НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГУМАНИТАРНЫЙ
УНИВЕРСИТЕТ ПРОФСОЮЗОВ»

Кафедра Информатики и математики

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ
Объектно-ориентированное программирование

Основная профессиональная образовательная программа
высшего образования программы бакалавриата
по направлению

09.03.03 «Прикладная информатика»

Профиль подготовки «Прикладная информатика в экономике»

Квалификация:
Бакалавр

Согласовано:
Руководитель ОПОП по направлению
09.03.03 – «Прикладная информатика»
Профиль «Прикладная информатика
в экономике»
_____/Путькина Л.В.

Рассмотрена и утверждена на заседании кафедры

«01» июня 2020 г., протокол № 10

Зав. кафедрой _____/Путькина Л.В.

Рекомендована решением
Методического совета

«15» июня 2020 г., протокол № 10

Секретарь МС _____Волкова А.М.

Авторы-разработчики:

_____/Мокрый В.Ю.

Санкт-Петербург

СТРУКТУРА

1. Цель и задачи освоения дисциплины
2. Место дисциплины в структуре ОПОП
3. Требования к результатам освоения дисциплины
4. Тематический план изучения дисциплины
5. Содержание разделов и тем дисциплины
6. План подгрупповых (лабораторных) занятий
7. Образовательные технологии
8. План самостоятельной работы студентов
9. Контроль знаний по дисциплине
10. Учебно-методическое и информационное обеспечение дисциплины
11. Материально-техническое обеспечение дисциплины

Учебно-методическое обеспечение самостоятельной работы студентов
1. Методические рекомендации по организации самостоятельной работы студентов
2. Методические рекомендации по подготовке к практическим (подгрупповым) занятиям
3. Методические рекомендации по написанию контрольных работ
4. Методические рекомендации по написанию курсовой работы

Оценочные и методические материалы
1. Перечень компетенций с указанием этапов их формирования в процессе освоения образовательной программы
2. Описание показателей и критериев оценивания компетенций, шкал оценивания
3. Типовые контрольные задания и методические материалы, процедуры оценивания знаний, умений и навыков

Глоссарий

Методические рекомендации для преподавателя по дисциплине

1. Цель и задачи освоения дисциплины:

Целью изучения дисциплины является изучение концепции объектно-ориентированного программирования (ООП), основных ее понятий, свойств, методики анализа и проектирования объектно-ориентированных программ, способов составления объектно-ориентированных программ на языке программирования С++.

Основные **задачи** дисциплины - получение систематических знаний об объектно-ориентированных системах, средствах и технологиях; приобретение специальных знаний и умений, необходимых для формирования навыков проектирования, программирования и отладки объектно-ориентированные программ.

2. Место дисциплины в структуре ОПОП

Междисциплинарные связи с обеспечиваемыми (последующими) дисциплинами

№ п/п	Наименование обеспечиваемых (последующих) дисциплин	№ № разделов данной дисциплины, необходимых для изучения обеспечиваемых дисциплин			
		1	2	3	4
1.	Интеллектуальные информационные системы	+	+		
2.	Проектирование информационных систем	+	+		
3.	Проектный практикум	+	+	+	+
4.	Высокоуровневые методы информатики и программирования	+	+	+	+

3. Требования к результатам освоения дисциплины

Процесс изучения дисциплины направлен на формирование следующих компетенций с установленными к ним индикаторами:

Компетенции и индикаторы их достижения

Категория компетенций	Код и наименование компетенции	Код и наименование индикатора достижения компетенции
Проектная деятельность	ПК-3. Способен проектировать ИС по видам обеспечения	ПК-3.1. Знает концепцию объектно-ориентированного программирования, методику анализа и проектирования объектно-ориентированных программ, основные понятия, синтаксис и семантику конструкций языка программирования.
		ПК-3.2. Умеет проектировать, программировать и отлаживать объектно-ориентированные программы
		ПК-3.3. Владеет навыками объектно-ориентированного программирования, отладки и тестирования программного обеспечения навыками программирования в современных средах.

4. Тематический план изучения дисциплины
См. приложение
5. Содержание разделов и тем дисциплины

РАЗДЕЛ 1 (Модуль 1). Объектный подход

Тема 1. Основные понятия дисциплины

Предмет дисциплины, ее структура и содержание. Цели и содержание дисциплины для подготовки студентов по выбранному направлению. Структура и план учебной деятельности студентов, рекомендуемая литература. Связь дисциплины с предшествующими и последующими дисциплинами и ее место в подготовке специалиста по прикладной информатике. Краткие сведения об истории развития и становления проблематики дисциплины. Стандарты, используемые в ООП.

Семантика и интерпретация языка программирования. Основные положения объектного подхода к разработке программ. Принципы объектного подхода, абстрагирование, ограничение доступа, модульность, иерархия, типизация, параллелизм, устойчивость.

Понятие объекта и класса. Инкапсуляция и спецификация правил доступности элементов класса. Конструкторы и деструкторы. Наследование. Иерархия классов. Одиночное и множественное наследование. Способы реализации множественного наследования, их достоинства и недостатки. Отношения между объектами и/или классами. Полиморфизм. Ранее и позднее связывание.

Управление последовательностью действий в объектно-ориентированной программе. Объект и процесс. Инициализация и взаимодействие объектов и процессов. Сообщения. Реализация механизмов посылки сообщений. Примеры функционирования объектно-ориентированной программы.

Формальные модели объектов и классов: автоматная и алгебраическая модели объектов, исчисления типов.

Тема 2. Введение в объектно-ориентированный анализ и проектирование

Системный анализ. Принципы объектно-ориентированного анализа и их обсуждение. Основные определения: система, домен, подсистема, элемент, связи, среда. Структура системы, декомпозиция, иерархия элементов. Процессы в системе и потоки информации. Исследование действий. Построение моделей доменов и подсистем, связей и взаимодействия подсистем, взаимодействия объектов, событий, процессов, потоков данных, действий.

Описание классов и их взаимосвязей с использованием унифицированного языка моделирования UML. Моделирование статической структуры системы. Диаграммы классов. Механизм пакетов. Моделирование поведения системы. Диаграммы взаимодействия (диаграммы последовательности и кооперативные диаграммы). Диаграммы состояний. Диаграммы действий. Моделирование реализации системы. Диаграммы компонентов. Диаграммы размещения.

РАЗДЕЛ 2 (Модуль 2). Объектно-ориентированное программирование

Тема 3. Классы

Краткие сведения об истории развития и текущем состоянии языка программирования C++. Реализация парадигм программирования на C++.

Способы реализации инкапсуляции. Определение класса. Личная и общая части определения класса. Функции-элементы класса и функции-друзья. Объекты класса. Создание и уничтожение объектов класса. Конструкторы и деструкторы.

Интерфейс и реализация контейнерных классов для моделирования структур данных. Статические члены объектов класса. Вложенные и локальные классы. Примеры описания и использования классов.

Базовый и производный классы. Функции-элементы и функции-друзья. Правила

доступа к элементам производного класса. Конструкторы и деструкторы. Иерархия классов. Одиночное и множественное наследование. Виртуальные базовые классы. Особенности доступа при множественном наследовании. Правила преобразования указателей.

Виртуальные функции. Таблицы виртуальных функций. Распределение таблиц виртуальных функций в многофайловых программах. Чистые виртуальные функции и абстрактные базовые классы.

Полиморфные контейнерные классы, итераторы и аппликаторы. Абстрактные и конкретные контейнерные классы.

Виды классов: конкретный тип, абстрактный тип, узловой класс, интерфейсный класс и другие.

Инициализация объектов. Отличия инициализации от присваивания. Инициализация баз и членов. Полный объект конечного производного класса.

Примеры описания и использования классов с наследованием.

Тема 4. Шаблоны и обработка исключительных ситуаций

Шаблоны классов и функций. Наследование шаблонных классов. Правила отождествления параметров шаблона.

Применение шаблонных классов для создания контейнерных классов. Анализ вариантов шаблонных классов на примере шаблона вектора с операцией сортировки.

Анализ различных моделей обработки исключительных ситуаций.

Стандартные средства контроля подтверждений. Проверка предусловий и постусловий, вычисления инвариантов. Контроль асинхронных событий. Сигналы.

Реализация модели обработки синхронных ситуаций с завершением в C++. Возбуждение ситуации, Описание блоков с контролем и реакций на ситуации. Система классов для описания исключительных ситуаций. Обработка исключительных ситуаций при обработке исключительных ситуаций. Примеры контроля ситуаций в контейнерных классах.

Разработка объектно-ориентированных программ на C++. Особенности разработки объектно-ориентированных программ на C++. Проектирование классов. Механизмы функционирования программ. Инструментальные программные средства для C++. Библиотеки классов для C++.

Тенденции развития ООП. Пути дальнейшего изучения вопросов ООП студентами.

6. План практических (лабораторных) занятий

№ п/п	Наименование темы дисциплины	Наименование и содержание подгрупповых (лабораторных) занятий, литература для подготовки к занятиям	Формируемые компетенции	Формы контроля усвоения знаний
1.	**Тема 3.** Классы	Выполнение задания «Программирование классов с использованием наследования». Программирование системы классов с использованием наследования. Описание полей, алгоритмизация и программирование методов Литература: 1-5.	ПК-3	Устный опрос Проверка файлов общих и индивидуальных заданий, устный опрос, оперативное исправление ошибок
2.	**Тема 4.** Шаблоны и обработка исключительных ситуаций	Программирование шаблонов контейнерных классов Проектирование и программирование приложений. Литература: 1-5.	ПК-3	Проверка индивидуальных заданий

7. Образовательные технологии

При проведении учебных занятий по дисциплине для успешного освоения применяются различные образовательные технологии, которые обеспечивают развитие навыков командной работы, межличностной коммуникации, принятия решений, лидерских качеств.

Методы / Формы	Лекции (Л)	Семинарские занятия (С)
Диалого-дискуссионное обсуждение проблем	+	+
Работа в команде		+
Поисковый метод	+	+
Проектный метод		+

8. План самостоятельной работы студентов

№ п/п	Содержание самостоятельной работы студентов	Формируемые компетенции	Форма отчетности студента
1	Изучение литературы по теме	ПК-3	Представления обоснованной и развернутой структуры работы
2	Подготовка к практическому занятию	ПК-3	Представление выполненных работ
3	Выполнение заданий лабораторных работ	ПК-3	Отчеты о выполнении лабораторных работ
4	Выполнение дополнительных заданий для самостоятельной работы	ПК-3	Отчеты о выполнении самостоятельных работ, тест-опрос

9. Контроль знаний по дисциплине

По дисциплине предусмотрены текущий контроль и промежуточная аттестация.

Текущий контроль успеваемости студента – одна из составляющих оценки качества усвоения образовательных программ. Текущий контроль проводится в течение семестра в результате проверки выполнения циклов лабораторных работ.

Промежуточная аттестация проводится по окончании изучения дисциплины в виде зачета. Вопросы к промежуточной аттестации сформулированы в **Оценочных и методических материалах**.

