

KOMPENDIUM

Programowanie obiektowe w C#

Kompletny kurs dla osoby, która zaczyna od zera

15 MIN	A / B / C / D	BLOKI KODU
strategia czasu	reguły i pułapki	uzupełnianie i śledzenie

Opracowano na podstawie ośmiu prezentacji kursowych oraz formatu zadań w lokalnej aplikacji egzaminacyjnej.

START

Jak korzystać z tego kursu

Najpierw zbuduj prosty obraz w głowie. Dopiero potem ucz się składni.

CEL

Po przerobieniu kursu masz umieć nie tylko rozpoznać poprawną odpowiedź, ale też **wykluczyć trzy błędne** i spokojnie przejść po kodzie linia po linii.

Trzy przejścia przez materiał

- 1. Obraz:** przeczytaj wyjaśnienie bez zapamiętywania szczegółów. Pytaj: kto ma dane i kto może ich użyć?
- 2. Reguła:** naucz się jednego zdania z ramki „Zapamiętaj”. To zdanie ma uruchamiać właściwy tok myślenia.
- 3. Zadanie:** zasłóń rozwiązanie, odpowiedź i dopiero potem sprawdź. Błąd jest informacją, którą regułę trzeba powtórzyć.

Plan minimum - 90 minut

0-20 min	Widoczność, klasa, obiekt, pole, metoda, właściwość	Rozumiesz słowa użyte w większości pytań
20-45 min	Konstruktory, dziedziczenie, abstract, sealed	Rozpoznajesz poprawne tworzenie obiektów i hierarchie
45-65 min	new kontra virtual/override, typ referencji i typ obiektu	Umiesz przewidywać wynik polimorfizmu
65-80 min	Interfejsy, bloki do uzupełniania, delegaty	Nie gubisz punktów na składni i stanie programu
80-90 min	Ściągą końcowa i próbny egzamin	Sprawdzasz gotowość pod presją czasu

WAŻNE

Nie próbuj zapamiętać całych programów. Na tym egzaminie wygrywa kilka stałych reguł: dostęp, kolejność konstruktorów, wybór metody, kolejność instrukcji i aktualna wartość zmiennej.

Spis treści

Układ od absolutnych podstaw do zadań egzaminacyjnych.

Jak korzystać z tego kursu	1
Spis treści	2
Anatomia 15-minutowego egzaminu	4
1. Obiektość bez technicznego żargonu	5
2. Typ wartościowy, typ referencyjny i new	6
3. Widoczność: public, protected, private	7
4. Enkapsulacja i właściwości	8
5. static: należy do typu, nie do obiektu	9
6. Konstruktory bez tajemnic	10
7. Kolejność konstruktorów i zadania z new	11
8. Dziedziczenie, abstract i sealed	12
9. Polimorfizm: typ referencji kontra typ obiektu	13
10. new kontra virtual/override	14
11. Śledzenie wywołań w hierarchii	15
12. Interfejsy i implementacja	16
13. Bloki do uzupełniania - gramatyka zadania	17
14. Delegaty: zmienna przechowująca metodę	18
15. Śledzenie stanu w zadaniach z delegatem	19
16. if, else if, argument domyślny i dzielenie	20
17. struct kontra class	21
18. Wyjątki, using i IDisposable	22
19. Kolekcje i typy generyczne	23
20. Operatory, indeksatory, konwersje i boxing	24
21. Zdarzenia, serializacja i metody Object	25

22. Jak rozpoznawać składowe klasy	26
23. Najczęstsze pułapki egzaminacyjne	27
24. Algorytmy rozwiązywania każdego typu zadania	28
25. Ściągą jednej strony	29
26. Trening A / B / C / D	30
27. Trening kodowy	31
28. Trening dopasowania i luk	33
29. Rozwiązania treningu - definicje	34
30. Rozwiązania treningu - kod	35
31. Rozwiązania treningu - dopasowanie i luki	36
32. Egzamin próbny - 15 minut	37
33. Egzamin próbny - ciąg dalszy	38
34. Egzamin próbny - bloki	40
35. Klucz egzaminu próbnego - 1-7	42
36. Klucz egzaminu próbnego - 8-12	43
37. Fiszki - szybkie aktywne przypominanie	44
38. Fiszki - składnia i śledzenie	45
39. Plan nauki na 5 dni	46
40. Gotowość do egzaminu	47

JAK CZYTAĆ

Rozdziały 1-12 to rdzeń egzaminu. Rozdziały 13-16 domykają zakres prezentacji. Potem przechodzisz do algorytmów rozwiązywania, treningu i egzaminu próbnego.

STRATEGIA

Anatomia 15-minutowego egzaminu

Krótki czas oznacza, że najpierw rozpoznajesz typ zadania, a dopiero potem rozwiązujesz.

A / B / C / D	Jedną precyzyjną regułę	Podkreśl słowa: public, private, static, abstract, virtual
Wynik liczbowy	Kolejność wykonania i zmianę stanu	Zapisz wartość początkową i rób tabelkę
Dopasowanie	Rozpoznanie rodzaju składowej	Czytaj od nazwy typu do nazwy zmiennej
Bloki do luk	Gramatykę hierarchii i implementacji	Najpierw ustal rolę luki: typ, słowo kluczowe czy identyfikator

Podział 15 minut

0:00-0:40	Szybki skan. Zaznacz zadania łatwe, kodowe i niepewne.
0:40-5:30	Rozwiąż definicje i widoczność. To najszybsze punkty.
5:30-10:30	Zrób konstruktory, dziedziczenie i polimorfizm.
10:30-13:30	Wypełnij bloki oraz policz delegaty w tabeli.
13:30-15:00	Sprawdź tylko miejsca oznaczone znakiem zapytania. Nie zmieniaj odpowiedzi bez konkretnej reguły.

SŁOWA ALARMOWE

zawsze, tylko, wszystkie, nigdy - odpowiedź bywa zbyt szeroka. Szukaj wyjątku.

SŁOWA DECYDUJĄCE

bezpośrednio, z zewnątrz, klasa potomna, przez referencję - często zmieniają prawidłową odpowiedź.

REGUŁA 20 SEKUND

Jeśli po 20 sekundach nie wiesz, **zaznacz zadanie i idź dalej**. Jedno trudne śledzenie kodu nie może zabrać czasu na cztery proste definicje.

1. Obiektość bez technicznego żargonu

Program obiektowy opisuje rzeczy, ich dane i zachowania.

Najprostszy obraz

KLASA = PROJEKT

Opisuje, jakie dane i działania będzie miał obiekt. Jak formularz, który mówi, jakie pola mają istnieć.

OBIEKT = KONKRETNY EGZEMPLARZ

Powstaje na podstawie klasy. Dwa obiekty tej samej klasy mogą przechowywać inne wartości.

ANATOMIA KLASY

```
class Konto
{
    private decimal saldo;        // pole - dane obiektu

    public Konto(decimal start)    // konstruktor - start obiektu
    {
        saldo = start;
    }

    public void Wplac(decimal x)   // metoda - zachowanie
    {
        saldo += x;
    }

    public decimal Saldo           // właściwość - kontrolowany dostęp
    {
        get { return saldo; }
    }
}

Konto moje = new Konto(100);     // obiekt
```

Cztery filary w jednym zdaniu

Abstrakcja	Pokazuję to, co ważne, ukrywam niepotrzebne szczegóły	abstract, interface
Enkapsulacja	Chronię dane i daję kontrolowany dostęp	private pole + public właściwość
Dziedziczenie	Nowy typ korzysta z cech typu bazowego	class B : A
Polimorfizm	To samo wywołanie może uruchomić inną wersję metody	virtual + override

KOTWICA

Klasa opisuje, obiekt istnieje. Pole pamięta, metoda robi, właściwość pilnuje dostępu, konstruktor rozpoczyna życie obiektu.

