УТВЕРЖДЕНО
решением Ученого совета факультета математики, информационных и авиационных технологий
от «21» 06 2019 г., протокол № 5/19
Председатель _____ М.А. Волков
(подпись, расшифровка подписи)
«21» 06 2019 г

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Дисциплина:	Объектно-ориентированное программирование
Факультет	Математики, информационных и авиационных технологий
Кафедра	Информационных технологий
Курс	3

Направление (специальность): 10.05.03 Информационная безопасность автоматизированных систем
(код специальности (направления), полное наименование)

Направленность (профиль): Безопасность открытых информационных систем
(полное наименование)

Форма обучения: очная
(очная, заочная, очно-заочная (указать только те, которые реализуются)

Дата введения в учебный процесс УлГУ: «_01_»_____09_____2019_г.

Программа актуализирована на заседании кафедры: протокол № 1__ от 31 августа 2020 г.
Программа актуализирована на заседании кафедры: протокол № 1___ от 31 августа 2020 г.
Программа актуализирована на заседании кафедры, протокол №__ от «__»_____20__г.

Сведения о разработчиках:

ФИО	Кафедра	Должность, ученая степень, звание
Головин Вячеслав Александрович	Информационных технологий	доцент, к.т.н.

СОГЛАСОВАНО	СОГЛАСОВАНО
Заведующий кафедрой, реализующей дисциплину	Заведующий выпускающей кафедрой
/_____/__Волков М.А._/ *(подпись)* *(Ф.И.О.)* «21» июня 2019 г.	/_____/__Андреев А.С._/ *(подпись)* *(Ф.И.О.)* «21» июня 2019 г.

1. ЦЕЛИ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Дисциплина «Объектно-ориентированное программирование» имеет целью:

- обучить студентов принципам объектно-ориентированного программирования;
- обучить студентов принципам декомпозиции при решении поставленных задач;
- обучить студентов принципам современным методам написания кода.

2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ОПОП

Дисциплина «Объектно-ориентированное программирование» входит в состав вариативной части Основной Профессиональной Образовательной Программы по специальности 10.05.03 Информационная безопасность автоматизированных систем.

В соответствии с учебным планом образовательной программы изучение данной дисциплины предусмотрено в 5-м семестре и логически взаимосвязано с изучающимися ранее дисциплинами.

Данная дисциплина базируется на учебных дисциплинах, указанных в фонде оценочных средств – далее ФОС, пункт 1).

Результаты освоения дисциплины будут необходимы для дальнейшего процесса обучения в рамках поэтапного формирования компетенций при изучении последующих дисциплин (указаны в ФОС, пункт 1), а также для прохождения всех видов практик и государственной итоговой аттестации.

3. ПЕРЕЧЕНЬ ПЛАНИРУЕМЫХ РЕЗУЛЬТАТОВ ОСВОЕНИЯ ДИСЦИПЛИНЫ (МОДУЛЯ), СООТНЕСЕННЫХ С ПЛАНИРУЕМЫМИ РЕЗУЛЬТАТАМИ ОСВОЕНИЯ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

Процесс изучения дисциплины, в соответствии с целями основной профессиональной образовательной программы и задачами профессиональной деятельности, направлен на формирование следующих компетенций:

Код и наименование реализуемой компетенции	**Перечень планируемых результатов обучения по дисциплине (модулю), соотнесенных с индикаторами достижения компетенций**
Способностью использовать основы правовых знаний в различных сферах деятельности (ОК-4)	**знать:** основы правовых знаний в различных сферах деятельности **уметь:** использовать правовые знания в различных сферах деятельности **владеть:** навыками использования правовых знаний в различных сферах деятельности
Способностью применять методы научных исследований в профессиональной деятельности, в том числе в работе над междисциплинарными и инновационными проектами (ОПК-5)	**знать:** основные методы научных исследований в профессиональной деятельности, в том числе в работе над междисциплинарными и инновационными проектами **уметь:** применять основные методы научных исследований в профессиональной деятельности

	владеть: навыками применения методов научных исследований в профессиональной деятельности
Способностью осуществлять поиск, изучение, обобщение и систематизацию научно-технической информации, нормативных и методических материалов в сфере профессиональной деятельности, в том числе на иностранном языке (ПК-1)	**знать:** основные нормативные и методические материалы в сфере профессиональной деятельности **уметь:** осуществлять поиск, изучение, обобщение и систематизацию научно-технической информации, нормативных и методических материалов в сфере профессиональной деятельности, в том числе на иностранном языке **владеть:** навыками поиска, изучения, обобщения и систематизации научно-технической информации, нормативных и методических материалов в сфере профессиональной деятельности, в том числе на иностранном языке

4. ОБЩАЯ ТРУДОЕМКОСТЬ ДИСЦИПЛИНЫ

4.1 Объем дисциплины в зачетных единицах (всего) 3 зачетных единицы

4.2 По видам учебной работы (в часах):

Вид учебной работы	**Количество часов (форма обучения очная)**	
	Всего по плану	**В т.ч. по семестрам**
		5
Контактная работа обучающихся с преподавателем	36/36	36/36
Аудиторные занятия:	36/36	36/36
Лекции	18/18	18/18
практические и семинарские занятия	-	-
лабораторные работы (лабораторный практикум)	36/36	36/36
Самостоятельная работа	54	54
Текущий контроль (количество и вид: конт.работа, коллоквиум,реферат)	Индивидуальное практическое задание	Индивидуальное практическое задание.
Курсовая работа		
Виды промежуточной аттестации (экзамен, зачет)	зачёт	зачёт
Всего часов по дисциплине	108	108

В случае необходимости использования в учебном процессе частично/исключительно дистанционных образовательных технологий в таблице через слеш указывается количество часов работы ППС с обучающимися для проведения занятий в дистанционном формате с применением электронного обучения.

4.3 Содержание дисциплины (модуля). Распределение часов по темам и видам учебной работы:

Форма обучения: очная

Название и разделов и тем	Всего	Виды учебных занятий				
		Аудиторные занятия				Самостоятельная работа
		лекции	Практические занятия, семинар	Лабораторная работа	в т.ч. занятия в интерактивной форме	
Раздел 1. Введение						
Тема 1. Парадигмы программирования. Абстракция данных. Объектно-ориентированное программирование.	6	2				4
Тема 2. Уровни доступа, статические поля класса.	6	2				4
Тема 3. Роль UML в ОПОП.	6	2				4
Раздел 2. Методы и классы						
Тема 4. Перегрузка методов, конструкторы, пакеты, финализация	26	2		12	6	12
Тема 5. Наследование, композиция, подклассы, многоуровневое наследование.	18	2		8	4	8
Тема 6. Восходящее преобразование типов, полиморфизм.	18	2		8	4	8
Тема 7. Абстрактные классы, интерфейсы	12	2		4	4	6
Раздел 3. Программное проектирование						
Тема 8. Паттерны проектирования.	6	2				4
Тема 9. Параметризованные классы.	10	2		4		4
Итого	**108**	**18**		**36**	**18**	**54**

5. СОДЕРЖАНИЕ КУРСА

Раздел 1. Введение

Тема 1. Парадигмы программирования. Процедурное. Модульное. Абстракция данных. Объектно-ориентированное программирование.

