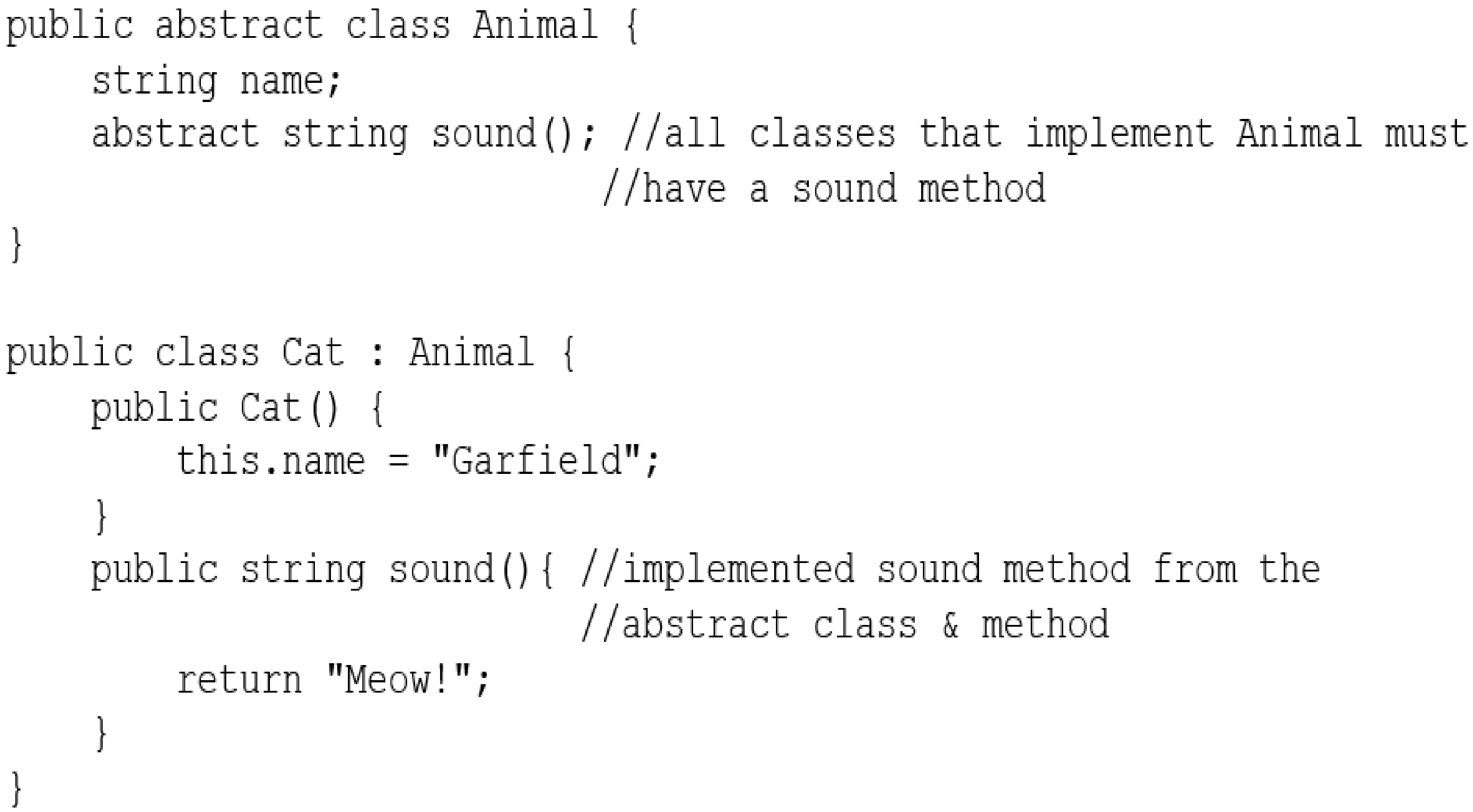
- They are declared with the purpose of having the **child class provide implementation**.

- They **must** be declared within an **abstract class**.

