OOP-principper i Zoo Management System

Abstraktion (Abstraction)

Abstraktion betyder, at man konceptualiserer objekter og fokuserer på fælles træk ved en gruppe objekter og skjuler implementeringsdetaljerne. Man skaber overordnede typer (abstrakte kasser eller interfaces), som definerer karaktertræk/attributter og adfærd/metoder, som objekter af samme type har til fælles. Man laver ikke en klasse for hvert enkelt objekt, men grupperer typer af objekter i abstraktioner/*klasser*. Det gør koden mere overskuelig og fleksibel.

Eksempler i Zoo Management System (ZMS)
ZMS bruger klasser til abstraktion. **Animal** har sin egen abstrakte klasse, hvor det er defineret, at alle objekter af typen Animal har nogle fælles attributter – noget der kendetegner Animals (*fields* og *properties*) – og fælles måder at opføre sig på (*metoder*). De har alle et *navn*, en *art* og en *fødselsdato*, og de kan alle *spise*, *sove* og *lave lyde*. Abstrakte klasser kan ikke instansieres direkte.

```
abstract class Animal { … }

    private string _name;
    private string _species;
    private DateTime _birthdate;

    public virtual void Sleep() { … }
    public virtual string Eat() { … }
```

Eksempler på abstraktioner (typer/klasser) i ZMS er *Zookeeper*, *Zoo* og *Enclosure*. Subklasserne *Lion*, *Giraffe*, *Penguin* og *Elephant* arver attributter og metoder fra basen *Animal*, og implementerer alle de samme abstrakte metoder.

Polymorfi (Polymorphism)

Polymorfi betyder, at man kan bruge et objekt uden at kende dets konkrete type, og at metoder kan opføre sig forskelligt afhængigt af, hvilken klasse de er i.

Eksempler i Zoo Management System (ZMS)
Brug af polymorfe objekter i Zoo.cs: Når man har objekter af typerne Lion, Giraffe, Elephant og Penguin, der nedarver fra Animal og man behandler dem som Animal-objekter (gennem en List af typen Animal), er objekterne polymorfe, f.eks.:

```
Enclosure:        public List<Animal> Animals => _animals
Zoo:            foreach (var enclosure in _enclosures)
                    foreach (var animal in enclosure.Animals)
```

Metoden GetSound() i Animals subklasser tillader hver undertype af Animal få Animals nedarvede metode MakeSound(), der kalder Animals abstrakte metode GetSound() til at opføre sig forskelligt. Animal-objekter laver forskellige lyde afhængigt af hvilken subklasse, de tilhører. En metode er polymorfisk, når den er overskrevet og kaldes gennem en baseklasse-reference, og det først afgøres, når programmet kører, hvilken konkret metode, der implementeres, f.eks.:

```
Animal dumbo = new Elephant("Dumbo", new DateTime(1941, 10, 23));
//Polymorfi: dumbo instansieret som Animal, hvor metode kaldes. (Animal.MakeSound()).
//Præcis type og metode (Elephant) afgøres først, når program kører.
dumbo.MakeSound();

Animal:         public virtual void MakeSound() { … says: {GetSound() … }…}
Animal:         public abstract string GetSound();
Elephant:       public override string GetSound() { … }
```

Nedarvning (Inheritance)

Nedarvning betyder, at attributter og metoder fra base-klassen kan arves og genbruges og specialiseres af dens subklasser i en hierakisk struktur. Fælles attributter og metoder er i base-klassen, og de enkelte subklasser specialiserer sig i det, der adskiller subklasserne fra hinanden.

Eksempler i Zoo Management System (ZMS)
Base-klassen Animal specificerer, at alle Animal-objekter og objekter af dens subtyper (Lion, Elephant, Giraffe og Penguin) har et navn, en art og en fødselsdato, hvilket nedarves til subtyperne, som gennem fields og metoder definererer, hvad det betyder for dem, og gennem constructors, hvad det betyder for det enkelte objekt, når det instansieres. Subklasserne specificerer her selv, at de har lyde, og hvilke lyde de har. De ting, der er unikke for hvert enkelt objekt specificeres, når objektet instansieres – på nær deres art, som bliver fastsat i base-kaldet i constructoren i hver subklasse.

```
abstract class Animal { … }
    private string _name;

enum ElephantSounds { … }
internal class Elephant : Animal { … }
    private string _sound = "Tooooooot";

    public Elephant(string name, DateTime birthdate)
        : base(name, species: "Elephant", birthdate) { … }
```

Indkapsling (Encapsulation)

Indkapsling betyder, at objekternes data (felter) og adfærd (metoder) holdes samlet og skjult og kun kan tilgås via klart definerede grænseflader – properties og metoder.

Eksempler i Zoo Management System (ZMS)

```
internal class Giraffe : Animal

    private string _sound = "Sighhhhh";
    public string Sound { get { return _sound; } set { _sound = value; } }
```

Felter, f.eks. _sound og _animals er private og skjulte for omverdenen. Adgang sker gennem properties, Sound og Animals, der kontrollerer, hvordan data læses og skrives. Property-listen Animal er en readonly property, og giver kun adgang til listen, ikke mulighed for at ændre den; en form for beskyttet adgang.

```
internal class Enclosure

    private List<Animal> _animals = new();
    public List<Animal> Animals => _animals; //shortform getter: kun læsbar
    udenfor klassen
```