Совокупность принципов, методов и понятий, определяющих способ конструирования

программ.

Тема 2. Уровни доступа, статические поля класса.

Переменные, описание которых создает программист при создании класса. Все данные объекта хранятся в его полях. Доступ к полям осуществляется по их имени. Обычно тип данных каждого поля задаётся в описании класса.

Тема 3. Роль UML в ОПОП.

Описывается язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур

Раздел 2. Методы и классы

Тема 4. Перегрузка методов, конструкторы, пакеты, финализация.

Описывается приём программирования, который позволяет разработчику в одном классе для методов с разными параметрами использовать одно и то же имя. А также метод класса, который автоматически вызывается средой исполнения в промежутке времени между моментом, когда объект этого класса опознаётся сборщиком мусора как неиспользуемый, и моментом удаления объекта (освобождения занимаемой им памяти).

Тема 5. Наследование, композиция, подклассы, многоуровневое наследование.

О концепция объектно-ориентированного программирования, согласно которой абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа, способствуя повторному использованию компонентов программного обеспечения.

Тема 6. Восходящее преобразование типов, полиморфизм.

О степени отделения интерфейса от реализации, разъединения что от как. Полиморфизм улучшает организацию кода и его читаемость, а также способствует созданию расширяемых программ, которые могут «расти» не только в процессе начальной разработки проекта, но и при добавлении новых возможностей.

Тема 7. Абстрактные классы, интерфейсы

Описание базовых классов, которые не предполагают создания экземпляров. Абстрактные классы реализуют на практике один из принципов ООП — полиморфизм. Абстрактные классы представляют собой наиболее общие абстракции, то есть имеющие наибольший объём и наименьшее содержание.

Раздел 3. Программное проектирование

Тема 8. Паттерны проектирования.

Описывается повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

Тема 9. Параметризованные классы.

Описывается шаблон, на основе которого можно строить другие классы. Этот класс можно рассматривать как некоторое описание множества классов, отличающихся только типами их данных.

6. ТЕМЫ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Данный вид работы не предусмотрен УП.

7. ЛАБОРАТОРНЫЕ РАБОТЫ, ПРАКТИКУМЫ

Лабораторная работа №1

Цель: Написать консольное приложение.

В рамках данной лабораторной работы необходимо создать структуру согласно вашему

варианту и реализовать следующий набор функций для работы с созданным пользовательским типом данных:

- ввод данных из потока;
- вывод данных в поток;
- инициализация полей структуры;
- получение отдельных полей структуры.

Результат: Продемонстрировать на примере работу с созданным пользовательским типом.

Лабораторная работа №2

Цель: Реализовать класс с разделением полей и методов класса по секциям public, private. Перегрузить метод toString(). Использовать ключевое слово this.

Результат: Продемонстрировать на примере работу с созданным пользовательским классом.

Лабораторная работа №3

Цель: В классе реализовать перегруженные конструкторы и методы. Реализовать методы для реализации операций: сложения, вычитания, умножения и деления или аналогичные им. Реализовать метод compareTo(). Расширить класс. Добавить в класс поле в котором будет храниться дата и время последней модификации экземпляра класса. Правильно реализовать клонирование (не поверхностное).

Результат: Продемонстрировать на примере работу с созданным пользовательским классом.

Методические указания:

Лабораторная работа № 1.

Объекты важны при описании выполнения ОО-систем. Но базовым понятием объектной технологии является класс.

Класс — это абстрактный тип данных, поставляемый с возможно частичной реализацией. Класс — это тип, описывающий множество возможных структур данных, называемых экземплярами класса. Экземпляры являются абстракциями — элементами математического множества. Экземпляр класса конкретен — это структура данных, размещаемая в памяти компьютера и обрабатываемая программой.

Например, если определить класс POINT, моделирующий точку на плоскости, если для представления точки выбрана декартова система координат, то каждый экземпляр POINT представляет собой запись с полями x, y — абсциссой точки и ее ординатой, если же выбрана полярная система координат, то радиус и угол поворота.

Термин "объект" появляется как побочный продукт определения "класса". Объект — это просто экземпляр некоторого класса.

Устранение традиционной путаницы

Класс — это модель, а объект — экземпляр такой модели. Эта особенность настолько очевидна, что обычно не требует дополнительных комментариев. Однако, в литературе можно встретить смешение этих понятий. У этой путаницы два источника. Один — возникает из-за широкого толкования термина "объект" в естественном языке. Другой источник недоразумений связан с метаклассами, — с ситуациями, когда классы сами выступают в роли объектов. Классическим примером может служить транслятор объектного языка, для которого классы языка являются объектами трансляции. Метакласс — это класс, экземпляры которого сами являются классами.

Роль классов

Для понимания ОО-подхода необходимо ясно представлять, что классы выполняют две функции, которые до появления ОО-технологий всегда были разделены. Класс одновременно является модулем и типом.

Модули и типы

Модули — это структурные единицы, из которых состоит программа. Существуют различные виды модулей, такие как подпрограммы, пакеты и т.д. Независимо от конкретного выбора той или иной модульной структуры, модуль всегда рассматривается как синтаксическая концепция, т.е. может компилироваться независимо. Отсюда следует, что разбиение на модули влияет лишь на форму записи исходных текстов программ, но не определяет их функциональность.

Концепция типов на первый взгляд совершенно иная. Тип является статическим описанием вполне определенных динамических объектов — элементов данных, которые обрабатываются во время выполнения программной системы. Набор типов обычно содержит предопределенные типы, такие как INTEGER или CHARACTER, а также пользовательские типы: записи (структуры), указатели, множества, массивы и другие. Понятие типа является семантической концепцией, и каждый тип непосредственно влияет на выполнение программной системы, так как описывает форму объектов, которые система создает и которыми она манипулирует.

Унифицированная система типов

Важным аспектом ОО-подхода является простота и универсальность системы типов, которая строится на основе фундаментального принципа.

Каждый объект является экземпляром некоторого класса.

Объектный принцип будет распространяться не только на составные объекты, определяемые разработчиками, но и на базовые объекты — целые и действительные числа, булевы значения и т.д.

На первый взгляд подобное стремление превратить любое сколь угодно простое значение в экземпляр некоторого класса может показаться преувеличенным и даже экстравагантным. Однако настойчивое требование к унификации вполне окупается по ряду причин.

 Всегда желательно иметь простую и универсальную схему, нежели множество частных случаев. Предлагаемая система типов полностью опирается на понятие класса.

 Описание базовых типов как абстрактных структур данных и далее, как классов, является простым и естественным. Нетрудно представить, например, определение класса INTEGER с функциональностью, включающей арифметические операции, такие как "+", операции сравнения, такие как "=" и ассоциированные свойства, следующие из соответствующих математических аксиом.

 Определение базовых типов как классов позволяет использовать все возможности ОО, главным образом наследование и родовые средства. Если базовые типы не будут классами, то придется вводить ряд ограничений и рассматривать частные случаи.

Унифицированный доступ (функции и атрибуты)

Во многих случаях необходимо иметь возможность работать с объектом, например, с точкой p1, не заботясь о том, какое внутреннее представление используется для p1 — декартово, полярное или иное. Необходимо ли для этого отличать атрибуты от функций?