10. Учебно-методическое и информационное обеспечение дисциплины:

а) Основная литература

1. Зыков С. В. Программирование : учебник и практикум для академического бакалавриата / С. В. Зыков. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/433432
2. Кувшинов Д. Р. Основы программирования : учебное пособие для вузов / Д. Р. Кувшинов. — Москва : Издательство Юрайт, 2019 ; Екатеринбург : Изд-во Урал. ун-та. — Режим доступа: https://urait.ru/bcode/441475
3. Трофимов В. В. Алгоритмизация и программирование : учебник для академического бакалавриата / В. В. Трофимов, Т. А. Павловская ; под редакцией В. В. Трофимова. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/423824
4. Черпаков И. В. Основы программирования : учебник и практикум для прикладного бакалавриата / И. В. Черпаков. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/433423
5. Тузовский А. Ф. Объектно-ориентированное программирование : учебное пособие для прикладного бакалавриата / А. Ф. Тузовский. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/434045

б) Дополнительная литература:

1. Семакин И.Г. Программирование, численные методы и математическое моделирование : учебное пособие / Семакин И.Г., Русакова О.Л., Тарунин Е.Л., Шкарапута А.П. — Москва : КноРус, 2017. — Режим доступа: https://book.ru/book/920222
2. Хлебников А. А. Информационные технологии : учебник / Хлебников А.А. — Москва : КноРус, 2018. — Режим доступа: https://book.ru/book/927689

3. Гниденко И. Г. Технологии и методы программирования : учебное пособие для прикладного бакалавриата / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/433611
4. Казанский А. А. Программирование на Visual C# : учебное пособие для вузов / А. А. Казанский. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/447372
5. Нагаева И. А. Программирование: Delphi : учебное пособие для академического бакалавриата / И. А. Нагаева, И. А. Кузнецов ; под редакцией И. А. Нагаевой. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/444273
6. Огнева М. В. Программирование на языке C++: практический курс : учебное пособие для бакалавриата и специалитета / М. В. Огнева, Е. В. Кудрина. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/438987
7. Подбельский В. В. Программирование. Базовый курс C# : учебник для бакалавриата и специалитета / В. В. Подбельский. — Москва : Издательство Юрайт, 2019. — Режим доступа: https://urait.ru/bcode/439068

в) Лицензионное программное обеспечение

Перечень лицензионного и свободно распространяемого программного обеспечения:

1. Семейство программ Microsoft Office Standart Russian (Включает набор продуктов: Word, Excel, PowerPoint, Publisher, Outlook);
2. Mirapolis Virtual Room;
3. Антиплагиат;
4. КонсультантПлюс
5. Project Expert 7
6. Prime Expert
7. FineModel Expert
8. Обеспечено доступом к сети «Интернет» и электронной информационно-образовательной среде СПбГУП.

г) Современные профессиональные базы данных и информационные справочные системы

1. Официальный сайт СПбГУП: http://www.gup.ru/
2. Электронно-библиотечная система СПбГУП,
3. Системы поддержки самостоятельной работы СПбГУП: http://edu.gup.ru/
4. Справочная правовая система «Консультант плюс» http://www.consultant.ru
5. Российское образование http://www.edu.ru/
6. Единое окно доступа к образовательным ресурсам http://window.edu.ru/
7. Электронно-библиотечная система http://e.lanbook.com/

11. Материально-техническое обеспечение дисциплины

Аудиторный фонд, компьютерные классы, видео-залы, фонды Научной библиотеки, методические ресурсы кафедры

Изучение дисциплины инвалидами и обучающимися с ограниченными возможностями здоровья осуществляется с учетом особенностей психофизического развития, индивидуальных возможностей и состояния здоровья обучающихся.

Учебная аудитория для проведения занятий лекционного типа, семинарского типа, групповых занятий, текущего контроля и промежуточной аттестации, ГИА
(учебная аудитория «Компьютерный класс»)
Лаборатория «ООП Экономика», «ООП Менеджмент»
Специализированная мебель:
Маркерная доска– 1 шт.
Стол– 20 шт.

Стул- 20 шт.
Стол для переговоров- 2 шт.
Дополнительные стулья-10 шт.
Стол преподавателя– 1 шт.
Стул преподавателя– 1 шт. Информационная доска- 3 шт.
Перечень оборудования и технические средства обучения:
Экран– 1 шт.
Проектор– 1 шт.
Компьютер– 20 шт.
Компьютер преподавателя– 1 шт.
Сканер– 1 шт.
Колонки– 1 шт.
Камера– 1 шт.

УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

1. Методические рекомендации по организации самостоятельной работы студентов

Самостоятельная работа в высшем учебном заведении является важной организационной формой индивидуального изучения студентами программного материала. Эти слова особенно актуальны в наше время, когда в педагогике высококвалифицированных специалистов широко используется дистанционное обучение, предполагающее значительную самостоятельную работу студента на основе рекомендаций преподавателя.

2. Методические рекомендации по подготовке к практическим (подгрупповым) занятиям

Подгрупповые занятия — важная форма учебного процесса. Они способствуют закреплению и углублению знаний, полученных студентами на лекциях и в результате самостоятельной работы над научной и учебной литературой и нормативными источниками. Они призваны развивать самостоятельность мышления, умение делать выводы, связывать теоретические положения с практикой. На занятиях вырабатываются необходимые каждому специалисту навыки и умения, логика, культура профессиональной речи. Кроме того, семинары — это средство контроля преподавателей за самостоятельной работой студентов, они непосредственно влияют на уровень подготовки к итоговым формам отчетности — зачетам и экзаменам. В выступлении на занятии должны содержаться следующие элементы:

- четкое формулирование соответствующего теоретического положения в виде развернутого определения;
- приведение и раскрытие основных черт, признаков, значения и роли изучаемого явления или доказательства определенного теоретического положения;
- подкрепление теоретических положений конкретными фактами.

Для качественного и эффективного изучения дисциплины необходимо овладение навыками работы с книгой, воспитание в себе стремления и привычки получать новые знания из научной и иной специальной литературы. Без этих качеств не может быть настоящего специалиста ни в одной области деятельности.

Читать и изучать, следует, прежде всего, то, что рекомендуется к каждой теме программой, планом семинарских занятий, перечнем рекомендуемой литературы.

Когда студент приступает к самостоятельной работе, то он должен проявить инициативу в поиске специальных источников. Многие новейшие научные положения появляются, прежде всего, в статях, опубликованных в журналах.

Надо иметь в виду, что в каждом последнем номере издаваемых журналов публикуется библиография всех статей, напечатанных за год, это облегчает поиск нужных научных публикаций.

Для качественного и эффективного изучения дисциплины необходимо овладение навыками работы с книгой, воспитание в себе стремления и привычки получать новые знания из научной и иной специальной литературы. Без этих качеств не может быть настоящего специалиста ни в одной области деятельности.

Читать и изучать, следует, прежде всего, то, что рекомендуется к каждой теме программой, планом семинарских занятий, перечнем рекомендуемой литературы.

Когда студент приступает к самостоятельной работе, то он должен проявить инициативу в поиске специальных источников. Многие новейшие научные положения появляются, прежде всего, в статях, опубликованных в журналах.

Надо иметь в виду, что в каждом последнем номере издаваемых журналов публикуется библиография всех статей, напечатанных за год, это облегчает поиск нужных научных публикаций.

Работа с научной литературой, в конечном счете, должна привести к выработке у студента умения самостоятельно размышлять о предмете и объекте изучения, которое должно проявляться:

- в ясном и отчетливом понимании основных понятий и суждений, содержащихся в публикации, разработке доказательств, подтверждающих истинность тех или иных положений;
- в понимании студентами обоснованности и целесообразности, приводимых в книге и статье примеров, поясняющих доказательства и выводы автора. При этом будет уместно, если студент самостоятельно приведет дополнительные примеры к этим выводам;
- в отделении основных положений от дополнительных, второстепенных сведений;
- в способности студента критически разобраться в содержании публикации, определить свое отношение, дать общую оценку, характеристику.

Самостоятельная работа студентов предполагает тщательное освоение студентами учебной и научной литературы по изучаемой дисциплине. Изучение научной литературы – это серьезная работа, которую следует проводить по этапам:

- общее ознакомление с источником в целом по его оглавлению;
- беглый просмотр всего содержания;
- последовательное чтение материала;
- выборочное чтение какой-либо части текста;
- выписка материала, представляющего интерес;
- критическая оценка выписанного материала, его сравнение с другими точками зрения.

При самостоятельном изучении *основной рекомендуемой литературой* студентам необходимо обратить внимание на выделение основных понятий, их определение, узловых положений, представленных в изучаемом тексте.

В качестве информационно-справочного материала можно использовать глоссарий, в котором содержится перечень и определение ключевых понятий документооборота.

Нужно обратить внимание на образно-схематическое представление излагаемого материала в виде рисунков, схем, графиков и диаграмм, присутствующих в изучаемом тексте. Они способствуют более быстрому восприятию и запоминанию учебного материала.

Для контроля усвоения содержания темы в конце соответствующих глав и параграфов учебников и учебных пособий обычно дается перечень контрольных вопросов, на которые студент должен уметь дать четкие и конкретные ответы.

Самостоятельная работа с дополнительной литературой предполагает умение студента выделять в ней необходимый аспект изучаемой темы (сто в данном материале относится непосредственно к изучаемой теме и основным вопросам). к дополнительной литературе как правило относят широкий спектр текстов (учебных, научных, научно-популярных, художественных, публицистических и др.), в которых изучаемых вопрос рассматривается частично либо с нетрадиционной точки зрения.

Знакомство с дополнительной литературой, несомненно, обогащает знания и расширяет научный кругозор студентов.

Дополнительную литературу нужно обрабатывать после основной литературы и исследовать ее надо комплексно, рассматривая различные точки зрения на исследуемый вопрос.

При самостоятельном освоении информационных технологий и работе с применением программных продуктов следует использовать методическую литературу, содержащую детальное описание последовательности и технологии выполнения базовых операций.

Для закрепления теоретических знаний и практических навыков рекомендуется отрабатывать технологии на контрольном примере.

Общими правилами самостоятельного изучения элементов курса являются:

- изучение теоретических основ соответствующих разделов программы, изложенных в лекционном материале и литературных источниках;
- обучение на контрольных примерах (с заранее известным результатом – правильным ответом);
- эффективное использование системы помощи (HELP) применяемого программного обеспечения;
- заданная последовательность в изучении и освоении курса («от простого к сложному» — от отдельных элементарных операций к комплексным);
- использование индивидуальных заданий, уменьшающих вероятность списывания недобросовестными студентами;
- подготовка к ответам на контрольные вопросы.

Основные виды самостоятельной работы студентов:

1. ***Предварительная подготовка к занятиям, в том числе и к тем, на которых будет изучаться новый материал***. Такая подготовка предполагает изучение учебной программы, установление связи с ранее полученными знаниями, выделение наиболее значимых и актуальных проблем, на изучение которых следует обратить особое внимание, ознакомление с материалом учебника. Также рекомендуется работа со словарями по новым экономическим терминам.

2. ***Прослушивание и восприятие лекций***, что предполагает осмысление учебной информации, сообщаемой преподавателем, ее обобщение и краткую запись, своевременную доработку конспектов лекций. Основная задача студентов на лекционных занятиях – выделить главное в изучаемом материале, а также уяснить связь данной темы с логикой дисциплины в целом и предыдущих лекций. На лекционных занятиях приветствуются уточняющие вопросы со стороны студентов.

3. ***Подбор, изучение, анализ и конспектирование рекомендуемой литературы***. Данная форма самостоятельной работы позволяет углубить знания в рамках определенной темы учебной программы. Основные рекомендации для эффективного освоения дополнительного материала: 1)до чтения: осмысление заголовка, анализ оглавления и аннотации; 2)по ходу чтения: старайтесь понять все слова и предложения в тексте, выделить важные и ключевые из них; задавайте вопросы самому себе по содержанию прочитанного, стройте свои предположительные ответы и сверяйте их с текстом; по ходу чтения попытайтесь осознать, что Вам не понятно, в чем возникают сомнения; делайте выписки, выделяйте главные мысли, составляйте схемы, графики, таблицы; 3)после прочтения: сформулируйте главную мысль прочитанного; прочитайте повторно наиболее сложные для Вас части текста; задайте вопросы себе по всему тексту; составьте конспект прочитанного; через время проверьте себя. Предварительное изучение рекомендуемой литературы позволяет отобрать необходимую учебную информацию и выяснить, по каким вопросам следует подобрать дополнительные литературные источники.

4. ***Подготовка к семинарским, практическим занятиям и зачету***. Подготовку к семинару и практическому занятию лучше начинать сразу же после лекции по данной теме. Прежде всего, следует доработать текст лекции по соответствующей теме, внимательно изучить план семинара (практического занятия), содержания основных учебных вопросов, выносимых для обсуждения, а также список рекомендованной литературы и дополнительные задания, которые могут быть даны преподавателем. Далее необходимо подобрать просмотреть литературу, которая рекомендована для подготовки к занятию. Это даст возможность выбрать те источники, где имеются ответы на поставленные учебные вопросы. Затем более внимательно ознакомиться с содержанием книги или статьи, отметить те части текста, в которых вопросы семинара раскрываются наиболее глубоко и подробно. В ходе углубленного чтения выделяются главные мысли, составляются краткие конспекты, тезисы своих будущих выступлений. Конспекты, которые отражают содержание вопросов семинарского и практического занятия, лучше

всего вести в той же тетради, в которой конспектируются лекции по дисциплине. Концентрация всей информации по проблеме в одном месте позволяет студентам активно участвовать в обсуждении вопросов семинара. В дальнейшем такие записи значительно облегчат подготовку к зачету. Подготовка к семинарским, практическим занятиям, зачету не сводится только к поиску ответов на поставленные вопросы. Любая теоретическая проблема должна быть осмыслена с точки зрения ее связи с реальной жизнью и возможностью реализации на практике. По каждому вопросу студент должен быть готов высказать и свою точку зрения.