Syntax of Abstract Methods

```
modifier abstract class className {
    //declare fields
    //declare methods
    abstract dataType methodName();
 }

modifier class childClass : className {
    dataType methodName(){}
}
```

Example



```
public abstract class Animal {
    string name;
    abstract string sound(); //all classes that implement Animal must
                            //have a sound method
}

public class Cat : Animal {
    public Cat() {
        this.name = "Garfield";
    }
    public string sound(){ //implemented sound method from the
                           //abstract class & method
        return "Meow!";
    }
}
```

Interface

- Interfaces define properties, methods, which are the members of the interface.
- Interfaces contain only the **declaration** of the members.
- It is the responsibility of the deriving class to define implementation to the members.

Interface

- Abstract classes to some extent serve the same purpose,

- However, they are mostly used when only few methods are to be declared by the base class and the deriving class implements the functionalities.

Declaring Interfaces

- Interfaces are declared using the **interface** keyword.
- It is similar to class declaration.
- Interface statements are **public** by default. Following is an example of an interface declaration:

```
public interface ITransactions
{
// interface members
void showTransaction();
double getAmount();
}
```

Notes

- Abstract classes and methods are declared with the '**abstract**' keyword.
- Abstract classes can only be **extended**, and cannot be **directly instantiated**.
- Abstract classes provide a little more than interfaces.

- Interfaces do not include fields and super class methods that get inherited, whereas abstract classes do.

- This means that an **abstract class** is more closely related to **a class which extends it**, **than an interface** is to a **class that implements it**.

Array of Objects

```
Circle[] circleArray = new Circle[10];
```

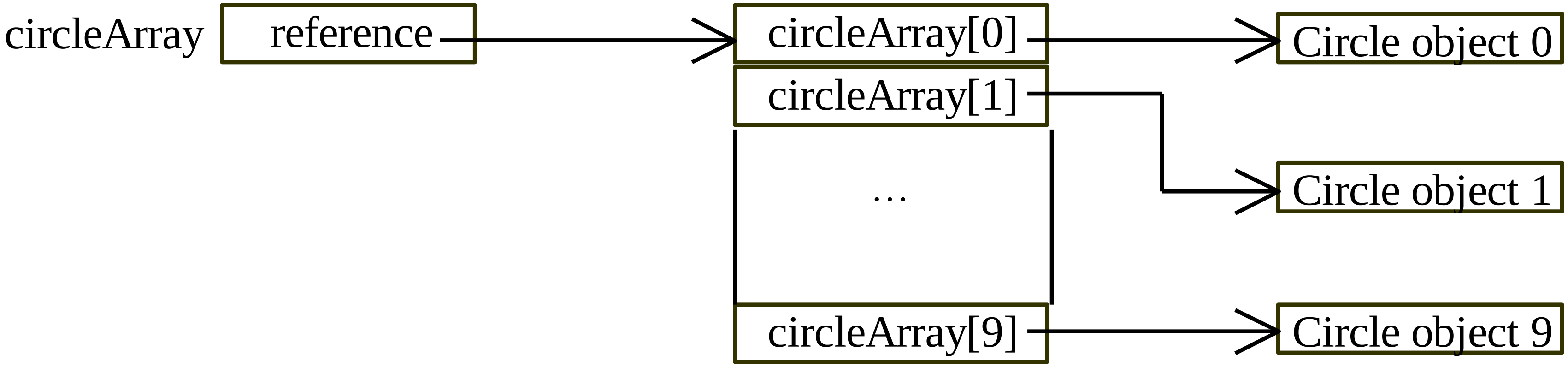
- An array of objects is actually an array of reference variables.

- So invoking circle [1].findArea() involves two levels of referencing as shown in the next figure.

- **circleArray** references to the entire array.
- **circle Array[1]** references to a **Circle object**.

