Pierwszy program w Visual Studio

Czas: 2 × 45 min | praca indywidualna | Wykład 1.01

Cel: Przygotujesz stanowisko pracy: sprawdzisz instalację Visual Studio 2026 i .NET 10, utworzysz pierwszą aplikację konsolową i zapiszesz ją w lokalnym repozytorium Git.

Po zajęciach potrafisz:

- sprawdza wersję zainstalowanego SDK poleceniem `dotnet`
- tworzy i uruchamia projekt konsolowy w Visual Studio
- rozróżnia uruchomienie z debugowaniem (F5) i bez (Ctrl+F5)
- zapisuje projekt w repozytorium Git i wykonuje commit

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0
- **Git for Windows** — lub Git wbudowany w Visual Studio git-scm.com/downloads/win

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Sprawdzenie środowiska

Upewnij się, że komputer ma potrzebne narzędzia.

1. Otwórz Terminal (Windows Terminal lub PowerShell) i wpisz `dotnet --list-sdks`. Zapisz numer wersji SDK 10.x.
2. Wpisz `dotnet --info` i odczytaj system operacyjny oraz architekturę (RID).
3. Uruchom Visual Studio 2026 i sprawdź wersję w menu Pomoc → Informacje o programie Microsoft Visual Studio.
4. Wpisz `git --version`. Jeśli polecenie nie działa, zgłoś to nauczycielowi.

2 Projekt konsolowy

Utwórz projekt **Wizytowka**.

1. Wybierz Utwórz nowy projekt → szablon **Aplikacja konsolowa** (C#) — nie „.NET Framework".
2. Nazwa projektu: `Wizytowka`, lokalizacja: Twój folder z projektami. Platforma docelowa: **.NET 10.0 (LTS)**.
3. Pozostaw instrukcje najwyższego poziomu (nie zaznaczaj „Nie używaj instrukcji najwyższego poziomu").
4. Uruchom program klawiszami Ctrl+F5, a potem F5. Zapisz, czym różni się zachowanie okna konsoli.

3 Wizytówka

Zastąp zawartość `Program.cs` własnym kodem wypisującym wizytówkę.

```
Console.Title = "Wizytówka";
string imie = "Jan";
Console.ForegroundColor = ConsoleColor.Cyan;
Console.WriteLine($"  {imie} ...");
Console.ResetColor();
Console.WriteLine($"Data: {DateTime.Now:dd.MM.yyyy}");
```

1. Zadeklaruj zmienne `imie`, `nazwisko`, `klasa` z Twoimi danymi.
2. Wypisz ramkę ze znaków `=` i dane w środku, używając interpolacji `$"..."`.
3. Zmień kolor tekstu wizytówki (`Console.ForegroundColor`), a potem przywróć domyślny (`Console.ResetColor()`).
4. Wypisz bieżącą datę w formacie dd.MM.yyyy i godzinę HH:mm oraz wersję .NET (`Environment.Version`).
5. Ustaw tytuł okna konsoli: `Console.Title = "Wizytówka";`. Wykonaj zrzut ekranu działającego programu.

4 Repozytorium Git

Zapisz pracę w historii wersji.

1. W folderze rozwiązania (tam, gdzie plik `.slnx`) wykonaj `dotnet new gitignore`.
2. Wykonaj `git init`, `git add .` i `git commit -m "Pierwsza wersja wizytówki"`.
3. Zmień kolor wizytówki, zatwierdź drugi commit i wyświetl historię `git log --oneline`.
4. Sprawdź `git status`, że foldery `bin` i `obj` nie są śledzone. Zrób zrzut ekranu terminala.

W sprawozdaniu umieść

- ☐ wersje SDK, Visual Studio i Gita
- ☐ kod `Program.cs`
- ☐ zrzut ekranu działającej wizytówki
- ☐ opis różnicy między F5 i Ctrl+F5
- ☐ zrzut `git log --oneline` z dwoma commitami

Pytania kontrolne

1. Do czego służy plik `.gitignore` i dlaczego nie zapisujemy w repozytorium folderów `bin` i `obj`?
2. Czym różni się uruchomienie programu klawiszem F5 od Ctrl+F5?
3. Co oznacza skrót LTS przy .NET 10 i dlaczego szkoła wybiera wersje LTS?
4. Gdzie w projekcie zapisana jest wersja platformy docelowej (`net10.0`)?

Kryteria oceny

Kryterium	Punkty
Sprawdzenie środowiska i zapis wersji	3
Projekt utworzony poprawnie (.NET 10, nazwa, lokalizacja)	3
Wizytówka: dane, ramka, kolory, data	6
Repozytorium z dwoma commitami i .gitignore	4
Sprawozdanie i odpowiedzi na pytania	4
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Dane z klawiatury i obliczenia

Czas: 2 × 45 min | praca indywidualna | Wykład 1.02–1.03

Cel: Napiszesz dwa programy obliczeniowe, które bezpiecznie wczytują liczby z klawiatury i odrzucają błędne dane.

Po zajęciach potrafisz:

- wczytuje dane metodą `Console.ReadLine` i zamienia je na liczby metodą `TryParse`
- powtarza pytanie do skutku w pętli przy błędnych danych
- wykonuje obliczenia na typie `double` i formatuje wynik
- testuje program na danych poprawnych, granicznych i błędnych

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Koszt przejazdu

Program pyta o długość trasy (km), spalanie (l/100 km), cenę paliwa (zł/l) i liczbę osób. Wypisuje zużycie paliwa, koszt całkowity i koszt na osobę.

1. Utwórz projekt konsolowy `Obliczenia` i zapisz wzór: paliwo = km × spalanie / 100, koszt = paliwo × cena.
2. Wczytaj dane metodą `double.TryParse(Console.ReadLine(), out double x)`.
3. Wynik wypisz z dwoma miejscami po przecinku (`{koszt:F2}`) i jednostką.
4. Sprawdź wynik na kalkulatorze dla danych: 250 km, 6,5 l, 6,19 zł, 3 osoby.

2 Odporność na błędy

Wydziel wczytywanie liczby do funkcji lokalnej, która pyta tak długo, aż dane będą poprawne.

```
double Wczytaj(string pytanie, double min, double max)
{
    while (true)
    {
        Console.Write(pytanie);
        if (double.TryParse(Console.ReadLine(), out double x) && x >= min && x <= max)
            return x;
        Console.WriteLine($"  Błąd: podaj liczbę od {min} do {max}.");
    }
}
```

1. Napisz funkcję `Wczytaj(string pytanie, double min, double max)` zgodnie z szablonem.
2. Dodaj sensowne zakresy (np. trasa 1–5000 km, spalanie 1–40 l).
3. Sprawdź działanie dla tekstu `abc`, pustego wiersza, liczby ujemnej i liczby z kropką zamiast przecinka.
4. Zapisz w tabeli, jak program zareagował na każdy przypadek.

3 Kalkulator BMI

W tym samym programie dopisz drugą część: BMI = masa / wzrost² (wzrost w metrach).

1. Wczytaj masę w kilogramach i wzrost w centymetrach funkcją `Wczytaj`; zamień centymetry na metry.
2. Wypisz BMI z jednym miejscem po przecinku.
3. Dodaj kategorię: poniżej 18,5 — niedowaga, poniżej 25 — waga prawidłowa, poniżej 30 — nadwaga, w przeciwnym razie otyłość (operator `?:`).
4. Przetestuj wartości graniczne: BMI dokładnie 18,5, 25 i 30 (dobierz masę dla wzrostu 200 cm).
5. Zatwierdź pracę w Git (`git commit`).

W sprawozdaniu umieść

- ☐ kod programu
- ☐ tabela testów z wynikami
- ☐ zrzut ekranu programu dla błędnych i poprawnych danych
- ☐ wyjaśnienie, dlaczego `double.Parse` jest gorszym wyborem niż `TryParse`

Pytania kontrolne

1. Co zwraca metoda `TryParse` i co trafia do parametru `out`, gdy konwersja się nie uda?
2. Dlaczego w polskim Windows liczba `6.5` może nie zostać rozpoznana?
3. Jakie dane testowe nazywamy granicznymi? Podaj przykład z tego zadania.
4. Dlaczego warto wydzielić wczytywanie do osobnej funkcji?

Kryteria oceny

Kryterium	Punkty
Koszt przejazdu — poprawne obliczenia i format	5
Funkcja Wczytaj z zakresem i pętlą	4
BMI z kategoriami	4
Tabela testów (poprawne, graniczne, błędne)	4
Sprawozdanie i pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Konwersje i typy liczbowe

Czas: 2 × 45 min | praca indywidualna | Wykład 1.03

Cel: Napiszesz przeliczniki jednostek i sprawdzisz w praktyce różnice między typami `int`, `double` i `decimal` oraz sposobami konwersji.

Po zajęciach potrafisz:

- dobiera typ liczbowy do rodzaju danych (pieniądze — `decimal`)
- porównuje dokładność obliczeń na `double` i `decimal`
- rozróżnia rzutowanie `(int)` od `Convert.ToInt32`
- wyjaśnia przepełnienie typu `int`

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Przelicznik temperatur

Program wczytuje temperaturę w °C i wypisuje ją w °F i K.

1. Utwórz projekt `Przeliczniki`. Wzory: F = C × 9/5 + 32, K = C + 273,15.
2. Wczytaj liczbę przez `double.TryParse`; przy błędzie wypisz komunikat.
3. Sprawdź wyniki dla 0, 100, −40 i 36,6 °C i wpisz je do tabeli.
4. Sprawdź, co się stanie, gdy zapiszesz wzór jako `C * (9 / 5) + 32` — wyjaśnij wynik.

2 Przelicznik walut

Kurs podaje użytkownik — program nie pobiera go z internetu (zrobimy to w roku 2).

1. Wczytaj kwotę w złotych i kurs euro jako `decimal`.
2. Odrzuć kurs mniejszy lub równy zero.
3. Wypisz wynik z dwoma miejscami po przecinku i symbolem EUR.

3 Eksperyment: double kontra decimal

Uruchom kod i zapisz wyniki.

```
double sd = 0; decimal sm = 0;
for (int i = 0; i < 10; i++) { sd += 0.1; sm += 0.1m; }
Console.WriteLine($"double:  {sd:R}, równe 1? {sd == 1.0}");
Console.WriteLine($"decimal: {sm}, równe 1? {sm == 1.0m}");
```

1. Dodaj dziesięć razy 0,1 do zmiennej `double` i do zmiennej `decimal`.
2. Wypisz obie sumy (dla `double` użyj formatu `:R`) i wynik porównania z 1.
3. Zapisz wniosek: którego typu użyć do kwot pieniężnych i dlaczego.

4 Eksperyment: konwersje i przepełnienie

Sprawdź, jak zachowują się różne konwersje.

1. Porównaj `(int)2.7` i `Convert.ToInt32(2.7)`.
2. Porównaj `Convert.ToInt32(2.5)` i `Convert.ToInt32(3.5)` — sprawdź w dokumentacji, co to jest „zaokrąglanie bankierskie".
3. Do zmiennej `int duza = int.MaxValue;` dodaj 1 w bloku `unchecked` i wypisz wynik.
4. Powtórz z blokiem `checked` i zapisz, jaki wyjątek wystąpił.
5. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod programu
- ☐ tabela wyników przeliczeń
- ☐ wyniki eksperymentów z wnioskami

Pytania kontrolne

1. Dlaczego `9 / 5` w C# daje 1, a `9.0 / 5` daje 1,8?
2. Dlaczego suma dziesięciu 0,1 w typie `double` nie jest równa 1?
3. Czym różni się `(int)x` od `Convert.ToInt32(x)`?
4. Co to jest przepełnienie i jak się przed nim zabezpieczyć?

Kryteria oceny

Kryterium	Punkty
Przelicznik temperatur i tabela	5
Przelicznik walut na decimal z walidacją	4
Eksperyment double/decimal z wnioskiem	4
Eksperyment konwersji i przepełnienia	4
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Formatowanie wyników — paragon

Czas: 2 × 45 min | praca indywidualna | Wykład 1.04

Cel: Wydrukujesz w konsoli czytelny paragon sklepowy z wyrównanymi kolumnami, kwotami w złotych, rabatem i podatkiem VAT.

Po zajęciach potrafisz:

- wyrównuje kolumny tekstu za pomocą formatu `{x,-20}` i `{x,10}`
- formatuje kwoty walutą zgodnie z kulturą pl-PL
- zaokrągla kwoty `Math.Round` z wybranym trybem `MidpointRounding`
- oblicza rabat i VAT zawarty w cenie brutto

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Dane i nagłówek

Utwórz projekt `Paragon`. Dane towarów zapisz w trzech tablicach.

```
Console.WriteLine($"{"Towar",-22}{"Ilość",6}{"Cena",11}{"Wartość",12}");
Console.WriteLine($"{nazwy[i],-22}{ilosci[i],6}{ceny[i],11:C}{wartosc,12:C}");
```

1. Ustaw kulturę: `CultureInfo.CurrentCulture = new CultureInfo("pl-PL");` (potrzebny `using System.Globalization;`).
2. Utwórz tablice: nazwy (`string[]`), ilości (`int[]`) i ceny brutto (`decimal[]`) dla co najmniej 4 towarów.
3. Wypisz nagłówek kolumn: Towar (22 znaki, do lewej), Ilość (6), Cena (11), Wartość (12) — do prawej.
4. Pod nagłówkiem narysuj linię `new string('-', 51)`.

2 Pozycje i suma

W pętli `for` wypisz pozycje paragonu.

1. Oblicz wartość pozycji: ilość × cena; dodaj ją do sumy.
2. Wypisz każdą pozycję w formacie z przykładu (`:C` — waluta).
3. Sprawdź, czy kolumny są równe także dla najdłuższej nazwy towaru. Jeśli nie — popraw szerokość.

3 Rabat i VAT

Podsumowanie paragonu.

1. Jeśli suma przekracza 100 zł, nalicz rabat 10% zaokrąglony do grosza: `Math.Round(x, 2, MidpointRounding.AwayFromZero)`.
2. Oblicz kwotę do zapłaty i zawarty w niej VAT 23%: netto = brutto / 1,23, VAT = brutto − netto.
3. Wypisz podsumowanie wyrównane do kolumny Wartość.
4. Sprawdź obliczenia na kalkulatorze dla swoich danych.

4 Zaokrąglanie

Eksperyment z trybami zaokrąglania.

1. Wypisz `Math.Round(2.345m, 2)` i `Math.Round(2.345m, 2, MidpointRounding.AwayFromZero)`.
2. Wypisz `Math.Round(2.5m)` i `Math.Round(3.5m)`.
3. Zapisz, który tryb jest domyślny w .NET i który stosuje się w handlu.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod programu
- ☐ zrzut ekranu paragonu
- ☐ obliczenia kontrolne rabatu i VAT
- ☐ wyniki eksperymentu z zaokrąglaniem

Pytania kontrolne

1. Co oznacza liczba ujemna w formacie `{x,-22}`?
2. Dlaczego do kwot pieniężnych używamy `decimal`?
3. Jak działa domyślne zaokrąglanie `Math.Round` w .NET (MidpointRounding.ToEven)?
4. Jak obliczyć VAT zawarty w cenie brutto?

