

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский ядерный университет «МИФИ»

Саровский физико-технический институт -
филиал федерального государственного автономного образовательного учреждения высшего образования
«Национальный исследовательский ядерный университет «МИФИ»
(САРФТИ НИЯУ МИФИ)

УТВЕРЖДАЮ

Зам. руководителя СарФТИ НИЯУ
МИФИ, к.э.н., доцент

_____ Т.Г. Соловьев

«11» августа 2025 г.

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

Объектно-ориентированное программирование

Специальность: 09.02.07 Информационные системы и программирование

Наименование образовательной программы: Информационные системы и программирование

Уровень образования: среднее профессиональное образование

Форма обучения: очная

г. Саров, 2025 г.

Паспорт фонда оценочных средств

Специальность:

09.02.07 Информационные системы и программирование

Учебная дисциплина: Объектно-ориентированное программирование

Требования ФГОС СПО к результатам освоения дисциплины

В результате освоения учебной дисциплины обучающийся должен обладать общими компетенциями, включающими в себя способность:

ОК 01. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 02. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.

ОК 09. Использовать информационные технологии в профессиональной деятельности.

Учащийся должен обладать профессиональными компетенциями, соответствующими основным видам профессиональной деятельности:

ПК 1.1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

ПК 1.2. Разрабатывать программные модули в соответствии с техническим заданием.

ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

ПК 1.4. Выполнять тестирование программных модулей.

ПК 1.5. Осуществлять рефакторинг и оптимизацию программного кода.

ПК 1.6. Разрабатывать модули программного обеспечения для мобильных платформ.

В результате освоения учебной дисциплины обучающийся должен **уметь**:

- ориентироваться в наиболее общих философских проблемах бытия, познания, ценностей, свободы и смысла жизни, составляющих основу формирования культуры гражданина и будущего специалиста.

В результате освоения дисциплины обучающийся должен уметь:

- Проектировать информационные системы на языке C++
- Проектировать и создавать программные системы Java

В результате освоения учебной дисциплины обучающийся должен **знать**:

- основ технологии объектно-ориентированной декомпозиции программных систем, базовых шаблонов проектирования (Наблюдатель, Итератор, Одиночка, Фабрика, Заместитель), отношений между классами и основ UML (диаграммы классов и последовательностей).
- особенности построения объектно-ориентированных программных систем на C++.
- основные инструментальные средства языка C++ и стандартной библиотеки
- базовые знания платформы Java, особенности построения программных систем Java
- средства реализации принципов ООП и инструментальные средства языка Java.
- основы технологий построения простейших распределенных информационных систем и обеспечения безопасности.

1. Общие положения

Фонд оценочных средств (ФОС) предназначен для контроля и оценки образовательных достижений обучающихся, освоивших программу учебной дисциплины «Объектно-ориентированное программирование».

ФОС включает контрольные материалы для проведения текущего контроля и промежуточной аттестации.

ФОС разработан на основании положений: основной профессиональной образовательной программы, рабочей программы учебной дисциплины «Объектно-ориентированное программирование» по направлению подготовки специальностей СПО 09.02.07 Информационные системы и программирование.

2. Результаты освоения дисциплины, подлежащие проверке.

Раздел 1. Основные принципы объектно-ориентированного программирования

Раздел 2. Объектно-ориентированная модель

Раздел 3. Средства объектного программирования языка C++

3. Фонд оценочных средств

Вопросы для проведения устного контроля по основным темам

Конечно, вот список из 60 вопросов для устного тестирования по объектно-ориентированному программированию, охватывающих основные концепции и их практическое применение.

Раздел 1. Основные принципы объектно-ориентированного программирования

1.1. Основы ООП

1. Что такое объектно-ориентированное программирование (ООП) и каковы его основные преимущества?
2. Назовите и кратко охарактеризуйте четыре основных принципа ООП.
3. Что такое парадигма программирования? Какие парадигмы вы знаете, кроме ООП?
4. Что такое объект? Из каких основных характеристик он состоит?
5. Что такое класс? Какова его роль в ООП?
6. Объясните разницу между классом и объектом (экземпляром класса).
7. Что такое инкапсуляция и какую проблему она решает?
8. Что такое наследование и как оно способствует повторному использованию кода?
9. Что такое полиморфизм? Приведите пример из реальной жизни.
10. Что такое абстракция? Как она помогает управлять сложностью программ?