Ответ зависит от того, с какой точки зрения рассматривать данную проблему — разработчика, автора данного класса POINT или клиента, создавшего класс, использующий POINT. Для разработчика разница между атрибутами и функциями принципиально важна и имеет смысл. Ему необходимо принимать решения о том, какие компоненты будут реализованы как данные в памяти и какие будут доступны в результате вычислений. Но заставлять клиента осознавать эту разницу, было бы серьезной ошибкой. Клиент должен обращаться к значениям x или ρ для точки p1, не заботясь и не имея информации о том, как реализованы соответствующие запросы.

Решение проблемы дает принцип унифицированного доступа, принцип декларирует, что клиент должен иметь возможность доступа к свойствам объекта, используя

одинаковую нотацию, вне зависимости от того, как это свойство реализовано — в памяти или как результат вычислений.

В объектно-ориентированном программировании (ООП) – класс это основной элемент, в рамках которого осуществляется конструирование программ. Класс содержит в себе данные и код, который управляет этими данными.

Класс зачастую описывает объект реального мира. Как и реальный объект, класс содержит свой набор параметров и характеристик. Каждый такой параметр называется поле класса (очень похоже на обычные переменные). Также класс способен манипулировать своими характеристиками (полями) с помощью методов класса (похожи на функции в процедурных языках).

Лабораторная работа №2

Класс — это абстрактный тип данных, снабженный некоторой (возможно частичной) реализацией

Полностью реализованный класс называется эффективным (effective). Класс, который реализован лишь частично или совсем не реализован, называется отложенным (deferred). Всякий класс является либо отложенным, либо эффективным.

Чтобы получить эффективный класс, требуется предусмотреть все детали реализации. Для отложенного класса можно выбрать определенный уровень реализации, но при этом оставить некоторые аспекты реализации незавершенными. В самом крайнем случае при частичной реализации можно вообще отказаться от принятия каких-либо решений о ее уточнении. В этом случае получившийся класс будет полностью отложенным и будет эквивалентен АТД.

Эффективные классы

Рассмотрим вначале эффективные классы. Что нужно сделать для реализации АТД? Результирующий эффективный класс будет формироваться из элементов трех видов:

- (E1) Спецификации АТД (множество функций с соответствующими аксиомами и предусловиями, описывающими их свойства).
- (E2) Выбора представления.
- (E3) Отображения из множества функций (E1) в представление (E2) в виде множества механизмов (или компонентов (features)), каждый из которых реализует одну из функций в терминах представления и при этом удовлетворяет аксиомам и предусловиям. Многие из этих компонентов будут методами — обычными процедурами, но некоторые могут появляться в качестве полей данных или "атрибутов".

Отложенные классы

Отложенные классы особенно полезны при анализе и проектировании:

- При ОО-проектировании многие аспекты реализации будут опущены, проектирование должно сосредотачиваться на архитектурных свойствах высокого уровня — на том, какую функциональность обеспечивает каждый модуль системы, а не на том, как он это делает.
- При постепенном продвижении к полной реализации будут добавляться все новые и новые ее свойства до тех пор, пока не будет получен эффективный класс.

Но на этом роль отложенных классов не завершается, даже в полностью реализованной системе можно часто обнаружить много таких классов. Кое-что следует из только что перечисленных применений: когда из отложенных классов получаются эффективные, то появляется желание сохранить их в качестве предков (в смысле наследования) эффективных классов как живую память о процессе анализа и проектирования.

Очень часто при разработке ПО с помощью не ОО-подходов система в окончательном виде не содержит никаких записей о тех значительных усилиях, которые были затрачены на ее получение. Для тех, кто вынужден будет обслуживать такую

систему — расширять, переносить, отлаживать — понять ее без этих записей будет затруднительно.

У отложенных классов имеется также применение, полностью связанное с реализацией. Они служат для классификации групп связанных типов объектов, предоставляют некоторые наиболее важные многократно используемые модули высокого уровня, фиксируют общие свойства поведения.

Перегрузка — это связывание с одним именем более одного содержания. Наиболее часто перегружаются имена переменных: почти во всех языках программирования различные по смыслу переменные могут иметь одно и то же имя, если они принадлежат различным модулям.

Для этого обсуждения более существенной является перегрузка подпрограмм, частным случаем которой является перегрузка операторов, которая позволяет использовать одинаковые имена для нескольких подпрограмм. Такая возможность почти всегда имеет место для арифметических операторов: одна и та же запись, a+b, означает различные виды сложения, в зависимости от типов a и b.

Роль перегрузки — Перегрузка подпрограмм является средством, предназначенным для клиентов. Она позволяет писать один и тот же текст, используя разные реализации некоторого понятия.

Так что же дает перегрузка подпрограмм решению проблемы повторного использования? Это — синтаксическое средство, освобождающее разработчиков от необходимости придумывать различные имена для разных реализаций некоторой операции и, по существу, перекладывает эту ношу на компьютер.

Лабораторная работа №3

Написать программу, демонстрирующую работу с объектами двух типов: T1 и T2, для чего создать систему соответствующих классов. Каждый объект должен иметь идентификатор (в виде произвольной строки символов) и одно или несколько полей для хранения состояния (текущего значения) объекта.

Клиенту (функции main) должны быть доступны следующие основные операции (методы): создать объект, удалить объект, показать значение объекта и прочие дополнительные операции (зависят от варианта). Операции по созданию и удалению объектов инкапсулировать в классе Factory. Предусмотреть меню, позволяющее продемонстрировать заданные операции.

При необходимости в разрабатываемые классы добавляются дополнительные методы (например, конструктор копирования, операция присваивания и т. п.) для обеспечения надлежащего функционирования этих классов.

Далее перечислены возможные типы объектов и возможные дополнительные операции над ними

Класс	**Объект**
SymbString	Символьная строка (произвольная строка символов)
BinString	Двоичная строка (изображение двоичного числа)
OctString	Восьмеричная строка (изображение восьмеричного числа)
DecString	Десятичная строка (изображение десятичного числа)
HexString	Шестнадцатеричная строка (изображение шестнадцатеричного числа)

Перечень дополнительных операций (методов)

Операция (метод)	**Описание**
ShowBinO	Показать изображение двоичного значения объекта
ShowOctO	Показать изображение восьмеричного значения объекта
ShowDec ()	Показать изображение десятичного значения объекта

ShowHexO Показать изображение шестнадцатеричного значения объект
operator +(T& s1, T& s2) Для объектов SymbString — конкатенация строк s1 и s2; для объектов прочих классов — сложение соответствующие численных значений с последующим преобразованием к типу Т.