Kryteria oceny

Kryterium	Punkty
Tablice danych i pętla	4
Wyrównane kolumny i format waluty	5
Rabat i VAT poprawnie zaokrąglone	5
Eksperyment z zaokrąglaniem	3
Sprawozdanie i pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Warunki i wyrażenie switch

Czas: 2 × 45 min | praca indywidualna | Wykład 1.05

Cel: Zapiszesz decyzje w programie instrukcją `if`, operatorem warunkowym i wyrażeniem `switch` z wzorcami, dbając o poprawne granice przedziałów.

Po zajęciach potrafisz:

- zamienia punkty na ocenę według progów procentowych
- stosuje wyrażenie `switch` z wzorcami relacyjnymi i krotkami
- testuje wartości graniczne warunków
- porównuje czytelność `if` i `switch`

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET”)** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Ocena z punktów

Program wczytuje liczbę punktów i maksimum, oblicza procent i wypisuje ocenę.

```
string Ocena(int punkty, int max)
{
    double procent = 100.0 * punkty / max;
    return procent switch
    {
        >= 95 => "celujący (6)",
        >= 85 => "bardzo dobry (5)",
        // ... uzupełnij
        _ => "niedostateczny (1)"
    };
}
```

1. Utwórz projekt `Decyzje`. Progi: od 95% celujący, od 85% bardzo dobry, od 70% dobry, od 50% dostateczny, od 30% dopuszczający, poniżej — niedostateczny.
2. Najpierw zapisz rozwiązanie łańcuchem `if … else if`.
3. Następnie zapisz je jako funkcję z wyrażeniem `switch` (szablon) i porównaj czytelność.
4. Odrzuć błędne dane: maksimum ≤ 0, punkty < 0, punkty > maksimum.
5. Przetestuj granice: 29, 30, 49, 50, 69, 70, 84, 85, 94, 95 punktów na 100.

2 Dzień tygodnia

Funkcja zamienia numer 1–7 na nazwę dnia.

1. Użyj `switch` ze wzorcem `6 or 7 => "weekend"`.
2. Dla liczby spoza zakresu zwróć komunikat „nie ma takiego dnia”.
3. Wypisz wynik dla 1, 3, 6 i 9.

3 Taryfa biletów

Funkcja `CenaBiletu(int wiek, bool student)` zwraca cenę biletu.

1. Zasady: dzieci do 3 lat i osoby od 70 lat — 0 zł; uczniowie poniżej 18 lat i studenci do 26 lat — 2,20 zł; pozostali — 4,40 zł.
2. Zapisz je wyrażeniem `switch` na krotce `(wiek, student)` ze wzorcami `( < 4, _)` i klauzulą `when`.
3. Sprawdź kolejność ramion: czy student w wieku 72 lat płaci 0 zł?
4. Przygotuj tabelę 6 przypadków testowych z oczekiwaną ceną i porównaj z programem.
5. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod obu wersji funkcji `Ocena`
- ☐ tabela testów granicznych
- ☐ tabela testów taryfy biletów
- ☐ wnioski: kiedy `switch`, kiedy `if`

Pytania kontrolne

1. Dlaczego progi w łańcuchu `if` sprawdzamy od największego?
2. Co oznacza znak `_` w wyrażeniu `switch`?
3. Jakie wartości warto przetestować dla warunku `procent >= 50`?
4. Do czego służy klauzula `when`?

Kryteria oceny

Kryterium	Punkty
Ocena: wersja if i wersja switch	6
Walidacja danych	2
Dzień tygodnia	3
Taryfa biletów na krotce	5
Testy graniczne i wnioski	4
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Pętle

Czas: 2 × 45 min | praca indywidualna | Wykład 1.06

Cel: Rozwiążesz cztery krótkie problemy, dobierając odpowiedni rodzaj pętli: `for`, `while` lub `do…while`.

Po zajęciach potrafisz:

- dobiera rodzaj pętli do problemu
- stosuje pętle zagnieżdżone
- przerywa pętlę instrukcją `break` i pomija krok `continue`
- wyszukuje błędy typu „o jeden za dużo” (off-by-one)

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET”)** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Tabliczka mnożenia

Wypisz tabliczkę mnożenia 10 × 10 w równych kolumnach.

1. Utwórz projekt `Petle`. Użyj dwóch zagnieżdżonych pętli `for`.
2. Każdą liczbę wypisz w polu o szerokości 4 (`{i * j,4}`).
3. Zmień program tak, aby rozmiar tabliczki podawał użytkownik (2–20).

2 Suma cyfr

Wczytaj liczbę naturalną i oblicz sumę jej cyfr.

1. Użyj pętli `do…while`: dodawaj `n % 10`, potem dziel `n /= 10`.
2. Dlaczego `do…while`, a nie `while`? Sprawdź dla liczby 0.
3. Użyj typu `long`, aby program działał dla dużych liczb.

3 Liczby pierwsze

Wypisz wszystkie liczby pierwsze od 2 do podanej granicy i ich liczbę.

1. Dla każdej liczby `k` sprawdzaj dzielniki `d` od 2, dopóki `d * d <= k`.
2. Po znalezieniu dzielnika przerwij pętlę `break`.
3. Sprawdź wynik dla granicy 50 (powinno być 15 liczb).
4. Zmierz „na oko", ile trwa obliczenie do 1 000 000 z warunkiem `d * d <= k` i z `d < k`.

4 Gra „Zgadnij liczbę"

Program losuje liczbę 1–100, a użytkownik zgaduje z podpowiedziami „za mało / za dużo".

```
var los = new Random();
int tajna = los.Next(1, 101);   // 1..100 — górna granica nie jest losowana!
```

1. Losuj liczbę: `var los = new Random(); int tajna = los.Next(1, 101);`
2. Użyj pętli `do…while` kończącej się po trafieniu; licz próby.
3. Gdy użytkownik wpisze tekst niebędący liczbą, wypisz komunikat i użyj `continue` (nie licz próby).
4. Po trafieniu wypisz liczbę prób. Ile prób wystarczy przy najlepszej strategii? Uzasadnij.
5. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod programu
- ☐ zrzuty ekranu każdej części
- ☐ porównanie czasu sprawdzania liczb pierwszych
- ☐ odpowiedź: liczba prób przy najlepszej strategii

Pytania kontrolne

1. Czym różni się pętla `while` od `do…while`?
2. Dlaczego przy sprawdzaniu liczby pierwszej wystarczy warunek `d * d <= k`?
3. Co robi `break`, a co `continue`?
4. Dlaczego `los.Next(1, 100)` nigdy nie wylosuje liczby 100?

Kryteria oceny

Kryterium	Punkty
Tabliczka mnożenia z rozmiarem od użytkownika	4
Suma cyfr (do...while)	3
Liczby pierwsze z break	5
Gra z licznikiem prób i obsługą błędów	5
Sprawozdanie i pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Tablice jednowymiarowe — statystyki ocen

Czas: 2 × 45 min | praca indywidualna | Wykład 1.07

Cel: Obliczysz statystyki ocen klasy zapisanych w tablicy, napiszesz własne sortowanie bąbelkowe i porównasz je z metodą `Array.Sort`.

Po zajęciach potrafisz:

- przechodzi po tablicy pętlami `for` i `foreach`
- wyznacza minimum, maksimum, średnią i medianę
- implementuje sortowanie bąbelkowe i porównuje je z `Array.Sort`
- zlicza wystąpienia wartości w tablicy pomocniczej

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Dane i podstawowe statystyki

Utwórz projekt `Statystyki` z tablicą co najmniej 12 ocen (1–6).

1. Wypisz oceny w jednym wierszu: `string.Join(", ", oceny)`.
2. W jednej pętli `foreach` wyznacz minimum, maksimum i sumę — bez metod `Min()`, `Max()`, `Sum()`.
3. Oblicz średnią jako `double` (uwaga na dzielenie całkowite) i wypisz z dwoma miejscami po przecinku.

2 Sortowanie bąbelkowe

Napisz własne sortowanie na kopii tablicy.

```
for (int i = 0; i < kopia.Length - 1; i++)
    for (int j = 0; j < kopia.Length - 1 - i; j++)
        if (kopia[j] > kopia[j + 1])
            (kopia[j], kopia[j + 1]) = (kopia[j + 1], kopia[j]);
```

1. Skopiuj tablicę: `int[] kopia = (int[])oceny.Clone();` — dlaczego nie `int[] kopia = oceny;`?
2. Zaimplementuj sortowanie bąbelkowe według szablonu; zamianę zapisz krotką.
3. Dodaj licznik zamian i wypisz go po sortowaniu.
4. Porównaj wynik z `Array.Sort` na drugiej kopii (`SequenceEqual`).

3 Mediana i rozkład ocen

Wykorzystaj posortowaną tablicę.

1. Oblicz medianę: dla nieparzystej liczby elementów — środkowy, dla parzystej — średnia dwóch środkowych.
2. Sprawdź oba przypadki (dodaj lub usuń jedną ocenę).
3. Utwórz tablicę `int[] ile = new int[7]` i zlicz, ile było jedynek, dwójek itd.
4. Wypisz prosty wykres z gwiazdek lub znaków `#`: `new string('#', ile[o])`.

4 Wyszukiwanie

Odszukaj ocenę w tablicy.

1. Napisz pętlę, która znajduje pierwszą pozycję podanej oceny lub wypisuje „brak”.
2. Porównaj wynik z `Array.IndexOf`.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod programu
- ☐ zrzut ekranu wyników
- ☐ liczba zamian w sortowaniu bąbelkowym
- ☐ obliczenia mediany dla liczby parzystej i nieparzystej

Pytania kontrolne

1. Dlaczego `int[] kopia = oceny;` nie tworzy kopii tablicy?
2. Ile porównań wykonuje sortowanie bąbelkowe dla 12 elementów?
3. Dlaczego do obliczenia mediany tablica musi być posortowana?
4. Jaki wyjątek wystąpi przy odwołaniu `oceny[oceny.Length]`?

Kryteria oceny

Kryterium	Punkty
Min, max, średnia w jednej pętli	4
Sortowanie bąbelkowe z licznikiem zamian	5
Mediana (oba przypadki)	3
Rozkład ocen z wykresem	4
Wyszukiwanie i pytania	4
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Tablice dwuwymiarowe — kółko i krzyżyk

Czas: 2 × 45 min | praca w parach (programista + tester, zamiana ról w połowie) | Wykład 1.07

Cel: Napiszesz grę w kółko i krzyżyk dla dwóch graczy w konsoli, przechowując planszę w tablicy dwuwymiarowej.

Po zajęciach potrafisz:

- deklaruje i wypełnia tablicę `char[,]`
- rysuje planszę pętlami zagnieżdżonymi
- sprawdza poprawność ruchu i warunki zwycięstwa
- dzieli program na funkcje lokalne

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Plansza

Utwórz projekt `KolkoKrzyzyk`.

```
    1   2   3
1   X | 0 |
   ---+---+---
2     | X |
   ---+---+---
3     |   | 0
```

1. Zadeklaruj `char[,] plansza = new char[3, 3];` i wypełnij ją spacjami.
2. Napisz funkcję `Rysuj()`, która wypisuje planszę z numerami wierszy i kolumn (wzór poniżej).
3. Sprawdź rysowanie, wpisując ręcznie kilka znaków do planszy.

2 Ruchy graczy

Gracze na zmianę podają wiersz i kolumnę, np. `2 3`.

1. Rozdziel tekst metodą `Split(' ', StringSplitOptions.RemoveEmptyEntries)` i zamień części na liczby.
2. Odrzuć ruch, gdy: dane nie są dwiema liczbami, liczby są spoza 1–3 albo pole jest zajęte. Pytaj ponownie.
3. Pamiętaj o przesunięciu indeksu: użytkownik podaje 1–3, tablica ma indeksy 0–2.
4. Po poprawnym ruchu zmień gracza: `gracz = gracz == 'X' ? 'O' : 'X';`.

3 Zwycięstwo i remis

Napisz funkcję `bool Wygral(char z)`.

1. Sprawdź w pętli trzy wiersze i trzy kolumny.
2. Sprawdź dwie przekątne.
3. Po każdym ruchu sprawdź zwycięstwo; po 9 ruchach bez zwycięzcy ogłoś remis.
4. Przetestuj z partnerem: wygraną w wierszu, w kolumnie, na każdej przekątnej, remis i wszystkie rodzaje błędnych ruchów.

4 Rozszerzenie (dla chętnych)

Wybierz jedno.

1. Kolorowe znaki: X na czerwono, O na niebiesko.
2. Licznik wygranych i ponowna gra bez zamykania programu.
3. Prosty przeciwnik komputerowy: losowe wolne pole.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod programu
- ☐ tabela testów wykonanych przez testera
- ☐ zrzut ekranu wygranej i remisu
- ☐ opis rozszerzenia (jeśli wykonane)

Pytania kontrolne

1. Czym różni się tablica `char[,]` od tablicy tablic `char[][]`?
2. Ile linii kodu trzeba by zmienić, aby plansza miała 4 × 4? Jak to uprościć?
3. Dlaczego sprawdzamy zwycięstwo tylko gracza, który właśnie wykonał ruch?
4. Jak program reaguje na pustą linię? Sprawdź.

Kryteria oceny

Kryterium	Punkty
Plansza i rysowanie	4
Ruchy z walidacją	5
Wykrywanie zwycięstwa i remisu	5
Testy w parach (tabela)	4
Rozszerzenie lub odpowiedzi na pytania	2
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Kolekcje List i Dictionary

Czas: 2 × 45 min | praca indywidualna | Wykład 1.08

Cel: Wykorzystasz listę i słownik w trzech programach: liście zakupów z menu, teście słówek i zliczaniu słów w tekście.

Po zajęciach potrafisz:

- dodaje, usuwa i przegląda elementy `List<T>`
- korzysta ze słownika `Dictionary<TKey,TValue>` do wyszukiwania i zliczania
- buduje menu tekstowe z pętlą i instrukcją `switch`
- dobiera kolekcję do problemu

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Lista zakupów

Utwórz projekt `Kolekcje`. Program wyświetla menu: [1] dodaj, [2] usuń, [3] pokaż, [0] koniec.

1. Utwórz `var zakupy = new List<string>();` i pętlę `while (true)` z menu.
2. Dodawanie: odrzuć pustą nazwę i duplikat bez względu na wielkość liter (`StringComparer.OrdinalIgnoreCase`).
3. Usuwanie po numerze z listy (1, 2, 3...) — sprawdź zakres numeru.
4. Pokazywanie: lista ponumerowana i liczba pozycji (`Count`).
5. Przetestuj nieznaną opcję menu, usunięcie z pustej listy i numer 0.

2 Test słówek

Słownik polsko-angielski z co najmniej 8 słówkami.

1. Utwórz słownik inicjalizatorem `["pies"] = "dog"`.
2. Przejdź po słowniku `foreach (var (pl, en) in slowka)` i pytaj o tłumaczenie.
3. Porównuj odpowiedź po `Trim()` i `ToLower()`; przy błędzie pokaż poprawną odpowiedź.
4. Na końcu wypisz wynik, np. 6/8.

3 Zliczanie słów

Policz, ile razy każde słowo występuje w tekście.

