KARTA OPISU MODUŁU KSZTAŁCENIA

Nazwa modułu/przedmiotu	Kod
Programowanie obiektowe	**1010514331010510084**

Kierunek studiów	Profil kształcenia (ogólnoakademicki, praktyczny)	Rok / Semestr
Informatyka	**ogólnoakademicki**	**2 / 3**
Ścieżka obieralności/specjalność	Przedmiot oferowany w języku:	Kurs (obligatoryjny/obieralny)
-	**polski**	**obligatoryjny**

Stopień studiów:	**Forma studiów** (stacjonarna/niestacjonarna)
I stopień	**niestacjonarna**

Godziny	Liczba punktów
Wykłady: **16** Ćwiczenia: **-** Laboratoria: **16** Projekty/seminaria: **-**	**3**

Status przedmiotu w programie studiów (podstawowy, kierunkowy, inny)	(ogólnouczelniany, z innego kierunku)
kierunkowy	**z danego kierunku**

Obszar(y) kształcenia i dziedzina(y) nauki i sztuki	**Podział ECTS (liczba i %)**
nauki techniczne	**3 100%**

Odpowiedzialny za przedmiot / wykładowca:

dr inż. Marek Wojciechowski
email: Marek.Wojciechowski@cs.put.poznan.pl
tel. 6652962
Instytut Informatyki
ul. Piotrowo 2, 60-965 Poznań

Wymagania wstępne w zakresie wiedzy, umiejętności, kompetencji społecznych:

1	**Wiedza:**	Student rozpoczynający ten przedmiot powinien posiadać podstawową wiedzę uzyskaną na przedmiotach: Wprowadzenie do informatyki oraz Algorytmy i Struktury Danych.
2	**Umiejętności:**	Powinien posiadać umiejętność rozwiązywania podstawowych problemów ze specyfikacją algorytmów, samodzielnego pisania, modyfikowania i testowania programów komputerowych oraz umiejętność pozyskiwania informacji ze wskazanych źródeł.
3	**Kompetencje społeczne**	Powinien również rozumieć konieczność poszerzania swoich kompetencji i mieć gotowość do podjęcia współpracy w ramach zespołu. Ponadto w zakresie kompetencji społecznych student musi prezentować takie postawy jak uczciwość, odpowiedzialność, wytrwałość, ciekawość poznawcza, kreatywność, kultura osobista i szacunek dla innych ludzi.

Cel przedmiotu:

1. Nauczenie studentów zasad tworzenia uniwersalnych modułów programowych zdatnych do wielokrotnego wykorzystania w różnych projektach programistycznych i łatwych w rozwoju i pielęgnacji, poprzez zastosowanie unikalnych rozwiązań dostępnych w językach obiektowych, które sprzyjają tworzeniu programów komputerowych o takich cechach. Ponadto celem jest nauczenie studentów tworzenia własnych bogatych semantycznie i uniwersalnych abstrakcyjnych typów danych.
2. Rozwijanie u studentów umiejętności projektowania i tworzenia systemów informatycznych o poprawnej architekturze, to jest takiej, która charakteryzuję się spoistością składowych modułów programowych i luźnych związków między tymi modułami.
3. Kształtowanie u studentów umiejętności komunikacji podczas niezależnego tworzenia modułów programów komputerowych, które mają być skomponowane w jedną całość. Ponadto uzyskanie umiejętności wyszukiwania optymalnych, gotowych i dostępnych komponentów, do wykorzystania we własnych złożonych programach komputerowych.