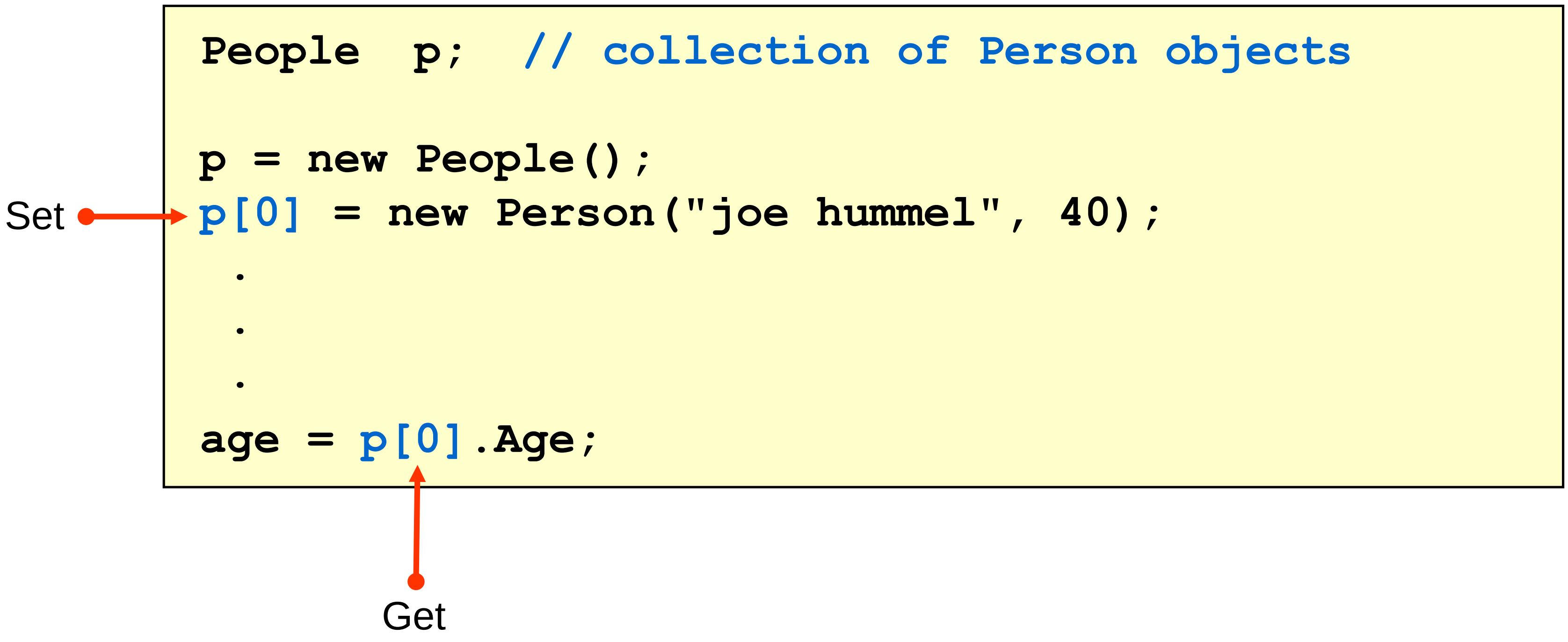
Array of Objects

```
Circle[] circleArray = new Circle[10];
```



Indexers

- **Enable array-like access with method-like semantics**



```
People  p;  // collection of Person objects

p = new People();
p[0] = new Person("joe hummel", 40);
  .
  .
  .
age = p[0].Age;
```