1. Podziel tekst na słowa (separator: spacja, kropka, przecinek), zamień na małe litery.
2. Zliczaj w słowniku: `licznik[s] = licznik.GetValueOrDefault(s) + 1;`.
3. Wypisz słowa od najczęstszego (`OrderByDescending(x => x.Value)`).
4. Wklej do programu fragment dowolnego tekstu (np. 3 akapity lektury) i znajdź 5 najczęstszych słów.
5. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod programu
- ☐ tabela testów menu listy zakupów
- ☐ zrzut wyniku testu słówek
- ☐ 5 najczęstszych słów z wybranego tekstu

Pytania kontrolne

1. Kiedy użyjesz `List<T>`, a kiedy `Dictionary<TKey,TValue>`?
2. Co się stanie przy odczycie `slownik["nieznane"]`? Jak tego uniknąć?
3. Czym różni się `Remove(element)` od `RemoveAt(indeks)`?
4. Dlaczego przed zliczaniem zamieniamy słowa na małe litery?

Kryteria oceny

Kryterium	Punkty
Lista zakupów: menu i operacje z walidacją	7
Test słówek	4
Zliczanie słów i sortowanie	5
Testy i pytania	4
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Metody i rekurencja

Czas: 2 × 45 min | praca indywidualna | Wykład 1.09

Cel: Zbudujesz statyczną klasę `Matematyka` z metodami obliczeniowymi w wersjach iteracyjnych i rekurencyjnych oraz użyjesz parametrów `out` i `ref`.

Po zajęciach potrafisz:

- definiuje metody statyczne z parametrami i wartością zwracaną
- zapisuje algorytm rekurencyjnie i iteracyjnie
- stosuje parametry `out` i `ref`
- dokumentuje metody komentarzem `///`

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Klasa Matematyka

Utwórz projekt `Biblioteka`. Metody umieść w statycznej klasie pod instrukcjami najwyższego poziomu.

```
Console.WriteLine($"NWD(84, 36) = {Matematyka.Nwd(84, 36)}");

static class Matematyka
{
    /// <summary>Największy wspólny dzielnik (algorytm Euklidesa).</summary>
    public static int Nwd(int a, int b)
    {
        while (b != 0) (a, b) = (b, a % b);
        return Math.Abs(a);
    }
}
```

1. Utwórz `static class Matematyka` na końcu pliku `Program.cs` (lub w osobnym pliku `Matematyka.cs`).
2. Napisz `Nwd(int a, int b)` algorytmem Euklidesa z resztą z dzielenia i `Nww(a, b) = a / Nwd(a, b) * b`.
3. Napisz `CzyPierwsza(int n)` (dla n < 2 zwróć false).
4. Nad każdą metodą dodaj komentarz `///` z opisem (Visual Studio wstawi szablon po wpisaniu trzech ukośników).

2 Iteracja i rekurencja

Każdą z metod napisz w dwóch wersjach.

1. `Silnia(int n)` — pętlą, `SilniaRek(int n)` — rekurencyjnie; wynik typu `long`.
2. `Potega(long a, int n)` i `PotegaRek(long a, int n)`.
3. Porównaj wyniki obu wersji dla kilku danych.
4. Oblicz 20! i 21!. Co się stało? Sprawdź wynik dla `checked`.
5. Wywołaj `SilniaRek(100000)` i zapisz, jaki błąd się pojawił.

3 Parametry out i ref

Metody zwracające więcej niż jedną wartość.

1. Napisz `MinMax(int[] t, out int min, out int max)`.
2. Napisz `Podwoj(ref int x)` i pokaż wartość zmiennej przed i po wywołaniu.
3. Napisz drugą wersję `MinMax`, która zwraca krotkę `(int Min, int Max)`, i porównaj wygodę użycia.

4 Testy ręczne

Sprawdź metody na przygotowanych danych.

1. Wypełnij tabelę testów w sprawozdaniu (wyniki oczekiwane oblicz wcześniej na kartce).
2. Popraw metody, jeśli wyniki się nie zgadzają.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klasy Matematyka z komentarzami
- ☐ tabela testów
- ☐ wyjaśnienie przepełnienia przy 21!
- ☐ opis błędu przy głębokiej rekurencji

Pytania kontrolne

1. Co oznacza słowo `static` przy klasie i metodzie?
2. Czym różni się parametr `out` od `ref`?
3. Co to jest warunek zakończenia rekurencji i co się dzieje, gdy go brakuje?
4. Kiedy rekurencja jest gorszym wyborem niż pętla?

Kryteria oceny

Kryterium	Punkty
NWD, NWW, CzyPierwsza	5
Silnia i potęga w dwóch wersjach	5
MinMax z out, Podwoj z ref, wersja z krotką	4
Komentarze /// i tabela testów	3
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Przetwarzanie tekstu

Czas: 2 × 45 min | praca indywidualna | Wykład 1.10

Cel: Napiszesz klasę `Tekst` z metodami do sprawdzania palindromów, liczenia samogłosek, szyfrowania szyfrem Cezara i weryfikacji numeru PESEL.

Po zajęciach potrafisz:

- przetwarza napisy znak po znaku i metodami klasy `string`
- korzysta z `StringBuilder` przy budowaniu napisu
- implementuje algorytm sumy kontrolnej
- testuje metody na danych poprawnych i błędnych

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Numer PESEL to dane osobowe — w testach używaj wyłącznie numerów przykładowych.

Przebieg zadania

1 Palindrom i samogłoski

Utwórz projekt `Teksty` i statyczną klasę `Tekst`.

1. Metoda `CzyPalindrom(string s)` ignoruje wielkość liter, spacje i znaki interpunkcyjne (zostaw tylko `char.IsLetterOrDigit`).
2. Sprawdź: „Kajak", „Kobyła ma mały bok", „Ala", „Programista".
3. Metoda `Samogloski(string s)` liczy samogłoski z polskimi znakami: a ą e ę i o ó u y.
4. Sprawdź na „Zażółć gęślą jaźń" (wynik: 5).

2 Szyfr Cezara

Każdą literę alfabetu łacińskiego przesuń o zadaną liczbę pozycji; inne znaki pozostaw.

```
if (z is >= 'a' and <= 'z')
    sb.Append((char)('a' + ((z - 'a' + przes) % 26 + 26) % 26));
```

1. Napisz `Cezar(string s, int przes)` z użyciem `StringBuilder`.
2. Obsłuż małe i wielkie litery oraz zawijanie (z → c dla przesunięcia 3).
3. Deszyfrowanie to szyfrowanie z przesunięciem ujemnym — upewnij się, że działa (reszta z dzielenia liczb ujemnych!).
4. Zaszyfruj i odszyfruj zdanie z wielkimi literami, cyframi i znakami interpunkcyjnymi.

3 PESEL

Metoda `PeselPoprawny(string p)` sprawdza długość, cyfry i cyfrę kontrolną.

1. Sprawdź, czy napis ma 11 znaków i same cyfry (`char.IsAsciiDigit`).
2. Oblicz sumę iloczynów pierwszych 10 cyfr przez wagi 1, 3, 7, 9, 1, 3, 7, 9, 1, 3.
3. Cyfra kontrolna = (10 − suma mod 10) mod 10; porównaj z ostatnią cyfrą.
4. Sprawdź przykładowy numer z dokumentacji MSWiA `44051401359` (poprawny) i ten sam ze zmienioną ostatnią cyfrą.
5. Nie używaj prawdziwych numerów PESEL swoich ani kolegów.

4 Menu programu

Połącz metody w program z menu.

1. Użytkownik wybiera funkcję i wpisuje tekst; program wypisuje wynik.
2. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klasy Tekst
- ☐ tabela testów wszystkich metod
- ☐ wyjaśnienie wzoru `((z - 'a' + przes) % 26 + 26) % 26`

Pytania kontrolne

1. Dlaczego `string` w C# jest niezmienny i jakie ma to znaczenie przy sklejaniu w pętli?
2. Co zwraca `-1 % 26` w C# i dlaczego dodajemy 26?
3. Jakie informacje zawiera numer PESEL poza cyfrą kontrolną?
4. Dlaczego szyfr Cezara nie jest bezpieczny?

Kryteria oceny

Kryterium	Punkty
Palindrom i samogłoski	4
Szyfr Cezara z zawijaniem i deszyfrowaniem	5
Weryfikacja PESEL	5
Menu i testy	3
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Wyjątki i odporny program

Czas: 2 × 45 min | praca indywidualna | Wykład 1.11

Cel: Napiszesz kalkulator, który nie kończy się błędem przy żadnych danych, obsługując wyjątki wbudowane i własny wyjątek.

Po zajęciach potrafisz:

- obsługuje wyjątki blokami `try`, `catch` i `finally`
- zgłasza wyjątek instrukcją `throw`
- definiuje własną klasę wyjątku
- projektuje komunikaty błędów zrozumiałe dla użytkownika

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Kalkulator bez zabezpieczeń

Utwórz projekt `Kalkulator`. Użytkownik wpisuje działanie w jednej linii: `12 / 4`.

1. Napisz klasę `Kalkulator` z metodą `decimal Oblicz(string wyrazenie)`: podział na 3 części, `decimal.Parse`, `switch` po znaku.
2. Uruchom kolejno: `12 / 4`, `1 / 0`, `abc`, `2 ^ 3`, `12/4` (bez spacji). Zapisz, jakie wyjątki przerwały program.

2 Obsługa wyjątków

Dodaj pętlę i obsługę błędów w programie głównym.

1. Obejmij wywołanie `Oblicz` blokiem `try` i dodaj osobne `catch` dla `FormatException`, `DivideByZeroException` i `OverflowException`.
2. Każdy komunikat ma mówić, co zrobić, np. „Liczby wpisz z przecinkiem, np. 2,5 * 4.”.
3. Dodaj blok `finally`, który wypisuje linię oddzielającą — sprawdź, że wykonuje się zawsze.
4. Gdy wyrażenie nie ma trzech części, zgłoś `throw new FormatException(...)`.

3 Własny wyjątek

Nieznany znak działania sygnalizuj własnym wyjątkiem.

```
class NieznaneDzialanieException(string znak) : Exception($"Nieznane działanie: {znak}")
{
    public string Znak { get; } = znak;
}
```

1. Utwórz klasę `NieznaneDzialanieException` dziedziczącą po `Exception` z właściwością `Znak`.
2. Zgłoś ją w ramieniu `_` wyrażenia `switch`.
3. Dodaj `catch` i komunikat z listą dozwolonych znaków.

4 Testy złych danych

Sprawdź odporność programu.

1. Przygotuj co najmniej 10 przypadków testowych (poprawne, błędne, graniczne, np. `79228162514264337593543950335 + 1`).
2. Upewnij się, że program nigdy się nie zatrzymuje z błędem.
3. Sprawdź, co się stanie z `double` zamiast `decimal` przy dzieleniu przez zero.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod programu
- ☐ lista wyjątków z wersji bez zabezpieczeń
- ☐ tabela 10 testów
- ☐ wyjaśnienie różnicy w dzieleniu przez zero dla `double` i `decimal`

Pytania kontrolne

1. Kiedy wykonuje się blok `finally`?
2. Dlaczego nie warto pisać jednego `catch (Exception)` dla wszystkich błędów?
3. Dlaczego `1.0 / 0` dla `double` nie zgłasza wyjątku?
4. Kiedy lepiej sprawdzić dane warunkiem (`TryParse`), a kiedy użyć wyjątku?

Kryteria oceny

Kryterium	Punkty
Metoda Oblicz i analiza wersji bez zabezpieczeń	4
Obsługa wyjątków z czytelnymi komunikatami	5
finally i throw	2
Własny wyjątek	4
Tabela 10 testów i pytania	5
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Debugowanie programu

Czas: 2 × 45 min | praca indywidualna | Wykład 1.12

Cel: Znajdziesz i poprawisz sześć błędów logicznych w gotowym programie, korzystając z debugera Visual Studio zamiast zgadywania.

Po zajęciach potrafisz:

- ustawia punkty przerwania i wykonuje program krokowo (F10, F11)
- obserwuje wartości zmiennych w oknach Locals i Watch
- używa warunkowego punktu przerwania
- opisuje błąd: objaw, przyczynę i poprawkę

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Przygotowanie przed zajęciami

- Pobierz plik `Program.cs` z folderu `pliki` przy tym zadaniu.

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Uruchomienie programu z błędami

Utwórz projekt konsolowy `Debugowanie` i zastąp `Program.cs` plikiem od nauczyciela.

1. Przeczytaj komentarz na początku pliku — opisuje, co program **powinien** robić.
2. Oblicz na kartce poprawne wyniki dla danych z programu (średnia, liczba pozytywnych, procent, najlepsza ocena, imię wspak).
3. Uruchom program (Ctrl+F5) i porównaj wyniki z obliczeniami. Zapisz objawy błędów w tabeli.

2 Narzędzia debugera

Przećwicz narzędzia na pierwszej pętli.

1. Ustaw punkt przerwania (F9) na wierszu `suma += oceny[i];` i uruchom program klawiszem F5.
2. Wykonuj program krokowo (F10). W oknie **Locals** obserwuj `i` i `suma` — od jakiej wartości zaczyna się `i`?
3. Dodaj do okna **Watch** wyrażenie `oceny[i]` oraz `suma / oceny.Length`.
4. Najedź kursorem na zmienną w kodzie, aby zobaczyć jej wartość (DataTip).

3 Szukanie błędów

Znajdź wszystkie 6 błędów. Każdy popraw dopiero po znalezieniu przyczyny w debugerze.

1. Dla każdego złego wyniku ustaw punkt przerwania w miejscu obliczenia i sprawdź wartości.
2. Ustaw warunkowy punkt przerwania (prawy przycisk na czerwonej kropce → Warunki) na `o == 2` w pętli liczącej oceny pozytywne.
3. W oknie **Immediate** (Debuguj → Okna → Natychmiastowe) wpisz `100.0 * pozytywne / oceny.Length` i porównaj z wynikiem programu.
4. Przy pętli odwracającej imię obserwuj ostatnią wartość `i` przed wyjściem z pętli.
5. Po każdej poprawce uruchom program i zapisz wynik w tabeli.

4 Zakończenie

Sprawdź, czy wszystkie wyniki są zgodne z obliczeniami na kartce.

1. Opisz każdy błąd w tabeli sprawozdania.
2. Zatwierdź poprawioną wersję w Git z opisem „Poprawione 6 błędów”.

W sprawozdaniu umieść

- ☐ tabela 6 błędów: objaw, przyczyna, poprawka
- ☐ zrzut ekranu debugera z oknem Locals lub Watch
- ☐ zrzut ekranu warunkowego punktu przerwania
- ☐ wyniki programu po poprawkach

Pytania kontrolne

1. Czym różni się F10 (Step Over) od F11 (Step Into)?
2. Do czego służy warunkowy punkt przerwania? Podaj przykład spoza zadania.
3. Dlaczego kompilator nie wykrył żadnego z tych błędów?
4. Jak test jednostkowy mógłby wykryć błąd w obliczaniu średniej?

Kryteria oceny

Kryterium	Punkty
Obliczenia na kartce i objawy błędów	3
Znalezione i poprawione błędy (po 2 pkt)	12
Zrzuty z debugera	2
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Pierwsza klasa — konto bankowe

Czas: 2 × 45 min | praca indywidualna | Wykład 1.13

Cel: Zaprojektujesz klasę `KontoBankowe` z hermetyzacją danych, konstruktorem, metodami wpłaty i wypłaty oraz historią operacji.