Особую роль самостоятельная работа играет при освоении материала данной дисциплины. Научить программированию нельзя. Программированию можно только научиться. Преподаватель в этом процессе самообучения играет роль консультанта, более опытного коллеги и заказчика одновременно.

Дисциплина обеспечивает развитие базовых знаний и навыков в области информатики и программирования.

- в способности студента критически разобраться в содержании публикации, определить свое отношение к ней в целом, дать ей общую оценку, характеристику.

Основные понятия дисциплины

Семантика и интерпретация языка программирования. Основные положения объектного подхода к разработке программ. Принципы объектного подхода - абстрагирование, ограничение доступа, модульность, иерархия, типизация, параллелизм, устойчивость.

Понятие объекта и класса. Инкапсуляция и спецификация правил доступности элементов класса. Конструкторы и деструкторы. Наследование. Иерархия классов. Одиночное и множественное наследование. Способы реализации множественного наследования, их достоинства и недостатки. Отношения между объектами и/или классами. Полиморфизм. Ранее и позднее связывание.

Управление последовательностью действий в объектно-ориентированной программе. Объект и процесс. Инициализация и взаимодействие объектов и процессов. Сообщения. Реализация механизмов посылки сообщений. Примеры функционирования объектно-ориентированной программы.

Формальные модели объектов и классов: автоматная и алгебраическая модели объектов, исчисления типов.

При изучении этой темы следует обратить особое внимание источники и объективные предпосылки объектного подхода в программировании. Должно быть достигнуто полное понимание принципов объектного подхода и формулировок основных определений. Более детального освоения требуют вопросы применения параллелизма в объектно-ориентированных программах. Можно более подробно изучить мониторы Хоара, семафоры Дейкстры и механизм рандеву из языка программирования Ада. Дополнительная информация о формальных подходах в объектно-ориентированном программировании м.б. получена в результате изучения рекомендованных источников и интернет-ресурсов.

Перечень контрольных вопросов.

1. Приведите основных характеристики следующих таких принципов объектного подхода, как абстрагирование, ограничение доступа, модульность, иерархия, типизация параллелизм, устойчивость.
2. Как связаны инкапсуляция и спецификации правил доступности элементов класса.
3. Виды конструкторов классов и правила их автоматического создания.
4. Виды наследования. Проблемы множественного наследования.
5. Способы реализации множественного наследования, их достоинства и недостатки.
6. Полиморфизм и его взаимосвязь с типизацией и технологиями раннего и позднего связывания.
7. Взаимоотношение понятий объект и процесс. Технологии инициализации и

взаимодействия объектов и процессов.

8. Реализация механизмов посылки сообщений.
9. Возможности и недостатки автоматной и алгебраической моделей объектов, исчисления типов.

Введение в объектно-ориентированный анализ и проектирование

Основные определения: система, домен, подсистема, элемент, связи. Структура системы, декомпозиция, иерархия элементов. Процессы в системе и потоки информации. Исследование действий. Построение моделей доменов и подсистем, связей и взаимодействия подсистем, взаимодействия объектов, событий, процессов, потоков данных, действий.

Описание классов и их взаимосвязей с использованием унифицированного языка моделирования UML. Моделирование статической структуры системы. Диаграммы классов. Механизм пакетов. Моделирование поведения системы. Диаграммы взаимодействия (диаграммы последовательности и кооперативные диаграммы). Диаграммы состояний. Диаграммы действий. Моделирование реализации системы. Диаграммы компонентов. Диаграммы размещения.

При изучении этой темы обратите особое внимание на понимание общих принципов анализа и проектирования программного обеспечения. Важно освоить все основные элементы нотации UML практически опробовать средства создания всех типов диаграмм UML. По дополнительным источникам рекомендуется составить список основных стереотипов языка и определить наиболее распространенные ограничения. В дальнейшем следует для всех разрабатываемых программ строить соответствующие диаграммы UML.

Перечень контрольных вопросов.

1. Определения системы, домена, подсистемы, элемента, связи, среды. Последовательность этапов анализа по методике Шлеер – Меллор.
2. Описание стереотипов классов и ограничений связей в унифицированном языке моделирования UML.
3. Моделирование статической структуры системы. Диаграммы классов. Механизм пакетов. Примеры диаграмм.
4. Принципиальные отличия диаграмм динамического типа в моделировании поведения системы. Диаграммы взаимодействия. Диаграммы состояний. Диаграммы действий..
5. Применение графических стереотипов в моделировании реализации системы. Диаграммы компонентов. Диаграммы размещения.

Классы

Способы реализации инкапсуляции. Определение класса. Личная и общая части определения класса. Функции-элементы класса и функции-друзья. Объекты класса. Создание и уничтожение объектов класса. Конструкторы и деструкторы.

Интерфейс и реализация контейнерных классов для моделирования структур данных. Статические члены объектов класса. Вложенные и локальные классы. Примеры описания и использования классов.

Базовый и производный классы. Функции-элементы и функции-друзья. Правила доступа к элементам производного класса. Конструкторы и деструкторы. Иерархия классов. Одиночное и множественное наследование. Виртуальные базовые классы. Особенности доступа при множественном наследовании. Правила преобразования указателей.

Виртуальные функции. Таблицы виртуальных функций. Распределение таблиц виртуальных функций в многофайловых программах. Чистые виртуальные функции и абстрактные базовые классы.

Полиморфные контейнерные классы, итераторы и аппликаторы. Абстрактные и

конкретные контейнерные классы.

Виды классов: конкретный тип, абстрактный тип, узловой класс, интерфейсный класс и другие.

Инициализация объектов. Отличия инициализации от присваивания. Инициализация баз и членов. Полный объект конечного производного класса.

Примеры описания и использования классов с наследованием.

Эта тема является определяющей при формировании навыков объектно-ориентированного программирования и при планировании самостоятельной работы должна рассматриваться совместно с темой 4. Дополнительные рекомендации аналогичны предыдущей теме – как можно больше самостоятельно написанных и отлаженных программ-примеров.

Перечень контрольных вопросов.

1. Примеры определение класса. Личная и общая части определения класса.
2. Функции-элементы класса и функции-друзья. Объекты класса.
3. Интерфейс и реализация контейнерных классов для моделирования структур данных.
4. Статические члены объектов класса.
5. Вложенные и локальные классы.
6. Базовый и производный классы. Функции-элементы и функции-друзья.
7. Виртуальные базовые классы. Особенности доступа при множественном наследовании. Правила преобразования указателей.
8. Виртуальные функции. Таблицы виртуальных функций.
9. Чистые виртуальные функции и абстрактные базовые классы.
10. Полиморфные контейнерные классы, итераторы и аппликаторы.
11. Абстрактные и конкретные контейнерные классы. Виды классов: конкретный тип, абстрактный тип, узловой класс, интерфейсный класс.
12. Инициализация баз и членов. Полный объект конечного производного класса.

Шаблоны и обработка исключительных ситуаций

Шаблоны классов и функций. Наследование шаблонных классов. Правила отождествления параметров шаблона.

Применение шаблонных классов для создания контейнерных классов. Анализ вариантов шаблонных классов на примере шаблона вектора с операцией сортировки.

Анализ различных моделей обработки исключительных ситуаций.

Стандартные средства контроля подтверждений. Проверка предусловий и постусловий, вычисления инвариантов. Контроль асинхронных событий. Сигналы.

Реализация модели обработки синхронных ситуаций с завершением в C++. Возбуждение ситуации, Описание блоков с контролем и реакций на ситуации. Система классов для описания исключительных ситуаций. Обработка исключительных ситуаций при обработке исключительных ситуаций. Примеры контроля ситуаций в контейнерных классах.

В отличие от материала темы 2, для тем 3 и 4, лабораторных работ никогда не бывает достаточно. Поэтому следует использовать любую возможность для самостоятельной практики в написании программ. Большую пользу приносит и изучение текстов готовых программ, которое, впрочем, не может заменить самостоятельного программирования. Синтаксис и определения конкретных структур языка программирования проще всего запоминаются при написании программ.

Перечень контрольных вопросов.

1. Шаблоны классов и функций. Наследование шаблонных классов. Правила отождествления параметров шаблона.
2. Примеры применения шаблонных классов для создания контейнерных классов.
3. Стандартные средства контроля подтверждений. Проверка предусловий и постусловий, вычисления инвариантов.

4. Контроль асинхронных событий. Особенности программирования сигналов.
5. Возбуждение ситуации, Описание блоков с контролем и реакций на ситуации. Система классов для описания исключительных ситуаций.
6. Обработка исключительных ситуаций при обработке исключительных ситуаций.

2. Методические рекомендации по подготовке к практическим (подгрупповым) занятиям

«Программирование классов с использованием наследования»

Для успешного выполнения заданий рекомендуется самостоятельно проработать следующий материал.

Определение класса. Личная и общая части определения класса. Функции-элементы класса и функции-друзья. Объекты класса. Создание и уничтожение объектов класса. Конструкторы и деструкторы.

«Программирование контейнерных классов»

Все задания построены методически однотипно: вначале предлагается разобрать готовое решение задачи, набрать и отладить одинаковую для всех студентов группы задачу; затем — самостоятельно решить несколько однотипных задач по индивидуальному заданию. Данный подход позволяет устранить зависимость сроков выполнения заданий от индивидуальных способностей студента к программированию. Как общие, так индивидуальные задания студент может выполнять самостоятельно. В последнем случае на преподавателя возлагается дополнительная ответственность по контролю самостоятельного выполнения заданий и проверке степени усвоения учащимся материалов соответствующей лабораторной работы.

«Проектирование и программирование приложений»

Для успешного выполнения заданий рекомендуется самостоятельно проработать возможности визуального программирования интерфеса приложений в используемой системе программирования на C++. Элементы интерфейса разрабатываемых приложений для разных студентов должны отличаться по дизайну.

3. Методические рекомендации по написанию контрольных работ

Важнейшей формой учебной отчетности студента является **контрольная работа**.

Выполнение контрольной работы является промежуточной формой отчетности по изучаемой дисциплине и преследует цель лишь оценить способность студента к самостоятельному поиску источников, формированию содержания и его письменного изложения по указанной проблеме. Это важная составляющая изучения дисциплины, а также эффективная форма контроля знаний. При заочном обучении она выступает как обязательная, основная форма самостоятельной работы. В курсовой работе (в соответствии с учебным планом) студент обязан самостоятельно глубоко разобраться в изучаемых проблемах, усвоить суть темы, уяснить ее содержание и только затем письменно представить свою отчетную работу.

Выполнение контрольной работы является одним из условий допуска студента к сдаче экзамена. Работа должна соответствовать установленным требованиям, то есть в ней должны быть раскрыты все проблемы, определенные темой. Для этого студент обязан самостоятельно проанализировать первоисточники и дать исчерпывающие ответы на вопросы темы. Контрольная работа — серьезное учебное задание, и чтобы написать ее как следует, необходимо использовать те первоисточники и учебные пособия, которые позволяют полнее разобраться в проблеме. Студент должен регулярно работать в

университетской и городской библиотеке, вдумчиво конспектировать лекции преподавателей.

При написании контрольной работы следует обращать особое внимание на грамотное использование терминологии. При употреблении впервые тех или иных терминов и понятий следует давать их определения либо в самом тексте, либо в сносках.

Приступая к контрольной работе, требуется сначала ознакомиться с имеющейся литературой по теме, изучить первоисточники и составить план. Здесь, в отличие от курсовой работы, план предполагает рассмотрение одной, причем довольно широкой, проблемы, и он может состоять из двух-трех вопросов. Минимальное количество первоисточников, привлекаемых для написания курсовой работы — пять наименований.