Efekty kształcenia i odniesienie do kierunkowych efektów kształcenia

Wiedza:

1. ma uporządkowaną, podbudowaną teoretycznie wiedzę ogólną w zakresie algorytmów, języków i paradygmatów programowania oraz inżynierii oprogramowania, - [K_W4]

2. zna podstawowe metody, techniki i narzędzia stosowane przy rozwiązywaniu prostych zadań informatycznych, algorytmów i problemów, budowy systemów komputerowych, implementacji języków programowania oraz inżynierii oprogramowania, - [K_W8]

3. ma wiedzę niezbędna do modelowania i analizy obiektowej niedużych fragmentów ?świata rzeczywistego? związanych z różnymi dziedzinami zastosowań, - [K_W8]

4. ma wiedzę niezbędna do transformacji modeli obiektowych fragmentów rzeczywistości do wybranych języków obiektowych, - [K_W8]

5. zna i rozumie składnię i semantykę podstawowych i złożonych mechanizmów obiektowych, takich jak: klasy, obiekty, interfejsy i implementacja obiektów, hermetyczność obiektów, dziedziczenie klas i relacja podtypów, zmienne i podstawienia polimorficzne, wiązanie dynamiczne, - [K_W8]

6. zna i rozumie klasy generyczne, obsługa wyjątków w językach obiektowych i potrafi zastosować te mechanizmy do tworzenia uniwersalnych modułów programowych do wielokrotnego użytku, gwarantujących budowę programów komputerowych o wysokiej jakości, - [K_W8]

7. zna i rozumie zasady konstrukcji programów komputerowych przetwarzających trwałe obiekty przechowywane w bazie danych oraz zasady konstrukcji programów o złożonej wieloaspektowej architekturze. - [K_W8]

Umiejętności:

1. potrafi stworzyć model obiektowy prostego systemu (np. w języku UML) - [K_U14]

2. potrafi wybrać język programowania odpowiedni do danego zadania programistycznego - [K_U20]

3. potrafi - zgodnie z zadaną specyfikacją - zaprojektować oraz zrealizować prosty system informatyczny, używając właściwych metod, technik i narzędzi - [K_U21]

4. ma umiejętność formułowania klas i ich programowania z użyciem przynajmniej dwóch popularnych narzędzi, tj. obiektowych języków programowania: C++ i Java, - [K_U22]

5. ma umiejętność tworzenia programów komputerowych o wysokiej jakości zgodnej z kryteriami zdefiniowanymi w normach ISO, a w szczególności charakteryzujących się: niezawodnością, łatwością utrzymania i rozwoju, elastyczności, łatwości testowania, przenośności i łatwości współużywania kodu, - [K_U22]

Kompetencje społeczne:

1. posiada umiejętność komunikacji typu ?burza mózgów? za pomocą wyspecjalizowanych narzędzi, takich jak karty CRC (Class Responsibility Collaboration) i język UML, - [K_K5]

2. posiada umiejętność dwóch specyficznych typów komunikacji: użytkownik ? analityk systemowy i analityk systemowy ? programista, za pomocą wyspecjalizowanych narzędzi, takich jak język UML, - [K_K5]

Sposoby sprawdzenia efektów kształcenia

Efekty kształcenia przedstawione wyżej weryfikowane są w następujący sposób:

Ocena formująca:

a) w zakresie wykładów:

- na podstawie odpowiedzi na pytania dotyczące materiału omówionego na poprzednich wykładach;

b) w zakresie ćwiczeń:

- na podstawie oceny bieżącego postępu realizacji zadań,

Ocena podsumowująca:

Sprawdzanie założonych efektów kształcenia realizowane jest przez:

- ocenę przygotowania studenta do poszczególnych sesji zajęć laboratoryjnych (sprawdzian

Treści programowe

Program wykładów z przedmiotu obejmuje następujące zagadnienia.

Przesłanki programowania obiektowego wynikające z analizy źródeł kryzysu oprogramowania. Idea nowego paradygmatu programowania, który wspiera tworzenie programów o wysokiej jakości. Poszukiwanie optymalnego języka programowania i metodyk właściwych dla budowy uniwersalnych modułów programowych do wielokrotnego użytku. Związki paradygmatu obiektowego z inżynierią oprogramowania. Metryki jakości architektury programów komputerowych: kohezja i niezależność modułów programowych. Języki programowania o rozwijalnym systemie typów danych. Implementacja pojęcia abstrakcyjnych typów danych. Krótkie przedstawienie historii języków obiektowych.