Po zajęciach potrafisz:

- definiuje klasę z polami prywatnymi, konstruktorem i metodami
- chroni stan obiektu przed niepoprawnymi zmianami (hermetyzacja)
- zgłasza wyjątek przy niepoprawnym argumencie
- nadpisuje metodę `ToString`

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Projekt klasy

Utwórz projekt `Bank`. Zanim napiszesz kod, zapisz w sprawozdaniu projekt klasy: pola, konstruktor, metody.

1. Pola prywatne: `_numer`, `_wlasciciel`, `_saldo` (`decimal`), `_historia` (`List<string>`).
2. Konstruktor przyjmuje numer i właściciela; saldo początkowe 0.
3. Właściwość tylko do odczytu `public decimal Saldo => _saldo;`.
4. Sprawdź, że instrukcja `konto._saldo = 1000000;` w programie głównym się nie kompiluje — zapisz komunikat błędu.

2 Wpłata i wypłata

Metody zmieniające saldo.

```
public bool Wyplac(decimal kwota)
{
    if (kwota <= 0) throw new ArgumentOutOfRangeException(nameof(kwota));
    if (kwota > _saldo) { /* zapisz odmowę */ return false; }
    _saldo -= kwota;
    // zapisz operację
    return true;
}
```

1. `Wplac(decimal kwota)` — dla kwoty ≤ 0 zgłoś `ArgumentOutOfRangeException(nameof(kwota))`.
2. `bool Wyplac(decimal kwota)` — gdy brak środków, nie zmieniaj salda i zwróć `false`.
3. Każdą operację (także odmowę) dopisz do historii z datą: `$"{DateTime.Now:dd.MM HH:mm} wpłata {kwota,10:F2} zł"`.

3 Historia i opis konta

Udostępnij dane do odczytu.

1. Dodaj właściwość `IReadOnlyList<string> Historia => _historia;` — dlaczego nie `List<string>`?
2. Nadpisz `ToString()`, aby zwracał właściciela, numer i saldo.
3. W programie głównym utwórz dwa konta i wykonaj na nich po kilka operacji, w tym nieudaną wypłatę i wpłatę ujemną w bloku `try`.

4 Przelew (rozszerzenie)

Dodaj metodę `bool Przelej(KontoBankowe cel, decimal kwota)`.

1. Przelew to wypłata z jednego konta i wpłata na drugie — wykorzystaj istniejące metody.
2. Sprawdź przelew na to samo konto i przelew bez pokrycia.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ projekt klasy (pola, metody) przed kodowaniem
- ☐ kod klasy i programu
- ☐ zrzut ekranu historii operacji
- ☐ komunikat błędu przy próbie dostępu do pola prywatnego

Pytania kontrolne

1. Co to jest hermetyzacja i jak ją zrealizowano w klasie KontoBankowe?
2. Dlaczego udostępniamy historię jako `IReadOnlyList<string>`?
3. Kiedy metoda powinna zgłosić wyjątek, a kiedy zwrócić `false`?
4. Czym różni się klasa od obiektu? Odpowiedz na przykładzie z zadania.

Kryteria oceny

Kryterium	Punkty
Projekt klasy przed kodowaniem	2
Pola prywatne, konstruktor, właściwość Saldo	4
Wpłata i wypłata z kontrolą	5
Historia, ToString, program testowy	4
Przelew	2
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Właściwości i składowe statyczne — klasa Kosc

Czas: 2 × 45 min | praca indywidualna | Wykład 1.14

Cel: Napiszesz klasę `Kosc` w stylu zadania egzaminacyjnego INF.04 — z polem statycznym liczącym obiekty, właściwościami i dokumentacją — oraz program testowy gry w kości.

Po zajęciach potrafisz:

- definiuje właściwości z prywatnym setterem i właściwości automatyczne
- używa składowej statycznej wspólnej dla wszystkich obiektów
- tworzy przeciążone konstruktory z wywołaniem `this(...)`
- dokumentuje klasę komentarzami `///` jak na egzaminie

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Klasa Kosc

Utwórz projekt `GraWKosci` i plik `Kosc.cs` (Projekt → Dodaj klasę).

```
public class Kosc
{
    private static readonly Random Losowanie = new();
    private readonly Random _los;
    public static int LiczbaKosci { get; private set; }
    public int Wartosc { get; private set; }
    public bool Zablokowana { get; set; }

    public Kosc() : this(Losowanie) { }
    public Kosc(Random los) { _los = los; LiczbaKosci++; }
    // ...
}
```

1. Właściwość `Wartosc` (1–6, 0 przed pierwszym rzutem) z prywatnym setterem.
2. Właściwość automatyczna `bool Zablokowana { get; set; }`.
3. Statyczna właściwość `LiczbaKosci` zwiększana w konstruktorze — liczy utworzone obiekty.
4. Metoda `int Rzuc()` losuje 1–6, chyba że kość jest zablokowana; zwraca aktualną wartość.
5. Metoda `Ustaw(int wartosc)` zgłasza `ArgumentOutOfRangeException` dla wartości spoza 1–6.

2 Punktacja

Dodaj statyczną metodę `Punkty(IEnumerable<int> wartosci)`.

1. Punkty to suma oczek tych kości, których wartość wystąpiła co najmniej dwa razy (np. 3, 3, 2, 4, 2 → 10).
2. Sprawdź ręcznie trzy przykłady: brak powtórzeń (0 pkt), wszystkie równe, dwie pary.

3 Program testowy

Program w `Program.cs` działa jak w arkuszu egzaminacyjnym.

1. Pytaj o liczbę kości (3–10) aż do skutku.
2. Utwórz listę kości i wypisz wartość `Kosc.LiczbaKosci`.
3. Rzuć wszystkimi kośćmi, wypisz „Kostka 1: 3”... i „Liczba uzyskanych punktów: ...”.
4. Pytaj „Jeszcze raz? (t/n)” i powtarzaj rzut.
5. Zablokuj w kodzie jedną kość przed drugim rzutem i sprawdź, że jej wartość się nie zmienia.

4 Dokumentacja

Na egzaminie INF.04 oceniana jest dokumentacja kodu.

1. Nad klasą i każdą metodą dodaj komentarz `///` z opisem, parametrami i wartością zwracaną.
2. W nagłówku pliku wpisz autora i datę.
3. Wykonaj zrzuty ekranu działania programu.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klasy Kosc z dokumentacją
- ☐ kod programu testowego
- ☐ zrzut ekranu z dwoma rzutami
- ☐ tabela ręcznych przykładów punktacji

Pytania kontrolne

1. Czym różni się składowa statyczna od składowej obiektu? Podaj przykład z klasy Kosc.
2. Dlaczego obiekt `Random` jest wspólny (statyczny), a nie tworzony w każdej kości?
3. Po co klasie drugi konstruktor przyjmujący `Random`?
4. Co daje prywatny setter właściwości `Wartosc`?

Kryteria oceny

Kryterium	Punkty
Właściwości i konstruktory	4
Licznik statyczny	2
Rzuc z blokowaniem, Ustaw z walidacją	4
Punktacja	3
Program testowy jak w arkuszu	4
Dokumentacja ///	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Dziedziczenie — pracownicy

Czas: 2 × 45 min | praca indywidualna | Wykład 1.15

Cel: Zbudujesz hierarchię klas pracowników z abstrakcyjną klasą bazową i metodą wirtualną `ObliczPensje`, a następnie przetworzysz listę różnych pracowników jedną pętlą.

Po zajęciach potrafisz:

- tworzy klasę abstrakcyjną i klasy pochodne
- nadpisuje metody słowem `override` i wywołuje `base`
- przetwarza listę obiektów typu bazowego (polimorfizm)
- dobiera dane i metody do odpowiedniego poziomu hierarchii

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Klasa bazowa

Utwórz projekt `Kadry`.

1. Klasa abstrakcyjna `Pracownik` z właściwościami `Imie`, `Nazwisko` ustawianymi w konstruktorze.
2. Metoda abstrakcyjna `decimal ObliczPensje()`.
3. Nadpisz `ToString()` — zwraca imię i nazwisko.
4. Sprawdź, że `new Pracownik(...)` się nie kompiluje. Zapisz komunikat.

2 Klasy pochodne

Utwórz trzy klasy dziedziczące po `Pracownik`.

```
abstract class Pracownik(string imie, string nazwisko)
{
    public string Imie { get; } = imie;
    public string Nazwisko { get; } = nazwisko;
    public abstract decimal ObliczPensje();
    public override string ToString() => $"{Imie} {Nazwisko}";
}

class Stazysta(string imie, string nazwisko, decimal stypendium)
    : Pracownik(imie, nazwisko)
{
    public override decimal ObliczPensje() => stypendium;
}
```

1. `Etatowy`: pensja zasadnicza + premia procentowa.
2. `Godzinowy`: stawka × godziny; godziny ponad 160 płatne 150%.
3. `Stazysta`: stałe stypendium; `ToString()` dopisuje „(staż)" z użyciem `base.ToString()`.
4. Obliczenia sprawdź ręcznie: Godzinowy 40 zł × 172 h = 6400 + 12 × 60 = 7120 zł.

3 Lista płac

Polimorfizm w praktyce.

1. Utwórz `List<Pracownik>` z co najmniej 5 pracownikami różnych typów.
2. W jednej pętli wypisz typ (`GetType().Name`), pracownika i pensję w kolumnach oraz sumę.
3. Wypisz tylko pracowników godzinowych, używając `if (p is Godzinowy g)`.
4. Narysuj w sprawozdaniu diagram klas (prostokąty i strzałki dziedziczenia).

4 Nowy typ (rozszerzenie)

Dodaj klasę `Kierownik` dziedziczącą po `Etatowy` z dodatkiem funkcyjnym.

1. Użyj `base.ObliczPensje()` w nadpisanej metodzie.
2. Sprawdź, czy pętla z kroku 3 działa bez żadnej zmiany.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ diagram klas
- ☐ kod klas i programu
- ☐ zrzut listy płac
- ☐ ręczne obliczenia pensji dla każdego typu

Pytania kontrolne

1. Czym różni się metoda `abstract` od `virtual`?
2. Dlaczego nie można utworzyć obiektu klasy abstrakcyjnej?
3. Co to jest polimorfizm? Wskaż go w swoim programie.
4. Do czego służy słowo `base`?

Kryteria oceny

Kryterium	Punkty
Klasa abstrakcyjna	3
Trzy klasy pochodne z poprawnymi obliczeniami	6
Lista płac i operator is	4
Diagram klas	2
Kierownik z base	2
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Interfejsy i polimorfizm — figury

Czas: 2 × 45 min | praca indywidualna | Wykład 1.16

Cel: Zdefiniujesz interfejs `IFigura` i kilka implementujących go klas, a następnie posortujesz listę figur według pola dzięki interfejsowi `IComparable<T>`.

Po zajęciach potrafisz:

- definiuje i implementuje interfejs
- wykorzystuje interfejs `IComparable<T>` do sortowania
- waliduje dane w konstruktorze
- korzysta z interfejsu jako typu zmiennej i elementu listy

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Interfejs i pierwsze klasy

Utwórz projekt `Figury`.

1. Interfejs `IFigura` z właściwością `string Nazwa` i metodami `double Pole()`, `double Obwod()`.
2. Klasy `Prostokat(a, b)` i `Kolo(r)` implementujące interfejs.
3. Klasa `Kwadrat(a)` dziedzicząca po `Prostokat` — nadpisz tylko nazwę (właściwość `virtual` w klasie bazowej).

2 Trójkąt z walidacją

Klasa `Trojkat(a, b, c)`.

1. W konstruktorze sprawdź warunek trójkąta (suma dwóch boków większa od trzeciego); w przeciwnym razie `ArgumentException`.
2. Pole oblicz wzorem Herona: p = obwód/2, P = √(p(p−a)(p−b)(p−c)).
3. Sprawdź: trójkąt 3-4-5 ma pole 6; boki 1, 2, 10 — wyjątek.

3 Sortowanie i raport

Lista różnych figur.

```
interface IFigura : IComparable<IFigura>
{
    string Nazwa { get; }
    double Pole();
    double Obwod();
    int IComparable<IFigura>.CompareTo(IFigura? inna) => Pole().CompareTo(inna?.Pole() ?? 0);
}
```

1. Rozszerz interfejs: `interface IFigura : IComparable<IFigura>` i zaimplementuj porównywanie po polu (szablon).
2. Utwórz `List<IFigura>` z co najmniej 6 figurami i wywołaj `figury.Sort()`.
3. Wypisz tabelę: nazwa, pole, obwód (2 miejsca po przecinku), sumę pól i figurę o największym obwodzie (`MaxBy`).
4. Posortuj listę także po obwodzie malejąco, używając `Sort((a, b) => b.Obwod().CompareTo(a.Obwod()))`.

4 Nowa figura

Sprawdź elastyczność rozwiązania.

1. Dodaj klasę `Romb(d1, d2)` lub `Szesciokat(a)`.
2. Sprawdź, czy raport działa bez zmian w pętli.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod interfejsu i klas
- ☐ tabela figur posortowana po polu
- ☐ obliczenia kontrolne dla trzech figur
- ☐ wyjaśnienie, dlaczego dodanie nowej figury nie wymagało zmian w raporcie

Pytania kontrolne

1. Czym różni się interfejs od klasy abstrakcyjnej?
2. Do czego służy interfejs `IComparable<T>`?
3. Dlaczego `Kwadrat` dziedziczy po `Prostokat`, a `Kolo` tylko implementuje interfejs?
4. Co zwraca metoda `CompareTo` i jak interpretujemy wynik?

Kryteria oceny

Kryterium	Punkty
Interfejs i klasy Prostokat, Kwadrat, Kolo	5
Trojkat z walidacją i wzorem Herona	4
Sortowanie IComparable i lambdą	4
Raport (suma, MaxBy)	2
Nowa figura	2
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Rekordy i wyliczenia — kawiarnia

Czas: 2 × 45 min | praca indywidualna | Wykład 1.17

Cel: Zamodelujesz zamówienia w kawiarni za pomocą rekordów i typów wyliczeniowych, z kontrolą dozwolonych zmian statusu zamówienia.

Po zajęciach potrafisz:

- definiuje rekord pozycyjny i tworzy jego zmodyfikowaną kopię wyrażeniem `with`
- porównuje rekordy według wartości
- stosuje typ wyliczeniowy `enum` w wyrażeniu `switch`
- pilnuje poprawnej kolejności stanów obiektu

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Wyliczenia i rekord

Utwórz projekt `Kawiarnia`.

1. `enum Rozmiar { Maly, Sredni, Duzy }` i `enum StatusZamowienia { Nowe, WPrzygotowaniu, Gotowe, Wydane }`.
2. Rekord `record Pozycja(string Nazwa, decimal CenaBazowa, Rozmiar Rozmiar)` z właściwością `Cena` = cena bazowa + dopłata za rozmiar.
3. Dopłaty (0 / 2 / 4 zł) zwracaj w statycznej metodzie z wyrażeniem `switch` na `Rozmiar`.
4. Wypisz rekord metodą `Console.WriteLine(pozycja)` — co wypisuje domyślny `ToString` rekordu?