PODSTAWY

2. Typ wartościowy, typ referencyjny i new

Najczęstszy błąd początkującego: mylenie obiektu z adresem prowadzącym do obiektu.

Wartościowy	Samą wartość	Powstaje kopia wartości	int, double, bool, char, struct
Referencyjny	Referencję do obiektu	Kopiuwana jest referencja - obie zmienne wskazują ten sam obiekt	class, array, string, delegate

KOPIA WARTOŚCI KONTRA KOPIA REFERENCJI

```
int a = 5;
int b = a;
b = 9;           // a nadal 5

Konto x = new Konto(100);
Konto y = x;
y.Wplac(50);     // x i y wskazują ten sam obiekt
```

Co robi new?

- rezerwuje miejsce na nowy obiekt,
- ustawia pola na wartości domyślne,
- uruchamia odpowiedni konstruktor,
- zwraca referencję, którą zapisujesz w zmiennej.

TYLKO DEKLARACJA

Konto k;
Masz zmienną, ale nie masz jeszcze obiektu. Nie wolno wywołać metody na niezainicjalizowanej zmiennej.

UTWORZENIE OBIEKTU

Konto k = new Konto(100);
Powstaje obiekt i jego konstruktor otrzymuje argument 100.

PYTANIE KONTROLNE

Czy przypisanie tworzy nowy obiekt? **Nie**, jeśli nie ma `new`. Przy klasach najczęściej tylko kopiuje referencję.

3. Widoczność: public, protected, private

Wyobraź sobie trzy strefy dostępu: świat, rodzina i wewnątrz klasy.

public	TAK	TAK	TAK	dla wszystkich
protected	TAK	TAK	NIE	dla rodziny
private	TAK	NIE bezpośrednio	NIE	tylko u siebie

DOMYŚLNIE W CLASS

Pole lub metoda bez modyfikatora dostępu jest **private**.

DOMYŚLNIE W STRUCT

Pole lub metoda bez modyfikatora dostępu także jest **private**.

DOSTĘP W HIERARCHII

```
class A
{
    public int p;        // widoczne wszędzie
    protected int r;    // A i klasy potomne
    private int v;      // tylko kod klasy A
    int x;              // też private
}

class B : A
{
    void Test()
    {
        p = 1;          // OK
        r = 2;          // OK
        // v = 3;       // B nie ma bezpośredniego dostępu
    }
}
```

PUŁAPKA Z ARKUSZA

Prywatna metoda klasy bazowej jest częścią implementacji obiektu, lecz kod klasy potomnej **nie może wywołać jej bezpośrednio**. Na pytanie w formule arkusza wybierasz: „tak, ale nie można bezpośrednio korzystać”.

SKRÓT

PU-PRO-PRI: public - wszyscy, protected - potomni, private - tylko klasa deklarująca.

4. Enkapsulacja i właściwości

Pole przechowuje dane. Właściwość kontroluje sposób ich odczytu i zapisu.

Dlaczego nie publiczne pole?

Jeśli każdy może dowolnie zmienić pole, obiekt łatwo wprowadzić w niepoprawny stan. Enkapsulacja ukrywa pole jako `private` i wystawia bezpieczną bramkę jako właściwość.

POLE + WŁAŚCIWOŚĆ

```
class Osoba
{
    private int wiek;           // pole zapasowe

    public int Wiek             // właściwość
    {
        get { return wiek; }    // odczyt
        set
        {
            if (value >= 0)      // value = przekazywana wartość
                wiek = value;
        }
    }
}

Osoba o = new Osoba();
o.Wiek = 20;                    // uruchamia set
int x = o.Wiek;                 // uruchamia get
```

get	zwraca wartość	odczyt: x = obiekt.Cena
set	ustawia wartość	zapis: obiekt.Cena = 50
value	wartość po prawej stronie przypisania	w set: cena = value
{ get; set; }	właściwość automatyczna	kompilator tworzy ukryte pole
{ get; private set; }	odczyt publiczny, zapis tylko w klasie	dobry sposób ochrony stanu

NAJPOPULARNIEJSZY BŁĄD

`get { return Wiek; }` odwołuje się znów do tej samej właściwości i tworzy nieskończoną rekurencję. Getter ma zwracać **pole zapasowe**, np. `wiek`.

5. static: należy do typu, nie do obiektu

Element statyczny istnieje w jednej wspólnej kopii.

INSTANCYJNE

Każdy obiekt ma własną kopię pola. Wywołujesz przez obiekt: `k.Saldo`.

STATYCZNE

Jedna wspólna kopia dla całej klasy. Wywołujesz przez typ: `Konto.Liczba`.

JEDNA KOPIA WSPÓLNA

```
class Licznik
{
    public static int Ile { get; private set; }
    public int Numer { get; private set; }

    static Licznik()           // konstruktor statyczny
    {
        Ile = 0;
    }

    public Licznik()
    {
        Ile++;
        Numer = Ile;
    }
}

Licznik a = new Licznik();    // a.Numer = 1
Licznik b = new Licznik();    // b.Numer = 2, Licznik.Ile = 2
```

Konstruktor statyczny - cztery fakty

- ma nazwę klasy i słowo `static`,
- nie ma parametrów ani modyfikatora dostępu,
- uruchamia się automatycznie najwyżej raz, przed pierwszym użyciem typu,
- inicjalizuje część wspólną, nie konkretny obiekt.

TEST 3 PYTAŃ

1) Czy potrzebuję konkretnego obiektu? 2) Czy każdy obiekt ma mieć inną wartość? 3) Czy metoda używa `this`? Jeśli odpowiedzi brzmią NIE, NIE, NIE - element może być statyczny.

KOTWICA

static = wspólne. Metoda statyczna nie ma `this` i bez obiektu nie może bezpośrednio używać pól instancyjnych.

6. Konstruktory bez tajemnic

Konstruktor nie zwraca wartości - przygotowuje nowy obiekt.

Nazwa taka jak klasa	<code>public A(...), nigdy void A(...)</code>
Brak typu zwrótnego	Wpisanie typu zmienia konstruktor w zwykłą metodę
Można przeciążać	Kilka konstruktorów różni się listą parametrów
Automatyczny bezparametrowy	Powstaje tylko wtedy, gdy nie napisano żadnego własnego konstruktora
Parametr domyślny	Można pominąć argument; użyta zostanie wartość po znaku =
Argument nazwany	Można jawnie wskazać nazwę parametru, np. <code>wiek: 20</code>

PRZECIĄŻENIA, DOMYŚLNE ARGUMENTY, THIS

```
class Punkt
{
    public int X, Y;

    public Punkt() : this(0, 0) { }

    public Punkt(int x, int y = 5)
    {
        X = x;
        Y = y;
    }
}

new Punkt();           // (0, 0)
new Punkt(3);          // (3, 5)
new Punkt(3, 7);       // (3, 7)
new Punkt(y: 4, x: 2); // (2, 4)
```

PUŁAPKA AUTOMATYCZNEGO KONSTRUKTORA

Gdy napiszesz choć jeden własny konstruktor, kompilator nie dopisze już publicznego konstruktora bez parametrów. Jeśli nadal go chcesz, musisz napisać go samodzielnie.

this(...) PO LEWEJ

Wywołuje inny konstruktor **tej samej klasy**. Wykona się przed ciałem bieżącego konstruktora.

base(...) PO PRAWEJ

Wywołuje konstruktor **klasy bazowej**. Część bazowa powstaje przed częścią potomną.

7. Kolejność konstruktorów i zadania z new

Rysuj drzewo wywołań od najbardziej zewnętrznego new do środka.

