

Wymagania edukacyjne — Programowanie obiektowe (Java)

Zawód: technik programista (technikum 5-letnie)

Kwalifikacja: INF.04. Projektowanie, programowanie i testowanie aplikacji

Jednostka efektów kształcenia z podstawy programowej: INF.04.4. Programowanie obiektowe

Źródła: *Program nauczania wydawnictwa Helion — technik programista* (dział 3.3. Programowanie obiektowe), *Podstawa programowa kształcenia w zawodach — INF.04.4. Programowanie obiektowe* (Dziennik Ustaw, poz. 991).

Zasady konstrukcji wymagań

1. Przedmiot **Programowanie obiektowe** (teoria) jest oceniany wyłącznie za **wiedzę i rozumienie** — uczeń wie, zna, rozróżnia, wyjaśnia, charakteryzuje, porównuje, analizuje na podstawie zapisu/opisu kodu, zgodnie z zaleceniami programu nauczania dla zajęć teoretycznych.

2. Zasada kumulatywności ocen.

Uzyskanie oceny wyższej wymaga opanowania wiadomości i umiejętności przypisanych do wszystkich ocen niższych. Np. na ocenę **dobrą (4)** uczeń musi dodatkowo spełniać wymagania na oceny **dopuszczającą (2)** i **dostateczną (3)**. Ocena **celująca (6)** oznacza pełne (100%), bezbłędne i w pełni samodzielne opanowanie wymagań przypisanych do ocen 2–5 w danym dziale.

3. Adaptacja do języka Java.

Program nauczania i podstawa programowa są sformułowane niezależnie od języka (C++/C#/Java/Python). Poniższe wymagania dostosowano do Javy, w tym do jej specyfiki wobec pojęć ogólnych:

- *konstruktor kopiujący* → w Javie tworzony ręcznie (konstruktor przyjmujący obiekt tego samego typu) lub przez `clone()/Cloneable` — Java nie generuje go automatycznie;
- *destruktor* → Java nie ma destruktorów w rozumieniu C++ (obiekty usuwa **Garbage Collector**); najbliższym odpowiednikiem jest zwalnianie zasobów przez `try-with-resources` i interfejs `AutoCloseable` (ewentualnie przestarzała metoda `finalize()`, wspominana wyłącznie historycznie);
- *funkcje/klasz zaprzyjaźnione* → Java nie zna słowa kluczowego `friend`; zbliżony efekt (uprzywilejowany dostęp) osiąga się przez dostęp pakietowy (brak modyfikatora) lub klasy zagnieżdżone (*nested/inner classes*).

4. Godziny zajęć wg planu nauczania: **Programowanie obiektowe** (teoria) — klasa III: 2 godz./tydz. (60 godz.), klasa IV: 1 godz./tydz. (30 godz.), razem 90 godz.

5. Ocena niedostateczna.

We wszystkich działach ocenę niedostateczną otrzymuje uczeń, który nie spełnia wymagań na ocenę dopuszczającą — nie posiada wiedzy/umiejętności koniecznych do kontynuowania nauki nawet przy pomocy nauczyciela, nie podejmuje samodzielnej pracy lub oddaje prace niesamodzielne.

KLASA III

Programowanie obiektowe

Dział 1. Środowisko Java, kompilacja, interpretacja i debugowanie

Zakres: JDK, JVM, JRE, kompilator i interpreter, etapy kompilacji i uruchamiania programu, rola debugera, rodzaje błędów (składniowe, wykonania, logiczne), pojęcie biblioteki.

