Objektorienterad programmering

www.everyloop.com

Objektorienterad Programmering

Fyra grundläggande principer

Inkapsling

(Gruppera information)

Abstraktion

(Gömma information)

Arv

(Dela information)

Polymorfism

(Omdefiniera information)

Inkapsling

Abstraktion

Arv

Polymorfism

Klass: Djur

Egenskaper

Namn (String)
Vikt (Double)
Färg (Color)

Interna tillstånd

Hungrig (Bool)
Trött (Bool)

Metoder

Spring();
Prata();

Private

Public

Protected

Internal

Klass: Katt

Egenskaper

Namn: Morris
Vikt: 3.8 kg
Färg: Svart
Har päls
Har 4 ben

Interna tillstånd

Hungrig: Nej
Trött: Nej

Saker objektet kan göra

Springa
Jama

Hungrig: Ja
Trött: Nej

Namn: Findus
Vikt: 3.3 kg
Färg: Brun

Hungrig: Ja
Trött: Nej

Namn: Leo
Vikt: 4.2 kg
Färg: Gul

Klass: Fågel

Namn: Fenix
Vikt: 38 g
Färg: Grön
Har fjädrar
Har 2 ben

Hungrig: Ja
Trött: Nej
Springa
Kvittra
Flyga

Hungrig: Ja
Trött: Ja

Namn: Pippin
Vikt: 33 g
Färg: Blå

Overloading

Method Overloading

Jama();

Jama(int volym);

Jama(double volym);

Operator Overloading

2 + 3 = 5

”abc” + ”def” = ”abcdef”

 +  = 

Vikt: 7.5 kg

Färg: Randig

Har 8 ben

Klassdefinition och instanser

Autoproperty

```
private string _name;
public string Name
    { get { return _name;  }
    { set _name = value; }
}
```

Klass: Cat

Egenskaper

```
public string Name { get; set; }
public double Weight
{
    get { return _weight + (hungry ? 0 : 0.5); }
    set { _weight = value; }
}
```

Fält

```
private bool hungry = false;
private bool tired = false;
private double _weight = 4.0;
```

Metoder

```
public void Talk()
{
    Console.WriteLine(”Mjaou!”);
}
```

Overloaded method

```
public void Talk(string text)
{
    Console.WriteLine(text);
}
```

Konstruktor

```
public Cat(string name)
{
    this.Name = name;
}
```

Overloaded constructor

```
public Cat()
{
    this.Name = ”Unknown”;
}
```

```
Cat blackCat = new Cat(”Morris”);
Cat brownCat = new Cat(”Findus”);
Cat yellowCat = new Cat();
```