Kolejność w dziedziczeniu

NAJPIERW BAZA

```
class A
{
    public A() { Console.Write(""A""); }
}

class B : A
{
    public B() { Console.Write(""B""); }
}

class C : B
{
    public C() { Console.Write(""C""); }
}

new C();           // ABC
```

KOTWICA

Budowa domu zaczyna się od fundamentu. **Najpierw konstruktor bazowy, potem potomny.**

Algorytm dla zagnieżdżonych konstruktorów

1. Zakreśl każdy fragment `new Typ(...)`.
2. Najpierw policz argumenty wewnętrzne, bo muszą powstać przed obiektem zewnętrznym.
3. Dopasuj liczbę i typ argumentów do konkretnego przeciążenia konstruktora.
4. Jeśli konstruktor ma `: this(...)` lub `: base(...)`, dopisz to wywołanie przed jego ciałem.
5. Zapisuj wynik po każdym ciele konstruktora, nie próbuj liczyć całości w pamięci.

MINI

Które wywołanie jest niepoprawne?

- A. `new A();` gdy klasa nie ma żadnego konstruktora
- B. `new A();` gdy klasa ma tylko `A(int x)`
- C. `new A(4);` gdy klasa ma `A(int x)`
- D. `new A();` gdy klasa jawnie ma `A()`

ODPOWIEDŹ	B
Dlaczego?	Własny konstruktor <code>A(int)</code> usuwa automatyczny bezparametrowy. Pozostałe wywołania mają pasujący konstruktor.
Kotwica pamięci	Własny konstruktor? Sprawdź, czy <code>A()</code> nadal istnieje jawnie.

8. Dziedziczenie, abstract i sealed

Dziedziczenie opisuje relację „jest rodzajem”, a nie „ma”.

IS-A - DZIEDZICZENIE

Pies **jest** zwierzęciem. `class Pies : Zwierze.`

HAS-A - KOMPOZYCJA

Samochód **ma** silnik. Silnik jest polem wewnątrz klasy Samochód.

abstract class	niepełny wzorzec dla potomków	NIE	TAK
zwykła class	pełny typ	TAK	TAK, jeśli nie sealed
sealed class	typ zamknięty na dziedziczenie	TAK	NIE

KLASA ABSTRAKCYJNA

```
abstract class Figura
{
    public abstract double Pole(); // bez ciała
    public void Opisz() { }        // metoda zwykła może mieć ciało
}

class Kolo : Figura
{
    public override double Pole()
    {
        return 3.14 * 2 * 2;
    }
}

Figura f = new Kolo();           // OK
// Figura x = new Figura();      // BŁĄD
```

DWA OBOWIĄZKI

Metoda `abstract` nie ma ciała. Pierwsza konkretna klasa potomna musi ją zaimplementować słowem `override`, chyba że sama także jest abstrakcyjna.

NIEDOZWOLONE POŁĄCZENIE

Element nie może być jednocześnie `virtual` i `abstract`. Metoda abstrakcyjna już z definicji oczekuje implementacji w potomku.

9. Polimorfizm: typ referencji kontra typ obiektu

To najważniejszy rozdział do zadań z wynikiem programu.

DWIE INFORMACJE W JEDNEJ LINII

```
A x = new B();
// ^      ^
// typ    rzeczywisty
// referencji typ obiektu
```

zwykła / ukryta przez new	typ referencji po lewej stronie	Jak zadeklarowano zmienną?
virtual / override	rzeczywisty typ obiektu po prawej stronie new	Jaki obiekt naprawdę utworzono?

ZDANIE, KTÓRE DAJE PUNKTY

new patrzy na etykietę zmiennej, override patrzy na obiekt.

TYP REFERENCJI

To typ przed nazwą zmiennej, np. `A x`. Określa, co kompilator pozwala wywołać i wybiera metody niepolimorficzne.

TYP OBIEKTU

To typ po `new`, np. `new B()`. Wybiera najbardziej szczegółowe `override`.

Rzutowanie nie zmienia obiektu

Zmiana sposobu patrzenia na referencję nie przebudowuje obiektu. `A x = new B()` nadal wskazuje obiekt B. Rzutowanie może tylko odsłonić członki dostępne dla innego typu referencji.

10. new kontra virtual/override

Dwa prawie podobne mechanizmy dają inny wynik.

Ukrywanie przez new

NEW = WYBÓR STATYCZNY

```
class A
{
    public void M() { Console.WriteLine("A"); }
}
class B : A
{
    public new void M() { Console.WriteLine("B"); }
}

A x = new B();
B y = new B();
x.M();           // A - patrzymy na typ referencji
y.M();           // B
```

Nadpisywanie przez override

OVERRIDE = WYBÓR DYNAMICZNY

```
class A
{
    public virtual void M() { Console.WriteLine("A"); }
}
class B : A
{
    public override void M() { Console.WriteLine("B"); }
}

A x = new B();
x.M();           // B - patrzymy na rzeczywisty obiekt
```

A x	new A()	A	A
A x	new B()	A	B
B x	new B()	B	B

PULAPKA

Samo istnienie metody o tej samej nazwie w potomku nie wystarcza. Dla polimorfizmu baza musi mieć `virtual` lub `abstract`, a potomek `override`.

11. Śledzenie wywołań w hierarchii

W trudnym zadaniu oddziel wybór metody od instrukcji wykonywanych wewnątrz niej.

Algorytm 4 kroków

1. Nad każdą zmienną zapisz: **typ referencji / typ obiektu**, np. A/B.
2. Przy wywołanej metodzie zaznacz: zwykła, new czy virtual/override.
3. Wybierz pierwsze ciało: dla new według referencji, dla override według obiektu.
4. Dopiero potem wejdź do środka. Jeśli występuje `base.M()`, dopisz wykonanie wersji bazowej dokładnie w tym miejscu.

PRZEJDŹ OD ZEWNĄTRZ DO BAZY

```
class A
{
    public virtual int M() { return 1; }
}
class B : A
{
    public override int M() { return base.M() + 2; }
}
class C : B
{
    public override int M() { return base.M() * 3; }
}

A x = new C();
Console.WriteLine(x.M());
```

1	x.M() wybiera C.M(), bo metoda jest wirtualna, a obiekt to C	jeszcze brak
2	C.M() potrzebuje base.M(), więc uruchamia B.M()	jeszcze brak
3	B.M() potrzebuje A.M()	A.M() zwraca 1
4	B dodaje 2	3
5	C mnoży przez 3	9

ODPOWIEDŹ

Program wypisze **9**. Zapisuj wynik przy każdym powrocie z metody - tak samo jak śledzi się nawiasy od środka.

12. Interfejsy i implementacja

Interfejs mówi „co trzeba umieć”, a klasa mówi „jak to zrobić”.

Rola	kontrakt	wspólna baza z kodem i stanem
Pola	nie	tak
Implementacja metod	nie w klasycznym zakresie prezentacji	może zawierać metody zwykłe
Wiele naraz	klasa może implementować wiele	klasa dziedziczy po jednej klasie
Słowo w klasie potomnej	bez override	override dla abstract/virtual

BAZA + INTERFEJS

```
public interface IRuch
{
    int Predkosc { get; set; }
    void Jedz();
}

abstract class Pojazd
{
    public abstract void Start();
}

class Rower : Pojazd, IRuch
{
    public int Predkosc { get; set; }
    public override void Start() { }
    public void Jedz() { }
}

// kolejność: jedna klasa bazowa, potem interfejsy
```

Lista kontrolna implementacji

- Po dwukropku najpierw wpisz klasę bazową, a potem interfejsy.
- Przy metodzie abstrakcyjnej z klasy bazowej użyj `override`.
- Przy metodzie z interfejsu nie dopisuj `override`.
- W klasycznej implementacji egzaminacyjnej człon interfejsu zaimplementuj publicznie.
- Nazwa, typ zwracany, parametry i akcesory właściwości muszą pasować do kontraktu.

KOTWICA

Klasa bazowa daje korzenie, interfejs daje obietnice. Korzeń jest jeden, obietnic może być wiele.

13. Bloki do uzupełniania - gramatyka zadania

Nie zgaduj słowa. Najpierw ustal, jakiej kategorii brakuje w danym miejscu.