Как правило, контрольные работы по дисциплине сугубо индивидуальны, то есть их тематика персонифицирована. Однако в отдельных случаях темы контрольных работ могут быть адресованы и сразу нескольким, и группе в целом. Таким приемом преподаватель выявляет степень усвоения какой-то важной учебной проблемы и определяет необходимость проведения дополнительных занятий по какой-либо теме. В настоящее время широко используется методика компьютерного тестирования знаний студентов по дисциплинам, в результате чего появляется возможность быстро проверять знания по наиболее важными темами и объективно оценивать их. Эта форма также может выступать как вид контрольной работы.

В качестве контрольной работы широко применяется самостоятельное изучение монографического исследования по конкретной, крайне важной проблеме, требующей глубокого рассмотрения. Этот вид работы предполагает не простое знакомство с определенным монографическим исследованием, а детальное его изучение. Для этого студенту важно знать некоторые правила работы с первоисточником, которым для него будет являться монография. Следует выяснить фамилию автора, его имя и отчество, ученую степень и звание, а также что побудило его взяться за изучение данной проблемы; обратить внимание на основные вопросы монографии и их разрешение автором, уметь раскрывать их в ходе собеседования с преподавателем.

Студенту следует письменно (предельно кратко) очертить те вопросы (полностью или частично), которые поставлены автором в монографическом исследовании; при изложении их следует указывать страницы источника.

Задания для написания контрольных работ (для заочной формы обучения)

Примерная тематика контрольных работ для студентов заочной формы обучения

Работа состоит из трех частей, которые выполняются последовательно в течении семестра.

Программирование классов с использованием наследования

При выполнении первой части контрольной работы студент должен выполнить следующие этапы работы.

- Написать классы для создания графических объектов. Классы должны иметь общий абстрактный базовый класс Shape с чистыми виртуальными функциями.
- Использовать множественное наследование. В классах должны быть предусмотрены виртуальные функции для вывода информации об объектах в поток, а Shape должен иметь дружественный перегруженный оператор <<.
- Исходный текст должен быть разделен на три файла .h, .cpp и .cpp с тестовой программой.
- По данной части контрольной работы представляется диаграмма классов графических фигур.

Последовательность выполнения задания для первой программы

1. Согласовать с руководителем вариант первой части задания контрольной работы.
2. Нарисовать диаграмму классов графических фигур.
3. Создать классы графических фигур.

4. Организовать наследование классов графических фигур.
5. Добавить поля в классы графических фигур.
6. Добавить описания методов в классы графических фигур.
7. Объявить и описать дружественные операторы ввода/вывода.
8. Реализовать тестирующую программу.
9. Рассмотреть работу программы в режиме отладки.

Варианты фигур для индивидуальных заданий по первой части контрольной работы

1. Квадрат, прямоугольник, текст, текст в прямоугольнике.
2. Окружность, эллипс, текст, текст в эллипсе.
3. Правильный треугольник, правильный пятиугольник (с поворотом), текст, текст в пятиугольнике.
4. Точка, отрезок, треугольник, текст, текст в треугольнике.
5. Прямоугольник, овал, текст, текст в овале.
6. Квадрат, квадрат с тенью, текст, текст в квадрате.
7. Правильный треугольник, правильный треугольник со сглаженными углами, текст, текст в треугольнике.
8. Прямоугольник, таблица, текст, текст в прямоугольнике.
9. Отрезок синусоиды, квадрат с волнистыми краями, текст, текст в квадрате.
10. Прямоугольник, рамка (прямоугольник с толщиной сторон), текст, текст в прямоугольнике.
11. Треугольник, трапеция, текст, текст в трапеции.
12. Прямоугольник, параллелограм, текст, текст в параллелограмме.
13. Окружность, сечение тора, текст, текст в окружности.
14. Круг, сектор, текст, текст в секторе.
15. Окружность, правильные фигуры, текст, текст в фигурах.
16. Круг, кольцо, текст, текст в кольце.
17. Эллипс, дуга эллипса, текст, текст в эллипсе.
18. Равнобедренный треугольник, равнобедренный треугольник со скошенными углами, текст, текст в треугольнике.
19. Квадрат, куб, текст, текст в квадрате.
20. Прямоугольник, параллелограмм, параллелепипед, текст , текст в параллелограмме.

Программирование контейнерных классов

При выполнении второй части контрольной работы студент должен выполнить следующие этапы работы.

- Использовать полиморфный контейнер для размещения объектов ранее созданных классов графических фигур.
- Произвести тестирование заполненного контейнера, применяя для этого наиболее характерный набор методов и итераторов.
- По данной части контрольной работы представляется диаграмма классов графических фигур, контейнера и итераторов, а также наиболее важные отношения UML.

Последовательность выполнения задания для второй программы

1. Согласовать с руководителем вариант задания второй части контрольной работы.
2. Изучить интерфейс контейнера в соответствии с индивидуальным заданием.
3. Опробовать работу контейнера и итераторов на примере стандартных типов данных.
4. Создать контейнер и необходимые итераторы.
5. Создать по два разных объекта каждого класса графических фигур.
6. Добавить указатели на созданные объекты в контейнер и с помощью итераторов вывести информацию о графических фигурах в поток.
7. Провести демонстрацию полезных функций контейнера, используя необходимые методы и итераторы.

Варианты контейнеров для индивидуальных заданий по второй части контрольной работы

1. Вектор.
2. Очередь с двумя концами.
3. Очередь с одним концом.
4. Стек.
5. Список
6. Мультимножество.
7. Множество.

Проектирование и программирование приложений

Индивидуальность задания третьей части контрольной работы полностью определяется вариантами заданий, полученных на первых двух этапах. При выполнении последней части контрольной работы студент должен выполнить следующие этапы работы.

- Создать диалоговое приложение с элементами интерфейса для тестирования полиморфного контейнера, заполненного графическими фигурами.
- Включить в программный код приложения классы графических фигур и операции по заполнению контейнера и его тестирования.
- Обеспечить визуализацию результатов тестирования операций с контейнером с помощью диалоговых окон.
- По данной части контрольной работы представляются диаграмма компонентов итогового проекта.

Последовательность выполнения задания для третьей программы

1. Согласовать с руководителем способ визуализации результатов тестирования контейнера.
2. Создать диалоговое приложение.
3. Добавить диалоговые элементы интерфейса и окна.
4. Создать контейнер и необходимые итераторы.
5. Создать по два разных объекта каждого класса графических фигур.
6. Добавить указатели на созданные объекты в контейнер.
7. Провести демонстрацию полезных функций контейнера, используя элементы интерфейса приложения.

8. Методические рекомендации по написанию курсовой работы

Курсовая работа учебным планом не предусмотрена.

ОЦЕНОЧНЫЕ И МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ

Оценочные и методические материалы включают в себя:
- перечень компетенций с указанием этапов их формирования в процессе освоения образовательной программы;
- показателей и критериев оценивания компетенций на различных этапах их формирования, описание шкал оценивания;
- типовые контрольные задания или иные материалы, необходимые для оценки знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций в процессе освоения образовательной программы;
- методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций.

1. Перечень компетенций с указанием этапов их формирования в процессе освоения образовательной программы

№ п\п	Контролируемые темы дисциплины	Код формируемой компетенции	Код и наименование индикатора достижения	Наименование оценочного средства
	Основные понятия дисциплины	ПК-3	ПК-3.1. Знает концепцию объектно-ориентированного программирования, методику анализа и проектирования объектно-ориентированных программ, основные понятия, синтаксис и семантику конструкций языка программирования.	Опрос, участие в коллоквиуме, кейсы
	Введение в объектно-ориентированный анализ и проектирование	ПК-3	ПК-3.1. Знает концепцию объектно-ориентированного программирования, методику анализа и проектирования объектно-ориентированных программ, основные понятия, синтаксис и семантику конструкций языка программирования.	Опрос, участие в коллоквиуме, кейсы
	Классы	ПК-3	ПК-3.1. Знает концепцию объектно-	Опрос, тестирование, коллоквиум,

			ориентированного программирования, методику анализа и проектирования объектно-ориентированных программ, основные понятия, синтаксис и семантику конструкций языка программирования. ПК-3.2. Умеет проектировать, программировать и отлаживать объектно-ориентированные программы ПК-3.3. Владеет навыками объектно-ориентированного программирования, отладки и тестирования программного обеспечения навыками программирования в современных средах.	кейсовые задания
	Шаблоны и обработка исключительных ситуаций	ПК-3	ПК-3.1. Знает концепцию объектно-ориентированного программирования, методику анализа и проектирования объектно-ориентированных программ, основные понятия, синтаксис и семантику конструкций языка программирования. ПК-3.2. Умеет проектировать, программировать и отлаживать объектно-ориентированные программы ПК-3.3. Владеет навыками объектно-ориентированного	Опрос, тестирование, коллоквиум, практические задания

			программирования, отладки и тестирования программного обеспечения навыками программирования в современных средах.	
Результат достижения планируемых результатов изучения дисциплины				зачет

2. Описание показателей и критериев оценивания компетенций, шкал оценивания

Критерии оценивания (текущий контроль)

1. Оценка **«отлично»** выставляется студенту, если студент имеет глубокие знания учебного материала по теме практического задания, в логической последовательности излагает материал; смог ответить на все уточняющие и дополнительные вопросы;
2. Оценка **«хорошо»** выставляется, если студент показал знание учебного материала, смог ответить почти полностью на все заданные дополнительные и уточняющие вопросы;
3. Оценка **«удовлетворительно»** выставляется, если студент в целом освоил материал; однако, ответил не на все уточняющие и дополнительные вопросы;
4. Оценка **«неудовлетворительно»** выставляется студенту, если он имеет существенные пробелы в знаниях основного учебного материала по теме практического задания, который полностью не раскрыл содержание вопросов, не смог ответить на уточняющие и дополнительные вопросы.

Критерии оценивания (зачет)

Знания, умения, навыки и компетенции студентов оцениваются следующими оценками: «зачтено», «не зачтено».

- **«зачтено» -** студент хорошо и прочно усвоил весь программный материал, исчерпывающе, последовательно, грамотно и логически стройно его излагает, увязывает с практикой, свободно справляется с решением ситуационных задач и тестовыми заданиями, правильно обосновывает принятие решений, умеет самостоятельно обобщать программный материал, не допуская ошибок, знает дополнительную литературу по изучаемой дисциплине.
- «**не зачтено»** - студент не знает значительной части основного программного материала, в ответах допускает существенные ошибки, не владеет умениями и навыками в выполнении тестовых заданий и решении задач, не способен ответить на дополнительные вопросы.

3. Типовые контрольные задания и методические материалы, процедуры оценивания знаний, умений и навыков

ТЕКУЩИЙ КОНТРОЛЬ

Тестовые материалы

Важными в методическом плане на семинарских занятиях являются проводимые тестовые задания, которые содействуют превращению теоретико-правовых знаний в

глубокие убеждения, дают простор для развития творческо-эмоциональной сферы, позволяют сделать выводы об эффективности занятий с учащимися, что в итоге повышает интерес к овладению знаниями.

Решение тестовых заданий является важным методическим приемом для закрепления и осмысления, полученных бакалаврами знаний по изучаемому предмету.

Студент тестируемой учебной группы получает 50 тестовых заданий. Для каждого из вопросов тестового задания предусмотрен только один правильный вариант ответа, который должен выбрать студент. Результаты тестирования оцениваются в зависимости от количества неверно выбранных ответов.

Итоги тестирования заносятся в ведомость, составляемую на всю учебную группу. Предоставленные сведения должны содержать данные о количестве опрошенных, о количестве отличных, хороших, удовлетворительных и неудовлетворительных оценок.

В заключение работы выводиться средний балл итогового контроля знаний студентов.