Ocena	Uczeń wie / rozumie / potrafi wyjaśnić
Dopuszczająca (2)	Wymienia podstawowe elementy środowiska Java (JDK, JVM, JRE); wyjaśnia w prosty sposób, czym jest kompilator i interpreter; wskazuje przykład błędu składniowego we wskazanym fragmencie kodu.
Dostateczna (3)	Opisuje zadania realizowane przez kompilator, interpreter i debugger; wyjaśnia kolejne etapy powstawania programu w Javie (kod źródłowy .java → kompilacja → plik .class → uruchomienie przez JVM); wyjaśnia pojęcie biblioteki (modułu) i jej rolę w programowaniu; rozróżnia błędy składniowe, logiczne i błędy wykonania na podstawie opisu lub fragmentu kodu.
Dobra (4)	Charakteryzuje różnicę między kompilacją a interpretacją kodu; analizuje (na podstawie zapisu, bez uruchamiania) prawdopodobną przyczynę typowego błędu w kodzie, np. brak średnika, niezgodność typów, nieistniejąca zmienna; wyjaśnia zastosowanie debugera (punkty przerwania, podgląd wartości zmiennych) jako narzędzia diagnostycznego.
Bardzo dobra (5)	Wyjaśnia rolę JVM w uruchamianiu i przenośności programu Java; analizuje przykładowe komunikaty wyjątków czasu wykonania (np. NullPointerException, ArrayIndexOutOfBoundsException, NumberFormatException) i wskazuje ich prawdopodobną przyczynę; porównuje możliwości różnych narzędzi diagnostycznych.
Celująca (6)	W pełni, bezbłędnie i samodzielnie opanował materiał działu: wyczerpująco wyjaśnia działanie kompilatora, interpretera, JVM i debugera; analizuje najbardziej złożone, wieloprzyczynowe przypadki błędów na podstawie przedstawionego kodu i bezbłędnie uzasadnia sposób ich usunięcia; porównuje możliwości różnych środowisk programistycznych i debuggerów w zakresie omawianym na zajęciach.

Dział 2. Funkcje (metody) i rekurencja

Zakres: definicja metody, parametry, wartość zwracana, void, zakres obowiązywania zmiennych, przeciążanie metod, pojęcie i zastosowania rekurencji.

Ocena	Uczeń wie / rozumie / potrafi wyjaśnić
Dopuszczająca (2)	Wskazuje definicję i wywołanie metody w gotowym kodzie; wyjaśnia, czym różni się metoda zwracająca wartość od metody void.
Dostateczna (3)	Wyjaśnia rolę parametrów formalnych i aktualnych; rozróżnia parametry i wartość zwracaną na podstawie przykładu; wyjaśnia, na czym polega rekurencja i wskazuje warunek stopu w prostym przykładzie.
Dobra (4)	Wyjaśnia cel dzielenia programu na metody (modularność, czytelność, wielokrotne użycie); wyjaśnia mechanizm przeciążania metod; opisuje zakres obowiązywania (widoczność) zmiennych lokalnych; analizuje przykład rekurencyjny (np. silnia) krok po kroku, bez uruchamiania kodu.
Bardzo dobra (5)	Porównuje rozwiązanie iteracyjne i rekurencyjne tego samego problemu; wyjaśnia zasady doboru parametrów i typu zwracanego do rozwiązywanego problemu; analizuje wywołania rekurencyjne (drzewo wywołań) na papierze.
Celująca (6)	W pełni, bezbłędnie i samodzielnie opanował materiał działu: trafnie wyjaśnia zasady dzielenia programu na metody, przeciążania metod oraz działania rekurencji, w tym ryzyko przepełnienia stosu (StackOverflowError); porównuje efektywność i czytelność rozwiązania iteracyjnego i rekurencyjnego dla najtrudniejszych przykładów omawianych na zajęciach i bezbłędnie uzasadnia wybór podejścia.

Dział 3. Typy danych, operatory i instrukcje sterujące

Zakres: typy proste (int, double, boolean, char) i String, tablice, operatory arytmetyczne/relacyjne/logiczne/bitowe, priorytety operatorów, instrukcje if, switch, pętle.

Ocena	Uczeń wie / rozumie / potrafi wyjaśnić
Dopuszczająca (2)	Wymienia podstawowe typy danych w Javie; wskazuje przykład zastosowania instrukcji warunkowej i pętli w gotowym kodzie.
Dostateczna (3)	Charakteryzuje podstawowe typy proste i typ String; wyjaśnia działanie operatorów arytmetycznych, porównania i logicznych; wyjaśnia różnicę między if...else, switch a pętlami for/while/do...while; opisuje budowę tablicy jednowymiarowej.