2 Kopie i porównania

Sprawdź cechy rekordów.

1. Utwórz duże latte jako kopię średniego: `menu[1] with { Rozmiar = Rozmiar.Duzy }`.
2. Porównaj `==` dwa rekordy o tych samych danych i dwa o różnych. Zapisz wyniki.
3. Zamień `record` na `class` i powtórz porównanie. Co się zmieniło? Przywróć `record`.

3 Zamówienie

Klasa `Zamowienie` z numerem, listą pozycji i statusem.

```
public void ZmienStatus(StatusZamowienia nowy)
{
    if (nowy != Status + 1)
        throw new InvalidOperationException($"Nie można przejść ze statusu {Status} do
{nowy}.");
    Status = nowy;
}
```

1. Metoda `Dodaj(Pozycja p)` — działa tylko dla statusu `Nowe`.
2. Metoda `Suma()` — suma cen pozycji.
3. Metoda `ZmienStatus(StatusZamowienia nowy)` pozwala tylko na przejście do następnego statusu; inaczej `InvalidOperationException`.
4. W programie złóż zamówienie z trzech pozycji, przeprowadź je przez wszystkie statusy i spróbuj cofnąć status.

4 Podsumowanie

Raport kawiarni.

1. Wypisz wszystkie wartości `Rozmiar` pętlą po `Enum.GetValues<Rozmiar>()` z liczbą (`(int)r`) i dopłatą.
2. Wypisz paragon zamówienia z pozycjami i sumą.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod programu
- ☐ wyniki porównań rekordów i klas
- ☐ zrzut paragonu
- ☐ zrzut komunikatu przy próbie cofnięcia statusu

Pytania kontrolne

1. Czym rekord różni się od klasy przy porównaniu `==`?
2. Do czego służy wyrażenie `with`?
3. Jakie wartości liczbowe mają kolejne elementy `enum` i jak je zmienić?
4. Dlaczego status zamówienia lepiej zapisać jako `enum` niż jako `string`?

Kryteria oceny

Kryterium	Punkty
Wyliczenia i rekord z ceną	4
Kopie with i porównania (z wnioskami)	4
Zamówienie z kontrolą statusu	6
Raport	3
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Delegaty i zdarzenia — czujnik temperatury

Czas: 2 × 45 min | praca indywidualna | Wykład 1.18

Cel: Zbudujesz klasę czujnika, która powiadamia inne części programu o nowych pomiarach i przekroczeniu progu za pomocą zdarzeń, oraz napiszesz filtr przyjmujący warunek jako parametr `Func<T,bool>`.

Po zajęciach potrafisz:

- deklaruje zdarzenie `event EventHandler<T>` i własną klasę argumentów
- subskrybuje zdarzenie wieloma metodami obsługi (`+=`)
- przekazuje funkcję jako argument (`Func<double,bool>`, wyrażenie lambda)
- wyjaśnia, po co oddzielać źródło zdarzenia od reakcji

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET”)** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Klasa argumentów i czujnik

Utwórz projekt `Czujnik`.

```
public event EventHandler<OdczytEventArgs>? PrzekroczonoProg;

public void Zmierz(double temperatura)
{
    _pomiary.Add(temperatura);
    var e = new OdczytEventArgs(temperatura);
    NowyOdczyt?.Invoke(this, e);
    if (temperatura > ProgAlarmu) PrzekroczonoProg?.Invoke(this, e);
}
```

1. Klasa `OdczytEventArgs : EventArgs` z właściwościami `Temperatura` i `Czas`.
2. Klasa `Czujnik` z nazwą, progiem alarmu i listą pomiarów (tylko do odczytu na zewnątrz).
3. Zdarzenia: `NowyOdczyt` i `PrzekroczonoProg` typu `EventHandler<OdczytEventArgs>?`.
4. Metoda `Zmierz(double t)` zapisuje pomiar, zgłasza `NowyOdczyt`, a gdy t > próg — także `PrzekroczonoProg`.

2 Subskrybenci

W programie głównym podłącz trzy reakcje.

1. Wypisywanie każdego odczytu z godziną.
2. Alarm na czerwono z nazwą czujnika (z parametru `sender`).
3. Dziennik alarmów: dopisywanie wpisu „godzina;temperatura” do listy `dziennik`.
4. Symuluj 12 pomiarów: `new Random(7)` i `22 + los.NextDouble() * 8`. Dlaczego ziarno 7 daje za każdym razem te same wyniki?
5. Odłącz alarm (`-=`) po trzecim alarmie i sprawdź, że dziennik nadal działa (potrzebna nazwana metoda zamiast lambdy).

3 Filtrowanie funkcją

Napisz metodę `Filtr(IEnumerable<double> dane, Func<double, bool> warunek)`.

1. Metoda zwraca listę elementów spełniających warunek — bez LINQ, pętlą `foreach`.
2. Wywołaj ją z lambdami: temperatury powyżej 27 °C i poniżej 24 °C.
3. Dodaj trzeci warunek zapisany jako zwykła metoda `static bool Normalna(double t)` i przekaż jej nazwę.
4. Porównaj wynik z `Where` z LINQ.
5. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klas i programu
- ☐ zrzut ekranu symulacji z alarmami
- ☐ zawartość dziennika
- ☐ wyniki filtrów

Pytania kontrolne

1. Co oznacza zapis `NowyOdczyt?.Invoke(this, e)` i dlaczego jest `?.`?
2. Czym różni się delegat `Func<double,bool>` od `Action<double>`?
3. Dlaczego lambdy nie da się łatwo odłączyć operatorem `-=`?
4. Podaj przykład zdarzenia z aplikacji okienkowej.

Kryteria oceny

Kryterium	Punkty
Klasa argumentów, czujnik i zdarzenia	6
Trzech subskrybentów i odłączanie	5
Metoda Filtr z Func i trzy warunki	5
Pytania	4
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

LINQ w praktyce — wyniki uczniów

Czas: 2 × 45 min | praca indywidualna | Wykład 1.19

Cel: Przeanalizujesz listę uczniów i ich ocen zapytaniami LINQ: filtrowaniem, sortowaniem, grupowaniem i agregacją.

Po zajęciach potrafisz:

- filtruje i sortuje dane metodami `Where`, `OrderBy`, `ThenBy`
- grupuje dane `GroupBy` i liczy agregaty (`Count`, `Average`, `MaxBy`)
- spłaszcza kolekcje zagnieżdżone `SelectMany`
- zapisuje zapytanie w składni metod i składni zapytania

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Dane

Utwórz projekt `Analiza` i rekord ucznia.

1. `record Uczen(string Imie, string Nazwisko, string Klasa, List<int> Oceny)` z właściwością `Srednia`.
2. Utwórz listę co najmniej 8 uczniów z 3 klas (możesz użyć danych z rozwiązania nauczyciela lub własnych).
3. Zabezpiecz średnią dla ucznia bez ocen (zwróć 0).

2 Zapytania podstawowe

Każde zapytanie wypisz z nagłówkiem.

```
var klasy = uczniowie.GroupBy(u => u.Klasa)
    .Select(g => new { Klasa = g.Key, Liczba = g.Count(),
                       Srednia = g.Average(u => u.Srednia) })
    .OrderBy(k => k.Klasa);
```

1. Lista uczniów posortowana malejąco po średniej, przy równych — alfabetycznie po nazwisku.
2. Uczniowie zagrożeni — mający choć jedną ocenę 1 (`Contains`).
3. Uczniowie ze średnią co najmniej 4,75 (kandydaci do wyróżnienia).
4. Trzech najlepszych (`Take(3)`) zapisanych w składni zapytania (`from … orderby … select`).

3 Grupowanie

Zestawienia dla klas.

1. Liczba uczniów i średnia w każdej klasie (szablon powyżej).
2. Najlepszy uczeń w każdej klasie (`MaxBy` w grupie).
3. Klasa z najwyższą średnią.

4 Statystyki ocen

Wszystkie oceny razem.

1. Spłaszcz oceny `SelectMany` i policz: liczbę ocen, średnią, liczbę szóstek.
2. Rozkład ocen: ile jedynek, dwójek… (`GroupBy(o => o)`).
3. Sprawdź `Any` i `All`: czy ktoś ma wyłącznie oceny 5 i 6?
4. Sprawdź wyniki dwóch zapytań ręcznie (kalkulator).
5. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod programu
- ☐ wyniki wszystkich zapytań
- ☐ ręczne sprawdzenie dwóch wyników
- ☐ porównanie składni metod i składni zapytania

Pytania kontrolne

1. Co to jest wykonanie odroczone (deferred execution) w LINQ?
2. Czym różni się `Select` od `SelectMany`?
3. Co zwraca `GroupBy` i jak odczytać klucz grupy?
4. Kiedy `First()` zgłosi wyjątek i czym go zastąpić?

Kryteria oceny

Kryterium	Punkty
Dane i rekord ze średnią	3
Zapytania podstawowe (4)	6
Grupowanie (3)	5
Statystyki ocen	3
Sprawdzenie ręczne i pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Pliki CSV i JSON — dziennik wydatków

Czas: 2 × 45 min | praca indywidualna | Wykład 1.20

Cel: Wczytasz dziennik wydatków z pliku CSV z pominięciem błędnych wierszy, obliczysz sumy w kategoriach, zapiszesz raport oraz wyeksportujesz dane do JSON i odczytasz je z powrotem.

Po zajęciach potrafisz:

- odczytuje plik tekstowy wiersz po wierszu i dzieli wiersze na pola
- przetwarza liczby i daty z jawnie podaną kulturą
- zapisuje raport do pliku CSV
- serializuje i deserializuje listę obiektów do JSON z polskimi znakami

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Przygotowanie przed zajęciami

- Pobierz plik `wydatki.csv` z folderu `pliki` przy tym zadaniu.

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Odczyt CSV

Utwórz projekt `Wydatki` i dodaj do niego plik `wydatki.csv`; we właściwościach pliku ustaw **Kopiuj do katalogu wyjściowego: Kopiuj, jeśli nowszy**.

1. Rekord `Wydatek(DateOnly Data, string Kategoria, string Opis, decimal Kwota)`.
2. Czytaj plik `File.ReadLines`; pomiń nagłówek i puste wiersze.
3. Dziel wiersz `Split(';')`; datę czytaj `DateOnly.TryParseExact(p[0], "yyyy-MM-dd", out var d)`, kwotę `decimal.TryParse` z kulturą pl-PL.
4. Wiersz błędny pomiń i wypisz jego numer — w pliku jest jeden celowo błędny wiersz.
5. Wypisz liczbę wczytanych wydatków i błędnych wierszy.

2 Analiza i raport

Sumy w kategoriach.

1. Pogrupuj wydatki po kategorii, policz sumę i udział procentowy; posortuj malejąco.
2. Wypisz tabelę w konsoli (kwoty `:C`, udział `:P1`) i sumę całkowitą.
3. Zapisz raport do `raport.csv` (`File.WriteAllLines`) z nagłówkiem `kategoria;suma;udział`.
4. Otwórz `raport.csv` w Excelu lub LibreOffice Calc — czy kolumny rozdzieliły się poprawnie?

3 JSON

Eksport i import.

```
var opcje = new JsonSerializerOptions
{
    WriteIndented = true,
    Encoder = JavaScriptEncoder.Create(UnicodeRanges.All)   // polskie znaki bez \u0105
};
File.WriteAllText("wydatki.json", JsonSerializer.Serialize(wydatki, opcje));
```

1. Zapisz listę wydatków do `wydatki.json` metodą `JsonSerializer.Serialize` z opcjami poniżej.
2. Otwórz plik JSON w Visual Studio i sprawdź, czy polskie znaki są czytelne. Usuń opcję `Encoder` i porównaj.
3. Odczytaj plik `JsonSerializer.Deserialize<List<Wydatek>>` i sprawdź zgodność `SequenceEqual`.
4. Zatwierdź pracę w Git (bez plików wynikowych z folderu `bin`).

W sprawozdaniu umieść

- ☐ kod programu
- ☐ zrzut tabeli sum
- ☐ zawartość `raport.csv` i fragment `wydatki.json`
- ☐ porównanie JSON z opcją Encoder i bez niej

Pytania kontrolne

1. Dlaczego przy odczycie liczb z pliku warto podać kulturę jawnie?
2. Czym różni się `File.ReadAllLines` od `File.ReadLines`?
3. Gdzie program szuka pliku podanego bez ścieżki i dlaczego trzeba ustawić „Kopiuj do katalogu wyjściowego”?
4. Czym różni się format CSV od JSON? Kiedy który wybrać?

Kryteria oceny

Kryterium	Punkty
Odczyt CSV z obsługą błędnych wierszy	6
Sumy, udziały i raport CSV	5
Eksport i import JSON z polskimi znakami	5
Pytania	4
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Testy jednostkowe MSTest

Czas: 2 × 45 min | praca indywidualna | Wykład 1.21

Cel: Utworzysz projekt testów MSTest dla klas `Kosc` (zadanie 15) i `Matematyka` (zadanie 10), korzystając z testów parametryzowanych i testów wyjątków.

Po zajęciach potrafisz:

- tworzy projekt testowy i dodaje odwołanie do testowanego projektu
- pisze testy według wzorca Arrange–Act–Assert
- stosuje `[DataRow]` do testów parametryzowanych
- testuje zgłaszanie wyjątków `Assert.ThrowsExactly<T>`

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Przygotowanie przed zajęciami

- Przygotuj projekt z klasą `Kosc` z zadania 15 i klasę `Matematyka` z zadania 10.

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Struktura rozwiązania

Biblioteka klas + testy.

1. Utwórz rozwiązanie z projektem **Biblioteka klas** `GraLogika` (.NET 10) i skopiuj do niego `Kosc.cs` oraz `Matematyka.cs` (klasa `public static`).
2. Dodaj projekt **Projekt testowy MSTest** `GraLogika.Tests`.
3. W projekcie testów: Dodaj → Odwołanie do projektu → `GraLogika`.
4. Otwórz **Eksplorator testów** (Test → Eksplorator testów) i uruchom przykładowy test z szablonu.

2 Testy klasy Kosc

Klasa `KoscTests` z atrybutem `[TestClass]`.

```
[TestMethod]
[DataRow(new[] { 3, 3, 2, 4, 2 }, 10)]
[DataRow(new[] { 1, 2, 3, 4, 5 }, 0)]
public void Punkty_SumujePowtarzajaceSieWartosci(int[] wartosci, int oczekiwane)
{
    Assert.AreEqual(oczekiwane, Kosc.Punkty(wartosci));
}

[TestMethod]
[DataRow(0)] [DataRow(7)]
public void Ustaw_WartoscSpozaZakresu_ZglaszaWyjatek(int w)
{
    var kosc = new Kosc();
    Assert.ThrowsExactly<ArgumentOutOfRangeException>(() => kosc.Ustaw(w));
}
```

1. Test: 1000 rzutów daje wartości 1–6 (użyj `new Kosc(new Random(1))` — wynik powtarzalny).
2. Test: zablokowana kość nie zmienia wartości.
3. Test: nowa kość ma wartość 0; konstruktor zwiększa `LiczbaKosci`.
4. Test parametryzowany `Ustaw` dla 0, 7 i −1 — oczekiwany `ArgumentOutOfRangeException`.
5. Test parametryzowany `Punkty` dla co najmniej 4 zestawów (szablon).

