1. \*\*Понятие жизненного цикла ПО и его этапы:\*\*

- \*\*Жизненный цикл программного обеспечения (ПО)\*\* охватывает все фазы, начиная с идеи создания программы и заканчивая её выводом из эксплуатации.

- \*\*Этапы жизненного цикла ПО:\*\*

- \*\*Планирование:\*\* Определение целей, задач, бюджета и ресурсов для разработки.

- \*\*Анализ:\*\* Исследование требований пользователя и определение функциональных и нефункциональных характеристик ПО.

- \*\*Проектирование:\*\* Разработка архитектуры, определение структуры и интерфейсов программы.

- \*\*Разработка:\*\* Непосредственное создание программного кода и тестирование отдельных модулей.

- \*\*Тестирование:\*\* Проверка работоспособности и выявление ошибок в программе.

- \*\*Внедрение:\*\* Внедрение программы в рабочую среду пользователя.

- \*\*Эксплуатация:\*\* Поддержка и обновление программы в процессе её использования.

- \*\*Снятие с эксплуатации:\*\* Окончательное вывод из эксплуатации, если программа устарела или не соответствует требованиям.

2. \*\*Технологии структурного программирования:\*\*

- \*\*Структурное программирование\*\* — методология разработки программ, основанная на использовании структурных элементов.

- \*\*Основные технологии:\*\*

- \*\*Последовательность:\*\* Выполнение команд в определенном порядке.

- \*\*Ветвление:\*\* Использование условных операторов для выбора между альтернативными путями выполнения.

- \*\*Циклы:\*\* Итеративное выполнение блока команд до выполнения определенного условия.

- \*\*Подпрограммы (процедуры и функции):\*\* Разделение кода на логические блоки для упрощения понимания и обеспечения возможности повторного использования кода.

3. \*\*Инструментальные средства оформления и документирования алгоритмов программ:\*\*

- \*\*Комментарии:\*\* Вставка текстовых пояснений в код программы для описания его структуры и функций.

- \*\*Диаграммы:\*\* Использование графических представлений, таких как блок-схемы, для визуализации алгоритмов и структуры программы.

- \*\*Документация:\*\* Создание текстовых документов, описывающих цели, функциональность, использование и технические аспекты программы.

- \*\*Интегрированные среды разработки (IDE):\*\* Предоставление средств для автоматического создания документации, а также подсказок и анализа кода.

- \*\*Системы контроля версий:\*\* Позволяют отслеживать изменения в коде, облегчая управление и поддержку программного продукта.

4. \*\*Классификация алгоритмов:\*\*

- \*\*По времени выполнения:\*\*

- \*Полиномиальные:\* Время выполнения зависит от полинома от размера входных данных.

- \*Экспоненциальные:\* Время выполнения растет экспоненциально с увеличением размера входных данных.

- \*Логарифмические:\* Время выполнения зависит от логарифма от размера входных данных.

- \*\*По степени детерминированности:\*\*

- \*Детерминированные:\* Всегда дают одинаковый результат при одних и тех же входных данных.

- \*Недетерминированные:\* Могут давать различные результаты при одинаковых входных данных.

- \*\*По типу данных:\*\*

- \*Числовые алгоритмы:\* Работают с числовыми данными.

- \*Текстовые алгоритмы:\* Ориентированы на обработку текстовой информации.

- \*Графовые алгоритмы:\* Применяются к работе с графами и сетями.

- \*\*По методу решения:\*\*

- \*Прямые алгоритмы:\* Решают задачу непосредственно.

- \*Итерационные (итеративные) алгоритмы:\* Решают задачу путем многократного применения одного и того же процесса.

- \*Рекурсивные алгоритмы:\* Решают задачу, разбивая её на подзадачи того же типа.

5. \*\*Основные принципы объектно-ориентированного программирования:\*\*

- \*\*Инкапсуляция:\*\* Сокрытие деталей реализации и предоставление интерфейса для взаимодействия.

- \*\*Наследование:\*\* Позволяет создавать новые классы на основе существующих, наследуя их свойства и методы.

- \*\*Полиморфизм:\*\* Возможность использования одного и того же интерфейса для объектов разных классов, что позволяет обрабатывать их единообразно.

6. \*\*Классы. Основные понятия:\*\*

- \*\*Класс:\*\* Абстрактный тип данных, описывающий набор свойств (полей) и методов.

- \*\*Объект:\*\* Экземпляр класса, содержащий конкретные данные.

- \*\*Поля (атрибуты):\*\* Переменные, хранящие состояние объекта.