Dobra (4)	Wyjaśnia zasady konwersji (rzutowania) typów, w tym utratę precyzji przy rzutowaniu; opisuje budowę i zastosowanie tablicy dwuwymiarowej; wyjaśnia działanie pętli for-each; analizuje (na podstawie zapisu) wynik działania fragmentu kodu z instrukcjami sterującymi.
Bardzo dobra (5)	Wyjaśnia priorytety operatorów i ich wpływ na wynik wyrażenia złożonego; przewiduje skutki dzielenia całkowitego, przepełnienia zakresu oraz niejawnej konwersji typów; wyjaśnia zastosowanie wybranych metod klasy String do analizy tekstu.
Celująca (6)	W pełni, bezbłędnie i samodzielnie opanował materiał działu: analizuje wszystkie, również nietypowe, przypadki działania operatorów i konwersji typów (np. utrata precyzji, przepełnienie) omawiane na zajęciach; porównuje alternatywne zapisy tego samego warunku/pętli pod względem czytelności; bezbłędnie uzasadnia wybór struktury danych do opisanego problemu.

Dział 4. Klasy, obiekty i hermetyzacja

Zakres: pojęcie klasy i obiektu, pola, metody instancyjne, konstruktor, słowo kluczowe this, modyfikatory dostępu (public, private, protected, domyślny), hermetyzacja, gettery/settery.

Ocena	Uczeń wie / rozumie / potrafi wyjaśnić
Dopuszczająca (2)	Rozróżnia pojęcia klasy i obiektu; wskazuje pole, metodę i konstruktor w gotowym kodzie klasy.
Dostateczna (3)	Wyjaśnia budowę klasy (pola, konstruktor, metody); wyjaśnia zastosowanie słowa kluczowego this; wyjaśnia znaczenie modyfikatorów public i private.
Dobra (4)	Wyjaśnia pojęcie hermetyzacji i jej cel (ochrona danych, kontrolowany dostęp); wyjaśnia rolę metod get/set; opisuje przeznaczenie konstruktora z parametrami; wyjaśnia znaczenie metody toString() odziedziczonej po klasie Object.
Bardzo dobra (5)	Uzasadnia podział składowych klasy na pola prywatne i metody publiczne; wyjaśnia zasady walidacji danych przekazywanych do konstruktora lub setterów; wyjaśnia, kiedy należy nadpisać metodę toString(), a kiedy equals().
Celująca (6)	W pełni, bezbłędnie i samodzielnie opanował materiał działu: analizuje przykładowy model kilku współpracujących klas i trafnie ocenia poprawność podziału odpowiedzialności; wyczerpująco wyjaśnia różnicę między relacją „jest rodzajem” a „ma” (dziedziczenie a kompozycja) na przykładzie opisanego problemu.

Dział 5. Wprowadzenie do dziedziczenia i polimorfizmu

Zakres: extends, klasa bazowa i pochodna, super, przesłanianie metod, adnotacja @Override, podstawowe pojęcie polimorfizmu.

Ocena	Uczeń wie / rozumie / potrafi wyjaśnić
Dopuszczająca (2)	Wskazuje klasę bazową i pochodną w gotowym programie; rozpoznaje słowo kluczowe extends.
Dostateczna (3)	Wyjaśnia, które pola i metody są dziedziczone przez klasę pochodną; wyjaśnia zastosowanie super() w konstruktorze; wskazuje przesłoniętą metodę w kodzie.
Dobra (4)	Wyjaśnia znaczenie adnotacji @Override; wyjaśnia, na czym polega polimorficzne wywołanie metody; wyjaśnia, co oznacza przypisanie obiektu klasy pochodnej do zmiennej typu klasy bazowej.
Bardzo dobra (5)	Uzasadnia, które elementy powinny znaleźć się w klasie bazowej, a które w klasie pochodnej dla opisanego problemu; wyjaśnia praktyczne zastosowanie polimorfizmu na przykładzie.
Celująca (6)	W pełni, bezbłędnie i samodzielnie opanował materiał działu: analizuje konsekwencje wprowadzenia dziedziczenia w opisanym projekcie i trafnie porównuje je z zastosowaniem kompozycji jako alternatywnego podejścia do tego samego problemu.