3 Testy klasy Matematyka

Klasa `MatematykaTests`.

1. `Nwd` dla 4 par, w tym z zerem i liczbą ujemną.
2. `CzyPierwsza` dla 2, 97, 1, 91 i −7.
3. Zgodność `Silnia` i `SilniaRek` dla 0, 1, 5, 15.
4. `MinMax` — dwa asercje w jednym teście.

4 Test, który wykrywa błąd

Sprawdź, czy testy naprawdę działają.

1. Wprowadź celowo błąd do `Kosc.Punkty` (np. `>= 3` zamiast `>= 2`) i uruchom testy.
2. Zrób zrzut Eksploratora testów z czerwonymi testami, popraw błąd i zrób zrzut z zielonymi.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ struktura rozwiązania (zrzut Eksploratora rozwiązań)
- ☐ kod klas testowych
- ☐ zrzut Eksploratora testów: testy niezaliczone i zaliczone
- ☐ lista przypadków testowych

Pytania kontrolne

1. Co oznacza wzorzec Arrange–Act–Assert?
2. Dlaczego w testach kości przekazujemy `Random` z ziarnem?
3. Po co projekt logiki jest osobną biblioteką klas?
4. Co powinien sprawdzać dobry test — przypadki typowe, graniczne czy błędne? Uzasadnij.

Kryteria oceny

Kryterium	Punkty
Struktura rozwiązania i odwołanie	3
Testy Kosc (min. 6, w tym DataRow i wyjątek)	6
Testy Matematyka (min. 4)	4
Wykrycie celowego błędu (zrzuty)	3
Pytania	4
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Aplikacja konsolowa jak na egzaminie — parking

Czas: 2 × 45 min

praca samodzielna na czas — warunki egzaminacyjne (dozwolona tylko dokumentacja Microsoft Learn)

Wykład 1.13–1.21

Cel: W 90 minut wykonasz zadanie w stylu części konsolowej egzaminu INF.04: klasę z polem statycznym i walidacją, program główny, testy jednostkowe i dokumentację ze zrzutami.

Po zajęciach potrafisz:

- realizuje specyfikację klasy w wyznaczonym czasie
- pisze program obsługujący błędne dane wejściowe
- tworzy testy jednostkowe dla wartości granicznych
- przygotowuje dokumentację kodu i zrzuty ekranu jak na egzaminie

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Specyfikacja klasy Bilet

Utwórz rozwiązanie `Parking` z aplikacją konsolową, biblioteką klas i projektem MSTest (jak w zadaniu 22).

1. Statyczna właściwość `LiczbaBiletow` (tylko do odczytu z zewnątrz) — liczba wydanych biletów.
2. Właściwość `Numer` — kolejny numer biletu nadawany w konstruktorze (1, 2, 3...).
3. Właściwość `Minuty` — czas postoju podany w konstruktorze; dla wartości spoza 0–10080 konstruktor zgłasza `ArgumentOutOfRangeException`.
4. Metoda `decimal Oplata()`: do 15 minut włącznie — 0 zł; powyżej — 3 zł za każdą rozpoczętą godzinę liczoną od początku postoju, najwyżej 40 zł za dobę; za każdą pełną dobę 40 zł, a pozostałe minuty liczone jak wyżej (z limitem 40 zł).

2 Program główny

Kasa parkingu.

1. Program pyta o czas postoju w minutach, aż użytkownik wpisze pusty wiersz.
2. Dla każdego poprawnego czasu wypisuje: `Bilet nr 1: 90 min, do zapłaty 6,00 zł`.
3. Dla tekstu niebędącego liczbą i czasu spoza zakresu wypisuje zrozumiały komunikat i pyta ponownie.
4. Na końcu wypisuje liczbę wydanych biletów i utarg.

3 Testy jednostkowe

Co najmniej 10 przypadków dla metody `Oplata` i testy wyjątków.

1. Przed pisaniem testów oblicz ręcznie oczekiwane opłaty dla: 0, 15, 16, 60, 61, 780, 781, 1440, 1441 i 3000 minut.
2. Zapisz je jako jeden test z `[DataRow]`.
3. Dodaj testy wyjątku dla −1 i 10081 oraz test kolejnych numerów biletów.
4. Jeśli test nie przechodzi — sprawdź najpierw, czy dobrze policzyłeś oczekiwany wynik, potem kod.

4 Dokumentacja

Jak na egzaminie.

1. Komentarze `///` przy klasie, konstruktorze, właściwościach i metodzie (opis, parametry, wynik, wyjątki).
2. W nagłówku plików wpisz zamiast nazwiska numer (np. numer z dziennika) — na egzaminie podaje się numer PESEL lub numer zdającego.
3. Zrzuty ekranu: działanie programu (poprawne i błędne dane) oraz wynik testów w Eksploratorze testów.
4. Spakuj folder rozwiązania (bez `bin` i `obj`) i oddaj zgodnie z poleceniem nauczyciela.

W sprawozdaniu umieść

- ☐ kod klasy Bilet z dokumentacją
- ☐ kod programu głównego
- ☐ tabela ręcznych obliczeń opłat
- ☐ zrzuty: program i Eksplorator testów
- ☐ czas wykonania każdej części (samoocena)

Pytania kontrolne

1. Która część zadania zajęła najwięcej czasu i jak możesz to przyspieszyć na egzaminie?

2. Dlaczego numer biletu można nadać na podstawie licznika statycznego?
3. Jakie wartości graniczne ma metoda Oplata i dlaczego to one są najważniejsze w testach?

Kryteria oceny

Kryterium	Punkty
Klasa Bilet zgodna ze specyfikacją	6
Program główny z obsługą błędów	4
Testy (min. 10 przypadków, wyjątki)	4
Dokumentacja /// i zrzuty	4
Organizacja rozwiązania i pytania	2
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Pierwsza aplikacja WPF

Czas: 2 × 45 min | praca indywidualna | Wykład 1.22–1.23

Cel: Utworzysz pierwszą aplikację okienkową WPF, poznasz pliki projektu i projektant XAML oraz obsłużysz kliknięcie przycisku.

Po zajęciach potrafisz:

- tworzy projekt WPF dla .NET 10 i uruchamia go
- rozróżnia pliki XAML i code-behind
- ustawia właściwości okna i kontrolek w XAML
- obsługuje zdarzenie Click i zmienia treść kontrolek w kodzie
- korzysta z funkcji Przeładowywanie na gorąco (Hot Reload)

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET”)** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Rób przerwy dla oczu (zasada 20-20-20) — praca z projektantem okien wymaga długiego patrzenia w ekran.

Przebieg zadania

1 Nowy projekt

Utwórz projekt z szablonu **Aplikacja WPF** (nie „WPF App (.NET Framework)”).

1. Nazwa: `Powitanie`, platforma: .NET 10.0 (LTS).
2. Obejrzyj w Eksploratorze rozwiązań pliki `App.xaml`, `MainWindow.xaml`, `MainWindow.xaml.cs` i `Powitanie.csproj`.
3. W pliku `.csproj` znajdź `UseWPF` i `TargetFramework` — zapisz ich wartości.
4. Uruchom pusty projekt (F5).

2 Okno i kontrolki

Zbuduj okno według wzoru — w XAML, nie przeciągając kontrolek.

```
<StackPanel Margin="16">
    <TextBlock Text="Jak masz na imię?" FontSize="16" />
    <TextBox x:Name="TxtImie" FontSize="16" Padding="4" />
    <Button x:Name="BtnPowitaj" Content="Powitaj" Width="120"
            HorizontalAlignment="Left" Margin="0,10,0,0"
            Click="BtnPowitaj_Click" />
    <TextBlock x:Name="TxtPowitanie" FontSize="20" FontWeight="Bold" />
</StackPanel>
```

1. Ustaw okno: `Title="Powitanie"`, wysokość 260, szerokość 420, `WindowStartupLocation="CenterScreen"`.
2. Zastąp `Grid` panelem `StackPanel` z marginesem 16.
3. Dodaj: TextBlock z pytaniem, TextBox `TxtImie`, Button `BtnPowitaj`, TextBlock `TxtPowitanie` (duża, pogrubiona czcionka w kolorze #512BD4).
4. Nadaj kontrolkom nazwy `x:Name` z przedrostkami (Txt, Btn).

3 Obsługa kliknięcia

Kod w `MainWindow.xaml.cs`.

1. W metodzie `BtnPowitaj_Click` wypisz „Witaj, [imię]!" albo „Wpisz imię.", gdy pole jest puste.
2. Dodaj pole `_kliknięcia` i wypisuj liczbę kliknięć w drugim TextBlocku.
3. Ustaw `IsDefault="True"` na przycisku i sprawdź działanie klawisza Enter.
4. Po uruchomieniu ustaw kursor w polu imienia: `TxtImie.Focus();` w konstruktorze.

4 Hot Reload

Zmiany bez ponownego uruchamiania.

1. Uruchom aplikację (F5) i nie zamykając jej, zmień w XAML kolor i rozmiar napisu. Zapisz plik i obserwuj okno.
2. Zmień tekst powitania w kodzie C# i użyj przycisku Hot Reload (ikona płomienia).
3. Zrób zrzut działającej aplikacji. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ wartości `UseWPF` i `TargetFramework`
- ☐ kod XAML i code-behind
- ☐ zrzut działającej aplikacji
- ☐ opis działania Hot Reload

Pytania kontrolne

1. Do czego służy plik XAML, a do czego plik `.xaml.cs`?
2. Co oznacza `x:Name` i jak korzystamy z niego w kodzie?
3. Czym różni się TextBlock od TextBox?
4. Co robi metoda `InitializeComponent()`?

Kryteria oceny

Kryterium	Punkty
Projekt WPF .NET 10 i opis plików	3
Okno zbudowane w XAML zgodnie z opisem	6
Obsługa kliknięcia, licznik, Enter, fokus	6
Hot Reload i zrzuty	2
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Układ okna formularza

Czas: 2 × 45 min | praca indywidualna | Wykład 1.24

Cel: Zbudujesz okno formularza rejestracji, które poprawnie zachowuje się przy zmianie rozmiaru, używając kontenerów Grid, StackPanel i DockPanel.

Po zajęciach potrafisz:

- definiuje wiersze i kolumny Grid z rozmiarami Auto, liczbą i *
- umieszcza kontrolki w komórkach (`Grid.Row`, `Grid.Column`, `Grid.ColumnSpan`)
- buduje typowy układ okna z menu i paskiem stanu w DockPanel
- sprawdza okno przy różnych rozmiarach

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Rób przerwy dla oczu (zasada 20-20-20) — praca z projektantem okien wymaga długiego patrzenia w ekran.

Przebieg zadania

1 Szkic

Narysuj na kartce układ okna z zaznaczonymi wierszami i kolumnami Grida.

1. Okno: menu u góry, pasek stanu na dole, przyciski Zapisz/Anuluj nad paskiem stanu po prawej.
2. Formularz: Imię, Nazwisko, E-mail, Hasło (PasswordBox), zgoda (CheckBox), Uwagi (wielowierszowe, zajmują resztę miejsca).
3. Zaplanuj: kolumna etykiet `Auto`, kolumna pól `*`; wiersz Uwag `*`.

2 DockPanel

Utwórz projekt WPF `Rejestracja`.

1. Zastąp domyślny Grid panelem `DockPanel`.
2. Dodaj `Menu` (Plik: Wyczyść, Zakończ; Pomoc) z `DockPanel.Dock="Top"`.
3. Dodaj `StatusBar` z tekstem „Wypełnij wszystkie pola” (`Dock="Bottom"`).
4. Dodaj poziomy `StackPanel` z przyciskami wyrównany do prawej (`Dock="Bottom"`) — kolejność elementów w XAML ma znaczenie!

3 Grid formularza

Ostatnie dziecko DockPanel wypełnia środek — to będzie Grid.

```
<Grid Margin="12">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />  <!-- ×5 -->
        <RowDefinition Height="*" />     <!-- Uwagi -->
    </Grid.RowDefinitions>
    <Label Content="_Imię:" Target="{Binding ElementName=TxtImie}" />
    <TextBox x:Name="TxtImie" Grid.Column="1" Margin="4" />
</Grid>
```

1. Zdefiniuj 2 kolumny i 6 wierszy (szablon).
2. Etykiety jako `Label` ze skrótem klawiszowym (`_Imię:`) i `Target` wskazującym pole.
3. Pole Uwagi: `AcceptsReturn="True"`, `TextWrapping="Wrap"`, przewijanie `Auto`.
4. Ustaw `MinWidth` i `MinHeight` okna.

4 Test rozmiarów

Sprawdź okno przy różnych rozmiarach.

1. Zrób zrzuty okna: rozmiar domyślny, zmaksymalizowane, minimalne.
2. Sprawdź skróty Alt+I, Alt+N... i kolejność klawisza Tab.
3. Popraw układ, jeśli coś się rozjeżdża. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ szkic układu
- ☐ kod XAML
- ☐ trzy zrzuty przy różnych rozmiarach
- ☐ opis poprawek

Pytania kontrolne

1. Co oznaczają rozmiary Auto, 100 i * w Grid?
2. Dlaczego kolejność elementów w DockPanel ma znaczenie?
3. Czym różni się Margin od Padding?
4. Jak działa skrót klawiszowy w Label z właściwością Target?

Kryteria oceny

Kryterium	Punkty
Szkic	2
DockPanel: menu, pasek stanu, przyciski	5
Grid formularza z poprawnymi rozmiarami	6
Zachowanie przy zmianie rozmiaru (zrzuty)	4
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Kontrolki wyboru — konfigurator pizzy

Czas: 2 × 45 min | praca indywidualna | Wykład 1.25

Cel: Zbudujesz konfigurator pizzy z przyciskami opcji, polami wyboru, listą rozwijaną i suwakiem, a cenę obliczy osobna klasa logiki.

Po zajęciach potrafisz:

- odczytuje stan kontrolek RadioButton, CheckBox, ComboBox i Slider
- aktualizuje wynik przy każdej zmianie wyboru
- oddziela obliczenia od interfejsu (klasa Cennik)
- zabezpiecza obsługę zdarzeń zgłaszanych podczas wczytywania okna

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Rób przerwy dla oczu (zasada 20-20-20) — praca z projektantem okien wymaga długiego patrzenia w ekran.

Przebieg zadania

1 Klasa Cennik

Utwórz projekt WPF `Pizzeria` i plik `Cennik.cs` (bez WPF).

1. `enum Rozmiar { Mala, Srednia, Duza }`; ceny bazowe 24 / 32 / 42 zł.
2. Metoda `Cena(Rozmiar r, int liczbaDodatkow, bool grubeCiasto, int ilosc)`: dodatek 4 zł, grube ciasto +3 zł; od 3 sztuk rabat 10% (zaokrąglenie do grosza).
3. Dla ilości spoza 1–10 zgłoś `ArgumentOutOfRangeException`.
4. Policz ręcznie: średnia, 2 dodatki, grube, 1 szt. = 43 zł; średnia, 1 dodatek, 3 szt. = 97,20 zł.