Modelowanie i analiza obiektowa za pomocą języka UML w zakresie diagramów klas. Poznanie podstawowych konstruktorów modelu obiektowego: klasy, obiektu, zmiennych i operacji klas, relacji generalizacji, związków między klasami. Przykłady prostych modeli fragmentów rzeczywistości. Metodyki dla modelowania i analizy. Transformacja diagramów obiektowych do obiektowych języków programowania.

Definicje podstawowych pojęć obiektowych: obiekt, atrybuty (zmienne) obiektu, metody obiektu, przesyłanie komunikatów wyzwalających wywołania metod obiektów, interfejsy klas, obiekty jako wystąpienia klas, Przykłady definiowania klas obejmujące: definicje konstruktorów i destruktorów klas, operatorów przeciążonych, zmiennych i metod klasowych. Hermetyczność implementacji klas jako mechanizm ograniczania związków między modułami programowymi. Relacja przyjaźni między klasami. Dualne spojrzenie na klasę jako nowy typ danych i jako moduł programowy. Porównanie rozwiązań prostych problemów w sposób funkcjonalny i obiektowy. Implementacja obiektów złożonych i związków między obiektami. Poznanie typów operatorów kopiowania obiektów złożonych. Architektura obiektowej maszyny wirtualnej.

Dziedziczenie klas i relacja podtypu między klasami. Definicja nowych cech klas pochodnych, przesłanianie metod i zmiennych, kowariantna redeklaracja zmiennych i metod oraz implementacja klas abstrakcyjnych. Przegląd topologii sieci dziedziczenia klas w różnych jeżykach programowania. Dziedziczenie wirtualne w języku C++. Dziedziczenie konstruktorów i destruktorów klas. Metodyki stosowania mechanizmu dziedziczenia klas.

Relacje podtypu między klasami. Definiowanie zmiennych polimorficznych i podstawienia polimorficzne. Zwiększanie uniwersalności i elastyczności klas przez stosowanie późnego wiązania komunikatów. Implementacja i przykłady zastosowania mechanizmu późnego wiązania. Późne wiązanie, a mechanizmy refleksji typów danych. Dynamiczne rzutowanie typów danych.

Dalsze zwiększenie stopnia uniwersalności klas przez definiowane klas generycznych. Tworzenie uniwersalnych programów przy zachowaniu silnego typowania danych. Granice stosowalności klas generycznych: generyczność ograniczona i nieograniczona. Typowe przykładu klas generycznych. Wzorce klas w języku C++. Klasy parametryzowane wytycznymi (ang. Policy-based design).

Tworzenie niezawodnych programów komputerowych. Poziomy bezpieczeństwa kodu. Podstawowe strategie tworzenia programów odpornych na występowanie błędów i wyjątków. Metodyki i techniki obsługi wyjątków w językach obiektowych. Definiowanie i zgłaszanie wyjątków. Wyłapywanie wyjątków i ich obsługa. Przykłady zastosowań obsługi wyjątków. Ważność obsługi wyjątków dla modułów programowych przeznaczonych do wielokrotnego użytku.

Metodyka tworzenia oprogramowania zgodnego z jego specyfikacją. Formalna specyfikacja semantyki abstrakcyjnych typów danych. Programowanie przez kontrakt: analiza i programowanie aksjomatów klas oraz warunków początkowych i końcowych metod. Definiowanie i zastosowania dla asercji w językach obiektowych.

Zapewnienie trwałości obiektom przez ich przechowywanie w bazie danych. Funkcjonalność oprogramowania systemowego do odwzorowania obiektowo-relacyjnego (OR/M). Metodyki poprawnego odwzorowania sieci klas w schemat relacyjnej bazy danych.

