# Модуль 3. ООП. Вопросы.

1. **Событийно-ориентированное программирование**

   a. **Библиотека SFML**
   Основной цикл программы. Метод `setFramerateLimit`. Использование классов `sf::Time` и `sf::Clock` и функции `sf::sleep` для задания максимального количества кадров в секунду. Двойная буферизация. Как двойная буферизация реализована в SFML? Антиалиасинг. Как включить антиалиасинг в SFML? Системы координат SFML: система координат пикселей, глобальная система координат, локальные системы координат). Методы `mapPixelToCoords` и `mapCoordsToPixel`.

   b. **Проверка на нажатия клавиш и кнопок**
   Проверка на нажатия клавиш клавиатуры и кнопок мыши. Функции `sf::Keyboard::isKeyPressed` и `sf::Mouse::isButtonPressed`. Чем использование этих функций отличается от использования соответствующих событий?

   c. **Событийно-ориентированное программирование в библиотеке SFML**
   Понятие событий. Событийно-ориентированное программирование. Очередь событий. Цикл обработки событий. Класс `sf::Event` и цикл обработки событий в SFML. События SFML: `Closed`, `Resized`, `KeyPressed`, `KeyReleased`, `MouseButtonPressed`, `MouseButtonReleased`, `MouseMoved`. Получение координат мыши в событиях, связанных с мышью. Автоповтор при нажатиях на клавиши клавиатуры. Repeat delay и Repeat rate.

   d. **Реализация простейших элементов графического интерфейса** Реализация класса перетаскиваемой таблички (`Draggable`). Реализация класса кнопки. Реализация класса слайдера.

2. **Наследование**

   a. **Основы наследования**
   Наследование в языке `C++`. Базовый и производный классы. Добавление новых полей и методов в производный класс.

   b. **Затенение**
   Затенение методов и полей в производном классе. Чем затенение методов отличается от перегрузки методов? Как получить доступ к затенённым именам из базового класса? Использование ключевого слова `using` для отмены затенения.

   c. **Наследование и конструкторы/деструкторы**
   Наследуются ли конструкторы по умолчанию? Порядок вызова конструкторов при создании объекта производного класса. Как сделать так, чтобы вызывалась необходимая перегрузка конструктора базового класса при создании объекта производного класса? Использование ключевого слова `using` для наследования конструкторов. Порядок вызова деструкторов при уничтожении объекта производного класса.

   d. **Модификация доступа**
   Защищённые члены класса. Модификатор доступа `protected`. Публичное, приватное и защищённое наследование. Имеют ли друзья базового класса доступ к приватным полям производного класса?

   e. **Приведение типов при наследовании**
   Приведение объектов базового класса к объектам производного класса и наоборот. Срезка объекта. Приведение указателей на объекты базовых классов к указателям на объекты производных классов и наоборот. Использование `static_cast` для приведения указателей. В каких случах это может привести к неопределённому поведению?

   f. **Строение объекта производного класса**
   Строение объекта производного класса. Размер объекта производного класса. Как получить объект базового класса, находящийся внутри объекта производного класса. Размер объектов класса, не содержащего полей. Empty base optimisation.

   g. **Обычное множественное наследование**
   Строение объекта производного класса при обычном (не виртуальном) множественном наследовании. Сдвиг указателей при множественном наследовании.

   h. **Виртуальное множественное наследование**
   Проблема ромбовидного наследования. Как в языке C++ решается проблема ромбовидного наследования? Ключевое слово `virtual`. Строение объекта производного класса при виртуальном множественном наследовании.

3. **Динамический полиморфизм**

   a. **Определение полиморфизма**
   Определение понятия *полиморфизм*. Статический и динамический полиморфизм. Примеры статического полиморфизма в C++. Какие возможности даёт динамический полиморфизм по сравнению со статическим?

   b. **Виртуальные функции**
   Виртуальные функции. Указатели на базовый класс, хранящие адрес объекта производного класса (`Base* pbase = &derived`). Методы какого класса будут вызываться, если мы будем вызывать их через такой указатель? Понятие *полиморфный класс*. Вызов виртуальных функций из методов классов. Виртуальные функции в конструкторах и деструкторах. Виртуальный деструктор. Переопределение методов в производных классах. Ключевые слова `override` и `final`. Приватность и виртуальные функции.

   c. **Применение динамического полиморфизма**
   Контейнер указателей на базовый класс, хранящий адреса объектов производных классов. Хранение в поле типа "указатель на базовый класс"адреса объектов производных классов. Полиморфизм и умные указатели.

   d. **Основы реализации механизма виртуальных функций**
   Размер объектов полиморфных классов. Таблица виртуальных функций. Скрытое поле – указатель на таблицу виртуальных функций. Сколько таблиц виртуальных функций хранится в памяти при работе программы? Как устроены таблицы виртуальных функций? Как программа понимает какой виртуальный метод вызывать?

   e. **Абстрактные классы**
   Чистая виртуальная функция. Абстрактный класс. Интерфейс. Наследование от интерфейсов. Ошибка pure virtual call. Множественное наследование от интерфейсов.

   f. **RTTI**
   Использование `static_cast` для приведения типов и указателей на типы в иерархии наследования. Когда использование `static_cast` может привести к неопределённому поведению? Оператор `dynamic_cast`. Чем он отличается от `static_cast` и в каких случая он используется? Что происходит если `dynamic_cast` не может привести тип (рассмотрите случай приведения указателей и случай приведения ссылок)? Оператор `typeid` и класс `std::type_info`.