Findus

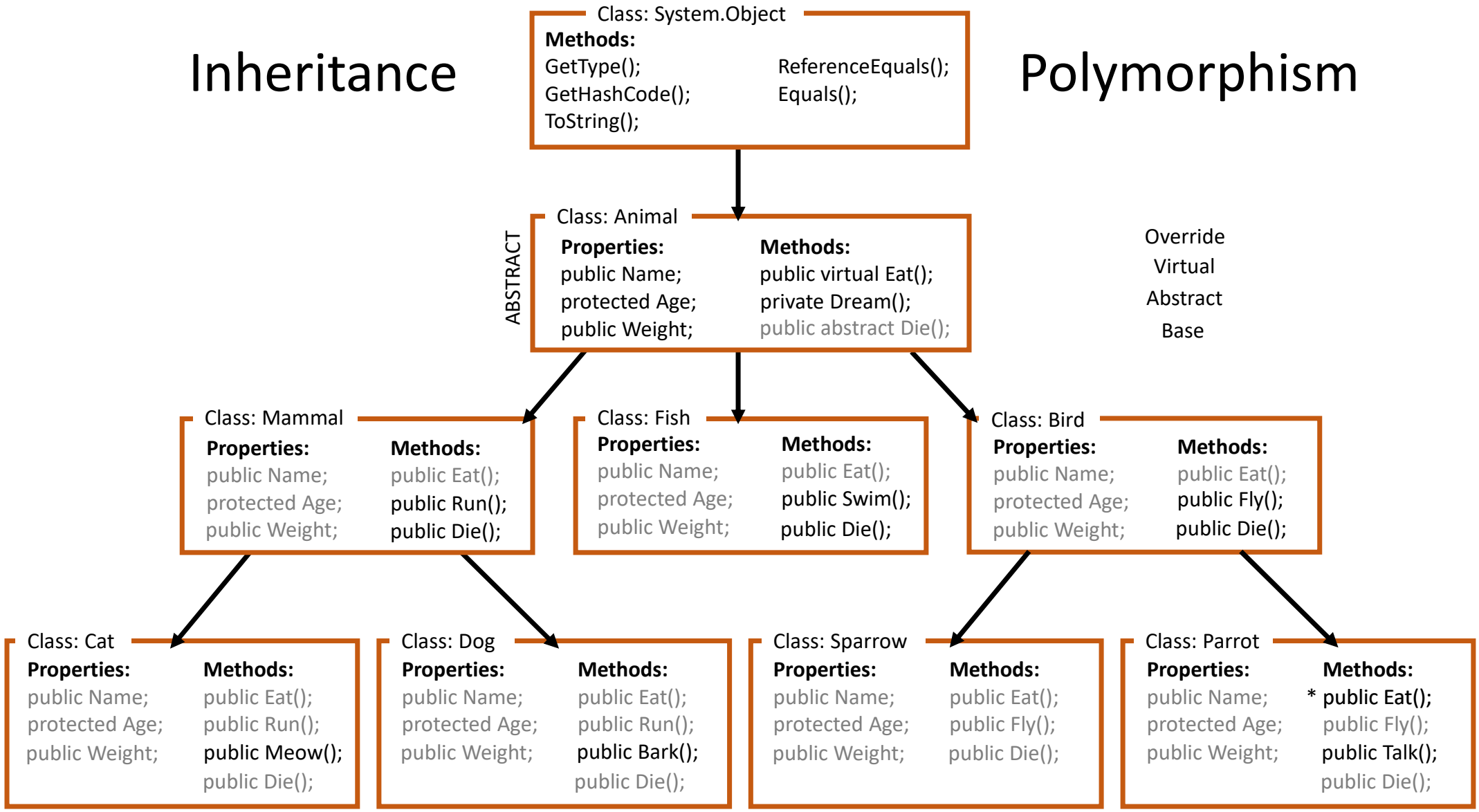
Findus

Leo

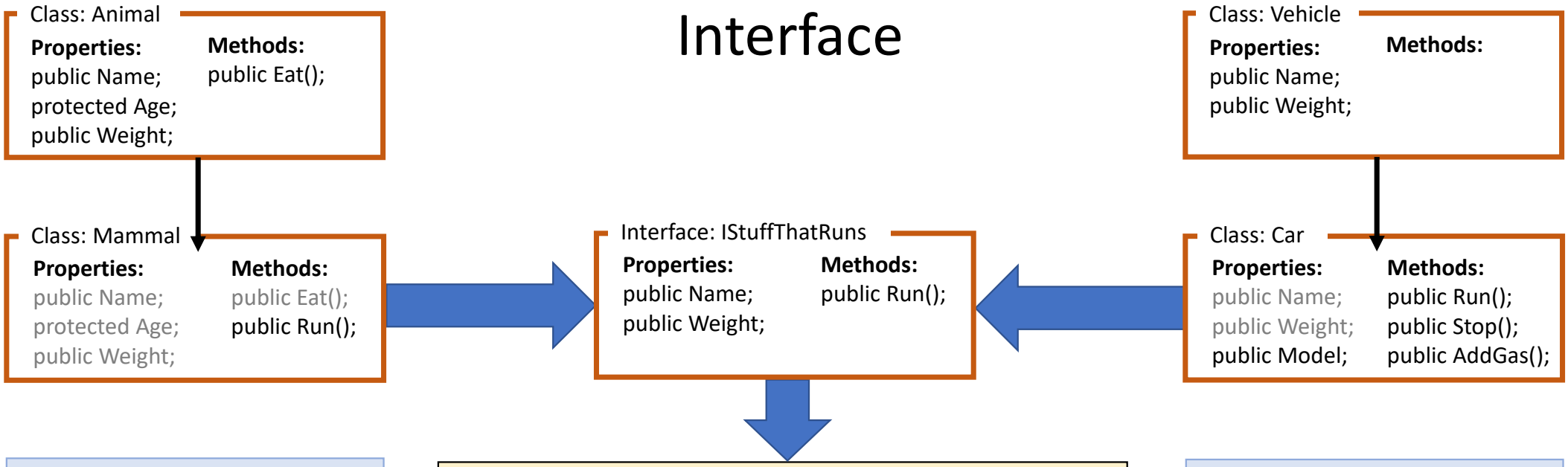
```
yellowCat.Name = ”Leo”;
blackCat.Name = brownCat.Name;
blackCat.Talk();
Console.WriteLine($”A cat has {Cat.GetLives()} lives.”);
```