ПАСПОРТ ТЕСТОВЫХ ЗАДАНИЙ

1. Общее количество тестовых заданий в базе – 128
2. Ограничение времени выполнения теста (в мин) – 30
3. Автоматическое перемешивание вопросов в тесте: □ **(да)**
4. Случайный порядок ответов в тестовом задании: □ **(да)**
5. Критерии оценки результатов тестирования :_свыше 50% правильных ответов – зачет

Пример тестовых заданий для текущего контроля представлен ниже:

1. Какая функция объявляется с целью получить доступ к защищенным и внутренним атрибутам другого класса?
   - (•) дружественная
   - ( ) шаблонная
   - ( ) макрофункция
   - ( ) статическая
   - ( ) защищенная

2. Если имеется код
```
class A { public: int a, b, c; };
A *obj;
```
как обратиться к переменной b?
   - ( ) obj.a
   - ( ) obj->a->b
   - (•) obj->b
   - ( ) obj->a.b

3. Размер объекта класса в памяти определяется
   - ( ) суммой размеров методов класса и атрибутов класса
   - ( ) суммой размеров методов класса
   - (•) суммой размеров атрибутов класса

- не зависит от размеров атрибутов и методов класса

4. Какой из стандартных классов используется для вывода строк на терминал:
   - strstream
   - ostream
   - ofstream
   - istream
   - ifstream

5. После компиляции программы
   - ее можно выполнять многократно без перекомпиляции
   - перед каждым последующим запуском ее нужно перекомпилировать
   - ее можно выполнять только с одним набором исходных данных

6. Процесс компиляции программы
   - переводит исходный текст в исполняемый файл
   - проверяет программу на наличие ошибок
   - приводит программы к единообразному внешнему виду
   - для языка Си++ необязателен

7. В программе на языке Си++ обязательно имеется функция
   - head
   - start
   - prime
   - main
   - finish

8. Какой правильный заголовок шаблона
   - template <class t1, class t2>
   - template <class t1,t2>
   - template <class t, class t>
   - template <class t,t>

9. Можно ли использовать класс-шаблон в качестве базового класса?
   - да
   - нет

10. Цель введения . . . функций – автоматизация создания функций, которые могут обрабатывать разнотипные данные.
    - шаблонов
    - прототипов
    - подписей (сигнатур)

- аргументов

11. Если заданы классы

```
class A {... };
class B : public A { ... };
class C : public A { ... };
```

то что будет выведено при выполнении оператора

```
throw (A);
```

а обработка исключительной ситуации записана

```
catch (B& b) { cout << 1; }
catch (C& c) { cout << 2; }
catch (A& a) { cout << 3; }
catch (...) { cout << 4; }
```

- 1
- 2
- 3
- 4
- 3 4
- 2 3 4

12. Что может быть аргументом оператора **throw**?

- [x] целое число
- [x] объект класса Exception
- [x] строка
- [x] ноль
- [ ] условный оператор
- [ ] вызов деструктора объекта
- [ ] вызов оператора return

13. Какой термин не имеет отношения к объектно-ориентированному программированию?

- инкапсуляция
- индексация
- наследование
- полиморфизм

14. Каким может быть аргумент деструктора?

- адрес объекта
- указатель this
- аргумента не может быть
- уничтожаемый объект

15. Отметьте все правильные варианты продолжения предложения: виртуальный деструктор

- [x] может использоваться с абстрактными классами
- [x] не нужен, если класс не наследуется
- [ ] должен быть определен как чисто виртуальный в абстрактном классе

16. Что выведет следующая программа ?

```
#include <iostream.h>
int main() {
  int i ;
  for(i = 0; i < 9; i++)
  cout << i+1;
  return 1;
}
```

- ( ) цифры от 0 до 8
- (•) цифры от 1 до 9
- ( ) программа не будет построена из-за ошибок

17. Абстрактный класс – это класс, в котором
- ( ) есть хотя бы один виртуальный метод
- ( ) есть виртуальный конструктор
- ( ) есть виртуальный деструктор
- (•) есть чисто виртуальный метод

18. Если не объявить тип возврата функции, то какой тип будет принят по умолчанию?
- ( ) int;
- (•) void;
- ( ) float;
- ( ) это вызовет ошибку при компиляции

19. Какими по умолчанию объявляются элементы структуры?
- ( ) private
- (•) public
- ( ) protected
- ( ) по умолчанию не объявляются

20. Какой правильный вызов функции базового класса из объекта производного класса, если в производном классе эта функция была замещена?
- ( ) FunctionName();
- (•) Base::FunctionName();
- ( ) Base.FunctionName();
- ( ) такую функцию вызывать нельзя.

21. Методы класса определяют

- [x] какие операции можно выполнять с объектами данного класса
- [ ] какие значения может принимать переменная данного класса
- [x] каким образом можно создавать объекты данного класса

22. Что является коллекцией переменных, объединенных с набором связанных функций

- (•) класс
- ( ) массив
- ( ) модуль
- ( ) пространство имен

23. Какие основные области применения языка Си++?

- ( ) системное программирование
- ( ) прикладное программирование
- (•) системное и прикладное программирование

24. Программа на языке Си++ начинает выполняться с:

- ( ) первой функции в программе
- (•) функции main
- ( ) той функции, которая указана как стартовая при компиляции программы

25. Какой правильный вариант создания класса из шаблона?

- ( ) template<char> Array;
- ( ) template Array<char>;
- ( ) Array template<char>;
- (•) Array<char> A

26. Какие требования предъявляются к классу исключительных ситуаций?

- [ ] он должен быть наследован от специального класса exception
- [ ] он не может использовать множественное наследование
- [ ] он должен содержать атрибуты только встроенных типов
- [x] он может быть произвольным классом

27. Переопределение операции сложения приведет к…(отметьте все правильные варианты)

- [ ] ее вызову при выполнении операции ++ с объектом класса
- [x] ее вызову при выполнении операции сложения с объектом класса
- [ ] преобразованию целых чисел к объекту данного класса при выполнении сложения
- [x] возможному преобразованию объектов других классов к данному при выполнении операции сложения

28. Если определена операция вычитания для двух объектов класса **A** и операция преобразования к **int**, что будет вызвано при

```
A a;
int x;
int y = a – x;
```

- преобразование к целому
- (•) операция вычитания, а затем преобразование к целому
- только операция вычитания
- произойдет ошибка

29. Какой тип будет у результата выражения?

- такой же, как тип операндов выражения
- (•) зависит от типа операндов выражения и выполняемой операции
- такой, как у переменной, которой присваивается значение выражения

30. Механизм позднего связывания применяется для реализации

- (•) полиморфизма
- инкапсуляции
- наследования

31. Какой класс может использоваться в качестве типа атрибута класса?

- базовый класс данного класса
- производный от данного класса
- пользовательский класс
- (•) произвольный класс

32. В результате компоновки (редактирования связей) получается:

- отредактированный исходный файл
- список имеющихся в исходном файле ошибок
- объектный файл
- промежуточное представление программы
- (•) исполняемый файл

33. Какова последовательность создания исполняемого файла:

1) Создать файл с исходным текстом программы, который будет иметь расширение **.cpp**.
2) Скомпоновать объектный файл с необходимыми библиотеками.
3) Скомпилировать исходный код.

- (•) 1-3-2
- 2-3-1
- 1-2-3
- 3-2-1

- 2-1-3

34. Если записано
class A { public: void f() { cout << 1; } };
class B : public A { public: void f() { cout << 2; } };
то что будет напечатано ?
B b; b.f();
    - 2
    - 2 1
    - 1 2
    - 1
    - ошибка

35. Какая из записей является правильной записью абстрактного класса?
    - abstract class A { virtual f() = 0; };
    - class A { virtual f() = 0; };
    - class A { virtual f(); };

36. Какой класс объявляется таким образом, чтобы все его функции - члены были дружественными по отношению к другому классу?
    - дружественный
    - базовый
    - абстрактный
    - виртуальный
    - глобальный

ПРОМЕЖУТОЧНАЯ АТТЕСТАЦИЯ

Вопросы для подготовки к промежуточной аттестации по дисциплине (зачету)

1. Принципы объектного подхода, абстрагирование, ограничение доступа, модульность, иерархия, типизация параллелизм, устойчивость.
2. Инкапсуляция и спецификация правил доступности элементов класса.
3. Конструкторы и деструкторы.
4. Наследование. Иерархия классов.
5. Одиночное и множественное наследование.
6. Способы реализации множественного наследования, их достоинства и недостатки.
7. Полиморфизм.
8. Ранее и позднее связывание.
9. Объект и процесс. Инициализация и взаимодействие объектов и процессов. Сообщения. Реализация механизмов посылки сообщений.
10. Формальные модели объектов и классов: автоматная и алгебраическая модели объектов, исчисления типов.
11. Определения системы, домена, подсистемы, элемента, связи, среды.
12. Описание классов и их взаимосвязей с использованием унифицированного языка моделирования UML.

13. Моделирование статической структуры системы. Диаграммы классов. Механизм пакетов.
14. Моделирование поведения системы. Диаграммы взаимодействия. Диаграммы состояний. Диаграммы действий.
15. Моделирование реализации системы. Диаграммы компонентов. Диаграммы размещения.
16. Имена и их объявления. Инициализация значений.
17. Область и время действия имени.
18. Классы памяти.
19. Явное управление видимостью, пространства имен.
20. Определение класса. Личная и общая части определения класса.
21. Функции-элементы класса и функции-друзья.
22. Объекты класса.
23. Интерфейс и реализация контейнерных классов для моделирования структур данных.
24. Статические члены объектов класса.
25. Вложенные и локальные классы.
26. Примеры описания и использования классов.
27. Базовый и производный классы. Функции-элементы и функции-друзья.
28. Виртуальные базовые классы.
29. Особенности доступа при множественном наследовании. Правила преобразования указателей.
30. Виртуальные функции. Таблицы виртуальных функций.
31. Чистые виртуальные функции и абстрактные базовые классы.
32. Полиморфные контейнерные классы, итераторы и аппликаторы.
33. Абстрактные и конкретные контейнерные классы.
34. Виды классов: конкретный тип, абстрактный тип, узловой класс, интерфейсный класс.
35. Инициализация баз и членов.
36. Полный объект конечного производного класса.
37. Шаблоны классов и функций.
38. Наследование шаблонных классов. Правила отождествления параметров шаблона.
39. Применение шаблонных классов для создания контейнерных классов.
40. Стандартные средства контроля подтверждений.
41. Проверка предусловий и постусловий, вычисления инвариантов.
42. Контроль асинхронных событий. Сигналы.
43. Возбуждение ситуации, описание блоков с контролем и реакций на ситуации.
44. Система классов для описания исключительных ситуаций.
45. Обработка исключительных ситуаций при обработке исключительных ситуаций.
46. Инструментальные программные средства для C++.
47. Библиотеки классов для C++.

ГЛОССАРИЙ

Агрегирование (Aggregation)
Агрегирование - это вид ассоциации, определяющей отношение "часть - целое". Для указания на элемент играющий роль целого к концу линии присоединяется терминатор в виде полого ромба.
Актор (Actor)
Актор является ролью объекта или объектов вне системы, которая напрямую взаимодействует с системой. Один физический объект может играть несколько ролей и, следовательно, моделируется несколькими акторами. Актор может показываться как прямоугольник класса со стереотипом “actor”. Стандартная картинка стереотипа для актора рисунок человечка с названием под ним.
Ассоциация (Association)
Ассоциации - есть статические связи между классами, изображаются в виде связующей линии. Возможны рефлексивные ассоциации от класса к самому себе. Тернарные и более высокого порядка ассоциации показываются как ромбы, соединенные линиями с символами классов.
Конец ассоциации, которым она присоединяется к классу, называется ролью ассоциации (см. подробнее Роль ассоциации).
На линии связи показывается строка названия. Строка названия может содержать необязательный маленький черный сплошной треугольник; указывающий направление, в котором понимается название. Классы в ассоциации упорядочиваются, как указано стрелкой направления названия.
Для реализации этой связи может быть описан специальный ассоциированный класс. Он обозначает ассоциацию, которая имеет подобные классу свойства, такие как атрибуты, операции и, возможно, другие ассоциации. Ассоциированный класс изображается как класс, присоединенный пунктирной линией к пути ассоциации.
Атрибут (Atribute)
Атрибут - свойство объекта или класса. Атрибут семантически эквивалентен построению ассоциаций. Тип атрибута - может быть именем класса или быть составным, как array[String] of Point. В любом случае детали выражений, описывающих тип атрибута, не определяются в UML; они зависят от синтаксиса выражений, поддерживаемых используемым языком или программирования. Атрибут отображается как текстовая строка, которая может разбираться на различные свойства атрибута как элемента модели.
visibility name : type-expression = initial-value {property-string}
видимость имя : выражение-типа = начальное-значение {строка-свойств}
где видимость принимает одно из следующих значений: **+**общедоступный (public) **#**защищенный (protected) **-**частный (private) Глобальные для класса атрибуты отображаются с помощью подчеркивания имени и выражения типа, иначе атрибут является локальным для экземпляра. Неизменяемые атрибуты описываются с помощью свойства {frozen} ({заморожен}). В отсутствии указателя множественности атрибут принимает точно одно значение. Множественность может быть задана помещением указателя множественности в скобках после имени.
Диаграмма действий (Activity diagram)
Диаграмма активности является частным случаем диаграммы состояний, в которой все или большинство состояний являются состояниями действий и, в которой переходы, инициируются завершением действий в состояниях источниках. Диаграмма активности связывается (через модель) с классом, с реализацией операции или с использования. Назначение этой диаграммы сконцентрироваться на потоках управляемых внутренней обработкой (в противоположность внешним событиям).
Диаграмма классов (Class diagram)

Диаграмма классов это набор декларативных (статических) элементов модели, таких как классы, интерфейсы и их отношения, соединенные друг с другом в виде графа. Диаграмма классов представляет собой граф из элементов соединенных различными постоянными отношениями.