Set

Get

Example

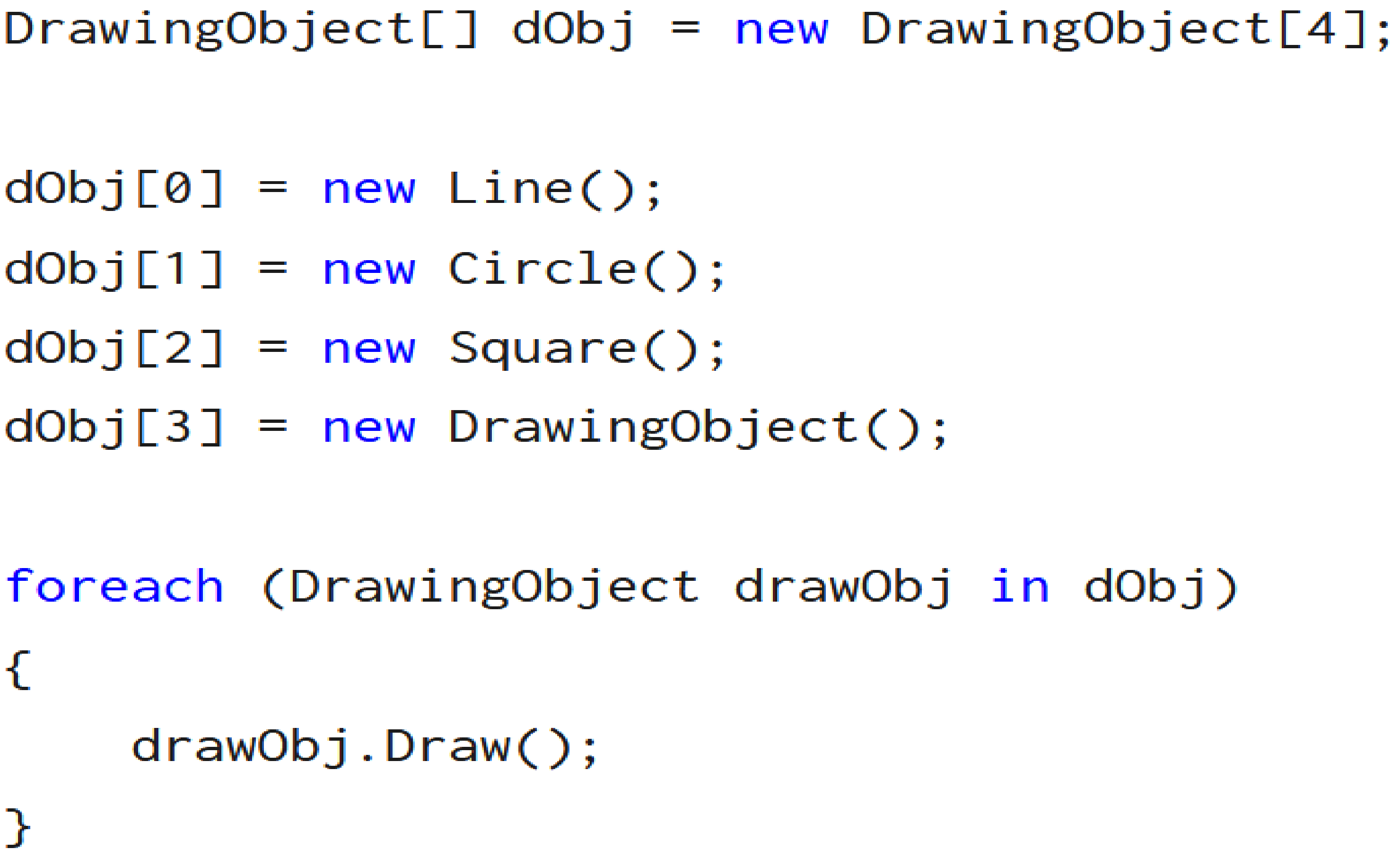
- **Implemented like properties, with Get and Set methods:**

```
public class People
{
  private Person[]  m_people;  // underlying array
    .
    .
    .

  public Person this[int i]          // int indexer
  {
    get { return this.m_people[i];  }
    set { this.m_people[i] = value; }
  }

  public Person this[string name]  // string indexer
  {
    get { return ...; }
  }
}
```

read-write →

read-only →

Example



```
DrawingObject[] dObj = new DrawingObject[4];

dObj[0] = new Line();
dObj[1] = new Circle();
dObj[2] = new Square();
dObj[3] = new DrawingObject();

foreach (DrawingObject drawObj in dObj)
{
    drawObj.Draw();
}
```

- We shall have an example about **polymorphism using array of objects**

- Then another example to use **abstract class with polymorphism**

Thank You