WZORZEC ZADANIA

```
abstract class Baza
{
    public abstract int A();
    protected abstract char B();
    public abstract uint C { get; set; }
}

interface ITest
{
    int D();
    int E { get; set; }
}

class Potomna : [[1]], [[2]]
{
    public [[3]] int A() { return 1; }
    protected [[4]] char B() { return &#x27;b&#x27;; }
    public int D() { return 1; }
    public override uint C
    {
        get { return c; }
        set { c = value; }
    }
    public int E
    {
        get { return e; }
        set { e = value; }
    }
}
```

[[1]]	Jaka klasa jest bazą?	Baza
[[2]]	Jaki kontrakt implementujemy?	ITest
[[3]]	A jest abstract w bazie - czym staje się implementacja?	override
[[4]]	B jest abstract w bazie - czym staje się implementacja?	override

Kolejność rozwiązywania bloków

1. Najpierw linia po dwukropku: **baza przed interfejsem**.
2. Potem metody z `abstract` w bazie: w potomku `override`.
3. Następnie kontrakt interfejsu: zgodna sygnatura i dostęp publiczny, ale bez `override`.
4. Na końcu `get/set`: getter zwraca pole, setter przypisuje `pole = value`.

FILTR SKŁADNIOWY

Po dwukropku nie może pojawić się `override`, bo tam oczekiwany jest typ. Między `public` a typem zwracanym nie może pojawić się nazwa klasy, bo tam oczekiwane jest słowo kluczowe.

14. Delegaty: zmienna przechowująca metodę

Delegat to bezpieczna referencja do metody o określonej sygnaturze.

Sygnatura musi pasować

DEKLARACJA, PRZYPISANIE, WYWOŁANIE

```
public delegate int Operacja(int x);

static int Podwoj(int x)
{
    return x * 2;
}

Operacja op = Podwoj;
int wynik = op(4);           // 8
```

int przed nazwą delegata	typ wyniku metody
(int x)	lista parametrów, która musi pasować
op = Podwoj	zapis referencji do metody, bez nawiasów
op(4)	wywołanie metody przez delegat
+=	dodanie kolejnej metody do listy wywołań
-=	usunięcie metody z listy wywołań

MULTICAST

Jeśli delegat ma kilka metod, zostaną wykonane w kolejności dodania. Gdy delegat coś zwraca, na zewnątrz otrzymasz wynik **ostatniej** wykonanej metody, choć skutki uboczne wszystkich wcześniejszych pozostają.

Gotowe delegaty generyczne

Action	przyjmuje dane, nic nie zwraca	Action
Func	ostatni typ to wynik	Func
Predicate	przyjmuje T i zwraca bool	Predicate

KOTWICA

Delegat = pilot do metody. Pilot działa tylko z urządzeniem o zgodnym „kształcie”: ten sam typ wyniku i parametry.

15. Śledzenie stanu w zadaniach z delegatem

Jedna tabelka usuwa większość pomyłek z pamięci roboczej.

PRZYKŁADOWY STAN

```
int Result = 8;

void Add(int x = 1)
{
    Result += x;
}

int Run(int arg)
{
    Add();
    if (arg > 1) Add(arg);
    else if (arg == 1) Add(2);
    return Result--;
}
```

Przykład dla arg = 2

start	wartość początkowa	-	8	-
Add()	domyślnie x = 1	8	9	-
arg > 1	warunek prawdziwy	9	9	-
Add(2)	dodajemy 2	9	11	-
return Result--	najpierw zwróć, potem zmniejsz	11	10	11

x++	użyj starej wartości	zwiększ x	najpierw oddaj, potem dodaj
++x	zwiększ x	użyj nowej wartości	najpierw dodaj, potem oddaj
x--	użyj starej wartości	zmniejsz x	post = po użyciu
--x	zmniejsz x	użyj nowej wartości	pre = przed użyciem

NIE LICZ W GŁOWIE

Zapisuj **stan przed i po każdej instrukcji**. W wariantach z kilkoma wywołaniami wartości zwykle przechodzą między pytaniami, więc nowy test nie zawsze startuje od 8.

16. if, else if, argument domyślny i dzielenie

Małe symbole decydują o całym wyniku.

DWA NIEZALEŻNE IF

Każdy warunek jest sprawdzany. Jeśli oba są prawdziwe, wykonają się oba bloki.

IF + ELSE IF

Po pierwszym prawdziwym warunku pozostała część łańcucha jest pomijana.

NIEZALEŻNE KONTRA ŁAŃCUCH

```
int x = 3;

if (x > 0) wynik += 10;
if (x > 2) wynik += 20;      // oba if: razem +30

if (x > 0) wynik += 10;
else if (x > 2) wynik += 20; // tylko pierwszy blok: +10
```

Trzy drobne reguły

Add() przy Add(int x = 1)	x otrzymuje wartość 1	brak argumentu nie oznacza zera
5 / 2 dla int	wynik 2	część ułamkowa jest odcinana
5.0 / 2 lub (double)5 / 2	wynik 2.5	wystarczy jeden operand zmiennoprzecinkowy
return x++	zwracana jest stara wartość	x po instrukcji jest większe
return ++x	zwracana jest nowa wartość	zwiększenie następuje przed return

SKAN PRZED LICZENIEM

Zakreśl wszystkie: ++, --, argumenty domyślne, osobne if oraz else if. To są przełączniki wyniku.

17. struct kontra class

Struktura jest typem wartościowym; klasa jest typem referencyjnym.

Rodzaj typu	wartościowy	referencyjny
Przypisanie	kopiuje dane	kopiuje referencję
Domyślny dostęp pól/metod	private	private
protected	nie ma sensu - brak dziedziczenia po strukturze	tak
Dziedziczenie po własnym typie	nie	tak
Implementacja interfejsów	tak	tak
Człony static	tak	tak

KOPIOWANIE STRUKTURY

```
struct Punkt
{
    public int X;
}

Punkt a = new Punkt();
a.X = 5;
Punkt b = a;
b.X = 9;

// a.X nadal 5, bo b jest kopią wartości
```

WERSJA POD MATERIAŁ

W zakresie prezentacji przyjmij klasyczną regułę: struktura ma dostępny publiczny konstruktor bezparametrowy ustawiający wartości domyślne, nie dziedziczy po klasach ani innych strukturach, ale może implementować interfejsy.

KOTWICA

struct kopiuje kartkę, **class** kopiuje adres do jednej kartki.

18. Wyjątki, using i IDisposable

Wyjątek sygnalizuje sytuację, której zwykły przebieg programu nie potrafi obsłużyć.

TRY - CATCH - FINALLY

```
try
{
    int x = int.Parse(""abc"");
}
catch (FormatException ex)
{
    Console.WriteLine(""Zły format"");
}
finally
{
    Console.WriteLine(""Koniec próby"");
}
```

try	kod, w którym może pojawić się wyjątek
catch	obsługa określonego typu wyjątku
throw	jawne zgłoszenie wyjątku
finally	kod wykonywany niezależnie od powodzenia, np. sprzątanie
Exception	bazowy typ wielu wyjątków

USING I ZWALNIANIE ZASOBU

```
using (Plik p = new Plik())
{
    p.Zapisz();
} // tutaj automatycznie p.Dispose()
```

PROSTY OBRAZ

Pamięć zarządzana jest sprzątana przez garbage collector, ale plik, połączenie lub uchwyt trzeba zamknąć przewidywalnie. `using` gwarantuje wywołanie `Dispose()` dla obiektu implementującego `IDisposable`.

19. Kolekcje i typy generyczne

Temat poboczny na tym formacie, ale łatwy do zdobycia, gdy rozumiesz nazwę operacji.

