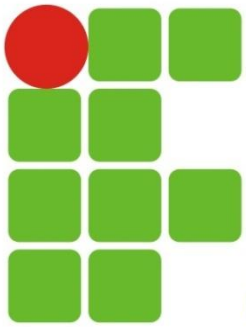


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

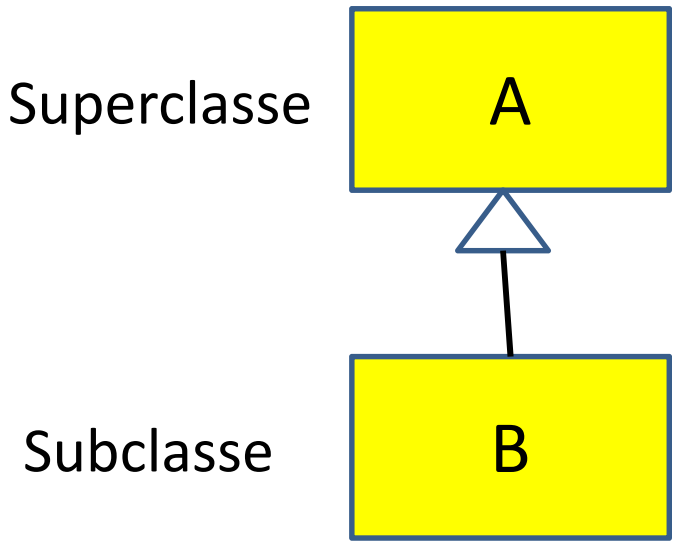
Herança em C#

Diretoria Acadêmica de Gestão e Tecnologia da Informação

Introdução

- Herança é um recurso que permite a uma classe absorver a estrutura de uma outra classe.
- A classe que é herdada é chamada de **classe base** ou **superclasse**.
- A classe que herda é chamada de **classe derivada** ou **subclasse**.
- C# não permite herança múltipla. Uma classe pode possuir apenas uma superclasse.

Introdução

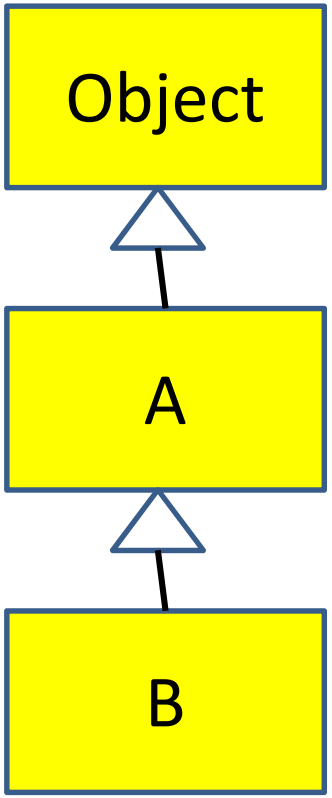



```
class A{
  ...
}

class B : A{
  ...
}
```

Classe **Object**

- **Object** é a classe mais básica em C#.
  - Todas as classes são subclasses de **Object**.



A herda **Object** de forma direta.

B herda **Object** de forma indireta.

Classe **Object**

- Quando não se indica a herança, o compilador assume que **Object** é a superclasse.
- Define alguns métodos:
  - **public virtual string ToString()**
  - **public virtual bool Equals(Object o)**

```
class A {
  ...
}
```

Neste exemplo, **A** tem **Object** como superclasse.

Herdando membros

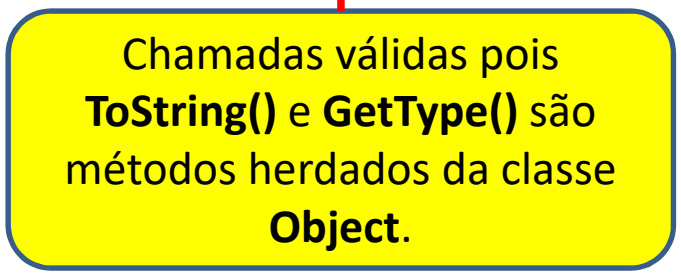
- Todos os membros da superclasse são herdados pela subclasse.
  - Assim, todas as classes herdam os membros definidos em **Object**.

