



<p>1. ma uporządkowaną, podbudowaną teoretycznie wiedzę ogólną w zakresie algorytmów, języków i paradygmatów programowania oraz inżynierii oprogramowania, - [K1st_W4]</p> <p>2. 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, - [K1st_W6]</p> <p>3. zna i rozumie zasady konstrukcji programów komputerowych przetwarzających trwałe obiekty przechowywane w bazie danych oraz zasady konstrukcji programów o złożonej wieloaspektowej architekturze. - [K1st_W6]</p> <p>4. zna podstawowe metody, techniki i narzędzia stosowane przy rozwiązywaniu prostych zadań informatycznych, algorytmów i problemów, budowy systemów komputerowych, implementacji języków programowania oraz inżynierii oprogramowania, - [K1st_W7]</p> <p>5. ma wiedzę niezbędną do transformacji modeli obiektowych fragmentów rzeczywistości do wybranych języków obiektowych, - [K1st_W7]</p> <p>6. ma wiedzę niezbędną do modelowania i analizy obiektowej niedużych fragmentów ?świata rzeczywistego? związanych z różnymi dziedzinami zastosowań, - [K1st_W7]</p> <p>7. 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, - [K1st_W7]</p>
<p><b>Umiejętności:</b></p> <p>1. 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, - [K1st_U9]</p> <p>2. potrafi stworzyć model obiektowy prostego systemu (np. w języku UML) - [K1st_U10]</p> <p>3. potrafi wybrać język programowania odpowiedni do danego zadania programistycznego - [K1st_U10]</p> <p>4. potrafi - zgodnie z zadaną specyfikacją - zaprojektować oraz zrealizować prosty system informatyczny, używając właściwych metod, technik i narzędzi - [K1st_U11]</p> <p>5. ma umiejętność formułowania klas i ich programowania z użyciem przynajmniej dwóch popularnych narzędzi, tj. obiektowych języków programowania: C++ i Java, - [K1st_U11]</p> <p>6. potrafi pracować w grupie - [K1st_U18]</p>
<p><b>Kompetencje społeczne:</b></p> <p>1. rozumie, że w języku programowania dynamicznie się rozwijają i część umiejętności związanych z programowaniem bardzo szybko staje się przestarzała - [K1st_K1]</p> <p>2. ma świadomość ważności poprawnego modelowania rzeczywistości w rozwiązywaniu problemów inżynierskich oraz zna przykłady i rozumie przyczyny wadliwie działających systemów informatycznych - [K1st_K2]</p>

### Sposoby sprawdzenia efektów kształcenia

Efekty kształcenia przedstawione wyżej weryfikowane są w następujący sposób:

Ocena formująca:

- a) w zakresie wykładów:
  - aktywność w trakcie wykładów wykładach
- b) w zakresie ćwiczeń:
  - na podstawie oceny bieżącego postępu realizacji zadań - ćwiczeń oraz projektu zaliczeniowego

Ocena podsumowująca:

- a) w zakresie wykładów weryfikowanie założonych efektów kształcenia realizowane jest przez:
  - ocenę wiedzy i umiejętności wykazanych na zaliczeniu pisemnym
  - omówienie wyników egzaminu
- b) w zakresie laboratoriów weryfikowanie założonych efektów kształcenia realizowane jest przez:
  - ocenianie ciągle, na każdych zajęciach (odpowiedzi ustne), premiowanie przyrostu umiejętności posługiwania się poznanymi zasadami i metodami oraz narzędziami programowymi
  - ocenę z realizacji projektów zaliczeniowych

### 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 metodok 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 kart CRC i języka UML w zakresie diagramów klas i obiektów oraz diagramów współpracy. 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 (zmiennie) obiektu, metody obiektu, przesyłanie komunikatów wywołujących 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, przestanie metod i zmiennych, kowariantna redeklaracja zmiennych i metod oraz implementacja klas abstrakcyjnych. Przegląd topologii sieci dziedziczenia klas w różnych językach 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 definiowanie klas generycznych. Tworzenie uniwersalnych programów przy zachowaniu silnego typowania danych. Granice stosowalności klas generycznych: generyczność ograniczona i nieograniczona. Typowe przykłady klas generycznych. Wzorce klas w języku C++.

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 realcyjnej bazy danych.

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

Powyższe zagadnienia są ilustrowane przykładami w obiektowych językach programowania: C++, Java, C# i Eiffel.

W ramach zajęć laboratoryjnych studenci zapoznają się dogłębnie 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. Zapoznanie się z każdym z dwóch języków programowania, kończy się samodzielną realizacją niewielkich projektów obejmujących analizę obiektową i implementację programów.

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 Meyer, Helion, Warszawa, 2005
2. Metody obiektowe w teorii i praktyce, Ian Graham, WNT, Warszawa, 2004
3. Język C++, Bjarne Stroustrup, WNT, Warszawa, 1994
4. The Java(TM) Programming Language (3rd Edition), Ken Arnold, James Gosling, David Holmes, Addison Wesley Professional, 2000
5. Programowanie obiektowe, Peter Coad, Edward Yourdon, Read Me, 1994
6. Analiza obiektowa, Peter Coad, Edward Yourdon, Read Me, 1994
7. Nowoczesne projektowanie w C++, Andrei Alexandrescu, WNT, 2005

<b>Literatura uzupełniająca:</b>		
1. Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów. Robert C. Martin, Helion 2018		
2. <a href="http://java.sun.com/docs/books/jls/download/langspec-3.0.pdf">http://java.sun.com/docs/books/jls/download/langspec-3.0.pdf</a>		
3. <a href="http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf">http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf</a>		
<b>Bilans nakładu pracy przeciętnego studenta</b>		
<b>Czynność</b>	<b>Czas (godz.)</b>	
1. udział w zajęciach laboratoryjnych:	30	
2. przygotowanie do ćwiczeń laboratoryjnych:	5	
3. dokończenie (w ramach pracy własnej) sprawozdań z ćwiczeń laboratoryjnych:	5	
4. udział w konsultacjach związanych z realizacją procesu kształcenia, w szczególności ćwiczeń laboratoryjnych / projektu	2	
	5	
5. napisanie programu / programów, uruchomienie i weryfikacja (czas poza zajęciami laboratoryjnymi)	3	
6. przygotowanie do sprawdzianów / kolokwium	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		
<b>Obciążenie pracą studenta</b>		
<b>forma aktywności</b>	<b>godzin</b>	<b>ECTS</b>
Łączny nakład pracy	80	3
Zajęcia wymagające bezpośredniego kontaktu z nauczycielem	47	2
Zajęcia o charakterze praktycznym	45	2