Inheritance

Polymorphism



Interface



IEnumerable
För att en klass ska kunna användas ihop med *foreach* krävs det att den är "uppräkningsbar", d.v.s. att den implementerar Ienumerable.

IComparable
Implementeras för att en klass ska vara sorterbar i en lista. Om vi sorterar en lista med katter, ska de sorteras efter namn eller vikt?

```
List<IStuffThatRuns> items = new List<IStuffThatRuns>();
items.Add(new Mammal());
items.Add(new Car());

foreach (IStuffThatRuns item in items)
{
    Console.WriteLine($"Make {item.Name} run!");
    item.Run;
}
```

IFormattable
.ToString()-metoden tillåter att man anger en formateringssträng som säger vad som ska skrivas, om klassen implementerat denna.

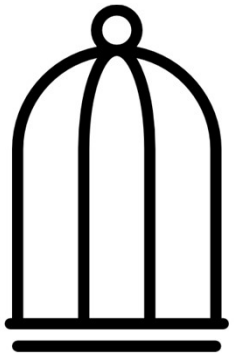
IDisposable
Används tillsammans med *using*
För att garantera att resurser frigörs när de lämnar "using"-scope.

Generics

Cage<Ball> ballCage = new Cage<Ball>();

ballCage.Inhabitant = ball;

Bird pippin = new Bird();



```
public class  Cage<T>
{
    public T Inhabitant { get; set; }
}
```

Bird fenix = new Bird();

Cat findus = new Cat();

Ball ball = new Ball();

Generic collections

List

```
List<Cat> cats = new List<Cat>();
cats.Add(leo);
cats.Add(morris);
cats.Add(findus);
Cat firstCat = cats[0];
```

LIFO = Last In First Out

Stack

```
Stack<Cat> stack = new Stack<Cat>();
stack.Push(leo);
stack.Push(morris);
stack.Push(findus);
Cat topCat = stack.Pop();
```

Key-Value Pair

Dictionary

```
Dictionary<Cat, Bird> dict = new Dictionary<Cat, Bird>();
dict.Add(morris, fenix);
dict.Add(findus, pippin);
dict.Add(leo, fenix);
Bird findusBird = dict[findus];
```

FIFO = First In First Out

Queue

```
Queue<Cat> queue = new Queue<Cat>();
queue.Enqueue(morris);
queue.Enqueue(leo);
queue.Enqueue(findus);
Cat nextCat = queue.Dequeue();
```

Extensions

Static Class: CatMethods

```
public static void Drink(Cat cat)
{
    Console.WriteLine($"{cat.Name} is drinking!");
    cat.Weight += 0.1;
}
```

```
Cat myCat = new Cat("Morris");
CatMethods.Drink(myCat);

myCat.Drink();

Cat cat2 = new Cat("Findus");
cat2.Hug(myCat);

string s = myCat.CompareTo(cat2);
Console.WriteLine(s);
```

```
Morris is drinking!
Morris is drinking!
Findus hugs Morris!
Both have the same weight.
```

Class: Cat

Properties:

public Name;
protected Age;
public Weight;

Methods:

public Eat();
public Run();
public Meow();
public Die();

Extension Methods:

public void Drink();
public void Hug(Cat c2);
public string CompareTo(Cat c2);

Static Class: Extensions

```
public static void Drink(this Cat cat)
{
    Console.WriteLine($"{cat.Name} is drinking!");
    cat.Weight += 0.1;
}

public static void Hug(this Cat c1, Cat c2)
{
    Console.WriteLine($"{c1.Name} hugs {c2.Name}!");
}

public static string CompareTo(this Cat c1, Cat c2)
{
    if (c1.Weight > c2.Weight)
    {
        return $"{c1.Name} weighs more.";
    }
    else if (c1.Weight < c2.Weight)
    {
        return $"{c2.Name} weighs more.";
    }
    else return "Both have the same weight.";
}
```