Диаграмма компонентов (Component diagram)

Компонентная диаграмма показывает зависимости между программными компонентами, включая компоненты исходного текста, компоненты двоичного кода и выполнимые компоненты. Компонентная диаграмма содержит только представление типа, а не экземпляра. Чтобы показать экземпляры компонентов, используется диаграмма развертывания. Компонентная диаграмма изображается как граф компонентов, соединенных отношениями зависимости. Компонент может также быть присоединен к компонентам с помощью физического включения, представляющего отношение композиции. Содержащая типы компонентов и типы узлов диаграмма может использоваться для показа зависимостей компилятора, которые показываются пунктирными стрелками от компонента клиента к компоненту поставщику, от которого он неким образом зависит. Виды зависимостей являются специфическими для языка и могут показываться как стереотипы зависимостей. Диаграмма может использоваться так же для показа интерфейсов и зависимостей вызова между компонентами, с помощью пунктирных стрелок от компонентов к интерфейсам других компонентов.

Диаграмма объектов (Object diagram)

Диаграмма объектов - это граф экземпляров, включая объекты и значения данных. Статическая диаграмма объектов - это экземпляр диаграммы классов; она показывает моментальный снимок детального состояния системы в определенный момент времени. Использование диаграммы объектов довольно ограничено. В основном она показывает примеры структуры данных.

Диаграмма последовательности (Sequence diagram)

Диаграмма последовательности представляет собой взаимодействие, которое является набором передаваемых между объектами сотрудничества сообщений для достижения желаемой операции или цели. Диаграмма последовательности имеет два измерения: вертикальное измерение представляет время, горизонтальное измерение представляет различные объекты. Обычно время распространяется вниз страницы. (При необходимости размерность может быть обращена.) Обычно важна только последовательность событий, однако в приложениях реального времени ось времени может иметь реальную метрику. Горизонтальное упорядочение объектов не имеет никакого значения. На диаграмме объекты могут быть сгруппированы в "дорожки процессов".

Диаграмма прецедентов использования (Use-case diagram)

Диаграммы прецедентов использования показывает отношение акторов (действующих лиц) и прецедентов использования внутри системы. С помощью такой диаграммы представляются функциональные возможности системы или класса как они проявляются при внешнем взаимодействии с системой. Диаграмма изображается в виде графа акторов, набора прецедентов использования, заключенных в границы системы, ассоциаций связи (участия) между акторами и прецедентами, и обобщений между прецедентами.

Диаграмма размещения (Deployment diagram)

Диаграммы размещения показывают конфигурацию обрабатывающих элементов времени выполнения и программных компонентов, процессов и объектов, размещенных в них. Компоненты, которые не являются сущностями времени выполнения показываться не должны. Диаграмма развертывания представляет граф узлов соединенных ассоциациями связи. Узлы могут содержать экземпляры компонентов; они показывают, что компоненты размещены или работают в узле. Компоненты соединяются с другими компонентами пунктирными стрелками зависимостей (возможно через интерфейс). Это значит, что один компонент использует сервисы другого компонента; для показа точного вида зависимости может использоваться стереотип.

Диаграмма состояний (State diagram)

Диаграмма состояний показывает последовательность состояний, через которые проходит объект или взаимодействие за период жизни. Диаграмма состояния представляет собой конечный автомат. Состояния показываются символами состояний, переходы - соединенными с ними стрелками. Состояния могут также содержать поддиаграммы путем физического включения и разделения на ячейки. Состояние - ситуация в жизни объекта или взаимодействия за период которой, они удовлетворяют некоторым условиям, выполняют некоторые действия или ждут некоего события. Объект находится в состоянии конечное (не нулевое) время. Действия являются атомарными и не могут прерываться.

Диаграмма сотрудничества (Collaboration diagram)

Диаграмма сотрудничества показывает взаимодействие организованное вокруг объектов во взаимодействии и их связи друг с другом. Поведение обеспечивается множеством объектов, которые для достижения цели обмениваются сообщениями. Для понимания используемых в проекте механизмов важно видеть только объекты и сообщения, связанные с достижением цели. Такая статическая конструкция называется сотрудничеством. Сотрудничество может быть присоединено к операции или прецеденту использования для описания окружения, в котором возникает их поведение. Параметризованное сотрудничество представляет проектную структуру, которая может многократно использоваться в различных проектах.

Диаграмма сотрудничества представляет **сотрудничество**, которое является множеством объектов связанных в конкретном окружении, и **взаимодействие**, которое является множеством обмениваемых между объектами сотрудничества сообщений для достижения требуемых операции или результата.

Сотрудничество показывает окружение, в котором происходит взаимодействие. Динамическое поведение обмениваемой между объектами для достижения конкретной цели последовательности сообщений называется взаимодействием. Сотрудничество показывается диаграммой сотрудничества без сообщений. Добавляя сообщения, мы получаем взаимодействие. Различные совокупности сообщений могут применяться к одному и тому же сотрудничеству для создания различных взаимодействий. Диаграмма сотрудничества является графом ссылок на объекты и связи с потоками сообщений присоединенных к этим связям.

Зависимость (Dependency)

Зависимость показывает семантическое отношение между двумя (или более) элементами модели. Она указывает ситуацию, в которой изменение целевого элемента может потребовать изменения элемента источника зависимости. Зависимость показывается пунктирной стрелкой между двумя элементами модели. Элемент модели на хвосте стрелки зависит от элемента модели на острие стрелки. Стрелка может быть помечена необязательным стереотипом и необязательным названием. Следующие виды зависимости предопределены и могут обозначаться ключевыми словами:

Trace - След. Историческое соединение между двумя элементами, которые представляют одинаковое понятие на разных уровнях абстракции.

Refine - Уточнение. Историческое или возникшее соединение между двумя элементами с отображением (не обязательно полным) между ними. Описание отображения может быть присоединено к зависимости в примечании.

Uses - Использование. Ситуация в которой элемент требует присутствия другого элемента для корректной реализации и функционирования. Может стереотипироваться дальше для указания точного характера зависимости, например, вызов операций другого класса, предоставление прав доступа, иллюстрирующий объект другого класса, и т.д.

Bind - Связывание. Связывание параметров шаблона с действительными значениями для создания не параметризованного элемента.

Интерфейс (Inteface)

Интерфейс описывает видимые снаружи операции класса, компоненты или другие сущности без задания их внутренней структуры. Каждый интерфейс часто описывает только ограниченную часть поведения реального класса. Интерфейсы не имеют

реализаций - они не имеют атрибутов состояний или ассоциаций, а только операции. Интерфейс может иметь отношение обобщения. Интерфейс формально эквивалентен абстрактному классу без атрибутов и методов и имеющему только абстрактные операции.
Интерфейс может быть показан как символ прямоугольника, разделенный на секции и имеющий соответствующий стереотип. Секция атрибутов может опускаться, так как она всегда пуста.
Интерфейс также может быть отображен как маленькая окружность с помещенным под символом названием интерфейса. Окружность может соединяться сплошной линией с классами, которые поддерживают интерфейс. Это означает, что класс поддерживает все операции интерфейса (и возможно другие). Класс, который использует или требует поддерживаемые интерфейсом операции, может быть соединен с окружностью пунктирной стрелкой, указывающей на окружность.
Отношение реализации классом интерфейса изображается пунктирной линией с треугольным терминатором.

Класс (Class)

Класс - это описатель для набора объектов с общей структурой, поведением и отношениями. UML представляет нотацию для объявления классов и определения их свойств. Некоторые моделирующие элементы, которые имеют общую с классом форму (такие как интерфейсы, сигналы или утилиты), описываются с использованием соответствующих стереотипов. Классы объявляются на диаграмме классов и используются в большинстве других диаграмм.
Название класса ограничено пакетом, в котором он объявлен, и оно должно быть уникальным (среди названий классов) в этом пакете. Класс изображается как сплошной прямоугольник, разделенный горизонтальными линиями на три секции. Верхняя секция названия содержит название класса и другие основные свойства класса (включая стереотип); средняя секция списка содержит список атрибутов; нижняя секция списка содержит список операций. Чтобы показать ссылку на класс, определенный в другом пакете, используется следующий синтаксис:

Package-name::Class-name (Название_пакета::Название_класса)

В качестве расширения программное обеспечение может добавлять дополнительные секции, показывающие предопределенные или определенные пользователем свойства модели, например, для показа бизнес правил, обязанностей, изменений, управления событиями, вызовов исключений и т.п. Большинство подобных секций представляют собой просто список строк.

Классов диаграмма (Class diagram)

Диаграмма классов это набор декларативных (статических) элементов модели, таких как классы, интерфейсы и их отношения, соединенные друг с другом в виде графа. Диаграмма классов представляет собой граф из элементов соединенных различными постоянными отношениями.

Композиция (Composition)

Композиция - это форма агрегирования с сильным отношением собственности и совпадающим сроком жизни ее части со сроком жизни целого. Множественность цели композиции не может превышать единицы. Композиция может изображаться с помощью терминитора в виде сплошного заполненного ромба. UML также обеспечивает вложенную форму, когда композиция может показываться графическим вложением символов элементов частей в символ элемента целого. Вложенный элемент может иметь множественность в составном элементе. Атрибуты, в действительности, являются отношениями композиции между классом и классами этих атрибутов.

Компонентов диаграмма(Component diagram)

Компонентная диаграмма показывает зависимости между программными компонентами, включая компоненты исходного текста, компоненты двоичного кода и выполнимые компоненты. Компонентная диаграмма содержит только представление типа, а не экземпляра. Чтобы показать экземпляры компонентов, используется диаграмма

развертывания. Компонентная диаграмма изображается как граф компонентов, соединенных отношениями зависимости. Компонент может также быть присоединен к компонентам с помощью физического включения, представляющего отношение композиции. Содержащая типы компонентов и типы узлов диаграмма может использоваться для показа зависимостей компилятора, которые показываются пунктирными стрелками от компонента клиента к компоненту поставщику, от которого он неким образом зависит. Виды зависимостей являются специфическими для языка и могут показываться как стереотипы зависимостей. Диаграмма может использоваться так же для показа интерфейсов и зависимостей вызова между компонентами, с помощью пунктирных стрелок от компонентов к интерфейсам других компонентов.