Операция (метод) Описание
operator –(T& s1, T& s2) Для объектов SymbString — если s2 содержится как подстрока в s1, то результатом является строка, полученная из s1 удалением подстроки s2; в противном случае возвращается значение s1;
для объектов прочих классов — вычитание соответствующих численных значений с последующим преобразованием к типу Т
Примечание: Первые четыре операции могут применяться к объектам любых классов, за исключением класса SymbString. Таблица 2.3 содержит спецификации вариантов.
Спецификации вариантов

Вариант	T1	T2	**Операции (методы)**
	SymbString	BinString	ShowOct(), ShowDec(),ShowHex()
2	SymbString	BinString	operator +(T&, T&)
з	SymbString	BinString	operator -(T&, T&)
4	SymbString	OctString	operator +(T&, T&)
5	SymbString	OctString	operator -(T&, T&)
6	SymbString	DecString	ShowBin(), ShowOct(), ShowHex()
4	SymbString	DecString	operator +(T&, T&)
6	SymbString	DecString	operator -(T&, T&)
9	SymbString	HexString	operator +(T&, T&)
10	SymbString	HexString	operator -(T&, T&)

Перечень типов объектов

Класс	**Объект**
Triangle	Треугольник
Quadrate	Квадрат
Rectangle	Прямоугольник
Tetragon	Четырехугольник
Pentagon	Пятиугольник

Перечень дополнительных операций (методов)

Операция (метод)	**Описание**
MoveO	Переместить объект на плоскости
Compare(T& ob1, T& ob2)	Сравнить объекты ob1 и ob2 по площади
IsIntersect(T& ob1, T& ob2)	Определить факт пересечения объектов ob1 и ob2 (есть пересечение или нет)
IsInclude(T& ob1, T& ob2)	Определить факт включения объекта ob2 в объект ob1

8. ТЕМАТИКА КУРСОВЫХ, КОНТРОЛЬНЫХ РАБОТ, РЕФЕРАТОВ

Данный вид работы не предусмотрен УП.

9. ПЕРЕЧЕНЬ ВОПРОСОВ К ЭКЗАМЕНУ

1. JVM, байт-код, JIT-компиляция, JDK, JRE, native-methods.
2. Понятие сборщика мусора.
3. Первая программа на Java.
4. Code Conventions for the Java Programming Language

Таблица 6. Спецификации вариантов 11-20

Вариант	**T1**	**T2**	**Операции (методы)**	
11	Triangle	Quadrate	Move().	Compare(T&, T&)
12	Quadrate	Pentagon	Move().	IsIntersect(T&, T&)
13	Triangle	Rectangle	Move().	Compare(T&, T&)
14	Triangle	Rectangle	Move().	IsIntersect(T&, T&)
15	Rectangle	Pentagon	Move().	IsInclude(T&, T&)
16	Triangle	Tetragon	Move().	Compare(T&, T&)
17	Triangle	Tetragon	Move(),	IsIntersect(T&, T&)
18	Triangle	Tetragon	Move().	IsInclude(T&, T&)
19	Triangle	Pentagon	Move().	Compare(T&, T&)
20	Triangle	Pentagon	Move().	IsIntersect(T&, T&)

5. Комментарии, аннотации
6. Встроенные типы данных и операции над ними. Приоритеты операций.
7. Константы: целые, действительные, символы, строки.
8. Примитивные и ссылочные типы данных.
9. Приведение типов.
10. Условный оператор.
11. Операторы цикла, break, continue, goto.
12. Массивы: объявление, определение, инициализация, цикл for-each, многомерные массивы.
13. Оператор варианта.
14. Процедурное программирование, модульное программирование, объектно-ориентированное программирование.
15. Принципы ОПОП: абстракция, иерархия, ответственность, модульность, KISS, инкапсуляция, наследование, полиморфизм.
16. Статические поля класса
17. Уровни доступа public, protected, private.
18. Сеттеры и геттеры.
19. Ключевое слово this.
20. Роль UML в ОПОП.
21. Перегрузка методов.
22. Конструкторы. Конструкторы по умолчанию. Вызов конструктора из конструктора.
23. Способы инициализации полей.

24. Инициализация статических полей.
25. Пакеты.
26. Спецификаторы доступа.
27. Доступ к классам.
28. finalize() и сборка мусора.
29. Понятие композиции классов.
30. Подклассы.
31. Делегирование.
32. Доступ к элементам суперкласса.
33. Конструкторы и наследование.
34. Переопределение методов при наследовании.
35. Многоуровневое наследование.
36. Восходящее преобразование типов
37. Ключевое слово final:
    для примитивных типов;
    для ссылочных типов;
38. пустые константы;
39. неизменные аргументы;
40. неизменные методы;
41. неизменные классы.
42. Позднее связывание, полиморфизм.
43. Конструкторы и полиморфизм.
44. Абстрактные классы.
45. Интерфейсы.
46. Отделение интерфейса от реализации.
47. Паттерны.
48. Параметризация классов и интерфейсов.
49. Итераторы.
50. List.
51. LinkedList.
52. Стек.
53. Множество.
54. Карта.
55. Очередь.
56. Обработка исключительных ситуаций.

10. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ

Форма обучения: очная

Название разделов и тем	**Вид самостоятельной работы** ***(проработка учебного материала, решение задач, реферат, доклад, контрольная работа, подготовка к сдаче зачета, экзамена и др.)***	**Объем в часах**	**Форма контроля (*проверка решения задач, реферата и др.)***
Раздел 1-3	– для овладения знаниями: чтение текста (учебника, первоисточника, дополнительной литературы): составление схем и таблиц по тексту, конспектирование текста; выписки	См. табл. 4.3	Проверка домашних и практических работ, заданий,

	из текста; работа со словарями и справочниками, ознакомление с нормативными документами; учебно-исследовательская работа; использование аудио- и видеозаписей, компьютерной техники и Интернета и др.; – для закрепления и систематизации знаний: работа с конспектом лекции (обработка текста); повторная работа над учебным материалом (учебника, первоисточника, дополнительной литературы, аудио- и видеозаписей); составление плана и тезисов ответа; составление таблиц для систематизации учебного материала; ответы на контрольные вопросы; подготовка сообщений к выступлению на семинаре, конференции; подготовка рефератов, докладов; составление библиографии, тематических кроссвордов; тестирование и др.; – для формирования умений: решение задач и упражнений по образцу; решение вариативных задач и упражнений; подготовка и проектирование, а также моделирование разных видов и компонентов профессиональной деятельности, выполнение практических работ; рефлексивный анализ профессиональных умений с использованием аудио- и видеотехники и др.		сообщений и др.

Текущий контроль знаний проводится преподавателем, ведущим праетические занятия. Текущий контроль проводится путем индивидуального опроса студентов по результатам освоения тем, вынесенных на практические занятия (по материалам, изложенным в лекционном курсе).

Методические рекомендации к самостоятельной работе студентов.

Для лабораторной работы №1.

Изучая C#, разработчик сталкивается с классами и объектами. Например, вот как выглядит первая программа любого новичка:

```
using System;

namespace OOPConsole
{
	class Program
	{
		static void Main(string[] args)
		{
			Console.WriteLine("Hello World!");
		}
```

```
        }
}
```

Здесь создается класс Program, у которого есть метод Main () — с него начинается выполнение программы, поэтому его называют точкой входа.

Для вывода текста используется следующий оператор:
Console.WriteLine("Hello, World!");

Тут программа обращается к объекту Console и вызывает метод WriteLine (), который выводит переданное значение в консоль.
Также у объекта Console есть разные свойства:

Console.ForegroundColor = ConsoleColor.Red; //Цвет шрифта — красный
Console.BackgroundColor = ConsoleColor.Blue; //Цвет фона — синий
Если бы не было объекта, было бы сложно определить, цвет какого фона и какого шрифта будет указываться, потому что их в программе может быть несколько.
Приведём пример простого класса Character:

```
namespace OOPConsole
{
        class Character
        {
        }
}
```

Всё, что находится внутри фигурных скобок, относится к этому классу. Несмотря на то что он пустой, уже можно создать его экземпляр — объект. Это называется объявлением или инстанцированием.

