Politechnika Wrocławska
Wydział Mechaniczny
Katedra Konstrukcji i Badań Maszyn i Pojazdów

# Programowanie obiektowe Lab 10

Dr inż. Paweł Maślak

pawel.maslak@pwr.edu.pl

## Laboratorium 10: Klasy abstrakcyjne, Interfejsy

### 1. Klasa abstrakcyjna

Po pierwsze klasa abstrakcyjna, jest pewnego rodzaju uogólnieniem klas pochodnych. Najprościej rzecz ujmując, klasa taka ma sens w przypadku, gdy kilka klas pochodnych wymaga użycia tej samej funkcji, ale z innymi jej implementacjami. Można przedstawić to na wiele sposobów. Częste przykłady klasy abstrakcyjnej to Figura lub Zwierzę.

**Implementacja klas abstrakcyjnej:**

Poniższy kod jest trywialny, ale oddaje w pełni ideę klasy abstrakcyjnej. Na pierwszy rzut oka widać słowo kluczowe **abstract** w tworzeniu obiektu klasy. Wewnątrz dodane zostały cztery metody abstrakcyjne do pobierania *nazwy*, *ceny*, *procentu rabatu* oraz *kategorii*. Ostatnia metoda w klasie służy do pobrania *nazwy magazynu*. Z racji założeń poczynionych przez biznes, okazało się, że jest tylko jeden magazyn i więcej nie będzie w planach. Zostało więc zaproponowane, że metoda z pobraniem nazwy magazynu będzie zaimplementowana w klasie abstrakcyjnej **ProductToSell**.

Następnie dwie publiczne klasy **Book** oraz **Cup** dziedziczą z klasy abstrakcyjnej **ProductToSell**. Powoduje to sytuację, gdzie wszystkie metody abstrakcyjne muszą zostać nadpisane i zaimplementowane. Służy do tego słowo kluczowe **override**. W klasie **Program** następuje deklaracja dwóch wyżej przedstawionych klas i wypisanie wyników na ekran konsoli. Warto skopiować kod do IDE i spróbować na przykład zadeklarować w funkcji **Main** obiekt klasy **ProductToSell**.

```
namespace Klasa_abstrakcyjna
{
    public abstract class ProductToSell
    {
        public abstract string getName();
        public abstract double getPrice();
        public abstract int getPercentDiscount();
        public abstract string getCategory();

        public string getWarehouse()
        {
            return "Warehouse number 1";
        }
    }

    public class Book : ProductToSell
    {
        public override string getName()
        {
            return "Prawdziwa historia McDonald's";
        }

        public override double getPrice()
        {
            return 24.99;
        }

        public override int getPercentDiscount()
        {
            return 5;
        }
        public override string getCategory()
        {
            return "Książka";
        }
    }

    public class Cup : ProductToSell
    {
        public override string getName()
        {
            return "Kubek programisty";
        }

        public override double getPrice()
        {
```

```
            return 21.99;
        }

        public override int getPercentDiscount()
        {
            return 0;
        }
        public override string getCategory()
        {
            return "Kubek";
        }
    }

    public class Program
    {
        static void Main(string[] args)
        {
            Book book = new Book();
            Console.WriteLine("Kategoria: " + book.getName());
            Console.WriteLine("Nazwa: " + book.getCategory());
            Console.WriteLine("Cena: " + book.getPrice().ToString());
            Console.WriteLine("Rabat [%]: " + book.getPercentDiscount().ToString());
            Console.WriteLine("Magazyn: " + book.getWarehouse());
            Console.WriteLine(Environment.NewLine);

            Cup cup = new Cup();
            Console.WriteLine("Kategoria: " + cup.getName());
            Console.WriteLine("Nazwa: " + cup.getCategory());
            Console.WriteLine("Cena: " + cup.getPrice().ToString());
            Console.WriteLine("Rabat [%]: " + cup.getPercentDiscount().ToString());
            Console.WriteLine("Magazyn: " + cup.getWarehouse());
            Console.ReadLine();
        }
    }
}
```

## 2. Interfejsy

Interfejsy są deklarowane za pomocą słowa kluczowego interface. Deklaracja ta jest podobna do deklaracji klasy. Interfejsy domyślnie ustawiane są jako publiczne.