Обобщение (Generalization)

Обобщение является таксономическим отношением между общим элементом и специфичным элементом, который полностью непротиворечив первому элементу и добавляет к нему дополнительную информацию. Оно используется для классов, пакетов, прецедентов использования и других элементов. Обобщение показывается как сплошная линия от специфичного элемента (например, подкласса) к общему элементу (например, суперклассу), с большим полым треугольником на конце пути в точке соединения с общим элементом. Для указания семантических ограничений среди подклассов могут использоваться предопределенные ограничения. Разделенный запятыми список ключевых слов помещается в скобки либо рядом с разделяемым треугольником (если несколько путей имеют один общий треугольник) либо рядом с пунктирной линией, которая пересекает все включаемые строки обобщения. Следующие ограничения (среди прочих) могут использоваться:

Overlapping - перекрывающийся. Потомок может происходить более чем из одного подкласса.

Disjoint - непересекающийся. Потомок не может происходить более чем из одного подкласса.

Complete - завершенный. Все подклассы определены (во всяком случае, показаны). Дополнительных подклассов не ожидается.

Incomplete - незавершенный. Некоторые подклассы определены, но известно, что их список незавершен. Имеются дополнительные подклассы, которые пока отсутствуют в модели.

Объект (Object)

Объект представляет собой отдельный экземпляр класса. Он имеет значения атрибутов. Объект изображается как класс подчеркиванием элементов уровня экземпляра, но в прямоугольнике с только двумя секциями. Верхняя секция показывает название объекта и его класс (оба подчеркнутые), м.б. включая полный путь содержащего его пакета. Например:

display_window: WindowingSystem::GraphicWindows::Window.

Вторая секция показывает список атрибутов объекта и их значений. Тип атрибутам объекта уже присвоен при объявлении атрибута класса и поэтому может быть опущен.

Возможно определение анонимного объекта заданного класса.

Объектов диаграмма (Object diagram)

Диаграмма объектов - это граф экземпляров, включая объекты и значения данных. Статическая диаграмма объектов - это экземпляр диаграммы классов; она показывает моментальный снимок детального состояния системы в определенный момент времени. Использование диаграммы объектов довольно ограничено. В основном она показывает примеры структуры данных.

Ограничение (Constraint)

Ограничение - это семантическое отношение между элементами модели, определяющее всегда истинные условия и теоремы. Основные виды ограничений (такие как ограничение - соединение "или") предопределены в UML, другие определяются пользователем. Для описания определенных пользователем ограничений в конкретном программном

обеспечении системы моделирования может использоваться один или более формальных языков. Предопределенным языком для записи ограничений является OCL. При отсутствии поддержки формального языка ограничений, они описываются на естественном языке. Ограничение изображается как текстовая строка в фигурных скобках ({ }).

Операция (Operation)

Операция - это сервис, выполнение которого может быть запрошено для экземпляра класса. Она имеет название и список аргументов. visibility name (parameter-list): return-type-expression {property-string} видимость имя (список-параметров): выражение-возвращаемого-типа {строка-свойств} где список-параметров - это разделенный запятыми список формальных параметров, для описания которых используется следующий синтаксис:

kind name : type-expression = default-value

вид имя : выражение-типа = значение-по-умолчанию

где вид - это in, out или inout, по умолчанию in; где имя - название формального параметра; Глобальные для класса операции отображаются с помощью подчеркивания имени и выражения типа. По умолчанию операции являются локальными для экземпляра и не маркируются. Операция, которая не модифицирует состояние системы (не имеет побочных эффектов) описывается свойством {query}; иначе, операция может изменять состояние системы. Семантика параллелизма операций определяется строкой свойств с одним из следующих имен: sequential, guarded, concurrent. В отсутствии описания семантика параллелизма неопределенна или в худшем случае должна быть назначена последовательная.

Сигнатура операции в базовом классе, наследуется всеми его наследниками). Если класс не реализует операцию (т.е. отсутствует метод), то операция либо маркируется как {abstract} либо сигнатура операции пишется курсивом, чтобы показать ее абстрактность. Любое последующее появление сигнатуры операции показывает, что подчиненный класс реализует метод операции.

Текст или алгоритм метода может быть показан в сноске, присоединенной к записи операции. Запись операции со стереотипом {signal} показывает, что класс принимает определенный сигнал. Описание поведения операции дается как присоединенная к ней сноска. Если описание сделано на формальном языке, то его текст должен быть оформлен как ограничение, иначе, если для описания поведения используется естественный язык, то должен применяться простой текст (комментарий).

Пакет (Pakage)

Пакеты являются основой управления и организации модели в UML. Пакеты предназначены для группировки элементов модели. В них могут быть упакованы все виды элементов и диаграмм UML. В качестве элемента может выступать другой пакет, т.е. пакеты могут быть вложенными друг в друга. Пакет изображается как большой прямоугольник с маленьким прямоугольником ("ярлыком"), присоединенным к одному из углов (обычно слева/сверху) большого прямоугольника. Для пакетов предопределены следующие стереотипы.

System (система) - стереотип пакета, который представляет набор моделей одной и той же моделируемой системы. Модели описывают моделируемую систему с различных точек зрения, которые не обязательно непересекающиеся. Поэтому система содержит всестороннее описание моделируемой системы и является самым верхним структурным элементом спецификации

Facade (фасад) является стереотипом пустого пакета, который ссылается на элементы модели, содержащиеся в другом пакете. Он обеспечивает общедоступное представление содержимого пакета.

Framework (скелет) является стереотипом пакета, в основном содержащего шаблоны.

Последовательности диаграмма(Sequence diagram)

Диаграмма последовательности представляет собой взаимодействие, которое является

набором передаваемых между объектами сотрудничества сообщений для достижения желаемой операции или цели. Диаграмма последовательности имеет два измерения: вертикальное измерение представляет время, горизонтальное измерение представляет различные объекты. Обычно время распространяется вниз страницы. (При необходимости размерность может быть обращена.) Обычно важна только последовательность событий, однако в приложениях реального времени ось времени может иметь реальную метрику. Горизонтальное упорядочение объектов не имеет никакого значения. На диаграмме объекты могут быть сгруппированы в "дорожки процессов".

Прецедент использования (Use-case)

Прецедент использования - это блок функциональных возможностей обеспеченных системой или классом, которые проявляются при обмене сообщений между системой и одним или более внешними объектами (называемыми акторами) одновременно с действиями выполняемыми системой. Прецедент использования изображается как эллипс, содержащий название. Точка расширения - место в случае использования, в которое может быть вставлена последовательность действий из других случаев использования. Каждая точка использования должна иметь уникальное название внутри прецедента использования. Точки расширения могут быть перечислены в секции с заголовком extension points.

Прецедентов использования диаграмма (Use-case diagram)

Диаграммы прецедентов использования показывает отношение акторов (действующих лиц) и прецедентов использования внутри системы. С помощью такой диаграммы представляются функциональные возможности системы или класса как они проявляются при внешнем взаимодействии с системой. Диаграмма изображается в виде графа акторов, набора прецедентов использования, заключенных в границы системы, ассоциаций связи (участия) между акторами и прецедентами, и обобщений между прецедентами.

Размещения диаграмма (Deployment diagram)

Диаграммы размещения показывают конфигурацию обрабатывающих элементов времени выполнения и программных компонентов, процессов и объектов, размещенных в них. Компоненты, которые не являются сущностями времени выполнения показываться не должны. Диаграмма развертывания представляет граф узлов соединенных ассоциациями связи. Узлы могут содержать экземпляры компонентов; они показывают, что компоненты размещены или работают в узле. Компоненты соединяются с другими компонентами пунктирными стрелками зависимостей (возможно через интерфейс). Это значит, что один компонент использует сервисы другого компонента; для показа точного вида зависимости может использоваться стереотип.

Роль ассоциации (Role)

Роль ассоциации описывает конец ассоциации, которым она присоединяется к классу. Роль может включать следующие элементы.

Множественность - определяется с помощью текстового описания .

Упорядочение - если множественность больше единицы, то набор связанных элементов может быть упорядоченным или нет. Виды упорядочения могут определяться в виде ограничения.

Сортировка. Правило сортировки, определяется как отдельное ограничение.

Навигация показывается присоединенной к концу линии стрелкой.

Название роли - это строка названия недалеко от конца линии. Показывает роль играемую классом, присоединенным к ближайшему от названия роли концу.

Изменчивость. Свойство {frozen} показывает, что связи объекта не могут быть добавлены, удалены или перемещены после того, как объект создан и инициализирован. Свойство {addOnly} показывает, что связи могут быть добавлены , но не могут быть модифицированы или удалены.

Видимость определяется индикатором видимости перед названием роли. Определяет видимость ассоциации со стороны заданного наименования роли.

Спецификатор - атрибут или список атрибутов, чьи значения служат для разделения

множества объектов связанных с объектом с помощью ассоциации. Спецификаторы - атрибуты ассоциации. Спецификатор показывается как маленький прямоугольник, присоединенный к концу пути ассоциации между заключительным сегментом линии и символом класса, к которому он присоединяется.

Сноска (Note)

Сноска - это содержащий текстовую информацию (возможно и изображения) графический символ. Сноска используется для представления различных видов текстовой информации из метамодели, таких как ограничения, комментарии, тела методов и теговые значения. Сноска изображается в виде прямоугольника с подогнутым верхним правым углом. Она содержит произвольный текст. Сноска появляется на диаграммах и может быть, как присоединена к одному или более элементам модели пунктирными линиями, так и не присоединена ни к одному из элементов.

Состояний диаграмма (State diagram)

Диаграмма состояний показывает последовательность состояний, через которые проходит объект или взаимодействие за период жизни. Диаграмма состояния представляет собой конечный автомат. Состояния показываются символами состояний, переходы - соединенными с ними стрелками. Состояния могут также содержать поддиаграммы путем физического включения и разделения на ячейки. Состояние - ситуация в жизни объекта или взаимодействия за период которой, они удовлетворяют некоторым условиям, выполняют некоторые действия или ждут некоего события. Объект находится в состоянии конечное (не нулевое) время. Действия являются атомарными и не могут прерываться.

Сотрудничества диаграмма (Collaboration diagram)

Диаграмма сотрудничества показывает взаимодействие организованное вокруг объектов во взаимодействии и их связи друг с другом. Поведение обеспечивается множеством объектов, которые для достижения цели обмениваются сообщениями. Для понимания используемых в проекте механизмов важно видеть только объекты и сообщения, связанные с достижением цели. Такая статическая конструкция называется сотрудничеством. Сотрудничество может быть присоединено к операции или прецеденту использования для описания окружения, в котором возникает их поведение. Параметризованное сотрудничество представляет проектную структуру, которая может многократно использоваться в различных проектах.

Диаграмма сотрудничества представляет **сотрудничество**, которое является множеством объектов связанных в конкретном окружении, и **взаимодействие**, которое является множеством обмениваемых между объектами сотрудничества сообщений для достижения требуемых операции или результата.

Сотрудничество показывает окружение, в котором происходит взаимодействие. Динамическое поведение обмениваемой между объектами для достижения конкретной цели последовательности сообщений называется взаимодействием. Сотрудничество показывается диаграммой сотрудничества без сообщений. Добавляя сообщения, мы получаем взаимодействие. Различные совокупности сообщений могут применяться к одному и тому же сотрудничеству для создания различных взаимодействий. Диаграмма сотрудничества является графом ссылок на объекты и связи с потоками сообщений присоединенных к этим связям.

Стереотип (Stereotype)

Стереотип - новый класс моделирующих элементов, который вводится в процессе моделирования. Он представляет собой подкласс уже существующих элементов с одинаковой формой (атрибутами и отношениями), но с различными целями. Название стереотипа заключается в согласованные кавычки. Допускается связывать со стереотипом особую пиктограмму (icon) или графический маркер (такой как текстура или цвет).

Теговые значения (Tagged values)

Многие виды элементов имеют подробные свойства, которые не имеют визуальной нотации. Теговое значение представляет собой пару ключевое слово - значение, которая может быть присоединена к любому элементу модели (включая как элементы диаграмм,

так и семантические элементы модели). Ключевое слово называется ярлыком (tag). Каждый ярлык представляет собой особый вид свойств, подходящий к одному или более элементу модели. И ярлык, и значение кодируются строками. Теговые значения являются механизмом расширения UML, позволяющим присоединять к модели произвольную информацию. Свойства (либо атрибуты метамодели, либо связанные значения) изображаются как заключенная в фигурные скобки ({ }), разделенная запятыми последовательность описаний свойств. Описание свойства имеет форму:

keyword = value (ключевое слово = значение)

где ключевое слово - это название свойства, а значение - произвольная строка, которая показывает его значение. Если тип свойства - логический, то в случае, когда значение не указано по умолчанию принимается "true" (истина). Это значит, что для описания истинного значение достаточно включить только ключевое слово, а для описания ложного значения допустимо совсем его не включать. Для других типов свойств требуется явное указание значения.