Чтобы объявить объект, нужно перейти к методу Main и использовать следующий код:

Character hero = new Character();
Это похоже на то, как создаются переменные, но вместо типа данных указывается название класса. После знака присваивания указываются ключевое слово new и конструктор — специальный метод, который позволяет создать объект (о нем читайте в блоке о методах).

Как добавить поля и свойства класса
Теперь можно добавить поля и свойства для класса. Поле объявляется как обычная переменная:

```
class Character
{
        int health = 100;
        int x = 50;
        int y = 25;
}
```

Теперь у объекта есть свои поля, но к ним нельзя обратиться извне, потому что закрыт доступ (подробнее об этом — в статье про инкапсуляцию). Чтобы его открыть, нужно поставить перед каждым полем ключевое слово public. То же слово нужно поставить перед словом class.

```
public class Character
{
	public int health = 100;
	public int x = 50;
	public int y = 25;
}
```

Теперь можно вернуться к созданному объекту и обратиться к его полю здоровья:

```
Console.WriteLine(hero.health);
```

Если доступ к полям открыт, то с ними можно проводить вычисления или просто получать их значение. Если же нужно запретить доступ к определенным полям — используйте свойства.

Это специальные конструкции, которые позволяют обращаться к полям. Чтобы создать свойства, нужно сначала закрыть доступ к полям с помощью уровня доступа private — тогда они будут доступны только изнутри класса:

```
public class Character
{
	private int health = 100;
	private int x = 50;
	private int y = 25;
}
```

Теперь нельзя узнать здоровье или координаты объекта или изменить их. Чтобы снова открыть к ним доступ, создайте свойства:

```
public int Health
{
	get
	{
		return this.health;
	}

	set
	{
		this.health = value;
	}
}

public int X {
	get
	{
		return this.x;
	}
}
```

Тут указывается уровень доступа public, затем тип свойства и его имя. Имена свойств принято начинать с заглавной буквы, и они должны соответствовать имени поля. Внутри свойства встречаются две конструкции:

get (геттер — позволяет получить значение свойства). В ней возвращается значение, которое идет после ключевого слова return;
set (сеттер — позволяет изменить значение). В ней указывается поле, которое нужно изменить, используя значение value.
Также тут можно заметить ключевое слово this, которое обозначает, что поле принадлежит этому объекту. Использовать его необязательно, но оно делает код более читаемым.

Такие манипуляции нужны для того, чтобы доступ к полям осуществлялся только так, как это нужно разработчику. Например, можно создать свойство, значение которого будет зависеть от выполнения условия:

```
public bool IsAlive
{
	get
	{
		return this.health > 0 ? true : false;
	}
}
```

Значение, которое будет возвращено, зависит от здоровья персонажа. Если оно ниже нуля, то будет передано false, а если выше — true.
Теперь можно приступить к работе с поведением объектов. Оно реализуется с помощью методов — специальных блоков кода, которые позволяют избежать повторений в проекте.

Методы являются аналогами функций (возвращают значение) и процедур (не возвращают), но с той разницей, что они являются частью какого-то класса. Например, можно в классе Character создать метод Move (), который будет отвечать за движение персонажа.

```
public void Move(string direction)
{
	switch (direction)
	{
		case "forward":
			this.x++;
			break;
		case "backward":
			this.x--;
			break;
		case "up":
			this.y++;
			break;
		case "down":
			this.y--;
			break;
	}
}
```

Сначала указывается уровень доступа public, затем тип возвращаемого значения (в данном случае используется void, что говорит компилятору о том, что ничего возвращать не нужно). Затем идет название метода и круглые скобки.

Внутри скобок указываются аргументы, которые принимает метод (в данном случае направление движения), — от переданных аргументов зависит результат работы метода.

Вот пример вызова метода:

```
Character hero = new Character();

string command = string.Empty;

while (command != "exit")
{
	Console.WriteLine($"You are at {hero.Coordinates}. Where to go?");
	command = Console.ReadLine();

	hero.Move(command);
}
```

Пользователь видит свои координаты и вводит направление движения, а персонаж двигается с помощью метода Move (). Всё это повторяется, пока не будет введена команда exit.
Если же нужно, чтобы метод что-то возвращал, то указывается его тип и используется оператор return:

```
public bool Collide(Character ch)
{
	if (ch.X == this.x && ch.Y == this.y)
	{
		return true;
	}
	else
	{
		return false;
	}
}
```

Этот метод принимает в качестве аргумента объект класса Character и сравнивает координаты. Если они равны, то метод возвращает значение true, а иначе — false.

Вот как это можно использовать:

```
Character hero = new Character();

Character npc = new Character();

bool collide = hero.Collide(npc);

if (collide)
{
	Console.WriteLine("Objects are on the same position.");
}
else
```

```
{
        Console.WriteLine("Objects are not on the same position.");
}

Console.ReadKey();
```

Для лабораторной работы №2

Создание класса

Для примера возьмём класс «Автомобиль». Что имеет автомобиль? В частности, это:

- марка;
- цвет;
- мощность (в л/с);
- максимальная скорость (км/ч);
- объём бака (л);
- расход топлива (л) на 100 км пути.

Напишем класс Car (автомобиль) на C# (аналогично на Java):

```
 public class Car
{
   private string brand;
   private string color;
   private int power;
   private int maxSpeed;
   private int volumeOfTank;
   private double fuelConsumption;
}
```

Как вы могли заменить класс объявляется так: модификатор доступа, ключевое слово class и имя класса. Тело класса определяется фигурными скобками. Внутри класса объявлены его поля.

Следует понимать, что класс – это каркас, иначе говоря, описание реального объекта. На основе этого “описания” создаются экземпляры реального объекта. Логично предположить, что необходим механизм для присваивания значениям полей характеристик объекта. Для этого существуют конструкторы класса.

Конструктор класса

Конструктор класса – это специальный метод, который вызывается при создании нового объекта и используется для инициализации полей класса значениями, а также для начальных вычислений, если они необходимы. После создания объекта конструктор вызвать нельзя. Кроме того, данный метод никогда не возвращает никакого значения.

Напишем конструктор для инициализации полей в классе Car:

```
public class Car
{
   private string brand;
   private string color;
   private int power;
   private int maxSpeed;
   private int volumeOfTank;
   private double fuelConsumption;

   //конструктор класса
   public Car(string newBrand, string newColor, int newPower, int newMaxSpeed,
      int newVolumeOfTank, double newFuelConsumption)
```

```
    {
        brand = newBrand;
        color = newColor;
        power = newPower;
        maxSpeed = newMaxSpeed;
        volumeOfTank = newVolumeOfTank;
        fuelConsumption = newFuelConsumption;
    }
}
```

Конструктор объявляется так: public Имя ([параметры]). Наличие параметров не обязательно. Соответственно выделяют конструкторы класса:

- без параметров
- с параметрами

Модификатор доступа обязательно public, поскольку конструктор всегда вызывается вне класса.

