# Интерфейси




**Учителски екип**

Обучение за ИТ кариера

https://it-kariera.mon.bg/e-learning/

# Интерфейс

Част от обект, чрез която можем да взаимодействаме с обекта

# Интерфейс

- Компилатора вътрешно добавя

Ключова дума

Модификатор за достъп

```
public interface IPrintable {
  void Print();
}
```

Име

компилатор

```
Public interface IPrintable {
  public abstract void Print();
}
```

Добавя public за всички членове

# Пример за интерфейс

- Имплементацията на Print() се задава в класа Document

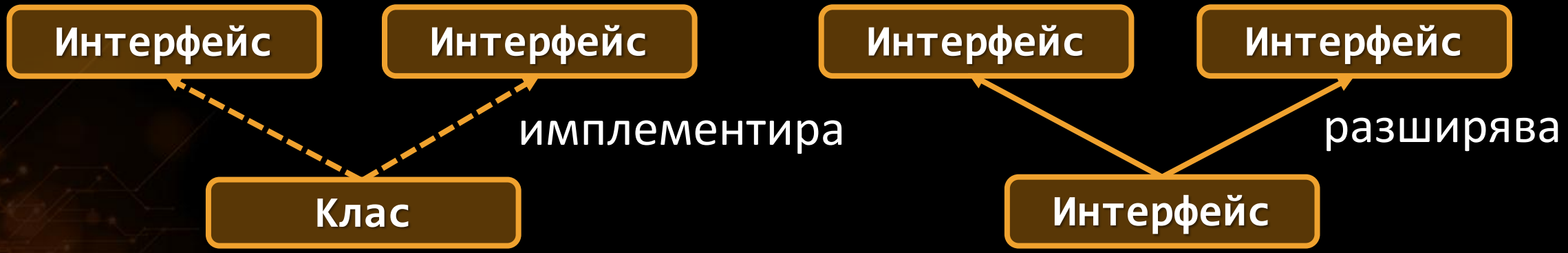
```
public interface IPrintable
{
  void Print();
}
```

```
class Document : IPrintable
{
  public void Print()
  { Console.ReadLine("Hello"); }
}
```

# Множествено наследяване

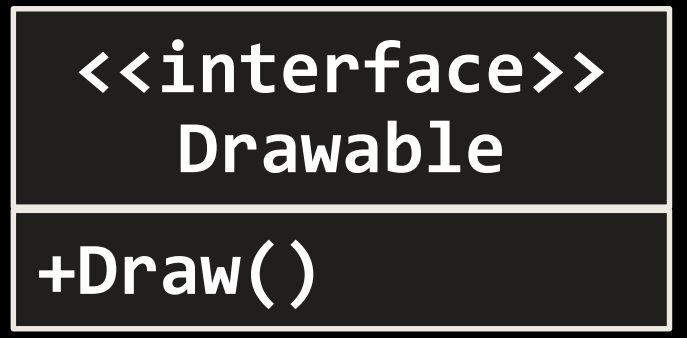
- Връзка между класове и интерфейси

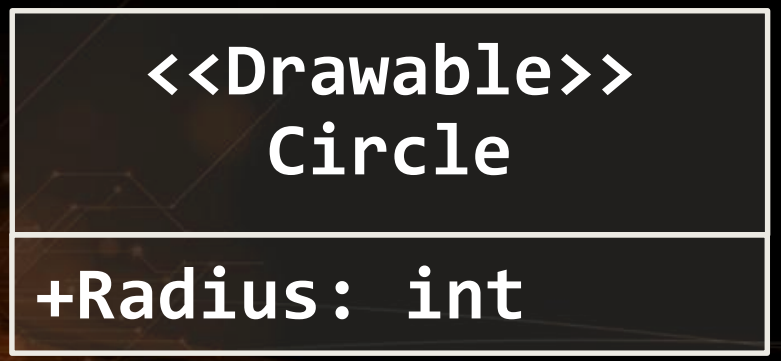

- Множествено наследяване

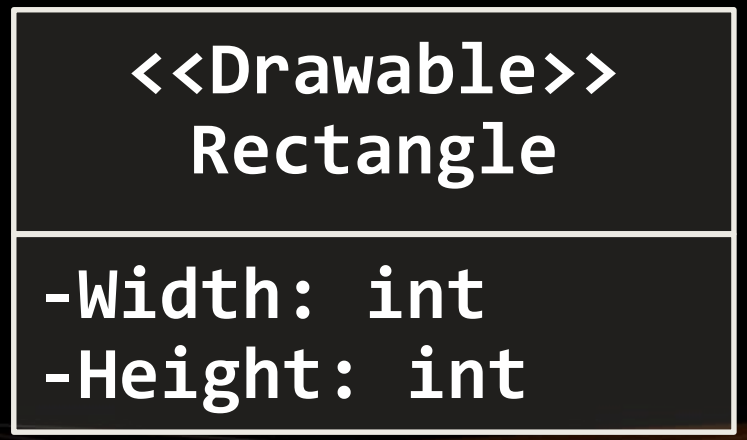


# Задача: Фигури

- Създайте проект, съдържащ интерфейс за рисуваеми обекти
- Имплементирайте два типа фигури:
  - Кръг
  - правоъгълник
- И двата класа трябва да отпечатват фигурата си със "*".