2 Okno

Zbuduj interfejs na Gridzie z dwiema kolumnami.

1. GroupBox „Rozmiar” z trzema RadioButton (wspólny `GroupName`, mała zaznaczona).
2. GroupBox „Dodatki” z WrapPanel `PnlDodatki` i co najmniej 5 CheckBox.
3. ComboBox `CmbCiasto` (cienkie, grube), Slider `SldIlosc` 1–10 z przeskokiem co 1 i wyświetlaną wartością.
4. ListBox zamówień, duży TextBlock z ceną i przycisk Zamów.

3 Cena na żywo

Cena zmienia się przy każdej zmianie wyboru.

```
private void Przelicz()
{
    // zdarzenia podczas wczytywania okna – kontrolki mogą jeszcze nie istnieć
    if (TxtCena is null || PnlDodatki is null || SldIlosc is null) return;
    TxtCena.Text = $"Do zapłaty: {AktualnaCena():C}";
}
```

1. Podłącz zdarzenia `Checked` / `Unchecked`, `SelectionChanged`, `ValueChanged` do metody `Przelicz()`.
2. Liczbę zaznaczonych dodatków policz: `PnlDodatki.Children.OfType<CheckBox>().Count(c => c.IsChecked == true)`.
3. Uruchom program. Jeśli pojawi się `NullReferenceException` przy starcie — dodaj w `Przelicz` sprawdzenie, czy kontrolki już istnieją (szablon).
4. Przycisk Zamów dopisuje do listy opis zamówienia z ceną.

4 Sprawdzenie

Porównaj ceny z obliczeniami ręcznymi.

1. Przetestuj co najmniej 5 konfiguracji i wpisz je do tabeli.
2. Zrób zrzut okna z kilkoma zamówieniami. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klasy Cennik
- ☐ kod XAML i obsługi zdarzeń
- ☐ tabela 5 konfiguracji z ceną ręczną i z programu
- ☐ zrzut okna

Pytania kontrolne

1. Dlaczego warunek zapisujemy `IsChecked == true`?
2. Dlaczego zdarzenie Checked może wystąpić, zanim istnieją inne kontrolki?
3. Co daje wydzielenie obliczeń do klasy Cennik?
4. Jak odczytać tekst wybranego elementu ComboBox?

Kryteria oceny

Kryterium	Punkty
Klasa Cennik z rabatem i walidacją	5
Okno z wszystkimi kontrolkami	5
Cena aktualizowana na żywo	4
Lista zamówień	2
Tabela testów i pytania	4
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Kalkulator WPF

Czas: 2 × 45 min | praca indywidualna | Wykład 1.26

Cel: Zbudujesz kalkulator z przyciskami cyfr i działań, w którym jedna metoda obsługuje wszystkie przyciski, a stan kalkulatora przechowuje osobna klasa.

Po zajęciach potrafisz:

- obsługuje wiele przycisków jedną metodą (parametr `sender`, `Tag`, zdarzenia trasowane)
- przechowuje stan aplikacji w klasie logiki
- prowadzi historię działań w ListBox
- obsługuje klawiaturę zdarzeniem KeyDown (rozszerzenie)

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Rób przerwy dla oczu (zasada 20-20-20) — praca z projektantem okien wymaga długiego patrzenia w ekran.

Przebieg zadania

1 Logika kalkulatora

Utwórz projekt WPF `Kalkulator` i klasę `KalkulatorLogika` (bez WPF).

1. Właściwość `Ekran` (string, początkowo „0").
2. Metody: `Cyfra(char)`, `Przecinek()`, `Dzialanie(char)`, `Rownasie()` (zwraca wpis do historii), `Wyczysc()`.
3. Dzielenie przez zero — komunikat na ekranie zamiast wyjątku; następny klawisz czyści błąd.
4. Działania wykonuj od lewej do prawej, jak w prostym kalkulatorze (2 + 3 × 4 = 20).

2 Okno

Ekran (TextBox tylko do odczytu, tekst do prawej) i siatka przycisków.

1. Użyj `UniformGrid` 5 × 4 z przyciskami C, ÷, ×, −, cyfry, +, =, przecinek.
2. Każdy przycisk ma `Tag` ze znakiem dla logiki (np. Content="×" Tag="*").
3. Po prawej: ListBox historii i przycisk czyszczenia historii.

3 Jedna metoda dla wszystkich przycisków

Zamiast 17 metod — jedna.

```
private void Przycisk_Click(object sender, RoutedEventArgs e)
{
    if (e.OriginalSource is Button b && b.Tag is string znak)
        Obsluz(znak[0]);
}
```

1. Podłącz zdarzenie na kontenerze: `<UniformGrid ButtonBase.Click="Przycisk_Click">`.
2. W metodzie odczytaj przycisk z `e.OriginalSource` i znak z `Tag`, a następnie wywołaj odpowiednią metodę logiki.
3. Po każdym klawiszu przepisz `Ekran` do TextBoxa; wynik `=` dopisz na górze historii.
4. Przetestuj: 12 + 7,5 =; 10 ÷ 4 =; 1 ÷ 0 =; 0,1 + 0,2 =; 7 ÷ 3 =.

4 Klawiatura (rozszerzenie)

Obsługa klawiatury numerycznej.

1. Dodaj zdarzenie `KeyDown` okna i zamień klawisze (`Key.NumPad0`… `Key.Add`, `Key.Enter`, `Key.Escape`) na znaki.
2. Ustaw `e.Handled = true` po obsłużeniu klawisza.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klasy KalkulatorLogika
- ☐ kod obsługi przycisków
- ☐ tabela testów
- ☐ zrzut kalkulatora z historią

Pytania kontrolne

1. Czym różni się `sender` od `e.OriginalSource` w zdarzeniu trasowanym?
2. Do czego służy właściwość `Tag`?
3. Dlaczego 0,1 + 0,2 daje w tym kalkulatorze dokładnie 0,3?
4. Jak zmienić logikę, aby mnożenie miało pierwszeństwo przed dodawaniem?

Kryteria oceny

Kryterium	Punkty
Klasa logiki z obsługą błędów	6
Okno z UniformGrid i Tag	4
Jedna metoda obsługi przycisków	4
Historia i testy	3
Klawiatura lub pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Walidacja formularza

Czas: 2 × 45 min | praca indywidualna | Wykład 1.27

Cel: Zbudujesz formularz zgłoszenia, który sprawdza wszystkie pola naraz, wyróżnia błędne pola i pokazuje czytelne komunikaty; reguły walidacji sprawdzisz testami jednostkowymi.

Po zajęciach potrafisz:

- wydziela reguły walidacji do klasy testowalnej
- pokazuje wszystkie błędy jednocześnie (MessageBox i komunikaty pod formularzem)
- wyróżnia błędne pola ramką i podpowiedzią ToolTip
- pisze testy dla wartości granicznych reguł

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Rób przerwy dla oczu (zasada 20-20-20) — praca z projektantem okien wymaga długiego patrzenia w ekran.

Przebieg zadania

1 Klasa Walidator

Utwórz projekt WPF `Zgloszenie` i statyczną klasę `Walidator`.

1. Metody zwracające komunikat błędu albo `null`: `SprawdzImie` (min. 2 litery), `SprawdzEmail` (wyrażenie regularne z lekcji), `SprawdzWiek` (13–120), `SprawdzZgode`.
2. Metoda `Sprawdz(...)` zwraca listę wszystkich błędów.
3. Komunikaty mają mówić, jak poprawić dane, np. „Wiek musi być liczbą od 13 do 120.".

2 Formularz

Okno na Gridzie.

```
private void Oznacz(Control pole, string? blad)
{
    pole.BorderBrush = blad is null ? _ramkaDomyslna : Brushes.Red;
    pole.ToolTip = blad;
}
```

1. Pola: imię, e-mail, wiek (wąskie, `MaxLength="3"`), CheckBox zgody, TextBlock na błędy (kolor czerwony), przyciski Wyczyść i Wyślij.
2. Po kliknięciu Wyślij: oznacz każde pole (czerwona ramka + ToolTip z opisem albo przywrócenie domyślnej ramki).
3. Gdy są błędy — pokaż je w TextBlocku i w jednym `MessageBox` z ikoną ostrzeżenia; w przeciwnym razie podziękowanie z ikoną informacji.
4. Domyślny kolor ramki zapamiętaj w zdarzeniu `Loaded` okna.

3 Czyszczenie

Przycisk Wyczyść.

1. Zapytaj `MessageBox` YesNo „Wyczyścić formularz?”.
2. Po potwierdzeniu wyczyść pola, odznacz zgodę, usuń oznaczenia błędów i ustaw kursor w polu imienia.

4 Testy jednostkowe

Projekt MSTest dla klasy Walidator (klasę przenieś do biblioteki klas lub dodaj plik jako link).

1. Test: poprawne dane — 0 błędów; puste dane — 4 błędy.
2. Test parametryzowany e-mail dla 4 adresów (poprawny, bez domeny, ze spacją, bez kropki).
3. Test parametryzowany wieku dla 12, 13, 120, 121 i tekstu.
4. Zrzut Eksploratora testów. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod klasy Walidator i testów
- ☐ zrzut formularza z oznaczonymi błędami
- ☐ zrzut MessageBox
- ☐ zrzut Eksploratora testów

Pytania kontrolne

1. Dlaczego walidację umieszczamy w osobnej klasie?
2. Dlaczego pokazujemy wszystkie błędy naraz?
3. Jak przywrócić pierwotny wygląd pola po poprawieniu błędu?
4. Jakie kontrolki mogłyby zastąpić TextBox wieku, aby uniknąć błędów?

Kryteria oceny

Kryterium	Punkty
Klasa Walidator z czytelnymi komunikatami	5
Oznaczanie pól i komunikaty	5
Czyszczenie z potwierdzeniem	2
Testy jednostkowe (min. 10 przypadków)	5
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Style i zasoby

Czas: 2 × 45 min | praca indywidualna | Wykład 1.28

Cel: Ujednolicisz wygląd aplikacji z zadania 26 za pomocą zasobów i stylów w App.xaml oraz dodasz przełącznik motywu jasnego i ciemnego.

Po zajęciach potrafisz:

- definiuje zasoby kolorów i rozmiarów w słownikach zasobów
- tworzy style z kluczem, style niejawne i dziedziczenie BasedOn
- łączy słowniki w App.xaml (MergedDictionaries)
- przełącza motyw w trakcie działania dzięki DynamicResource

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Przygotowanie przed zajęciami

- Przygotuj projekt Pizzeria z zadania 26 (lub inną swoją aplikację WPF z zadań 25–28).

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Rób przerwy dla oczu (zasada 20-20-20) — praca z projektantem okien wymaga długiego patrzenia w ekran.

Przebieg zadania

1 Inwentaryzacja

Znajdź w XAML powtarzające się ustawienia wyglądu.

1. Wypisz w tabeli powtarzające się wartości (kolory, Margin, FontSize, Padding) i liczbę ich wystąpień.
2. Zaplanuj nazwy zasobów: `KolorAkcentu`, `KolorTla`, `KolorTekstu`, `RozmiarTekstu`...

2 Słowniki i style

Dodaj pliki: Projekt → Dodaj → Nowy element → Słownik zasobów (WPF).

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="Themes/Jasny.xaml" />
            <ResourceDictionary Source="Styles.xaml" />
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

1. `Themes/Jasny.xaml` i `Themes/Ciemny.xaml` — te same klucze kolorów, różne wartości.
2. `Styles.xaml` — styl okna `StylOkna`, style niejawne dla TextBox, CheckBox, RadioButton, GroupBox, Label.
3. Style przycisków: `PrzyciskGlowny` i `PrzyciskUsun` z `BasedOn`.
4. W App.xaml połącz słowniki — motyw jako **pierwszy** (szablon).
5. Kolory zmieniające się z motywem wskazuj przez `DynamicResource`.

3 Zastosowanie

Przerób okno aplikacji.

1. Dodaj do okna `Style="{StaticResource StylOkna}"` i nagłówek ze stylem `Naglowek`.
2. Usuń z kontrolek wpisane na sztywno ustawienia, które przejęły style.
3. Dodaj przycisk Usuń zamówienie ze stylem `PrzyciskUsun`.
4. Policz, ile atrybutów wyglądu ubyło z MainWindow.xaml.

4 Motyw ciemny

Przełącznik CheckBox „Tryb ciemny”.

1. W obsłudze Checked/Unchecked podmień pierwszy słownik: `slowniki[0] = new ResourceDictionary { Source = new Uri(...) }`.
2. Sprawdź, które elementy zmieniły kolor, a które nie — popraw te, które używają `StaticResource` zamiast `DynamicResource`.
3. Zrób zrzuty w obu motywach. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ tabela powtarzających się ustawień przed zmianą
- ☐ kod słowników i App.xaml
- ☐ liczba usuniętych atrybutów
- ☐ zrzuty w motywie jasnym i ciemnym

Pytania kontrolne

1. Czym różni się StaticResource od DynamicResource?
2. Kiedy styl bez klucza dotyczy kontrolki, a kiedy nie?
3. Dlaczego niejawny styl TextBlock w App.xaml może zmienić tekst w przyciskach?
4. Która wartość wygrywa: ustawiona w stylu czy bezpośrednio na kontrolce?

Kryteria oceny

Kryterium	Punkty
Inwentaryzacja	2
Słowniki motywów i Styles.xaml	6
Zastosowanie stylów w oknie	4
Przełącznik motywu	5
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Wiele okien i pliki — notatnik

Czas: 2 × 45 min | praca indywidualna | Wykład 1.29

Cel: Napiszesz prosty notatnik z menu Plik, systemowymi oknami Otwórz i Zapisz, oknem „O programie" oraz pytaniem o zapis zmian przy zamykaniu.

Po zajęciach potrafisz:

- korzysta z OpenFileDialog i SaveFileDialog z filtrem plików
- tworzy dodatkowe okno i otwiera je modalnie (ShowDialog, Owner)
- śledzi niezapisane zmiany i pyta o zapis (MessageBox YesNoCancel)
- obsługuje wyjątki przy operacjach na plikach

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Rób przerwy dla oczu (zasada 20-20-20) — praca z projektantem okien wymaga długiego patrzenia w ekran.
- Testuj zapis w folderze `Dokumenty`, nie w folderach systemowych; nie nadpisuj plików innych osób.

Przebieg zadania

1 Okno główne

Utwórz projekt WPF `Notatnik`.

1. DockPanel: Menu (Plik: Nowy, Otwórz..., Zapisz, Zapisz jako..., Zakończ; Pomoc: O programie), StatusBar z liczbą znaków, TextBox na resztę okna.
2. TextBox: `AcceptsReturn`, `AcceptsTab`, zawijanie, przewijanie, czcionka Consolas.
3. Pola w klasie okna: `_sciezka` (null dla nowego pliku) i `_zmieniony`.
4. W `TextChanged` ustaw `_zmieniony = true`, odśwież licznik znaków i tytuł z gwiazdką: „Notatnik — notatka.txt *".

2 Otwórz i Zapisz

Systemowe okna dialogowe z `Microsoft.Win32`.