Exemplo 1

```
static void Main(string[] args){
  string s = "IFRN";
  Console.WriteLine(s.GetType());
  int i = 28;
  Console.WriteLine(i.ToString());
  Console.WriteLine(i.GetType());
}
```

Exemplo 1

```
static void Main(string[] args){
   string s = "IFRN";
   Console.WriteLine(s.GetType());
   int i = 28;
   Console.WriteLine(i.ToString());
   Console.WriteLine(i.GetType());
}
```



Exemplo 2

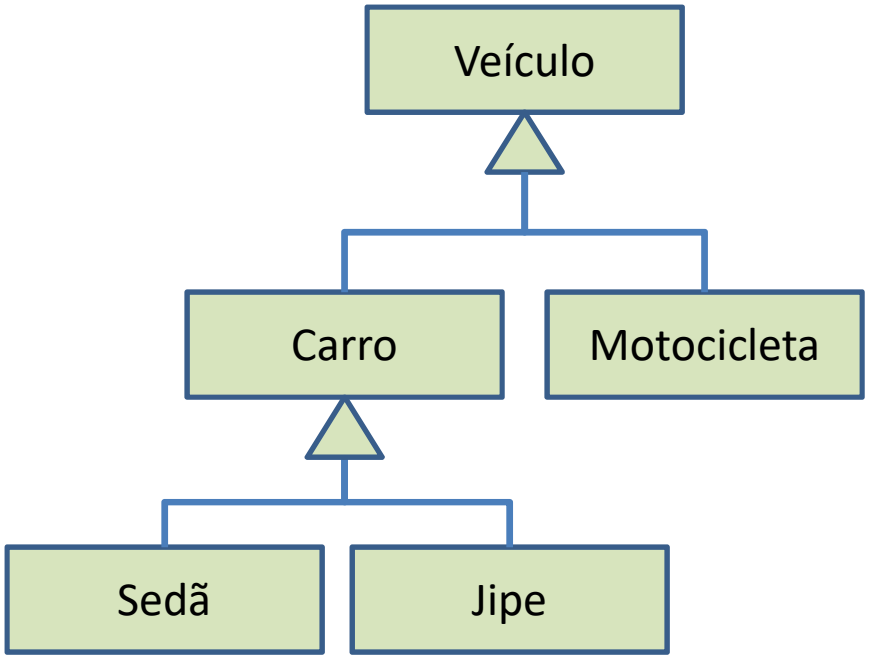
```
class MinhaClasse{
  static void Main(string[] args){
    MinhaClasse mc = new MinhaClasse();
    Console.WriteLine(mc.ToString());
    Console.WriteLine(mc.GetType());
  }
}
```

Hierarquia de classes

- Um conjunto de classes bases e classes derivadas formam uma **hierarquia de classes**.
- Por adicionar comportamento e dados adicionais, dizemos que as subclasses são mais especializadas do que as superclasses.
- Instâncias de classes derivadas podem ser tratadas como instâncias de sua classe base.

Hierarquia de classes

- Todo carro é um veículo.
- Toda motocicleta é um veículo.
- Todo sedã é um carro.
- Todo sedã é um veículo.
- Todo jipe é um carro.
- Todo jipe é um veículo.
- Nem todo veículo é um carro, motocicleta, sedã ou jipe.
- Nem todo carro é um sedã ou jipe.



Atribuição de supertipos e subtipos

- É possível atribuir uma instância de uma subclasse a uma referência de uma superclasse.
  - Todo carro é um veículo.
- A operação inversa é possível através de uma conversão de tipo (*type casting*).
  - Pode falhar em tempo de execução.
- Implementação do princípio do polimorfismo
  - Objeto pode ser enxergado como sendo uma instância de qualquer supertipo.