1.2. Классы и объекты (базовый уровень)

11. Какой метод называется конструктором? Каково его назначение?
12. Что такое конструктор по умолчанию? Всегда ли он создается компилятором?
13. Что такое деструктор? Когда он вызывается?
14. Что такое `this` (или `self` в некоторых языках) указатель/ссылка?
15. Что такое поле (атрибут) класса? Что такое метод класса?
16. Что такое статическое поле (static) и чем оно отличается от обычного?
17. Что такое статический метод? Имеет ли он доступ к нестатическим членам класса?
18. Что такое перегрузка методов (method overloading)?
19. Что такое переопределение метода (method overriding)?
20. Объясните разницу между перегрузкой (overloading) и переопределением (overriding).

Раздел 2. Объектно-ориентированная модель

2.1. Инкапсуляция и управление доступом

- 21. Какие модификаторы доступа вы знаете? (`'public'`, `'private'`, `'protected'` и т.д.)
- 22. Опишите уровень доступа для каждого модификатора (что и откуда видно).
- 23. Что такое "геттер" (getter) и "сеттер" (setter)? Для чего они используются?
- 24. Что такое свойство (property) и как оно связано с принципом инкапсуляции?
- 25. Что такое "дружественная" функция или класс (friend)? Является ли это нарушением инкапсуляции?

2.2. Наследование

- 26. Что такое базовый (родительский) и производный (дочерний) класс?
- 27. Каков порядок вызова конструкторов и деструкторов при создании и уничтожении объекта дочернего класса?
- 28. Что такое простое и множественное наследование? Какие в них проблемы?
- 29. Что такое виртуальное наследование (в C++)? Для чего оно нужно?
- 30. Что такое `'super()'` (или `'base()'`) и для чего используется?
- 31. Как предотвратить наследование от класса (на примере любого языка)?
- 32. Как запретить переопределение метода в дочерних классах? (Ключевое слово `'final'` или `'sealed'`).

2.3. Полиморфизм

- 33. Что такое виртуальный метод? Как он реализован "под капотом" (на примере vtable в C++)?
- 34. Что такое позднее (динамическое) связывание и как оно связано с полиморфизмом?
- 35. Что такое раннее (статическое) связывание?
- 36. Что такое абстрактный класс? Можно ли создать его экземпляр?
- 37. Что такое абстрактный метод? Чем он отличается от виртуального?
- 38. Что такое интерфейс? Чем он отличается от абстрактного класса?
- 39. Можно ли реализовать множественное наследование с помощью интерфейсов?
- 40. Что такое приведение типов? Какие виды приведения вы знаете (upcast, downcast)?

2.4. Дополнительные концепции и паттерны

41. Что такое класс-контейнер (коллекция)? Приведите примеры.
42. Что такое итератор и для чего он нужен?
43. Что такое перегрузка операторов? Какие ограничения на перегрузку существуют?
44. Что такое исключение (exception)? Как работает механизм обработки исключений (``try`/`catch`/`finally``)?
45. Что такое пространство имен (namespace) или пакет (package)? Какую проблему они решают?
46. Что такое обобщенное программирование (Generics / Шаблоны)? Какую проблему оно решает?
47. Что такое рефлексия (reflection)?
48. Что такое "поверхностное" (shallow) и "глубокое" (deep) копирование объектов?
49. Что такое композиция и агрегация? Чем они отличаются от наследования?
50. Объясните разницу между "is-a" (является) и "has-a" (имеет) отношениями.

Раздел 3. Средства объектного программирования языка

3.1. Вопросы на понимание и анализ

51. Когда следует использовать наследование, а когда — композицию?
52. Опишите проблему ромбовидного наследования и способы её решения.
53. В чем разница между интерфейсом и абстрактным классом? Когда что использовать?
54. Что такое "хрупкий базовый класс" (fragile base class) проблема?
55. Что такое Law of Demeter (Закон Деметры) или принцип "наименьшего знания"?
56. Опишите основные различия между ООП в C++, Java и C# (на примере ключевых концепций).
57. Что такое SOLID? Кратко раскройте смысл каждого принципа.
58. Что такое инверсия управления (IoC) и внедрение зависимостей (DI)? Как они связаны с ООП?
59. Что такое статический и динамический тип объекта?
60. Что такое сокрытие метода (method hiding) и чем оно отличается от переопределения?