ArrayList	lista bez kontroli typu	Add	Remove / RemoveAt	przechowuje object, wymaga rzutowań
Stack	stos talerzy	Push	Pop	LIFO - ostatni wchodzi, pierwszy wychodzi
Queue	kolejka ludzi	Enqueue	Dequeue	FIFO - pierwszy wchodzi, pierwszy wychodzi
List	lista jednego typu	Add	Remove / indeks	bezpieczna typowo

T = TYP PODSTAWIONY PÓŹNIEJ

```
static void Zamien<T>(ref T a, ref T b)
{
    T temp = a;
    a = b;
    b = temp;
}

List<int> liczby = new List<int>();
liczby.Add(5);
```

where T : class	T ma być typem referencyjnym
where T : struct	T ma być typem wartościowym
where T : ITest	T ma implementować interfejs ITest
where T : new()	T ma mieć publiczny konstruktor bez parametrów

DLACZEGO GENERICS

Typ generyczny daje kontrolę kompilatora, mniej rzutowań i zwykle mniej boxing/unboxing. `List<int>` przyjmie liczby całkowite, ale odrzuci napis.

20. Operatory, indeksatory, konwersje i boxing

Cztery krótkie tematy, które rozpoznasz po charakterystycznej składni.

Przeciążanie operatorów

OPERATOR JAK METODA STATYCZNA

```
public static Punkt operator +(Punkt a, Punkt b)
{
    return new Punkt(a.X + b.X, a.Y + b.Y);
}
```

Operator jest statyczny i co najmniej jeden operand musi mieć typ klasy lub struktury, która go definiuje. Operatory porównania występują parami: `==`, `z !=`, `< z >`, `<= z >=`.

Indeksator

THIS Z NAWIASAMI KWADRATOWYMI

```
public int this[int index]
{
    get { return dane[index]; }
    set { dane[index] = value; }
}

magazyn[0] = 12;
```

Konwersje i boxing

implicit	konwersja bezpieczna automatyczna	int do double
explicit	wymaga jawnego rzutowania, może stracić dane	(int)3.9 daje 3
boxing	wartość trafia do object	object o = 5
unboxing	object wraca do dokładnego typu wartościowego	int x = (int)o

PUŁAPKA UNBOXING

Najpierw trzeba odpakować dokładnie do typu, który został zapakowany. Jeśli w `object` jest `int`, bezpośrednie `(double)o` jest błędne.

ZAKRES

21. Zdarzenia, serializacja i metody Object

Wystarczy rozumieć ich rolę i charakterystyczne słowa.

event	obiekt ogłasza, że coś się stało; inni mogą się zapisać	delegate, event, +=, handler
serializacja	zamiana stanu obiektu na formę do zapisu lub przesłania	zapis/odczyt, strumień, odtworzenie obiektu
ToString	tekstowa reprezentacja obiektu	Console.WriteLine(obiekt)
Equals	logiczne porównanie obiektów	czy obiekty znaczą to samo
GetHashCode	liczba używana w kolekcjach haszujących	równe obiekty muszą mieć ten sam hash

ZDARZENIE OPARTE NA DELEGACIE

```
public delegate void AlarmHandler(string komunikat);
public event AlarmHandler Alarm;

void Uruchom()
{
    if (Alarm != null)
        Alarm("Start");
}
```

WAŻNA PARA

Jeśli nadpisujesz `Equals`, powinieneś spójnie nadpisać także `GetHashCode`. Dwa obiekty uznane za równe muszą zwracać ten sam kod skrótu.

PRIORYTET NA 15 MINUT

Te zagadnienia zwykle dają pytania definicyjne. Nie poświęcaj im więcej czasu niż polimorfizmowi, konstruktorom i blokom kodu.

22. Jak rozpoznawać składowe klasy

Czytaj deklarację warstwami od lewej do prawej.

<code>public static int v7 { get; set; }</code>	public	static	właściwość
<code>protected static int a3;</code>	protected	static	pole
<code>static int a1;</code>	private domyślnie	static	pole
<code>public static int a2;</code>	public	static	pole
<code>public int v4 { get; set; }</code>	public	instancyjna	właściwość
<code>public int M() { return 1; }</code>	public	instancyjna	metoda

Skan w czterech pytaniach

1. Czy na początku jest modyfikator dostępu? Jeśli nie, człon klasy jest domyślnie `private`.
2. Czy występuje `static`? Jeśli tak, składowa należy do typu.
3. Co jest po nazwie? Średnik oznacza pole, nawiasy okrągłe metodę, klamry z `get/set` właściwość.
4. Czy nazwa jest taka jak klasa i brak typu zwrotnego? To konstruktor.

POLE

Kończy się średnikiem: `int x;`. Przechowuje dane.

WŁAŚCIWOŚĆ

Ma `get` lub `set`: `int X { get; set; }`. Kontroluje dostęp.

KOTWICA

Średnik - magazyn. Nawiasy - działanie. `get/set` - bramka.

23. Najczęstsze pułapki egzaminacyjne

To lista rzeczy, które wydają się niewinnie, ale zmieniają odpowiedź.

1. Brak modyfikatora Człon class lub struct jest private, nie public.	2. Własny konstruktor Automatyczny publiczny A() znika w klasie.
3. private w bazie Potomek nie ma dostępu bezpośredniego, choć część bazowa obiektu istnieje.	4. abstract class Nie wolno utworzyć jej obiektu, ale wolno mieć referencję do konkretnego potomka.
5. interface Implementacja klasyczna jest publiczna i nie używa override.	6. new Wybór metody według typu referencji.
7. override Wybór metody według rzeczywistego obiektu.	8. base.M() Jawnie uruchamia wersję z klasy bazowej w tym miejscu.
9. x++ Wyrażenie dostaje starą wartość; zmienna rośnie dopiero potem.	10. ++x Zmienna rośnie przed użyciem w wyrażeniu.
11. if + if Oba warunki mogą wykonać kod.	12. if + else if Wykona się najwyżej jedna gałąź łańcucha.
13. int / int Wynik jest całkowity, część ułamkowa znika.	14. struct Przypisanie kopiuje wartość, nie referencję.
15. get { return X; } Jeśli X to ta sama właściwość, powstaje rekurencja; trzeba zwrócić pole.	16. delegat multicast Wszystkie metody się wykonują, ale zwracany wynik pochodzi z ostatniej.

ZASADA ZMIANY ODPOWIEDZI

Nie zmieniaj zaznaczenia dlatego, że „B brzmi mądrzej”. Zmień je tylko wtedy, gdy potrafisz wypowiedzieć regułę, która obala wcześniejszy wybór.

24. Algorytmy rozwiązywania każdego typu zadania

Procedura jest szybsza i pewniejsza niż intuicja.

A / B / C / D

1. Nazwij temat jednym słowem: dostęp, konstruktor, interfejs, polimorfizm, struct.
2. Wylap słowo decydujące: bezpośrednio, z zewnątrz, static, new, override.
3. Powiedz regułę bez patrzenia na odpowiedzi.
4. Skreśl odpowiedzi sprzeczne z regułą; dopiero potem wybierz.

Wynik programu

1. Zapisz stan początkowy każdej zmiennej.
2. Zapisz kolejność konstruktorów lub wywołanych metod.
3. Po każdej instrukcji wpisz nowy stan w osobnym wierszu.
4. Rozdziel wartość zwracaną od stanu po ++ lub --.

Dopasowanie

1. Najpierw rozpoznaj pole / metodę / właściwość / konstruktor.
2. Potem ustal public / protected / private.
3. Na końcu sprawdź static / instancyjne.

Bloki do luk

1. Określ kategorię gramatyczną każdej luki.
2. Uzupełnij hierarchię typów, następnie słowa override, na końcu pola i value.

MANTRA

Temat - reguła - tabela - odpowiedź. To cztery kroki, które warto przećwiczyć aż staną się automatyczne.