{author = "Joe Smith", deadline = 31-March-1997, status = analysis}

{abstract}

Утилита (Utility)

Утилита - группировка глобальных переменных и процедур в форме объявления класса. Это не фундаментальная конструкция, а средство повышения удобства программирования. Все атрибуты и операции утилиты являются глобальными переменными и процедурами. Утилита моделируется как соответствующий стереотип класса.

Шаблон (Template)

Шаблон - это описатель для класса с одним или более несвязанными параметрами. Он определяет семейство классов, каждый из которых определяется связыванием параметров с реальными значениями (инстанцированием). Параметры представляют собой типы или объекты. Атрибуты и операции в шаблоне определяются в терминах формальных параметров.

Так как шаблон содержит несвязанные параметры, то он непосредственно не используется как класс. Для создания шаблонного класса (класса инстанцированного по шаблону), параметры должны быть связаны с реальными значениями. Например, VArray обозначает класс, описанный шаблоном VArray. Количество и типы значений должны соответствовать количеству и типам параметрам шаблона с заданным названием.

Параметризация может применяться и к другим элементам модели, таким как сотрудничества или пакетам.

Для изображения шаблона в верхнем правом углу прямоугольника класса (или символа другого моделирующего элемента) добавляется маленький пунктирный прямоугольник. Он содержит список формальных параметров класса и типы их реализаций.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ ПРЕПОДАВАТЕЛЯ ПО ДИСЦИПЛИНЕ

Методические принципы и приемы построения учебной дисциплины «Объектно-ориентированное программирование». Ключевыми методическими способами подачи учебного материала по дисциплине «Объектно-ориентированное программирование» являются лекции и семинарские занятия.

Форма промежуточной аттестации знаний — **зачёт.**

Лекционное занятие — это систематическое, последовательное, устное изложение лектором учебного материала. Занятие «лекция» носит, прежде всего, обзорный характер, охватывая весь круг выносимых на изучение учебных вопросов. При проведении такого типа занятий очень важно живое слово лектора, его педагогическое мастерство как педагога, который дает студентам информационную базу. Лекции являются важной формой передачи преподавателем студентам общетеоретических знаний.

Лекции, как правило, читаются не по всем, а по наиболее сложным темам курса, не дублируют учебники, а содержат новейшие научные данные и примеры, которых может не быть в учебных пособиях. Для лучшего усвоения материала на лекционных занятиях целесообразно предварительно перед лекцией ознакомиться с положениями лекционной темы в конспекте лекций, содержащемся в данном учебно-методическом пособии либо в рекомендуемых учебниках.

Практическое (подгрупповое) занятие — другая важная форма учебного процесса. Они способствуют закреплению и углублению знаний, полученных студентами на лекциях и в результате самостоятельной работы над научной и учебной литературой и нормативными источниками. Они призваны развивать самостоятельность мышления, умение делать выводы, связывать теоретические положения с практикой, формировать профессиональное правовое сознание. На занятиях вырабатываются необходимые каждому бакалавру навыки и умения публично выступать, логика доказывания, культура профессиональной речи. Кроме того, занятия — это средство контроля преподавателей за самостоятельной работой студентов, они непосредственно влияют на уровень подготовки к итоговым формам отчетности — зачетам и экзаменам. В работе и выступлении на занятии должны содержаться следующие элементы:

- четкое формулирование соответствующего теоретического положения в виде развернутого определения;
- приведение и раскрытие основных черт, признаков, значения и роли изучаемого явления или доказательства определенного теоретического положения;
- подкрепление теоретических положений конкретными фактами.

Для качественного и эффективного изучения актуальных проблем теории необходимо овладение навыками работы с книгой, воспитание в себе стремления и привычки получать новые знания из научной и иной специальной литературы. Без этих качеств не может быть настоящего специалиста ни в одной области деятельности.

Читать и изучать, следует, прежде всего, то, что рекомендуется к каждой теме программой, планом семинарских занятий, перечнем рекомендуемой литературы.

Когда студент приступает к самостоятельной работе, то он должен проявить инициативу в поиске специальных источников. Надо иметь в виду, что в каждом последнем номере издаваемых журналов публикуется библиография всех статей, напечатанных за год, это облегчает поиск нужных научных публикаций.

Работа с научной литературой, в конечном счете, должна привести к выработке у бакалавра умения самостоятельно размышлять о предмете и объекте изучения, которое должно проявляться:

- в ясном и отчетливом понимании основных понятий и суждений, содержащихся в публикации, разработке доказательств, подтверждающих истинность тех или иных положений;

- в понимании студентами обоснованности и целесообразности, приводимых в книге и статье примеров, поясняющих доказательства и выводы автора. При этом будет уместно, если бакалавр самостоятельно приведет дополнительные примеры к этим выводам;
- в отделении основных положений от дополнительных, второстепенных сведений;
- в способности студента критически разобраться в содержании публикации, определить свое отношение к ней в целом, дать ей общую оценку, характеристику.

Другим важнейшим методическим приемом в учебном процессе является самостоятельная работа студента.

Самостоятельная работа в высшем учебном заведении, является важной организационной формой индивидуального изучения студентами программного материала.

В современных условиях дидактическое значение самостоятельной подготовки неизмеримо возрастает, а ее цели состоят в том, чтобы:

- повысить ответственность самих обучаемых за свою профессиональную подготовку, сформировать в себе личностные и профессионально-деловые качества;
- научить студентов самостоятельно приобретать знания, формировать навыки и умения, необходимы для программистской деятельности;
- развивать в себе самостоятельность в организации, планировании и выполнении заданий, определяемых учебным планом и указаниями преподавателя.

Достигнуть этих целей в ходе самостоятельной работы при изучении дисциплины возможно только при хорошей личной организации своего учебного труда, умении использовать все резервы имеющегося времени и подчинить их профессиональной подготовке.

Самостоятельная работа как метод обучения включает:

- изучение и конспектирование обязательной литературы в соответствии с программой дисциплины;
- ознакомление с литературой, рекомендованной в качестве дополнительной;
- изучение и осмысление специальной терминологии и понятий;
- изучение и отработка нормативных актов, комментариев к ним, проведение сравнительного анализа с предыдущим;
- сбор материала и написание контрольных, конкурсных и дипломных работ;
- изучение указанной литературы для подготовки к зачету.
- Основными компонентами содержания данного вида работы являются:
- творческое изучение учебных пособий и научной литературы;
- умелое конспектирование;
- участие в различных формах учебного процесса, научных конференциях, в работе кружков и т. д.;
- получение консультаций у преподавателя по отдельным проблемам курса;
- получение информации и опыта о работе профессионалов в процессе производственно-учебной практики;
- знакомство со специальной литературой при формировании своей личной библиотеки и др.

Данный комплекс рекомендаций позволяет студентам овладеть многими важными приемами самостоятельной работы и успешно использовать их при подготовке контрольных по дисциплине.

Важнейшей формой учебной отчетности студента являются индивидуальные задания по практическим работам. Выполнение индивидуальных заданий по практическим работам является промежуточной формой отчетности по изучаемой дисциплине и преследует цель лишь оценить способность студента к самостоятельному поиску источников, формированию содержания и его письменного изложения по указанной проблеме. Это важная составляющая изучения дисциплины, а также эффективная форма контроля знаний.

Как правило, индивидуальных заданий по практическим работам по дисциплине сугубо индивидуальны, то есть их тематика персонифицирована. Однако в отдельных случаях темы работ могут быть адресованы и сразу нескольким бакалаврам, и группе в целом. Таким приемом преподаватель выявляет степень усвоения какой-то важной учебной проблемы и определяет необходимость проведения дополнительных занятий по какой-либо теме.

Игра позволяет влиять на правовые установки студентов. Учебно-правовые ситуации относятся к тем методическим средствам, которые позволяют осуществлять взаимосвязь понятийно-категориального уровня правосознания с поведенческим. В результате достигается не только интеллектуальный, но и эмоциональный уровень усвоения правовых понятий и идей.

Учебно-тренировочные ситуации являются специфическим методическим приемом, одним из основных видов проблемно-развивающего обучения, благодаря которому усиливается практический интерес бакалавров к теоретико-правовым вопросам.

Эффективность применения учебных ситуаций зависит от соблюдения следующих условий: знание студентами теоретического материала и наличие достаточного личного опыта и жизненного опыта вообще.

Важными в методическом плане на семинарских занятиях являются проводимые **тестовые опросы** и решение задач, которые содействуют превращению знаний в глубокие убеждения, дают простор для развития творческо-эмоциональной сферы, позволяют сделать выводы об эффективности занятий с учащимися, что в итоге повышает интерес к овладению знаниями.

Только сочетая дидактически и органически все методические способны и приемы в их диалектическом единстве и взаимосвязи мы можем добиться должного уяснения учебного материала со стороны студентов.

Методические рекомендации для преподавателей

Тема занятия	**Виды учебных занятий**	**Способы учебной деят-сти**	**Методы обучения, формы педагогического общения**	**Средства обучения**	**Формы контроля**
Основные понятия дисциплины	Лекция	Коллективный	Методы: объяснительно-иллюстративный, репродуктивный. Формы: монолог/диалог	Проектор, компьютеры с установленным программным обеспечением, презентация, электронный курс по дисциплине	Устный опрос
Введение в объектно-ориентированный анализ и проектирование	Лекция	Коллективный	Методы: объяснительно-иллюстративный, репродуктивный. Формы: монолог/диалог	Проектор, компьютеры с установленным программным обеспечением, презентация, электронный курс по дисциплине	Устный опрос
Классы	Лекция, подгрупповое	Индивидуально-группово	Методы: объяснительно-иллюстративны	Проектор, компьютеры с установленным	Проверка индивидуальных заданий

	занятие	й	й, репродуктивный. Формы: монолог/диалог	программным обеспечением, презентация, электронный курс по дисциплине	
Шаблоны и обработка исключительных ситуаций	Лекция, подгрупповое занятие	Коллективный, Индивидуально-групповой	Методы: объяснительно-иллюстративный, репродуктивный. Формы: монолог/диалог	Проектор, компьютеры с установленным программным обеспечением, презентация, электронный курс по дисциплине	Проверка индивидуальных заданий

Приложение

Тематический план изучения дисциплины
«Объектно-ориентированное программирование»

Год набора с 2020, форма обучения очная

Наименование разделов и тем	**Всего**	**Трудоемкость по дисциплине**					**Формируемые компетенции**
		Контактная работа	в т.ч.			СР	
			Лекции	Подгр. \Лаб.	Пр.\Сем.		
Основные понятия дисциплины	42	12	4	8	0	30	ПК-3
Введение в объектно-ориентированный анализ и проектирование	42	12	4	8	0	30	ПК-3
Классы	54	18	6	12	0	36	ПК-3
Шаблоны и обработка исключительных ситуаций	42	12	4	8	0	30	ПК-3
Зачет							
Итого по дисциплине	**180**	**54**	**18**	**36**	**0**	**126**	
Зачетных единиц	**5**						

Тематический план изучения дисциплины
«Объектно-ориентированное программирование»

Год набора с 2020, форма обучения заочная

Наименование разделов и тем	**Всего**	**Трудоемкость по дисциплине**					**Формируемые компетенции**
		Контактная работа	в т.ч.			СР	
			Лекции	Подгр. \Лаб.	Пр.\Сем.		
Основные понятия дисциплины	32	2	2	0	0	30	ПК-3
Введение в объектно-ориентированный анализ и проектирование	52	2	2	0	0	50	ПК-3
Классы	52	6	0	6	0	46	ПК-3
Шаблоны и обработка исключительных ситуаций	40	4	0	4	0	36	ПК-3
Зачет	**4**	**4**					
Итого по дисциплине	**180**	**14**	**4**	**10**	**0**	**162**	
Зачетных единиц	**5**						
Контрольная работа	+						