1. Основы ООП (Вопросы 1-10)

Раскрываемые знания:

- **Знание** фундаментальных принципов, лежащих в основе объектно-ориентированного подхода.

- **Понимание** ключевых различий между ООП и другими парадигмами (процедурной, функциональной).

- **Понимание** базовых концепций: "класс" как шаблон и "объект" как конкретная сущность.

Формируемые умения:

- **Умение** объяснить преимущества ООП (сокрытие сложности, повторное использование кода, удобство поддержки).

- **Умение** на конкретных примерах проиллюстрировать каждый из четырех столпов ООП (Инкапсуляция, Наследование, Полиморфизм, Абстракция).

- **Умение** идентифицировать объекты и их свойства (состояние, поведение) в предметной области.

2. Классы и объекты (базовый уровень) (Вопросы 11-20)

Раскрываемые знания:

- **Знание** жизненного цикла объекта: создание (конструкторы), использование, уничтожение (деструкторы).

- **Понимание** различий между статическими (принадлежащими классу) и нестатическими (принадлежащими объекту) членами.

- **Знание** синтаксиса и семантики перегрузки и переопределения методов.

Формируемые умения:

- **Умение** правильно проектировать конструкторы (включая конструктор по умолчанию, копирования).

- **Умение** применять ключевое слово `this` для разрешения неоднозначностей.

- **Умение** выбирать между статическим и нестатическим членом в зависимости от решаемой задачи.

- **Умение** четко разграничивать концепции перегрузки (одно имя, разные параметры) и переопределения (изменение логики унаследованного метода).

3. Инкапсуляция и управление доступом (Вопросы 21-25)

Раскрываемые знания:

- **Глубокое понимание** принципа инкапсуляции как механизма защиты внутреннего состояния объекта.

- **Знание** модификаторов доступа и их семантики (private, protected, public).

- **Понимание** механизмов для контролируемого доступа к данным: аксессоры (геттеры/сеттеры) и свойства.

Формируемые умения:

- **Умение** правильно применять модификаторы доступа для сокрытия внутренней реализации.

- **Умение** проектировать классы, предоставляющие строго определённый публичный интерфейс.

- **Умение** реализовывать проверку корректности данных в сеттерах и свойствах для обеспечения целостности объекта.

- **Умение** аргументировать, почему прямое обращение к полям класса часто является плохой практикой.

4. Наследование (Вопросы 26-32)

Раскрываемые знания:

- **Знание** принципа наследования для создания иерархий классов "is-a" (является).

- **Понимание** механизма работы конструкторов и деструкторов в иерархии.

- **Знание** проблем множественного наследования и способов их обхода (например, через интерфейсы).

- **Понимание** ключевых слов для контроля наследования и переопределения (final, sealed).

Формируемые умения:

- **Умение** правильно строить иерархии классов, следуя отношениям "is-a".

- **Умение** предсказывать и объяснять порядок вызова конструкторов и деструкторов.

- **Умение** использовать super/base для явного вызова конструктора родительского класса.

- **Умение** применять механизмы запрета наследования или переопределения для создания стабильного и безопасного API.

5. Полиморфизм (Вопросы 33-40)

Раскрываемые знания:

- **Глубокое понимание** полиморфизма как возможности работать с объектами разных классов через единый интерфейс.

- **Знание** механизма виртуальных методов и динамического связывания (позднего связывания).

- **Понимание** различий и сходств между абстрактными классами и интерфейсами.

- **Знание** правил приведения типов (upcasting - безопасно, downcasting - опасно).

Формируемые умения:

- **Умение** проектировать полиморфные иерархии классов с использованием виртуальных и абстрактных методов.

- **Умение** правильно выбирать между абстрактным классом (общая реализация) и интерфейсом (контракт).

- **Умение** применять приведение типов, осознавая связанные с этим риски, и использовать безопасные формы приведения (is, as в C#).

- **Умение** объяснять, как полиморфизм делает код расширяемым и избавляет от больших условных конструкций.

6. Дополнительные концепции и паттерны (Вопросы 41-50)

Раскрываемые знания:

- **Знание** продвинутых концепций ООП и смежных тем, углубляющих понимание.