25. Ściąga jednej strony

Powtórz ją tuż przed wejściem na egzamin.

public	wszędzie	private	tylko klasa deklarująca
protected	klasa + potomni	brak modyfikatora członu	private
static	jedna kopia dla typu	instancyjne	osobna kopia w obiekcie
new Obiekt()	tworzy obiekt	samo przypisanie klasy	kopiuje referencję
abstract class	nie tworzysz obiektu	sealed class	nie wolno dziedziczyć
abstract method	bez ciała	override	implementuje/nadpisuje
new metoda	typ referencji	virtual/override	typ obiektu
this(...) konstruktor	ta sama klasa	base(...)	klasa bazowa
interfejs	kontrakt, wiele	klasa bazowa	jedna, przed interfejsami
get	zwraca pole	set	pole = value
x++	oddaj stare, potem +1	++x	+1, potem oddaj
if + if	możliwe dwa bloki	if + else if	najwyżej jeden blok
struct	kopiuwana wartość	class	kopiuwana referencja
Stack	LIFO, Push/Pop	Queue	FIFO, Enqueue/Dequeue
Action	brak wyniku	Func	ostatni typ to wynik
boxing	wartość do object	unboxing	dokładny typ z object

NAJGŁOŚNIEJSZA REGUŁA

Gdy widzisz `A x = new B()`: zwykła lub new metoda wybiera A, a virtual/override wybiera B.

26. Trening A / B / C / D

Najpierw rozwiąż bez zaglądania do odpowiedzi na następnych stronach.

1 Pola publiczne są dostępne:

- A. tylko w klasie deklarującej
- B. w klasie, klasach potomnych i z zewnątrz
- C. w klasie i klasach potomnych, ale nie z zewnątrz
- D. tylko przez właściwość

2 Które zdanie o własnym konstruktorze klasy jest prawdziwe?

- A. Kompilator zawsze zachowuje automatyczny publiczny konstruktor bez parametrów
- B. Każdy konstruktor musi mieć typ void
- C. Napisanie dowolnego konstruktora wyłącza automatyczny konstruktor bezparametrowy
- D. Konstruktor może mieć inną nazwę niż klasa

3 Co oznacza protected?

- A. dostęp tylko z zewnątrz
- B. dostęp w klasie i klasach potomnych, ale nie zwykle z zewnątrz
- C. dokładnie to samo co public
- D. dostęp tylko dla metod statycznych

4 Które utworzenie obiektu jest dozwolone?

- A. new Figura(), gdy Figura jest abstract
- B. new B(), gdy B jest sealed
- C. dziedziczenie C : B, gdy B jest sealed
- D. override metody, która w bazie jest zwykła i niewirtual

5 Klasa implementująca interfejs w klasycznej formule egzaminu:

- A. może pominąć dowolną metodę kontraktu i pozostać konkretna
- B. używa override przy każdej metodzie interfejsu
- C. może implementować wiele interfejsów
- D. musi dziedziczyć po interfejsie jako pierwszej pozycji przed klasą bazową

6 Co zwraca `return x++`, jeśli przed instrukcją `x = 7`?

- A. 6
- B. 7
- C. 8
- D. zależy od typu referencji

ĆWICZENIA

27. Trening kodowy

Zapisuj odpowiedzi na kartce. Nie licz w pamięci.

7 Co zostanie wypisane?

FRAGMENT DO ANALIZY

```
class A { public void M() { Console.Write("&quot;A&quot;"); } }  
class B : A { public new void M() { Console.Write("&quot;B&quot;"); } }  
A x = new B();  
x.M();
```

- A. A
- B. B
- C. AB
- D. błąd kompilacji

8 Co zostanie wypisane?

FRAGMENT DO ANALIZY

```
class A { public virtual void M() { Console.Write("&quot;A&quot;"); } }  
class B : A { public override void M() { Console.Write("&quot;B&quot;"); } }  
A x = new B();  
x.M();
```

- A. A
- B. B
- C. AB
- D. błąd kompilacji

9 Jaki jest wynik?**FRAGMENT DO ANALIZY**

```
int x = 0;
if (3 > 0) x += 2;
if (3 > 2) x += 3;
Console.Write(x);
```

- A. 2
- B. 3
- C. 5
- D. 6

10 Jaki jest wynik zwracany i stan x po return?**FRAGMENT DO ANALIZY**

```
int x = 4;
return x++;
```

- A. wynik 4, x = 4
- B. wynik 4, x = 5
- C. wynik 5, x = 5
- D. wynik 5, x = 4

11 Co stanie się z a.X?**FRAGMENT DO ANALIZY**

```
Punkt a = new Punkt();
a.X = 5;
Punkt b = a;
b.X = 9;
Console.Write(a.X); // Punkt jest struct
```

- A. 0
- B. 5
- C. 9
- D. błąd kompilacji

ĆWICZENIA

28. Trening dopasowania i luk

To dokładnie ten sposób myślenia, którego wymaga formularz z blokami.

Zadanie 12 - dopasuj deklaracje

public static int v7 { get; set; }	_____
protected static int a3;	_____
static int a1;	_____
public int v4 { get; set; }	_____
static A() {}	_____

Zadanie 13 - uzupełnij bloki

DOSTĘPNE: Urzadzenie, IStart, override, virtual, Telefon

```
abstract class Urzadzenie
{
    public abstract int Pobierz();
}

interface IStart
{
    void Start();
}

class Telefon : [[1]], [[2]]
{
    public [[3]] int Pobierz() { return 1; }
    public void Start() { }
}
```

Zadanie 14 - właściwość

DOSTĘPNE: punkty, Punkty, value, get, set

```
private int punkty;

public int Punkty
{
    get { return [[1]]; }
    set { [[2]] = [[3]]; }
}
```

SAMOKONTROLA

Czy potrafisz przy każdej luce powiedzieć: „tu oczekiwany jest typ”, „tu słowo kluczowe”, „tu pole”, „tu specjalna wartość settera”? Jeśli tak, nie zgadujesz - rozwiązujesz gramatycznie.

29. Rozwiązania treningu - definicje

Czytaj wyjaśnienie także wtedy, gdy odpowiedź była dobra.

ZADANIE 1	B
Dlaczego?	<code>public</code> nie ogranicza dostępu do klasy ani jej potomków i pozwala na użycie z zewnątrz.
Kotwica pamięci	<code>public</code> = dla wszystkich.

ZADANIE 2	C
Dlaczego?	Jeśli napiszesz dowolny konstruktor klasy, kompilator nie doda automatycznie publicznego <code>A()</code> .
Kotwica pamięci	Własny konstruktor wyłącza automat.

ZADANIE 3	B
Dlaczego?	<code>protected</code> chroni człon przed zwykłym kodem zewnętrznym, ale udostępnia go klasom potomnym.
Kotwica pamięci	<code>protected</code> = dla rodziny.

ZADANIE 4	B
Dlaczego?	<code>sealed</code> blokuje dziedziczenie, ale nie blokuje tworzenia obiektów. Abstrakcyjnej klasy nie tworzymy, a zwykłej metody nie nadpisujemy przez <code>override</code> .
Kotwica pamięci	<code>sealed</code> zamyka potomków, nie konstruktor.

ZADANIE 5	C
Dlaczego?	Jedna klasa może implementować wiele interfejsów. Klasyczna implementacja człon interfejsu nie używa <code>override</code> .
Kotwica pamięci	Jedna baza, wiele kontraktów.

ZADANIE 6	B - zwraca 7
Dlaczego?	Postinkrementacja używa starej wartości, a dopiero potem zwiększa zmienną do 8.
Kotwica pamięci	<code>post</code> = po użyciu.

30. Rozwiązania treningu - kod

Najpierw reguła wyboru, potem wynik.

ZADANIE 7	A
Dlaczego?	Metoda B ukrywa metodę A przez <code>new</code> . Referencja ma typ A, więc wybrana zostaje A.M().
Kotwica pamięci	<code>new</code> patrzy na etykietę A.

ZADANIE 8	B
Dlaczego?	Metoda jest wirtualna, a rzeczywisty obiekt ma typ B. Najbardziej szczegółowa implementacja to B.M().
Kotwica pamięci	<code>override</code> patrzy na obiekt B.