4. **Принципы проектирования**

   a. **Принципы проектирования**
   Какие свойства кода программ важны при написании больших программ. Принципы SOLID. Принцип единственной ответственности. Принцип открытости/закрытости. Принцип подстановки Барбары Лисков. Принцип разделения интерфейсов. Принцип инверсии зависимостей. Принцип "Инкапсулируйте то, что может измениться". Принцип "Предпочитайте композицию наследованию".

   b. **Паттерны проектирования**
   Что такое паттерны проектирования? Зачем нужно использовать паттерны проектирования? Поведенческие, структурные и порождающие паттерны.

   c. **UML-диаграммы**
   UML-диаграммы. Отношения между классами: наследование, зависимость, ассоциация, агрегация, композиция. Как эти отношения отображаются на UML-диаграммах.

5. **Поведенческие паттерны**

   a. **Стратегия**
   Какие проблемы решает использование паттерна Стратегия? Когда можно применять этот паттерн? Преимущества и недостатки паттерна. Примеры использования паттерна. Уметь самостоятельно написать код, использующий паттерн Стратегия.

   b. **Цепочка обязанностей**
   Когда можно применять паттерн Цепочка обязанностей? Преимущества и недостатки. Примеры использования.

   c. **Наблюдатель**
   Какие проблемы решает использование паттерна Наблюдатель? Когда можно применять этот паттерн? Преимущества и недостатки. Примеры использования. Уметь самостоятельно написать код, использующий паттерн Наблюдатель. Что такое Издатель и Подписчик в контексте этого паттерна?

d. **Команда**
Какие проблемы решает использование паттерна Команда? Преимущества и недостатки. Примеры использования. Уметь самостоятельно написать код, использующий паттерн Команда. Реализация класса мультикоманды (команды, состоящей из нескольких команд). Реализация операции отмены команд при использовании этого паттерна.

e. **Посетитель**
Когда можно применять паттерн Посетитель? Преимущества и недостатки. Примеры использования.

f. **Состояние**
Паттерн Состояние. Машина состояний (англ. *Finite-state machine*). Какие проблемы решает использование паттерна Состояние? Когда можно применять этот паттерн? Преимущества и недостатки. Примеры использования.

6. **Структурные паттерны**

a. **Адаптер**
Какие проблемы решает использование паттерна Адаптер? Преимущества и недостатки. Примеры использования.

b. **Фасад**
Какие проблемы решает использование паттерна Фасад? Преимущества и недостатки. Примеры использования.

c. **Заместитель (Прокси)**
Когда можно применять паттерн Заместитель? Преимущества и недостатки. Примеры использования. Виды заместителей. Виртуальный заместитель. Удалённый заместитель. Защищающий заместитель. Кэширующий заместитель. Логирующий заместитель.

d. **Декоратор**
Какие проблемы решает использование паттерна Декоратор? Когда можно применять этот паттерн? Преимущества и недостатки. Примеры использования. Уметь самостоятельно написать код, использующий паттерн Декоратор. В чём отличие паттерна Декоратор от паттернов Адаптер и Заместитель?

e. **Компоновщик**
Какие проблемы решает использование паттерна Компоновщик? Когда можно применять этот паттерн? Преимущества и недостатки. Примеры использования.

f. **Мост**
Какие проблемы решает использование паттерна Мост? Преимущества и недостатки. Примеры использования. В чём отличие паттерна Мост от паттерна Стратегия?

7. **Порождающие паттерны**

a. **Фабричный метод**
Какие проблемы решает использование паттерна Фабричный метод? Когда можно применять этот паттерн? Преимущества и недостатки. Примеры использования. Уметь самостоятельно написать код, использующий паттерн Фабричный метод.

b. **Абстрактная фабрика**
Какие проблемы решает использование паттерна Абстрактная фабрика? Когда можно применять этот паттерн? Преимущества и недостатки. Примеры использования. Уметь самостоятельно написать код, использующий паттерн Абстрактная фабрика.

c. **Строитель**
Какие проблемы решает использование паттерна Строитель? Преимущества и недостатки. Примеры использования.

d. **Прототип**
Какие проблемы решает использование паттерна Прототип? Преимущества и недостатки. Примеры использования.

e. **Синглтон**
Какие проблемы решает использование паттерна Синглтон? Преимущества и недостатки. Примеры использования. Уметь самостоятельно написать Синглтон.