- **Понимание** принципов организации кода (пространства имён) и обработки ошибок (исключения).

- **Знание** шаблонов проектирования и принципов (Композиция, Агрегация, SOLID), определяющих качество кода.

Формируемые умения:

- **Умение** использовать обобщенное программирование (Generics) для создания типобезопасных и повторно используемых компонентов.

- **Умение** грамотно обрабатывать исключительные ситуации, не нарушая работу программы.

- **Умение** выбирать композицию над наследованием там, где отношение "has-a" (имеет) уместнее, чем "is-a".

- **Умение** реализовывать глубокое копирование объектов, когда это необходимо.

7. Вопросы на понимание и анализ (Вопросы 51-60)

Раскрываемые знания:

- **Синтез** всех предыдущих знаний для анализа и проектирования сложных систем.

- **Понимание** не только "как", но и "почему" — причин, стоящих за теми или иными принципами ООП.

- **Знание** распространенных проблем проектирования (хрупкий базовый класс, нарушение Law of Demeter) и способов их избежать.

Формируемые умения:

- **Умение** проводить сравнительный анализ и принимать взвешенные архитектурные решения (наследование vs. композиция, интерфейс vs. абстрактный класс).

- **Умение** критически оценивать дизайн классов и находить в нём слабые места, нарушающие принципы SOLID.

- **Умение** анализировать предметную область и преобразовывать её требования в robust-ную (надежную, отказоустойчивую) и гибкую объектно-ориентированную модель.

- **Умение** аргументировать свои проектные решения, опираясь на фундаментальные принципы ООП.

Критерии оценки

1. Основы ООП

- **Отлично:** Четко определяет все 4 принципа, приводит точные примеры. Понимает разницу между классом и объектом.

- **Удовлетворительно:** Называет принципы, но объясняет их поверхностно. Путается в примерах.

- **Неудовлетворительно:** Не может назвать или объяснить ключевые принципы. Не видит разницы между классом и объектом.

2. Классы и объекты (базовый уровень)

- **Отлично:** Объясняет жизненный цикл объекта, разницу между static и не-static контекстом. Четко различает перегрузку и переопределение.

- **Удовлетворительно:** Понимает основы, но допускает ошибки в деталях (например, порядок вызова деструкторов).

- **Неудовлетворительно:** Не понимает назначение конструкторов. Путает перегрузку и переопределение.

3. Инкапсуляция и управление доступом

- **Отлично:** Объясняет инкапсуляцию как концепцию. Аргументирует использование геттеров/сеттеров. Правильно применяет модификаторы доступа.

- **Удовлетворительно:** Понимает модификаторы доступа, но не может объяснить их важность для проектирования.

- **Неудовлетворительно:** Не понимает, зачем ограничивать доступ к полям. Не знает разницы между public и private.

4. Наследование

- **Отлично:** Четко объясняет принцип "is-a", порядок вызова конструкторов. Понимает проблемы множественного наследования.

- **Удовлетворительно:** Понимает базовое наследование, но затрудняется с сложными иерархиями.

- **Неудовлетворительно:** Строит некорректные иерархии ("Круг" наследуется от "Точки"). Не знает порядок вызова конструкторов.

5. Полиморфизм

- **Отлично:** Объясняет механизм позднего связывания. Четко различает абстрактный класс и интерфейс, аргументирует их применение.

- **Удовлетворительно:** Понимает полиморфизм на примере, но не может объяснить его реализацию.

- **Неудовлетворительно:** Путает переопределение с перегрузкой. Не понимает, зачем нужны абстрактные методы.

6. Дополнительные концепции и паттерны

- **Отлично:** Понимает и применяет композицию, generics, исключения. Может объяснить разницу между глубоким и поверхностным копированием.

- **Удовлетворительно:** Знает концепции, но не всегда может применить их на практике.

- **Неудовлетворительно:** Не понимает практической пользы generics или исключений. Не видит разницы между наследованием и композицией.

7. Вопросы на понимание и анализ

- **Отлично:** Анализирует плюсы/минусы решений, применяет принципы SOLID. Аргументирует выбор композиции над наследованием.

- **Удовлетворительно:** Понимает вопросы, но дает упрощенные или шаблонные ответы без глубокого анализа.

- **Неудовлетворительно:** Не может провести анализ или сравнение. Не знаком с принципами проектирования.