# Решение: Фигури

```
public interface Drawable {
  void Draw();
}
```

```
public class Rectangle : Drawable {
  //TODO добавете полета и конструктор
public void Draw() { слайд 8 } }
```

```
public class Circle : Drawable {
  //TODO добавете полета и конструктор
public void Draw() { слайд 9 } }
```

# Решение: Фигури – Чертане на правоъгълник

```
public void Draw()
{
  DrawLine(this.Width, '*', '*');
  for (int i = 1; i < this.Height - 1; ++i)
    DrawLine(this.Width, '*', ' ');
  DrawLine(this.Width, '*', '*');
}
private void DrawLine(int width, char end, char mid)
{
  Console.Write(end);
  for (int i = 1; i < width - 1; ++i)
    Console.Write(mid);
  Console.WriteLine(end);
}
```

# Решение: Фигури – чертане на кръг

```
double r_in = this.Radius - 0.4;
double r_out = this.Radius + 0.4;
for (double y = this.Radius; y >= -this.Radius; --y)
{
  for (double x = -this.Radius; x < r_out; x += 0.5)
  {
    double value = x * x + y * y;
    if (value >= r_in * r_in && value <= r_out * r_out)
      Console.Write("*");
    else
      Console.Write(" ");
  }
  Console.WriteLine();
}
```

# Интерфейс с/у абстрактен клас

Абстрактен клас

- Класът може да наследи само един абстрактен клас.
- Абстрактните класове могат да предоставят целия код и/или само детайлите, които трябва да се презапишат.

Интерфейс

- Класът може да имплементира няколко интерфейса.
- Интерфейсът не може да предоставя никакъв код, предоставя само описание.

# Интерфейс с/у абстрактен клас (2)

Абстрактен клас

- Абстрактния клас може да съдържа модификатори за достъп
- Ако множество имплементации са от сходен вид и имат общо поведение или статут, то абстрактния клас е по-добър избор.

Интерфейс

- Интерфейсите нямат модификатори за достъп. Всичко е публично по подразбиране.
- Ако множество имплементации споделят само сигнатурата на методите и нищо друго, то тогава интерфейсът е по-добър избор.

# Интерфейс с/у абстрактен клас (3)

Абстрактен клас

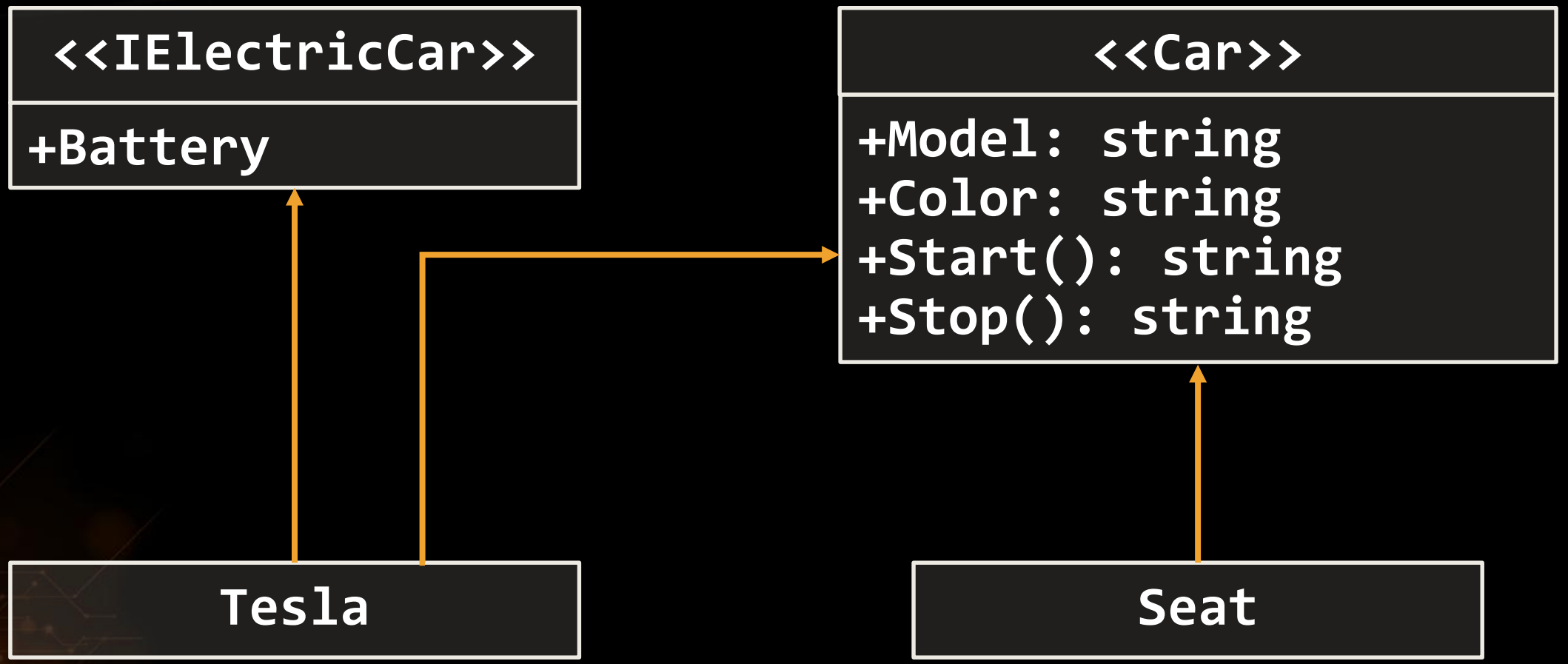
- Абстрактният клас може да притежава полета и константи
- Ако добавим нов метод към абстрактен клас, то имаме опцията да създадем имплементация по подразбиране и така съществуващият код ще може да работи коректно.