ZADANIE 9	C - wynik 5
Dlaczego?	To dwa niezależne warunki <code>if</code> . Oba są prawdziwe, więc x rośnie najpierw o 2, potem o 3.
Kotwica pamięci	<code>if + if</code> może wykonać dwa bloki.

1. `x = 0`

2. po pierwszym `if`: `x = 2`

3. po drugim `if`: `x = 5`

ZADANIE 10	B - wynik 4, potem <code>x = 5</code>
Dlaczego?	<code>x++</code> przekazuje do <code>return</code> starą wartość 4. Zwiększenie stanu następuje po użyciu.
Kotwica pamięci	Oddaj stare, potem dodaj.

ZADANIE 11	B - <code>a.X</code> wynosi 5
Dlaczego?	Punkt jest strukturą, więc przypisanie tworzy kopię wartości. Zmiana <code>b.X</code> nie dotyczy <code>a.X</code> .
Kotwica pamięci	<code>struct</code> kopiuje kartkę.

31. Rozwiązania treningu - dopasowanie i luki

W każdej odpowiedzi nazwij kategorię elementu.

Zadanie 12

<code>public static int v7 { get; set; }</code>	publiczna statyczna właściwość	get/set oznacza właściwość
<code>protected static int a3;</code>	chronione statyczne pole	średnik po nazwie oznacza pole
<code>static int a1;</code>	prywatne statyczne pole	brak modyfikatora członu class = private
<code>public int v4 { get; set; }</code>	publiczna właściwość instancyjna	brak static = osobna dla obiektu
<code>static A() { }</code>	konstruktor statyczny	nazwa klasy, brak typu wyniku, static

Zadanie 13

[[1]]	Urządzenie	klasa bazowa występuje pierwsza
[[2]]	IStart	interfejs po klasie bazowej
[[3]]	override	Pobierz jest abstract w bazie

Zadanie 14

[[1]]	punkty	getter zwraca pole zapasowe
[[2]]	punkty	setter przypisuje do pola
[[3]]	value	value oznacza wartość przekazaną do settera

FORMUŁA

`get { return pole; }` oraz `set { pole = value; }`. Jeśli obie strony settera są takie same, stan się nie zmieni; jeśli getter zwraca właściwość, powstanie rekurencja.

32. Egzamin próbny - 15 minut

Ustaw stoper. Nie zaglądaj do klucza przed końcem.

ZASADY

12 zadań: 7 pytań A/B/C/D, 2 wyniki kodu, 1 dopasowanie i 2 zestawy luk. Maksymalnie 15 minut. Przy wyniku kodu wolno robić tabelkę na marginesie.

1

Niestatyczna metoda `private` klasy bazowej:

- A. jest bezpośrednio dostępna w klasie potomnej
- B. jest dostępna z zewnątrz obiektu
- C. jest częścią implementacji bazowej, ale potomek nie wywoła jej bezpośrednio
- D. automatycznie zmienia dostęp na `protected`

2

Która deklaracja opisuje publiczną statyczną właściwość?

- A. `public static int X { get; set; }`
- B. `public int X;`
- C. `static int X;`
- D. `protected static int X;`

3

Które zdanie o konstruktorze statycznym jest prawdziwe?

- A. wywołuje się go przez `new`
- B. może otrzymywać dowolne argumenty
- C. uruchamia się automatycznie najwyżej raz przed pierwszym użyciem typu
- D. musi być publiczny

4

Który typ może być utworzony przez `new`?

- A. interfejs
- B. klasa abstrakcyjna
- C. klasa `sealed`
- D. metoda `abstract`

33. Egzamin próbny - ciąg dalszy

Pilnuj typu referencji i typu obiektu.

5 Co zostanie wypisane?

FRAGMENT DO ANALIZY

```
class A { public virtual void M() { Console.Write("&quot;A&quot;"); } }
class B : A { public override void M() { Console.Write("&quot;B&quot;"); } }
class C : B { public new void M() { Console.Write("&quot;C&quot;"); } }
A x = new C();
x.M();
```

- A. A
- B. B
- C. AB
- D. błąd

6 Co zostanie wypisane?

FRAGMENT DO ANALIZY

```
class A { public void M() { Console.Write("&quot;A&quot;"); } }
class B : A { public new void M() { Console.Write("&quot;B&quot;"); } }
class C : B { public new void M() { Console.Write("&quot;C&quot;"); } }
B x = new C();
x.M();
```

- A. A
- B. B
- C. C
- D. błąd

7 Która implementacja metody interfejsu jest poprawna w klasycznej formule materiału?

- A. private void Start() { }
- B. public override void Start() { }
- C. public void Start() { }
- D. abstract void Start() { }

8

Jaki wynik zwraca metoda?

FRAGMENT DO ANALIZY

```
int Result = 8;
void Add(int x = 1) { Result += x; }

int Run(int arg)
{
    Add();
    if (arg > 1) Add(arg);
    else if (arg == 1) Add(2);
    return Result--;
}

Run(2);
```

- A. 10
- B. 11
- C. 12
- D. 13

34. Egzamin próbny - bloki

Najpierw ustal rolę każdego miejsca.

9 Jaki jest wynik?

FRAGMENT DO ANALIZY

```
int x = 1;
if (x > 0) x++;
if (x > 1) ++x;
Console.Write(x);
```

- A. 1
- B. 2
- C. 3
- D. 4

10. Dopasuj

A. protected static int a;	_____
B. public int V { get; set; }	_____
C. static int x;	_____
D. public static void M() { }	_____

1 - prywatne pole statyczne | 2 - publiczna właściwość instancyjna | 3 - chronione pole statyczne | 4 - publiczna metoda statyczna

11. Uzupełnij hierarchię

DOSTĘPNE: Baza, IOpis, override, virtual, Wynik

```
abstract class Baza
{
    public abstract int Policz();
}

interface IOpis
{
    string Opis();
}

class Wynik : [[1]], [[2]]
{
    public [[3]] int Policz() { return 7; }
    public string Opis() { return "OK"; }
}
```

12. Uzupełnij właściwość

DOSTĘPNE: `liczba`, `Liczba`, `value`, `private`, `public`

```
private int liczba;  
public int Liczba  
{  
    get { return [[1]]; }  
    private set { [[2]] = [[3]]; }  
}
```

35. Klucz egzaminu próbnego - 1-7

Każda odpowiedź ma regułę, którą należy zapamiętać.

1	C
Dlaczego?	private pozostaje niedostępne bezpośrednio dla kodu potomka. Nie staje się protected.
Kotwica pamięci	Obecność w części bazowej nie oznacza dostępu.

2	A
Dlaczego?	get/set rozpoznaje właściwość, static oznacza wspólną kopię, public dostęp dla wszystkich.
Kotwica pamięci	get/set - właściwość; static - typ.

3	C
Dlaczego?	Konstruktor statyczny wywołuje środowisko automatycznie i najwyżej raz. Nie ma parametrów ani modyfikatora dostępu.
Kotwica pamięci	Nie tworzysz go przez new.

4	C
Dlaczego?	sealed blokuje dziedziczenie, ale obiekty klasy sealed można normalnie tworzyć.
Kotwica pamięci	sealed = ostatni typ w gałęzi.

5	B
Dlaczego?	A.M jest virtual, B.M ją nadpisuje. C jedynie ukrywa metodę przez new, nie tworzy nowego override w łańcuchu A/B. Wywołanie przez A trafia do najbardziej szczegółowego override, czyli B.M.
Kotwica pamięci	Najpierw znajdź łańcuch override; new tworzy osobną ścieżkę.

6	B
Dlaczego?	Metody są ukrywane przez new. Referencja ma typ B, więc wybierana jest B.M, mimo że obiekt jest C.
Kotwica pamięci	new patrzy na lewą stronę.

7	C
Dlaczego?	Klasyczna implementacja kontraktu jest publiczna i nie używa override.
Kotwica pamięci	Interfejs to kontrakt, nie metoda bazowa do nadpisania.