Класс может содержать несколько конструкторов с разными параметрами. При создании объекта будет вызван тот, который подходит по параметрам.

Создание экземпляра класса

Создадим экземпляр (объект) класса Car на примере автомобиля Форд. Для этого понадобится оператор new.

- Ford (марка)
- Серый (цвет)
- 150 (мощность)
- 210 (максимальная скорость)
- 55 (объём бака)
- 6.4 (расход топлива в смешанном цикле)

```
Car c = new Car("Ford", "Серый", 150, 210, 55, 6.4);
```

Сам по себе класс является ссылочным типом данных.

Если бы наш класс не содержал конструктора с параметрами, то создание нового объекта происходило бы так:

```
Car c = new Car();
```

Методы класса

В ООП методы класса схожи по назначению с функциями из процедурного программирования.

Общая схема объявления метода: [модификаторДоступа] типВозвращаемогоЗначения имяМетода ([аргументы]) { }.

- Тип возвращаемого значения может быть void (ничего не возвращает);
- Наличие аргументов необязательно.

Создадим метод, который рассчитает на сколько километров пути хватит полного бака бензина:

```
public double QuantityOfKilometers()
{
    double quantity = 100 * volumeOfTank / fuelConsumption;

    return quantity;
}
```

А также создадим метод с тем же именем, но с параметром – количеством топлива;

рассчитаем на сколько километров пути его хватит:

```
public double QuantityOfKilometers(double volume)
{
   double quantity = 100 * volume / fuelConsumption;

   return quantity;
}
```

Методы с одинаковым именем создавать можно, но они должны иметь разные аргументы (по количеству или типу данных; могут отсутствовать вовсе). Компилятор выберет тот, который подходит при вызове метода по аргументам. Методы с идентичными именами, но разными параметрами, иллюстрируют полиморфизм в ООП.

Вызовем данные методы для нашего Форда и выведем результат выполнения в консоль.

```
//C#
Console.WriteLine(c.QuantityOfKilometers());
Console.WriteLine(c.QuantityOfKilometers(10));

//Java
System.out.println(c.QuantityOfKilometers());
System.out.println(c.QuantityOfKilometers(10));
```

Результат вычислений:

Приведём общую схему вызова метода для экземпляра класса (используется оператор “точка”): имяЭкземпляраКласса.ИмяМетода([аргументы]);

```
c.QuantityOfKilometers();
c.QuantityOfKilometers(10);
```

Итак, подводя итоги, отметим, что базовая структура класса в ООП, это (в любом порядке):

- поля (данные);
- конструктор(ы);
- методы.

Для лабораторной работы №3

C# — строго типизированный язык. Это значит, что вы не можете поместить строку в переменную типа int — сначала нужно провести преобразование. Так же и в метод нельзя передать параметр типа float, если при объявлении метода был указан тип double.

Однако если вы экспериментировали с методом WriteLine() класса Console, то могли заметить, что в него можно передавать аргументы разных типов:

```
Console.WriteLine("Hello, World!"); //string

Console.WriteLine(50f); //float

Console.WriteLine(0.23); //double

Console.WriteLine(42); //int
```

Console.WriteLine("The numbers are {0} and {1}", 17, 3); //Строка с интерполяцией

Кажется, что нарушена типизация, но компилятор не выдаёт ошибку.

Так происходит потому, что у метода WriteLine() есть перегрузки — методы с таким же названием, но принимающие другие аргументы:

```
int Sum(int a, int b)
{
   return a + b; //Сумма двух целых чисел
}

double Sum(double a, double b)
{
   return a + b; //Сумма двух чисел с плавающей запятой (double)
}

float Sum(float a, float b, float c)
{
   return a + b + c; //Сумма трёх чисел с плавающей запятой (float)
}

int Sum(int[] array)
{
   int sum = 0;

   foreach (int num in array)
   {
      sum += num;
   }

   return sum; //Сумма всех целых чисел в массиве
}
```

Когда вы вызовете метод Sum(), компилятор по переданным аргументам узнает, какую из его перегрузок вы имели в виду — так же, как это происходит с методом WriteLine().

При этом стоит учитывать, что значение имеют только типы и количество передаваемых аргументов. Например, можно написать такие перегрузки:

```
int Multiply(int a, int b)
{
   return a * b;
}

void Multiply(int a, int b)
{
   Console.WriteLine(a * b);
}
```

У этих методов одинаковые параметры, но разный возвращаемый тип. Попытка скомпилировать такой код приведёт к ошибке — так же, как и создание перегрузки с такими же аргументами, но с другими названиями:

```
int Multiply(int a, int b)
{
   return a * b;
}

int Multiply(int num1, int num2)
{
   return num1 * num2;
}
```

То же самое можно сделать и с конструкторами классов:

```
class Item
{
   public string name;
   public int price;
   public int lvl;

   public Item()
   {
      this.name = "Item";
      this.price = 15;
      this.lvl = 1;
   }

   public Item(string name)
   {
      this.name = name;
      this.price = 15;
      this.lvl = 1;
   }

   public Item(string name, int price, int lvl)
   {
      this.name = name;
      this.price = price;
      this.lvl = lvl;
   }
}
```

Альтернатива этому решению — указать значения для аргументов по умолчанию:

```
public Item(string name = "Item", int price = 15, int lvl = 1)
{
   this.name = name;
   this.price = price;
   this.lvl = lvl;
}
```

Несмотря на, то что здесь меньше кода, на мой взгляд, это может запутать. Потому что придётся каждый раз заполнять все значения, даже если нужен только один аргумент из

конца списка. Перегрузка же позволяет определить и порядок параметров (если они разных типов).

Перегрузить можно даже операторы, то есть:

- +,
- ++,
- -,
- --,
- *,
- /,
- ==,
- >,
- <,
- >=,
- <=.

Для этого в определении класса нужно добавить вот такую конструкцию:

```
public static возвращаемый_тип operator оператор(аргументы)
{
   //Логика
}
```

Так как использоваться этот оператор должен без объявления экземпляра класса (item1 + item2, а не item1 item1.+ item2), то указываются модификаторы public static.

Например, мы хотим улучшать предметы в играх. Во многих MMO1 популярна механика, когда один предмет улучшается за счёт другого. Мы можем сделать это с помощью перегрузки оператора сложения:

```
public static Item operator +(Item i1, Item i2)
{
   return new Item(i1.name + "+", i1.price + i2.price / 2, ++i1.lvl);
}
```

Теперь при сложении двух объектов класса Item мы будем получать третий объект с улучшенными параметрами. Вот пример использования такого оператора:

```
//Создаём три предмета
Item i1 = new Item("Sword", 15, 1);
Item i2 = new Item();
Item i3 = new Item();

Console.WriteLine($"{i1.name} | Price: {i1.price} | Level: {i1.lvl}"); //Выводим параметры первого предмета

i1 += i2; //Улучшаем предмет с помощью другого предмета

Console.WriteLine($"{i1.name} | Price: {i1.price} | Level: {i1.lvl}"); //Снова выводим данные

i1 += i3; //Повторяем улучшение
```

Console.WriteLine($"{i1.name} | Price: {i1.price} | Level: {i1.lvl}"); //Выводим новые характеристики

Хотя типизация в C# строгая, типы можно преобразовывать. Например, мы можем конвертировать число типа float в число типа int:

```
float x = 50f;
int y = (int)y;
```

С помощью перегрузки операторов преобразования типов мы можем прописать любую логику для конвертации объектов. Для наглядности создадим класс Hero:

```
class Hero
{
   public string name = "";
   public int str = 1;
   public int dex = 1;
   public int intel = 1;
   public int lvl = 1;
}
```

В этом классе хранятся данные о персонаже. В MMO часто можно увидеть такой параметр, как мощь — это сумма всех характеристик героя или предмета. Например, её можно посчитать по следующей формуле:

Мощь = (сила + ловкость + интеллект) * уровень.