1. Otwórz: `OpenFileDialog` z filtrem `Pliki tekstowe (*.txt)|*.txt|Wszystkie pliki (*.*)|*.*`.
2. Zapisz: gdy plik ma już ścieżkę — zapisz bez pytania; w przeciwnym razie `SaveFileDialog`. „Zapisz jako" zawsze pyta.
3. Operacje na plikach w `try` / `catch` (`IOException`, `UnauthorizedAccessException`) z komunikatem o błędzie.
4. Sprawdź zapis do folderu bez uprawnień (np. `C:\Windows`) — program nie może się zakończyć błędem.

3 Niezapisane zmiany

Metoda `CzyMoznaPorzucic()` pyta „Zapisać zmiany w pliku?" (Tak/Nie/Anuluj).

```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    if (!CzyMoznaPorzucic()) e.Cancel = true;
}
```

1. Tak — zapisz (jeśli zapis się nie uda lub anulowano okno zapisu — nie porzucaj).
2. Nie — porzuć zmiany; Anuluj — przerwij operację.
3. Wywołuj ją przed Nowy, Otwórz i w zdarzeniu `Closing` okna (`e.Cancel = true`).
4. Zakończ wywołuje `Close()` — sprawdź, że pytanie pojawia się tylko raz.

4 Okno O programie

Projekt → Dodaj → Okno (WPF) `OknoOProgramie`.

1. Nazwa programu, wersja, wersja .NET (`Environment.Version`), autor, przycisk OK z `IsCancel="True"`.
2. Okno: `WindowStartupLocation="CenterOwner"`, `ResizeMode="NoResize"`, `ShowInTaskbar="False"`.
3. Otwórz je `ShowDialog()` z `Owner = this`. Sprawdź, że okno główne jest zablokowane.
4. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod obu okien
- ☐ zrzuty: okno Zapisz, pytanie o zapis zmian, okno O programie
- ☐ tabela testów scenariuszy

Pytania kontrolne

1. Czym różni się Show od ShowDialog?
2. Co robi `e.Cancel = true` w zdarzeniu Closing?
3. Dlaczego dialogi pochodzą z przestrzeni `Microsoft.Win32`, a nie z Windows Forms?
4. Jak przekazać dane do okna i odczytać je po zamknięciu?

Kryteria oceny

Kryterium	Punkty
Okno główne z menu, licznikiem i tytułem	4
Otwórz/Zapisz/Zapisz jako z obsługą błędów	6
Kontrola niezapisanych zmian	5
Okno O programie (modalne)	2
Tabela testów i pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Gra w kości w WPF

Czas: 2 × 45 min | praca indywidualna | Wykład 1.22–1.29

Cel: Zbudujesz aplikację okienkową gry w kości w stylu części desktopowej egzaminu INF.04, wykorzystując klasę `Kosc` z zadania 15 i obrazy kości.

Po zajęciach potrafisz:

- łączy klasę logiki z interfejsem WPF
- wyświetla i zmienia obrazy w kontrolce Image (Build Action: Resource)
- reaguje na kliknięcie obrazu i sygnalizuje stan przezroczystością
- kontroluje dostępność przycisków zależnie od stanu gry

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Przygotowanie przed zajęciami

- Przygotuj klasę `Kosc` z zadania 15 i pobierz obrazy `kosc0.png` – `kosc6.png` z folderu `pliki` przy tym zadaniu.

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.
- Rób przerwy dla oczu (zasada 20-20-20) — praca z projektantem okien wymaga długiego patrzenia w ekran.

Przebieg zadania

1 Projekt i zasoby

Utwórz projekt WPF `GraWKosci`.

1. Dodaj plik `Kosc.cs` z zadania 15.
2. Utwórz folder `Images`, dodaj obrazy i ustaw im **Akcja kompilacji: Resource**.
3. `kosc0.png` to kość przed rzutem (znak zapytania).

2 Okno

Układ okna według opisu.

1. Nagłówek i krótka instrukcja gry.
2. `UniformGrid` z pięcioma kontrolkami Image (`Img0`…`Img4`, Tag = numer kości).
3. Wiersz z wynikami: punkty w tym rzucie, wynik gry, numer rzutu („Rzut 2 z 3”).
4. Przyciski: Rzuć kośćmi, Zapisz punkty i nowa tura (nieaktywny na początku), Nowa gra.

3 Logika gry

Kod okna.

```
private void Odswiez()
{
    for (int i = 0; i < _kosci.Length; i++)
    {
        _obrazy[i].Source = new BitmapImage(
            new Uri($"Images/kosc{_kosci[i].Wartosc}.png", UriKind.Relative));
        _obrazy[i].Opacity = _kosci[i].Zablokowana ? 0.4 : 1.0;
    }
    // ... punkty, wynik, numer rzutu, IsEnabled przycisków
}
```

1. Tablica pięciu obiektów `Kosc` i tablica obrazów. W tytule okna pokaż `Kosc.LiczbaKosci`.
2. Rzuć: rzut wszystkimi kośćmi (zablokowane się nie zmieniają), zwiększ licznik rzutów w turze; po 3 rzutach przycisk nieaktywny.
3. Kliknięcie obrazu (`MouseLeftButtonDown`) blokuje lub odblokowuje kość — zablokowana ma `Opacity = 0.4`. Przed pierwszym rzutem kliknięcie nic nie robi.
4. Zapisz punkty: dodaj `Kosc.Punkty(...)` do wyniku gry, odblokuj kości, wyzeruj licznik rzutów.
5. Nowa gra: jeśli wynik > 0 — zapytaj o potwierdzenie.

4 Dokumentacja jak na egzaminie

Przygotuj materiały do oceny.

1. Komentarze `///` w kodzie okna i klasy Kosc.
2. Zrzuty: początek gry, kości po rzucie z dwiema zablokowanymi, wynik po kilku turach.
3. Zagraj 5 tur i zapisz wyniki w tabeli. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ kod okna (XAML i C#)
- ☐ trzy zrzuty z gry
- ☐ tabela 5 tur
- ☐ opis, jak klasa Kosc została użyta bez zmian

Pytania kontrolne

1. Dlaczego obrazy mają Akcję kompilacji Resource?
2. Jak program pilnuje, że w turze są najwyżej 3 rzuty?
3. Co by trzeba zmienić, aby gra miała 6 kości?
4. Dlaczego logikę punktacji można było przetestować bez uruchamiania okna?

Kryteria oceny

Kryterium	Punkty
Zasoby i układ okna	4
Rzut i wyświetlanie obrazów	4
Blokowanie kości z przezroczystością	4
Tury, wynik gry, dostępność przycisków	4
Dokumentacja i zrzuty	4
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt roczny — specyfikacja

Czas: 2 × 45 min

praca indywidualna lub w parach (za zgodą nauczyciela; w parze — podział zadań w specyfikacji) Wykład 1.30

Cel: Przygotujesz specyfikację własnej aplikacji WPF: wymagania, makiety okien, projekt klas, plan testów i harmonogram.

Po zajęciach potrafisz:

- formułuje sprawdzalne, ponumerowane wymagania
- projektuje makiety okien z nazwami kontrolek
- dzieli aplikację na klasy logiki i interfejs
- planuje testy i harmonogram prac

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0
- **Git for Windows** — lub Git wbudowany w Visual Studio git-scm.com/downloads/win
- **Narzędzie do makiet (dowolne): kartka, Paint, Figma lub Penpot** — Penpot — darmowy, open source penpot.app

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Temat i wymagania

Wybierz temat (przykłady w lekcji 1.30) i uzgodnij go z nauczycielem.

1. Zapisz 6–10 ponumerowanych wymagań funkcjonalnych (W1, W2...), każde sprawdzalne.
2. Wymagania obowiązkowe: co najmniej 2 okna, zapis i odczyt danych z pliku (CSV lub JSON), walidacja danych, klasy logiki z testami.
3. Dopisz 2–3 wymagania pozafunkcjonalne (np. minimalny rozmiar okna, polskie znaki, czas wczytania).

2 Makiety

Narysuj każde okno aplikacji.

1. Na makiecie podpisz nazwy kontrolek (TxtTytul, BtnDodaj...) i kontenery układu.
2. Zaznacz, które wymagania realizuje każde okno.
3. Opisz przejścia między oknami (co otwiera które okno, Show czy ShowDialog).

3 Projekt klas

Rozpisz klasy logiki.

1. Dla każdej klasy: nazwa, właściwości (z typami), metody (parametry, wynik, wyjątki).
2. Zaznacz, które klasy nie zależą od WPF (będą testowane).
3. Narysuj prosty diagram klas.

4 Testy i harmonogram

Plan pracy.

1. Zaplanuj co najmniej 5 testów jednostkowych: metoda, dane, oczekiwany wynik.
2. Rozpisz harmonogram na zadania 33 i 34 oraz pracę domową.
3. Utwórz repozytorium Git z `.gitignore`, folderem `docs` ze specyfikacją i pierwszym commitem.
4. Przedstaw specyfikację nauczycielowi do zatwierdzenia.

W sprawozdaniu umieść

- ☐ specyfikacja: temat, wymagania W1..., wymagania pozafunkcjonalne
- ☐ makiety okien
- ☐ projekt i diagram klas
- ☐ plan testów
- ☐ harmonogram
- ☐ adres lub zrzut repozytorium

Pytania kontrolne

1. Które z Twoich wymagań było najtrudniej sformułować w sposób sprawdzalny?
2. Które klasy będą testowane jednostkowo i dlaczego właśnie one?
3. Jakie ryzyko widzisz w harmonogramie i jak je ograniczysz?

Kryteria oceny

Kryterium	Punkty
Wymagania funkcjonalne i pozafunkcjonalne	6
Makiety z nazwami kontrolek	5
Projekt i diagram klas	4
Plan testów	3
Harmonogram i repozytorium	2
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt roczny — implementacja

Czas: 2 × 45 min | praca indywidualna | Wykład 1.30

Cel: Zrealizujesz aplikację WPF zgodnie ze specyfikacją z zadania 32, pracując małymi krokami z historią w Git, i przeprowadzisz przegląd kodu w parach.

Po zajęciach potrafisz:

- implementuje aplikację zgodnie z wymaganiami
- rozdziela logikę od interfejsu
- dokumentuje postęp commitami z opisowymi komunikatami
- przeprowadza i przyjmuje przegląd kodu

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Przygotowanie przed zajęciami

- Zatwierdzona specyfikacja z zadania 32 i repozytorium projektu.

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Klasy logiki

Zacznij od części bez interfejsu.

1. Zaimplementuj klasy logiki z projektu; walidacja danych w klasach (wyjątki lub komunikaty).
2. Zaimplementuj zapis i odczyt danych (JSON lub CSV) w osobnej klasie.
3. Sprawdź logikę krótkim programem konsolowym lub pierwszymi testami. Commit.

2 Okno główne

Interfejs według makiety.

1. Zbuduj układ okna kontenerami (bez sztywnych Margin).
2. Podłącz zdarzenia — code-behind tylko przekazuje dane do logiki i wyświetla wynik.
3. Dodaj style w App.xaml (zadanie 29).
4. Commit po każdym działającym wymaganiu, z numerem wymagania w opisie (np. „W3: filtr po gatunku”).

3 Kolejne okna i pliki

Pozostałe wymagania.

1. Okno dialogowe (np. dodawanie lub edycja elementu) otwierane ShowDialog z przekazywaniem danych.
2. Wczytywanie danych przy starcie i zapis przy zamykaniu lub na żądanie; obsługa błędów plików.
3. Uzupełnij tabelę realizacji wymagań.

4 Przegląd kodu w parach

Ostatnie 20 minut.

1. Zamień się z kolegą/koleżanką: przejrzyjcie nawzajem kod według listy: nazwy, podział logika/okno, obsługa błędów, powtórzenia kodu, komentarze.
2. Zapisz 3 uwagi otrzymane i 3 przekazane.
3. Popraw co najmniej jedną uwagę i zrób commit „Poprawki po przeglądzie”.

W sprawozdaniu umieść

- ☐ tabela realizacji wymagań
- ☐ historia commitów (`git log --oneline`)
- ☐ uwagi z przeglądu kodu i wprowadzone poprawki
- ☐ zrzuty działających okien

Pytania kontrolne

1. Które wymaganie zmieniłeś względem specyfikacji i dlaczego?
2. Która uwaga z przeglądu kodu była najbardziej przydatna?
3. Jak sprawdzasz, czy code-behind nie zawiera logiki, która powinna być w klasach?

Kryteria oceny

Kryterium	Punkty
Realizacja wymagań (wg tabeli)	8
Podział logika / interfejs	4
Historia commitów	3
Przegląd kodu i poprawki	3
Pytania	2
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt roczny — testy i prezentacja

Czas: 2 × 45 min | praca indywidualna | Wykład 1.30

Cel: Przetestujesz aplikację testami jednostkowymi i testami ręcznymi, przygotujesz dokumentację w stylu INF.04 i zaprezentujesz projekt.

Po zajęciach potrafisz:

- pisze testy jednostkowe logiki projektu
- przeprowadza test ręczny według scenariuszy z wymagań
- przygotowuje dokumentację kodu i instrukcję uruchomienia
- prezentuje działanie aplikacji i decyzje projektowe

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Testy jednostkowe

Projekt MSTest w rozwiązaniu projektu.

1. Zaimplementuj co najmniej 5 testów z planu z zadania 32 (w tym `[DataRow]` i test wyjątku).
2. Uruchom testy, popraw błędy w kodzie lub w testach, zrób zrzut Eksploratora testów.
3. Dopisz test dla błędu znalezionego podczas pracy (test regresji).

2 Test ręczny

Sprawdź aplikację z perspektywy użytkownika.

1. Dla każdego wymagania wykonaj scenariusz i zapisz wynik w tabeli.
2. Sprawdź złe dane, pusty plik danych, brak pliku, zmianę rozmiaru okna.
3. Popraw znalezione błędy; commit.

3 Dokumentacja

Jak na egzaminie.

1. Komentarze `///` przy klasach i publicznych metodach.
2. Plik `README.md`: opis, wymagania systemowe (.NET 10, Windows), sposób uruchomienia, autor.
3. Zrzuty ekranu wszystkich okien i wyniku testów w folderze `docs`.

4 Prezentacja

5 minut dla klasy.

1. Pokaż: problem, działanie aplikacji (najważniejsze wymagania), jedną ciekawą decyzję w kodzie, testy.
2. Odpowiedz na pytania klasy i nauczyciela.
3. Oddaj projekt: link do repozytorium lub archiwum bez folderów `bin` i `obj`.

W sprawozdaniu umieść

- ☐ lista testów jednostkowych i zrzut wyników
- ☐ tabela testów ręcznych
- ☐ README i dokumentacja kodu
- ☐ materiały prezentacji (opcjonalnie slajdy)

Pytania kontrolne

1. Jaki błąd znalazły testy, a jaki test ręczny?
2. Co zrobiłbyś inaczej, zaczynając projekt od nowa?
3. Która część aplikacji nadaje się do rozbudowy w roku 2 (np. MVVM, baza danych)?

Kryteria oceny

Kryterium	Punkty
Testy jednostkowe (min. 5, w tym regresji)	5
Test ręczny i poprawki	4
Dokumentacja (///, README, zrzuty)	5
Prezentacja i odpowiedzi	6
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).