KLASA IV

Programowanie obiektowe

Dział 1. Zaawansowane składniki klas

Zakres: konstruktor kopiujący (implementacja ręczna / clone()), zarządzanie cyklem życia obiektu i zwalnianie zasobów (odpowiednik destruktora — AutoCloseable/try-with-resources), składniki statyczne, klasy/funkcje zaprzyjaźnione i ich odpowiednik w Javie (dostęp pakietowy, klasy zagnieżdżone).

Ocena	Uczeń wie / rozumie / potrafi wyjaśnić
Dopuszczająca (2)	Wskazuje konstruktor i pole/metodę statyczną w gotowym kodzie klasy.
Dostateczna (3)	Wyjaśnia różnicę między polem/metodą instancyjną a statyczną; wyjaśnia, na czym polegałoby skopiowanie obiektu (konstruktor kopiujący) i dlaczego Java nie tworzy go automatycznie; wyjaśnia, że Java nie posiada destruktorów, a obiekty usuwa Garbage Collector.

Dobra (4)	Wyjaśnia zastosowanie interfejsu AutoCloseable i konstrukcji try-with-resources jako odpowiednika zwalniania zasobów; wyjaśnia sytuacje uzasadniające użycie składownika statycznego (np. licznik obiektów, stała klasowa).
Bardzo dobra (5)	Analizuje skutki płytkiego kopiowania obiektu (kopiowanie referencji) w porównaniu z kopiowaniem głębokim; wyjaśnia, że Java nie zna mechanizmu friend, i opisuje, jak podobny efekt (uprzywilejowany dostęp) uzyskuje się przez dostęp pakietowy lub klasy zagnieżdżone.
Celująca (6)	W pełni, bezbłędnie i samodzielnie opanował materiał działu: analizuje ryzyko wycieku zasobów przy niewłaściwym zarządzaniu cyklem życia obiektu; trafnie porównuje podejście Javy (Garbage Collector, try-with-resources) z ogólnym pojęciem destruktora, którym posługuje się program nauczania; ocenia zasadność zastosowania klasy zagnieżdżonej w opisanym przypadku.

Dział 2. Dziedziczenie zaawansowane, klasy abstrakcyjne i interfejsy

Zakres: hierarchie dziedziczenia, klasy i metody abstrakcyjne (abstract), metody wirtualne (w Javie domyślne dla metod niestatycznych), interfejsy (interface), typy wyliczeniowe (enum).

Ocena	Uczeń wie / rozumie / potrafi wyjaśnić
Dopuszczająca (2)	Rozpoznaje deklarację klasy abstrakcyjnej, interfejsu i enum w gotowym kodzie.
Dostateczna (3)	Wyjaśnia różnicę między klasą konkretną a abstrakcyjną; wyjaśnia, że implementacja interfejsu zobowiązuje klasę do zdefiniowania jego metod; wyjaśnia zastosowanie enum.
Dobra (4)	Wyjaśnia, że w Javie metody niestatyczne są domyślnie „wirtualne” (mechanizm dynamicznego wiązania); wyjaśnia, kiedy zastosować referencję typu klasy bazowej/interfejsu do obsługi obiektów różnych klas pochodnych.
Bardzo dobra (5)	Porównuje interfejs z klasą abstrakcyjną (<u>m.in.</u> możliwość wielokrotnej implementacji interfejsów) i uzasadnia wybór jednego z tych mechanizmów dla opisanego problemu; wyjaśnia konsekwencje zmiany w klasie bazowej dla klas pochodnych.
Celująca (6)	W pełni, bezbłędnie i samodzielnie opanował materiał działu: analizuje rozbudowany, wielopoziomowy model klas (abstrakcja, interfejsy, dziedziczenie) opisany w zadaniu i trafnie ocenia jego spójność; wyjaśnia ideę „programowania względem interfejsu, nie implementacji”.

Dział 3. Szablony klas i elementy programowania generycznego

Zakres: parametr typu (<T>), generyczna klasa/metoda, zastosowanie własnej klasy generycznej lub wybranej klasy generycznej biblioteki standardowej jako kontenera na obiekty.