Мы можем использовать преобразование типов, чтобы автоматически переводить объект в его мощь. Для этого нужно использовать такую конструкцию.

```
public static implicit|explicit operator новый_тип(исходный_тип arg)
{
   // логика
}
```

Модификатор implicit говорит компилятору, что преобразование может быть неявным. То есть оно сработает, если написать так:

```
Hero h1 = new Hero();
int power = h1;
```

Explicit, наоборот, означает, что преобразование должно быть явным:

```
Hero h1 = new Hero();
int power = (int)h1;
```

Вот как будет выглядеть перегрузка преобразования объекта класса Hero в int:

```
public static explicit operator int(Hero hero)
{
   return (hero.str + hero.dex + hero.intel) * hero.lvl;
}
```

Вот как она будет использоваться:

```
Hero h = new Hero //Создаём героя
{
```

```
    name = "Super Dragon Slayer 2000",
    str = 10,
    dex = 7,
    intel = 25,
    lvl = 5
};

int power = (int)h; //Конвертируем его в int

Console.WriteLine($"{h.name}'s power: {power}"); //Выводим результат
```

11. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

а) Список рекомендуемой литературы

основная

1. Тузовский, А. Ф. Объектно-ориентированное программирование: учебное пособие для прикладного бакалавриата / А. Ф. Тузовский. — Москва : Издательство Юрайт, 2018. — 206 с. — (Университеты России). — ISBN 978-5-534-00849-4. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://urait.ru/bcode/414163

2. Рацеев С. М. Программирование на языке СИ : учеб. пособие / С. М. Рацеев; УлГУ, ФМиИТ. -Ульяновск : УлГУ, 2015. - Загл. с экрана; Имеется печ. аналог. - Электрон. текстовые дан. (1 файл : 1,74 КБ). - Текст : электронный. http://lib.ulsu.ru/MegaPro/Download/MObject/325

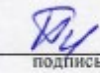
дополнительная

1. Ашарина И.В., Объектно-ориентированное программирование в С++: лекции и упражнения : Учебное пособие для вузов / Ашарина И.В. - М. : Горячая линия - Телеком, 2017. - 336 с. - ISBN 978-5-9912-0423-1 - Текст : электронный // ЭБС "Консультант студента": [сайт]. - URL : http://www.studentlibrary.ru/book/ISBN9785991204231.html

учебно-методическая

1. Жаркова Галина Алексеевна. Программная реализация конечных автоматов: учеб.-метод. пособие / Жаркова Галина Алексеевна, А. В. Жарков; УлГУ, Фак. матем. и информ. технологий, Каф. информ. технологий. - Ульяновск : УлГУ, 2011. - Имеется печ. аналог. - Электрон. текстовые дан. (1 файл : 350 Кб). - Текст : электронный. http://lib.ulsu.ru/MegaPro/Download/MObject/653
2. Жаркова Г. А. Методические указания для самостоятельной работы студентов по дисциплине «Объектно-ориентированное программирование» для студентов бакалавриата по направлениям подготовки 09.03.03 «Прикладная информатика», направленность (профиль/специализация) Информационная сфера и 02.03.03 «Математическое обеспечение и администрирование информационных систем», направленность (профиль/специализация) Технология программирования очной формы обучения / Г. А. Жаркова; УлГУ, ФМИиАТ. - Ульяновск: УлГУ, 2019. - Загл. с экрана; Неопубликованный ресурс. - Электрон. текстовые дан. (1 файл : 315 КБ). - Текст : электронный. http://lib.ulsu.ru/MegaPro/Download/MObject/7244

Согласовано:

Гл. библиотекарь НБ УлГУ / Полина Н.Ю. /  / 14.06.2019

Должность сотрудника научной библиотеки — ФИО — подпись — дата

б) Программное обеспечение:

Для проведения занятий требуются мультимедийные средства: компьютер с пакетом программ Open Office, Web браузер и проектор.

Для проведения лабораторных работ по курсу «Объектно-ориентированное программирование» требуется компьютерный класс, подключенный к ЛВС УлГУ с выходом в Интернет и с установленным ПО – MS Visual Studio.

в) Профессиональные базы данных, информационно-справочные системы

1.1. **IPRbooks** [Электронный ресурс]: электронно-библиотечная система / группа компаний Ай Пи Эр Медиа . - Электрон. дан. - Саратов , [2019]. - Режим доступа: http://www.iprbookshop.ru.

1.2. **ЮРАЙТ** [Электронный ресурс]: электронно-библиотечная система / ООО Электронное издательство ЮРАЙТ. - Электрон. дан. – Москва , [2019]. - Режим доступа: https://www.biblio-online.ru.

1.3. **Консультант студента** [Электронный ресурс]: электронно-библиотечная система / ООО Политехресурс. - Электрон. дан. – Москва, [2019]. - Режим доступа: http://www.studentlibrary.ru/pages/catalogue.html.

1.4. **Лань** [Электронный ресурс]: электронно-библиотечная система / ООО ЭБС Лань. - Электрон. дан. – С.-Петербург, [2019]. - Режим доступа: https://e.lanbook.com.

1.5. **Znanium.com** [Электронный ресурс]: электронно-библиотечная система / ООО Знаниум. - Электрон. дан. – Москва, [2019]. - Режим доступа: http://znanium.com.

2. **КонсультантПлюс** [Электронный ресурс]: справочная правовая система. /Компания «Консультант Плюс» - Электрон. дан. - Москва : КонсультантПлюс, [2019].

3. **База данных периодических изданий** [Электронный ресурс] : электронные журналы / ООО ИВИС. - Электрон. дан. - Москва, [2019]. - Режим доступа: https://dlib.eastview.com/browse/udb/12.

4. **Национальная электронная библиотека** [Электронный ресурс]: электронная библиотека. - Электрон. дан. – Москва, [2019]. - Режим доступа: https://нэб.рф.

5. **Электронная библиотека диссертаций РГБ** [Электронный ресурс]: электронная библиотека / ФГБУ РГБ. - Электрон. дан. – Москва, [2019]. - Режим доступа: https://dvs.rsl.ru.