36. Klucz egzaminu próbnego - 8-12

Przy wyniku kodu zawsze pokazuj sobie stan po każdej linii.

8	B - 11
Dlaczego?	Start 8. Add() używa domyślnego 1, więc jest 9. arg = 2 spełnia pierwszy warunek, Add(2) daje 11. Postdekrementacja zwraca 11, a dopiero potem ustawia Result na 10.
Kotwica pamięci	return Result-- oddaje stare 11.

1. 8 -> Add() -> 9

2. 9 -> Add(2) -> 11

3. zwróć 11, stan po return: 10

9	C - 3
Dlaczego?	Start 1. Pierwszy if jest prawdziwy: x++ zmienia stan do 2. To osobny drugi if, więc 2 > 1 jest prawdą; ++x daje 3.
Kotwica pamięci	Dwa niezależne if mogą wykonać oba bloki.

10. Dopasowanie

A	3	protected + static + średnik = chronione pole statyczne
B	2	public + get/set + bez static = publiczna właściwość instancyjna
C	1	brak modyfikatora + static + średnik = prywatne pole statyczne
D	4	public + static + nawiasy = publiczna metoda statyczna

11-12. Luki

11: **[[1]] Baza, [[2]] IOpis, [[3]] override.** Najpierw klasa bazowa, potem interfejs; metoda abstract wymaga override.

12: **[[1]] liczba, [[2]] liczba, [[3]] value.** Getter zwraca pole, setter wpisuje do pola wartość przekazaną przez value.

37. Fiszki - szybkie aktywne przypominanie

Zasłoń prawą kolumnę i odpowiedz własnymi słowami.

Czym różni się klasa od obiektu?	Klasa opisuje, obiekt jest konkretnym egzemplarzem.
Co robi pole?	Przechowuje stan obiektu lub typu.
Co robi metoda?	Opisuje działanie.
Co robi właściwość?	Kontroluje odczyt i zapis danych.
Jaki jest domyślny dostęp członu class?	private.
Kto widzi protected?	Klasa deklarująca i jej potomkowie.
Co znaczy static?	Jedna wspólna składowa należąca do typu.
Czy metoda static ma this?	Nie.
Kiedy istnieje automatyczny A()?	Gdy klasa nie ma żadnego własnego konstruktora.
Co wywołuje this(...)?	Inny konstruktor tej samej klasy.
Co wywołuje base(...)?	Konstruktor klasy bazowej.
Jaka jest kolejność konstruktorów?	Najpierw baza, potem potomek.
Czy tworzysz obiekt abstract class?	Nie.
Co robi sealed class?	Blokuje dalsze dziedziczenie.
Co wybiera metoda ukryta przez new?	Typ referencji.
Co wybiera virtual/override?	Rzeczywisty typ obiektu.
Czy metoda abstract ma ciało?	Nie.
Jak potomek implementuje abstract?	Przez override.
Ile klas bazowych może mieć klasa?	Jedną.
Ile interfejsów może implementować?	Wiele.

38. Fiszki - składnia i śledzenie

Powtarzaj aż odpowiedź pojawi się w mniej niż trzy sekundy.

Co robi get?	Zwraca wartość właściwości.
Co oznacza value w set?	Wartość po prawej stronie przypisania.
Jak implementujesz metodę interfejsu?	Publicznie, zgodna sygnatura, bez override w klasycznym zakresie.
Co przechowuje delegat?	Referencję do metody o zgodnej sygnaturze.
Co robi += dla delegata/zdarzenia?	Dodaje metodę obsługującą.
Jaki wynik zwraca delegat multicast?	Wynik ostatniej metody; wszystkie skutki uboczne zostają.
x++ w wyrażeniu?	Użyj starej wartości, potem zwiększ.
++x w wyrażeniu?	Najpierw zwiększ, potem użyj.
if + if?	Oba bloki mogą się wykonać.
if + else if?	Wykona się najwyżej jedna gałąź.
5 / 2 dla int?	2.
Jak przypisuje się struct?	Przez kopię wartości.
Jak przypisuje się class?	Przez kopię referencji.
Stack?	LIFO, Push i Pop.
Queue?	FIFO, Enqueue i Dequeue.
Action?	Delegat bez wartości zwracanej.
Func?	Ostatni parametr typu oznacza wynik.
Boxing?	Zapakowanie wartości do object.
Unboxing?	Jawne odpakowanie z object do dokładnego typu.
using z IDisposable?	Gwarantuje Dispose po wyjściu z bloku.

39. Plan nauki na 5 dni

Krótko, aktywnie i zawsze z zadaniem na końcu.

1 - język kodu	Rozdziały 1-5: klasa, obiekt, typy, dostęp, właściwości, static	Wyjaśnij każdą linię klasy Konto bez używania notatek
2 - tworzenie i hierarchie	Rozdziały 6-8: konstruktory, kolejność, dziedziczenie, abstract, sealed	Napisz z pamięci reguły this/base i automatycznego A()
3 - polimorfizm	Rozdziały 9-11: referencja/obiekt, new/override, base.M()	Rozwiąż 8 krótkich przykładów i uzasadnij wybór metody
4 - format egzaminu	Rozdziały 12-16: interfejsy, luki, delegaty, stan	Zrób trzy tabelki stanu z ++/-- i if/else if
5 - domknięcie	Rozdziały 17-25 i pełny egzamin próbny	15 minut bez przerw; potem popraw tylko konkretne reguły

System powtórek błędów

1. Przy błędzie zapisz nie pytanie, tylko **regułę**, której zabrakło.
2. Dopisz jeden własny mini-przykład kodu.
3. Powtórz po 10 minutach, następnego dnia i po trzech dniach.
4. Usuń fiszkę dopiero po trzech poprawnych odpowiedziach z rzędu.

DOBRA MIARA POSTĘPU

Nie pytaj: „czy to wygląda znajomo?”. Pytaj: „czy bez podpowiedzi potrafię powiedzieć regułę i przewidzieć wynik?”. Rozpoznawanie to nie to samo co przypominanie.

40. Gotowość do egzaminu

Jeśli potrafisz odpowiedzieć TAK na każdy punkt, masz opanowany rdzeń.

☐ Rozróżniam klasę, obiekt, pole, metodę, właściwość i konstruktor.	☐ Znam dostęp public, protected, private oraz domyślny private członów.
☐ Wiem, kiedy znika automatyczny konstruktor bezparametrowy.	☐ Potrafię ustalić kolejność this(...), base(...) i ciał konstruktorów.
☐ Rozróżniam abstract, virtual, override, new i sealed.	☐ Dla A x = new B() osobno zapisuję typ referencji i obiektu.
☐ Umiem wybrać ciało metody dla new oraz virtual/override.	☐ Wiem, że baza występuje przed interfejsami, a implementacja interfejsu nie używa override.
☐ Uzupełniam get/set jako pole oraz value, nie nazwę właściwości po obu stronach.	☐ Śledzę Result w tabeli i odróżniam x++ od ++x.
☐ Widzę różnicę między if + if i if + else if.	☐ Rozróżniam kopiowanie struct od kopiowania referencji class.
☐ Potrafię rozpoznać pole, metodę, właściwość i konstruktor po składni.	☐ Ukończyłem egzamin próbny w maksymalnie 15 minut.

OSTATNIA MANTRA

Czytam symbol, nazywam regułę, zapisuję stan, dopiero wybieram odpowiedź.

Zakres źródłowy

Kompendium opracowano na podstawie prezentacji Programowanie_obiektowe_CS_01 do CS_07-2 oraz pytań i mechaniki lokalnej aplikacji kursowej. Treści dokumentów źródłowych potraktowano jako materiał merytoryczny, nie jako instrukcje działania. W miejscach, w których współczesny C# rozszerza dawne możliwości języka, pierwszeństwo ma konwencja i zakres widoczny w przekazanych materiałach egzaminacyjnych.