Ocena	Uczeń wie / rozumie / potrafi wyjaśnić
Dopuszczająca (2)	Rozpoznaje zapis klasy/metody generycznej (<T>) w gotowym kodzie.
Dostateczna (3)	Wyjaśnia, czym różni się klasa generyczna od klasy dla konkretnego typu; wyjaśnia, po co stosuje się parametr typu zamiast wielu przeciążonych klas.
Dobra (4)	Wyjaśnia zasadę działania generycznego kontenera przechowującego obiekty jednego typu; wyjaśnia, dlaczego przechowywanie obiektów własnej klasy w kontenerze generycznym jest zastosowaniem hermetyzacji i polimorfizmu.
Bardzo dobra (5)	Analizuje zalety i ograniczenia programowania generycznego w porównaniu z użyciem typu ogólnego (Object); wyjaśnia znaczenie bezpieczeństwa typów (<i>type safety</i>) zapewnianego przez generyki.
Celująca (6)	W pełni, bezbłędnie i samodzielnie opanował materiał działu: analizuje projekt klasy generycznej opisany w zadaniu (np. z kilkoma parametrami typu) i trafnie ocenia jego uniwersalność oraz czytelność.

Dział 4. Obsługa wyjątków

Zakres: błędy wykonania, wyjątki kontrolowane i niekontrolowane, try, catch, finally, throw, throws, wyjątki własne.

Ocena	Uczeń wie / rozumie / potrafi wyjaśnić
Dopuszczająca (2)	Rozpoznaje blok try...catch w gotowym kodzie; wskazuje przykład sytuacji, która wywołuje wyjątek.
Dostateczna (3)	Wyjaśnia cel stosowania try, catch i finally; wyjaśnia znaczenie typowych wyjątków, np. InputMismatchException, NumberFormatException, ArithmeticException.
Dobra (4)	Wyjaśnia różnicę między wyjątkiem kontrolowanym (<i>checked</i>) a niekontrolowanym (<i>unchecked</i>); wyjaśnia zastosowanie throw i throws; wyjaśnia, dlaczego warto walidować dane przed wykonaniem operacji.

Bardzo dobra (5)	Wyjaśnia zasady definiowania własnego wyjątku (dziedziczenie po Exception/RuntimeException); analizuje, jak wyjątek propaguje się przez kolejne wywołania metod, jeśli nie zostanie przechwycony.
Celująca (6)	W pełni, bezbłędnie i samodzielnie opanował materiał działu: analizuje opisaną strategię obsługi błędów w aplikacji (rozdzielenie walidacji, logiki i komunikatów) i trafnie ocenia jej spójność oraz wpływ na niezawodność programu.

Dział 5. Projektowanie i implementacja aplikacji obiektowej

Zakres: synteza poznanych mechanizmów obiektowych (klasy, dziedziczenie, polimorfizm, wyjątki, generyki) w spójnym programie.

Ocena	Uczeń wie / rozumie / potrafi wyjaśnić
Dopuszczająca (2)	Wskazuje w opisie prostego problemu podstawowe klasy i obiekty.
Dostateczna (3)	Wyjaśnia, jakie pola i metody powinna mieć klasa reprezentująca opisany obiekt rzeczywisty; opisuje podstawowe relacje między dwiema-trzema klasami.
Dobra (4)	Analizuje opisane wymagania i proponuje podział na klasy oraz relacje między nimi (dziedziczenie lub kompozycja); wskazuje, gdzie w projekcie należałoby zastosować hermetyzację i obsługę wyjątków.
Bardzo dobra (5)	Ocena zaproponowany model klas pod kątem zgodności z zasadami programowania obiektowego (hermetyzacja, dziedziczenie, polimorfizm); uzasadnia decyzje projektowe dla opisanego problemu.
Celująca (6)	W pełni, bezbłędnie i samodzielnie opanował materiał działu: analizuje i krytycznie ocenia złożony, wielomodułowy model obiektowy opisany w zadaniu; proponuje jego refaktoryzację w granicach realizowanego programu i trafnie uzasadnia swoją ocenę.