Exemplo 3

```
static void Main(string[] args){
  string s = "IFRN";
  int i = 28;
  Object ob = s;
  Console.WriteLine(ob.GetType());
  ob = i;
  Console.WriteLine(ob.GetType());
  Print(i);
  Print(s);
}
static void Print(Object o){
  Console.WriteLine(o.ToString());
}
```

Exemplo 4

```
class MinhaClasse{
  public int GetX() { return 20; }
  static void Main(string[] args){
    Object ob = new MinhaClasse();
    ob.GetType();
    ob.GetX(); //gera erro de compilação
    MinhaClasse mc = (MinhaClasse)ob;
    mc.GetX();
    mc.GetType();
  }
}
```

Operador **is**

- Indica se um objeto se enquadra como sendo de um determinado tipo na hierarquia de classes.
  - Retorna **true** ou **false**

```
class MinhaClasse{
  public int GetX() { return 20; }
  static void Main(string[] args){
    Object ob = new MinhaClasse();
    if(ob is MinhaClasse)
    {
      MinhaClasse mc = (MinhaClasse)ob;
      mc.GetX();
    }
  }
}
```

Herança e visibilidade de membros

- A herança não altera a visibilidade dos membros. Assim, as subclasses não possuem acesso aos membros definidos como privados nas superclasses.

Exemplo 5 – parte 1

```
class Quadrilatero{
  private string nome;
  public Quadrilatero(){
    nome = "quadrilátero";
  }
  public string GetNome(){
    return nome;
  }
}
```

Exemplo 5 – parte 2

```
class Retangulo : Quadrilatero{
  private double altura, largura;
  public Retangulo(double l, double h){
    altura = h;
    largura = l;
    nome = "retângulo"; //erro ao compilar
  }
  public double GetArea(){ return largura * altura; }
  public double GetPerimetro() {
    return largura * 2 + altura * 2;
  }
```

Visibilidade protegida

- Membros com visibilidade protegida podem ser acessados pela própria classe e pelas subclasses.

Exemplo 6 – parte 1

```
class Quadrilatero{
  protected string nome;
  public Quadrilatero(){
    nome = "quadrilátero";
  }
  public string GetNome(){
    return nome;
  }
}
```

Exemplo 6 – parte 2

```
class Retangulo : Quadrilatero{
  private double altura, largura;
  public Retangulo(double l, double h){
    altura = h;
    largura = l;
    nome = "retângulo"; //ok
  }
  public double GetArea(){
    return largura * altura;
  }
  public double GetPerimetro() {
    return largura * 2 + altura * 2;
  }
}
```

Exemplo 6 – parte 3

```
static void Main(string[] args){
  Quadrilatero quad = new Quadrilatero();
  Console.WriteLine(quad.GetNome()); //imprime quadrilátero
  Retangulo ret = new Retangulo(10, 5);
  Console.WriteLine(ret.GetNome()); //imprime retângulo
  Console.WriteLine(ret.GetArea()); //imprime 50
  Console.WriteLine(quad.GetArea()); //erro de compilação
}
```

Construtores

- Construtores não são herdados pelas subclasses.
- Subclasses utilizam, implicitamente ou explicitamente, construtores da superclasse para inicializar os membros herdados.
- Na forma explícita o programador chama algum construtor da superclasse através da instrução **`base`**.
- Na forma implícita o compilador utiliza o construtor padrão da classe base.
  - Em caso de inexistência do construtor padrão na classe base, é gerado um erro de compilação.
- Construtores das classes derivadas devem obrigatoriamente chamar algum construtor da classe base, seja de forma implícita ou explícita.

Exemplo 7 – parte 1

```
class Quadrilatero{
   private string nome;
   public Quadrilatero(string n){
      nome = n;
   }
   public string GetNome(){
      return nome;
   }
}
```

Exemplo 7 – parte 2

```
class Retangulo : Quadrilatero{
  private double altura, largura;
  public Retangulo(double l, double h):base("retângulo"){
    altura = h;
    largura = l;
  }
  public double GetArea(){
    return largura * altura;
  }
  public double GetPerimetro() {
    return largura * 2 + altura * 2;
  }
}
```

Sobrescrita de métodos

- Classes derivadas podem redefinir métodos definidos como virtuais na classe base.
  - A sobrescrita substitui a implementação herdada.
- O método sobrescrito deve ser marcado com a instrução **override**.
- Mesmo após sobrescrever um método, uma classe derivada pode acessar métodos definidos na classe base através da instrução **base**.