**Deklaracja:**

```
public interface INazwaInterfejsu
{
  // składowe interfejsu
  void WyswietlDane();
  double PoliczPensje();
}
```

**Przykład:**

```
using System;
namespace Interfejs
{
  class Program
  {
    static void Main(string[] args)
    {
      Transakcje t1 = new Transakcje("01", "25/11/2023", 331);
      Transakcje t2 = new Transakcje("02", "26/11/2023", 3321);
      t1.WyswietlDane();
      t2.WyswietlDane();
      Console.ReadKey();
      // Wynik działania programu
      //Kod: 01
      //Data: 25/11/2023
      //Ilosc: 331
      //Kod: 02
      //Data: 26/11/2023
      //Ilosc: 3321
    }
  }
  public interface ITransakcje
  {
    // składowe interfejsu
    void WyswietlDane();
    int PoliczIlosc();
  }
  public class Transakcje : ITransakcje
  {
    private string kod;
    private string data;
    private int ilosc;
    public Transakcje()
    {
      kod = "";
      data = "";
      ilosc = 0;
    }
    public Transakcje(string k, string d, int i)
    {
      kod = k;
      data = d;
      ilosc = i;
    }
    public int PoliczIlosc()
    {
      return ilosc;
    }
    public void WyswietlDane()
    {
      Console.WriteLine("Kod: {0}", kod);
      Console.WriteLine("Data: {0}", data);
      Console.WriteLine("Ilość: {0}", ilosc);
    }
  }
}
```

## 3. Różnice między klasami abstrakcyjnymi a interfejsami

- W interfejsach wszystkie metody są abstrakcyjne, natomiast w klasie abstrakcyjnej można stworzyc metody posiadające ciało, jak i abstrakcyjne.
- W php można dzidziczyć jedynie po jednej klasie, natomiat interfejsów, można implementować wiele. Ponadto interfejsy mogą dziedziczyc wiele interfejów
- Klasa abstrakcyjna zazwyczaj jest mocno związana z klasami dziedziczącymi w sensie logicznym, czyli np. tworzymy klasę abstrakcyjną Planeta po której dziedziczą konkretne klasy planet (np Ziemia, Mars). Interfejs natomiast nie musi być już tak mocno związany z daną klasą, on określa jej cechy, np możesz stoworzyć interfejs Zniszczalny, który mówi że dany obiekt może zostać zniszczony. Taki interfejs możesz nadać zarówno klasą Planeta, Gwiazda, Budynek itp.

**Zadania do zrobienia**:

1. Napisz klasę IOString która będzie wyposażona w metody:

   - GetString – przyjmuje zmienną tekstową (string) od użytkownika
   - PrintString – wyświetla tę zmienną drukowanymi literami
   - IsPalindrome – sprawdzającą czy zmienna tekstowa jest palindromem.

2. Zaimplementuj kod:

```
abstract class Animal
{
  public virtual string Describe()
  {
    return "Not much is known about this animal!";
  }
}
class Dog : Animal
{
}
```

   a) Stwórz klasę abstrakcyjną i zobacz jaki jest wynik próby stworzenia jej instancji.
   b) Stwórz obiekt typu **Dog**, **Cat**, **Cow** i wywołaj metodę **Describe**.
   c) Wykorzystując zdobytą dotychczas wiedzę, rozszerz funkcjonalność klasy **Animal** w 3 utworzonych klasach o wypisanie dodatkowych informacji.
   d) Pobieraj od użytkownika informacje o dźwięku jaki wydaje dane zwierzę (w każdym obiekcie inny) i wyświetl nazwę zwierzęcia (**obiektu**) i jaki dźwięk wydaje (**Sound**) wszystkich obiektów.

3. Zaimplementuj interfejs **Shape**, który będzie zawierał metodę **Draw**. Następnie napisz klasy implementujące ten interfejs – **Rectangle** (dla prostokąta), **Triangle** (dla trójkąta), **Circle** (dla koła). Metoda **Draw**() ma wyrysować daną figurę za pomocą „*****”. Dodatkowo napisz klasę **Square**, dziedziczącą po klasie **Rectangle**. Dla klasy **Square** użyj metody klasy nadrzędnej **Draw**(), aby wyrysować kwadrat złożony z „*****”.