Интерфейс

- Не поддържа полета
- Ако добавим нов метод към интерйфес, то трябва да проследим всичките му имплементации и да дефинираме имплементация за новия метод.

# Задача: Коли

- Постройте йерархия от интерфейси и класове



# Решение: Коли

```
public interface ICar
{
   string Model { get; }
   string Color { get; }
   string Start();
   string Stop();
}
```

```
public interface IElectricCar
{
   int Batteries { get; }
}
```

# Решение: Коли (2)

```
public class Tesla : ICar, IElectricCar
{
  public string Model { get; private set; }
  public string Color { get; private set; }
  public int Batteries { get; private set; }
  public Tesla(string model, string color, int batteries)
  { //TODO: Add Logic here }
  public string Start()
  { //TODO: Add Logic here }
  public string Stop()
  { //TODO: Add Logic here }
}
```

# Решение: Коли (3)

```
public class Seat : ICar
{
  public string Model { get; private set; }
  public string Color { get; private set; }
  public Seat(string model, string color)
  { //TODO: Add Logic here }
  public string Start()
  { //TODO: Add Logic here }
  public string Stop()
  { //TODO: Add Logic here }
}
```

# Задача: Машина

- Създайте интерфейс IMachine, за машина, която трябва да има функционалност за пускане и спиране, добавете и свойство за вида на машината. Създайте класове Car, LawnMower, Truck, Airplane, които да имплементират интерфейса. Всеки от класовете трябва да има и конструктор, който задава вида на машината. Създайте клас MachineOperator, чиято цел е да пуска и спира машина, която му се подава чрез конструктора

# Решение: Машина (1)

- Нека да създадем интерфейса за машина и да заложим методите за пускане и спиране

```
interface IMachine
{
  string MachineType { get; set; }
  bool Start();
  bool Stop();
}
```

# Решение: Машина (2)

- Нека да създадем класа Car, имплементиращ интерфейса

```
public string MachineType { get; set; }
public Car() {
  this.MachineType = "Car";
}
public bool Start() {
  Console.WriteLine("Car starting...");
  return true;
}
public bool Stop() {
  Console.WriteLine("Car stopping...");
  return true;
}
```

# Решение: Машина (3)

- Останалите класове – Truck, Airplane, LawnMower, имат напълно аналогична имплементация с тази на Car.
- Следва дефиниране на класа за MachineOperator

# Решение: Машина (4)

- Нека да създадем класа MachineOperator

```
private IMachine entity;
public IMachine Entity { get { return this.value; }
  set {
    this.entity = value;
    Console.WriteLine("Machine operator is operating:
"+value.MachineType);
  }
}
public MachineOperator(IMachine entity){
  this.Entity = entity;
}
```

# Решение: Машина (5)

- А сега да добавим методи Start() и Stop():

```
public bool Start() {
  return Entity.Start();
}

public bool Stop() {
  return Entity.Stop();
}
```

# Решение: Машина (6)

- В Program.cs създаваме по един обект от всяка машина и един обект от MachineOperator