Exemplo 8 – parte 1

```
class Quadrilatero{
  public virtual string GetNome(){
    return "quadrilátero";
  }
}
```

Exemplo 8 – parte 2

```
class Retangulo : Quadrilatero{
  private double altura, largura;
  public Retangulo(double l, double h) {
    altura = h;
    largura = l;
  }
  public double GetArea(){ return largura * altura; }
  public double GetPerimetro() {
    return largura * 2 + altura * 2;
  }
  public override string GetNome(){ return "retângulo"; }
  public virtual void SetAltura(double h){ altura = h; }
  public virtual void SetLargura(double l){
    largura = l;
  }
}
```

Exemplo 8 – parte 3

```
class Quadrado : Retangulo{
  public Quadrado(double l) : base(l, l) { }
  public override string GetNome(){
    return "quadrado";
  }
  public override void SetAltura(double h){
    base.SetAltura(h);
    base.SetLargura(h);
  }
  public override void SetLargura(double l){
    SetAltura(l);
  }
}
```

Exemplo 8 – parte 3

```
class Quadrado : Retangulo{
  public Quadrado(double l) : base(l, l) { }
  public override string GetNome(){
    return "quadrado";
  }
  public override void SetAltura(double h){
    base.SetAltura(h);
    base.SetLargura(h);
  }
  public override void SetLargura(double l){
    SetAltura(l);
  }
}
```

Utiliza métodos **SetAltura** e **SetLargura** implementados na classe base.

Exemplo 8 – parte 3

```
class Quadrado : Retangulo{
  public Quadrado(double l) : base(l, l) { }
  public override string GetNome(){
    return "quadrado";
  }
  public override void SetAltura(double h){
    base.SetAltura(h);
    base.SetLargura(h);
  }
  public override void SetLargura(double l){
    SetAltura(l);
  }
}
```

Utiliza método **SetAltura** implementado na própria classe.

Exemplo 8 – parte 4

```
static void Main(string[] args){
  Quadrado q = new Quadrado(5);
  Console.WriteLine(q.GetNome()); //imprime quadrado;
  Console.WriteLine(q.GetArea()); //imprime 25
  q.SetAltura(10);
  Console.WriteLine(q.GetArea()); //imprime 100
}
```

Classes abstratas

- Não podem ser instanciadas.
- Geralmente são definidas para serem herdadas e especializadas pelas subclasses.
- São definidas com a instrução **`abstract`**.

Exemplo 9 – parte 1

```
abstract class Quadrilatero{
   public virtual string GetNome(){
      return "quadrilátero";
   }
}
```

Exemplo 9 – parte 2

```
static void Main(string[] args){
  Retangulo r = new Retangulo(2, 6); //ok
  Console.WriteLine(r.GetNome());
  Quadrilatero q = new Quadrilatero(); //erro
}
```

Métodos abstratos

- Não possuem implementação.
- Uma classe com pelo menos um método abstrato também deve ser abstrata.
- Devem ser implementados pelas subclasses.
  - Ou estas também devem ser abstratas.

Exemplo 10 – parte 1

```
abstract class Quadrilatero{
   public abstract string GetNome();
   public abstract double GetArea();
   public abstract double GetPerimetro();
}
```

Exemplo 10 – parte 2

```
class Retangulo : Quadrilatero{
  private double altura, largura;
  public Retangulo(double l, double h){
    altura = h;
    largura = l;
  }
  public override double GetArea() { return largura * altura; }
  public override double GetPerimetro(){
    return largura * 2 + altura * 2;
  }
  public override string GetNome() { return "retângulo"; }
  public virtual void SetAltura(double h) { altura = h; }
  public virtual void SetLargura(double l){
    largura = l;
  }
}
```

Referências

- http://www.tomasvasquez.com.br/cursocsharp/programacao_orientada_objetos_net/heranca-com-construtores
- http://www.linhadecodigo.com.br/artigo/316/introducao-a-orientacao-a-objetosparte-3.aspx
- http://msdn.microsoft.com/pt-br/library/bsc2ak47%28v=vs.90%29.aspx