Przypadki złożonych funkcjonalnie programów o przecinających się aspektach. Rozszerzenie języków obiektowych o jawne definiowanie aspektów. Przekazywanie sterowania między przecinającymi się aspektami. Przykłady zastosowania języków aspektowych.

Powyższe zagadnienia są ilustrowane przykładami w obiektowych językach programowania C++ i Java.

W ramach zajęć laboratoryjnych studenci zapoznają się z dwoma obiektowymi językami programowania: C++ i Java. Ćwiczenia polegają na samodzielnym tworzeniu programów zawierających podstawowe konstrukcje języków obiektowych przedstawianych na wykładach. Dodatkowo, przerabiane są biblioteki systemowe w zakresie: kolekcji, interfejsu graficznego, strumieni, wielowątkowości i serializacji stanu obiektów. Zapoznawanie się z każdym z dwóch języków programowania, kończy się samodzielną realizacją niewielkich projektów obejmujących analizę obiektową i implementację programów.

Cześć wymienionych wyżej treści programowych realizowana jest w ramach pracy własnej studenta.

Metody dydaktyczne:

1. wykład: prezentacja multimedialna, prezentacja ilustrowana przykładami podawanymi na tablicy, pokaz multimedialny,
2. ćwiczenia laboratoryjne: ćwiczenia praktyczne, dyskusja, praca w zespole, pokaz multimedialny, studium przypadków, demonstracja, burza mózgów,

Literatura podstawowa:

1. Programowanie zorientowane obiektowo, Bertrand Mayer, Helion, Warszawa, 2005
2. Metody obiektowe w teorii i praktyce, Ian Graham, WNT, Warszawa, 2004
3. Smalltalk-80: The Language and its Implementation, Goldberg A.J., A.D.Robson, Addison-Wesley, 1983
4. Język C++, Bjarne Stroustrup, WNT, Warszawa, 1994
5. The Java(TM) Programming Language (3rd Edition), Ken Arnold, James Gosling, David Holmes, Addison Wesley Professional, 2000
6. Programowanie obiektowe, Peter Coad, Edward Yourdon, Read Me, 1994
7. Analiza obiektowa, Peter Coad, Edward Yourdon, Read Me, 1994
8. Nowoczesne projektowanie w C++, Andrei Alexandrescu, WNT, 2005
9. http://java.sun.com/docs/books/jls/download/langspec-3.0.pdf
10. http://java.sun.com/docs/books/jls/download/langspec-3.0.pdf
11. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

Literatura uzupełniająca:

1. Simula Begin, Graham M. Birtwistle, O.J. Dahl, B. Myhrhaug, K. Nygaard, 1973

Bilans nakładu pracy przeciętnego studenta

Czynność	Czas (godz.)
1. udział w zajęciach laboratoryjnych:	30
2. przygotowanie do ćwiczeń laboratoryjnych:	7
3. dokończenie (w ramach pracy własnej) sprawozdań z ćwiczeń laboratoryjnych:	10
4. udział w konsultacjach związanych z realizacją procesu kształcenia, w szczególności ćwiczeń laboratoryjnych / projektu	2 8
5. napisanie programu / programów, uruchomienie I weryfikacja (czas poza zajęciami laboratoryjnymi)	6
6. przygotowanie do sprawdzianów / kolokwiów	15
7. udział w wykładach	10
8. zapoznanie się ze wskazaną literaturą / materiałami dydaktycznymi (10 stron tekstu naukowego = 1 godz.), 100 stron	5
9. przygotowanie do zaliczenia wykładów i udział w kolokwium zaliczeniowym	

Obciążenie pracą studenta

forma aktywności	godzin	ECTS
Łączny nakład pracy	78	3
Zajęcia wymagające bezpośredniego kontaktu z nauczycielem	34	1
Zajęcia o charakterze praktycznym	32	1