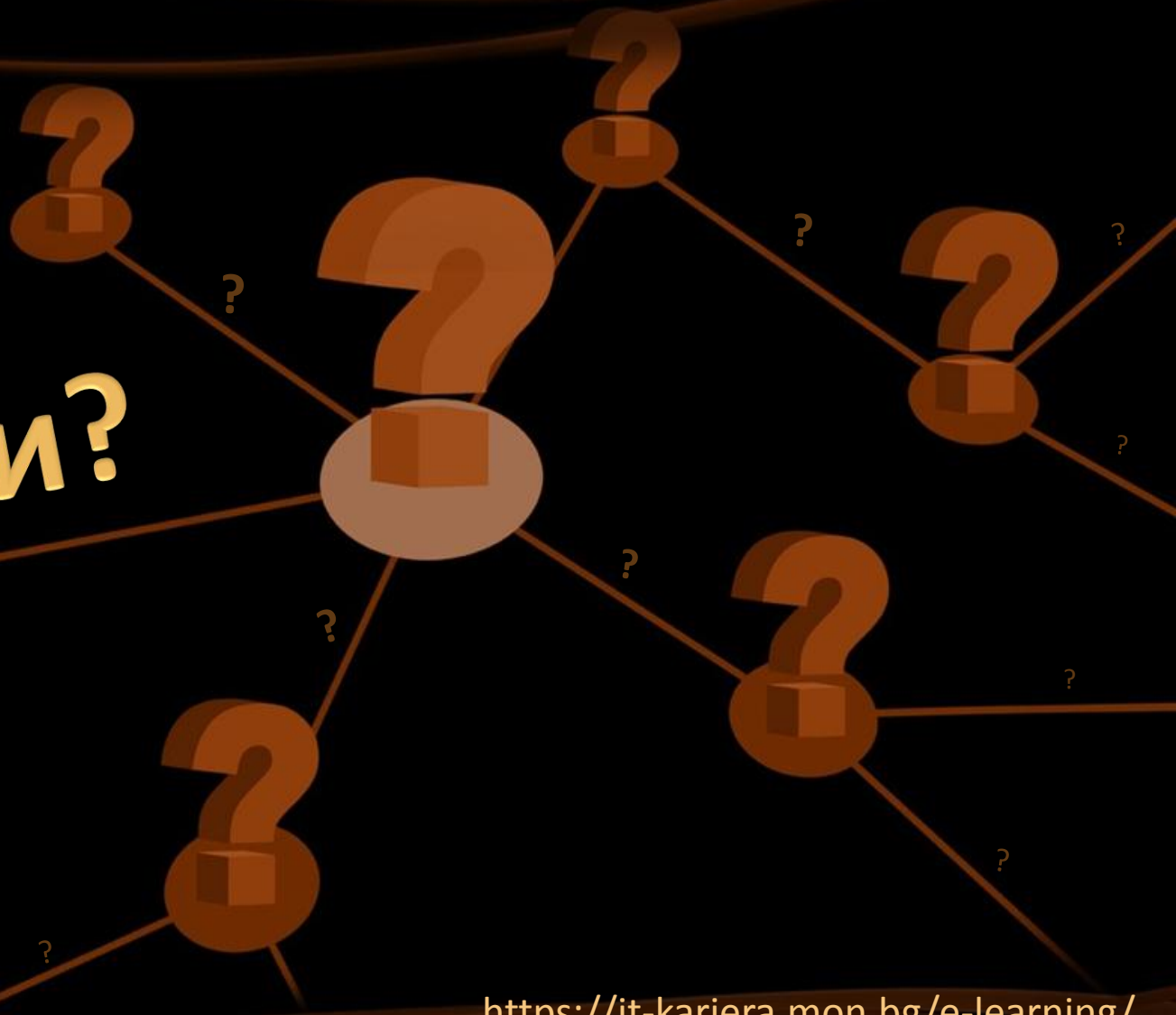
```
Car car = new Car();
LawnMower lawnMower = new LawnMower();
Airplane airplane = new Airplane();
Truck truck = new Truck();

MachineOperator mo = new MachineOperator(car);
mo.Start(); //пускаме машината
mo.Stop(); //спираме машината
mo.Entity = lawnMower; //сменяме машината
mo.Start();
mo.Stop();
//TODO: аналогично можем да сменим машината с другите
```

# Какво научихме днес?

- Интерфейсите ни позволяват да постигнем полиморфично поведение, което да не зависи от кода и да е лесно за промяна.
- Можем с лекота да добавим още класове, имплементиращи даден интерфейс без да счупим кода, като отразим и спецификите им

# Интерфейси

Въпроси?



https://it-kariera.mon.bg/e-learning/

# Лиценз

- Настоящият курс (слайдове, примери, видео, задачи и др.) се разпространяват под свободен лиценз "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International"


- Благодарности: настоящият материал може да съдържа части от следните източници
  - Книга "Основи на програмирането със C#" от Светлин Наков и колектив с лиценз CC-BY-SA