6. Федеральные информационно-образовательные порталы:

6.1. Информационная система Единое окно доступа к образовательным ресурсам. Режим доступа: http://window.edu.ru

6.2. Федеральный портал Российское образование. Режим доступа: http://www.edu.ru

7. Образовательные ресурсы УлГУ:

7.1. Электронная библиотека УлГУ. Режим доступа : http://lib.ulsu.ru/MegaPro/Web

7.2. Образовательный портал УлГУ. Режим доступа : http://edu.ulsu.ru

Согласовано:

Зам. нач. УИиТ	/Клочкова А.В.	/	14.06.2019
Должность сотрудника УИиТ	ФИО	подпись	дата

12. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ИЛИ ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Аудитории для проведения лекций и лабораторных занятий, для проведения текущего контроля и промежуточной аттестации, групповых и индивидуальных консультаций.

Аудитории укомплектованы специализированной мебелью, учебной доской. Аудитории для проведения лекций оборудованы мультимедийным оборудованием для предоставления информации большой аудитории. Помещения для самостоятельной работы оснащены компьютерной техникой с возможностью подключения к сети «Интернет» и обеспечением доступа к электронной инфромационно-образовательной среде, электронно-библиотечной системе. Перечень оборудования, используемого в учебном процессе, указывается в соответствии со сведениями о материально-техническом обеспечении и оснащенности образовательного процесса, размещенными на официальном сайте УлГУ в разделе «Сведения об образовательной организации».

13. СПЕЦИАЛЬНЫЕ УСЛОВИЯ ДЛЯ ОБУЧАЮЩИХСЯ С ОГРАНИЧЕННЫМИ ВОЗМОЖНОСТЯМИ ЗДОРОВЬЯ

В случае необходимости, обучающимся из числа лиц с ограниченными возможностями здоровья (по заявлению обучающегося) могут предлагаться одни из следующих вариантов восприятия информации с учетом их индивидуальных психофизических особенностей:

– для лиц с нарушениями зрения: в печатной форме увеличенным шрифтом; в форме электронного документа; в форме аудиофайла (перевод учебных материалов в аудиоформат); в печатной форме на языке Брайля; индивидуальные консультации с привлечением тифлосурдопереводчика; индивидуальные задания и консультации;

– для лиц с нарушениями слуха: в печатной форме; в форме электронного документа; видеоматериалы с субтитрами; индивидуальные консультации с привлечением сурдопереводчика; индивидуальные задания и консультации;

– для лиц с нарушениями опорно-двигательного аппарата: в печатной форме; в форме электронного документа; в форме аудиофайла; индивидуальные задания и консультации.

В случае необходимости использования в учебном процессе частично/исключительно дистанционных образовательных технологий, организация работы ППС с обучающимися с ОВЗ и инвалидами предусматривается в электронной информационно-образовательной среде с учетом их индивидуальных психофизических особенностей.

Разработчик _____ доцент Головин В.А.

подпись должность ФИО

ЛИСТ ИЗМЕНЕНИЙ

№ п/п	Содержание изменения или ссылка на прилагаемый текст изменения	ФИО заведующего кафедрой, реализующей дисциплину/ выпускающей кафедрой	Подпись	Дата
1.	Внесение изменений в пп. в) Профессиональные базы данных, информационно-справочные системы п. 11 «Учебно-методическое и информационное обеспечение дисциплины» с оформлением приложения 2	Волков М.А./ Андреев А.С.		31.08.2020
2.	Внесение изменений в п. 13 «Специальные условия для обучающихся с ограниченными возможностями здоровья» с оформлением приложения 3	Волков М.А./ Андреев А.С.		31.08.2020

1.1. **IPRbooks** [Электронный ресурс]: электронно-библиотечная система / группа компаний Ай Пи Эр Медиа . - Электрон. дан. - Саратов , [2019]. - Режим доступа: http://www.iprbookshop.ru.

1.2. **ЮРАЙТ** [Электронный ресурс]: электронно-библиотечная система / ООО Электронное издательство ЮРАЙТ. - Электрон. дан. – Москва , [2019]. - Режим доступа: https://www.biblio-online.ru.

1.3. **Консультант студента** [Электронный ресурс]: электронно-библиотечная система / ООО Политехресурс. - Электрон. дан. – Москва, [2019]. - Режим доступа: http://www.studentlibrary.ru/pages/catalogue.html.

1.4. **Лань** [Электронный ресурс]: электронно-библиотечная система / ООО ЭБС Лань. - Электрон. дан. – С.-Петербург, [2019]. - Режим доступа: https://e.lanbook.com.

1.5. **Znanium.com** [Электронный ресурс]: электронно-библиотечная система / ООО Знаниум. - Электрон. дан. – Москва, [2019]. - Режим доступа: http://znanium.com.

2. **КонсультантПлюс** [Электронный ресурс]: справочная правовая система. /Компания «Консультант Плюс» - Электрон. дан. - Москва : КонсультантПлюс, [2019].

3. **База данных периодических изданий** [Электронный ресурс] : электронные журналы / ООО ИВИС. - Электрон. дан. - Москва, [2019]. - Режим доступа: https://dlib.eastview.com/browse/udb/12.

4. **Национальная электронная библиотека** [Электронный ресурс]: электронная библиотека. - Электрон. дан. – Москва, [2019]. - Режим доступа: https://нэб.рф.

5. **Электронная библиотека диссертаций РГБ** [Электронный ресурс]: электронная библиотека / ФГБУ РГБ. - Электрон. дан. – Москва, [2019]. - Режим доступа: https://dvs.rsl.ru.

6. Федеральные информационно-образовательные порталы:

6.1. Информационная система Единое окно доступа к образовательным ресурсам. Режим доступа: http://window.edu.ru

6.2. Федеральный портал Российское образование. Режим доступа: http://www.edu.ru

7. Образовательные ресурсы УлГУ:

7.1. Электронная библиотека УлГУ. Режим доступа : http://lib.ulsu.ru/MegaPro/Web

7.2. Образовательный портал УлГУ. Режим доступа : http://edu.ulsu.ru

Согласовано:

Зам. нач. УИиТ /Клочкова А.В. / / 14.06.2019

Должность сотрудника УИиТ ФИО подпись дата

Приложение 2

13. СПЕЦИАЛЬНЫЕ УСЛОВИЯ ДЛЯ ОБУЧАЮЩИХСЯ С ОГРАНИЧЕННЫМИ ВОЗМОЖНОСТЯМИ ЗДОРОВЬЯ

В случае необходимости, обучающимся из числа лиц с ограниченными возможностями здоровья (по заявлению обучающегося) могут предлагаться одни из следующих вариантов восприятия информации с учетом их индивидуальных психофизических особенностей:

– для лиц с нарушениями зрения: в печатной форме увеличенным шрифтом; в форме электронного документа; в форме аудиофайла (перевод учебных материалов в аудиоформат); в печатной форме на языке Брайля; индивидуальные консультации с привлечением тифлосурдопереводчика; индивидуальные задания и консультации;

– для лиц с нарушениями слуха: в печатной форме; в форме электронного документа; видеоматериалы с субтитрами; индивидуальные консультации с привлечением сурдопереводчика; индивидуальные задания и консультации;

– для лиц с нарушениями опорно-двигательного аппарата: в печатной форме; в форме электронного документа; в форме аудиофайла; индивидуальные задания и консультации.

В случае необходимости использования в учебном процессе частично/исключительно дистанционных образовательных технологий, организация работы ППС с обучающимися с ОВЗ и инвалидами предусматривается в электронной информационно-образовательной среде с учетом их индивидуальных психофизических особенностей.