- \*\*Методы:\*\* Функции, связанные с классом, определяющие его поведение.

- \*\*Инкапсуляция:\*\* Способность объединения данных и методов внутри класса и сокрытие их от внешнего доступа.

- \*\*Наследование:\*\* Механизм создания нового класса на основе существующего, позволяющий повторно использовать код и расширять функциональность.

- \*\*Полиморфизм:\*\* Возможность использования объектов разных классов с использованием общего интерфейса.

7. \*\*Определение интерфейсов. Модификаторы доступа интерфейсов:\*\*

- \*\*Интерфейс:\*\* В ООП интерфейс представляет абстракцию, описывающую методы, которые класс должен реализовать. Интерфейсы не содержат реализации методов, только их сигнатуры.

- \*\*Модификаторы доступа интерфейсов:\*\*

- \*public:\* Методы интерфейса доступны из любого места программы.

- \*default (по умолчанию):\* Используется в Java и позволяет определить метод интерфейса с реализацией, которая может быть унаследована подклассами, но может быть переопределена.

- \*private:\* Методы интерфейса могут быть использованы только внутри самого интерфейса.

- \*protected:\* Недоступен для классов вне пакета, но доступен для подклассов.

8. \*\*Назначение и виды паттернов:\*\*

- \*\*Паттерн программирования:\*\* Общее решение типичной проблемы в конкретном контексте программирования.

- \*\*Виды паттернов:\*\*

- \*Порождающие (Creational):\* Отвечают за процесс создания объектов.

- \*Структурные (Structural):\* Описывают, как компоновать объекты в более крупные структуры.

- \*Поведенческие (Behavioral):\* Определяют методы взаимодействия объектов.

9. \*\*Порождающие паттерны: фабричный метод (Factory Method):\*\*

- \*\*Фабричный метод:\*\* Определяет интерфейс для создания объекта, но оставляет выбор подклассам, изменяя тип создаваемых объектов.

- \*\*Цель:\*\* Делегирование процесса создания экземпляра подклассам, предоставляя им возможность изменять тип создаваемого объекта.

10. \*\*Порождающие паттерны: Абстрактная фабрика (Abstract Factory):\*\*

- \*\*Абстрактная фабрика:\*\* Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, без указания их конкретных классов.

- \*\*Цель:\*\* Обеспечение создания объектов, которые взаимодействуют друг с другом, без необходимости указания их конкретных классов.

11. \*\*Порождающие паттерны: Строитель (Builder):\*\*

- \*\*Строитель (Builder):\*\* Позволяет создавать сложные объекты пошагово. Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться различные представления.

- \*\*Цель:\*\* Упростить создание сложных объектов, обеспечивая пошаговое конструирование.

12. \*\*Паттерны поведения: Стратегия (Strategy):\*\*

- \*\*Стратегия (Strategy):\*\* Определяет семейство алгоритмов, инкапсулирует каждый из них и обеспечивает их взаимозаменяемость. Позволяет изменять поведение объекта во время выполнения программы.

- \*\*Цель:\*\* Определение семейства алгоритмов, инкапсуляция каждого из них и обеспечение их взаимозаменяемости.

13. \*\*Паттерны поведения: Итератор (Iterator):\*\*

- \*\*Итератор (Iterator):\*\* Предоставляет механизм последовательного доступа к элементам объекта без раскрытия его внутренней структуры.

- \*\*Цель:\*\* Обеспечение единообразного способа доступа к элементам коллекции.

14. \*\*Паттерны поведения: Посетитель (Visitor):\*\*

- \*\*Посетитель (Visitor):\*\* Позволяет определить новую операцию без изменения классов объектов, над которыми эта операция выполняется.

- \*\*Цель:\*\* Обеспечение возможности добавления новых операций к объектам без изменения их классов.

15. \*\*Структурные паттерны: Компоновщик (Composite):\*\*

- \*\*Компоновщик (Composite):\*\* Позволяет клиентам обращаться к отдельным объектам и их композициям одинаковым образом.

- \*\*Цель:\*\* Обеспечение единообразного обращения к индивидуальным объектам и их композициям.

16. \*\*Структурные паттерны: Декоратор (Decorator):\*\*

- \*\*Декоратор (Decorator):\*\* Позволяет добавлять новое поведение или состояние объекту, не изменяя его структуры.

- \*\*Цель:\*\* Расширение функциональности объекта, делая его более гибким и удобным для использования.

17. \*\*Структурные паттерны: Приспособленец (Flyweight):\*\*

- \*\*Приспособленец (Flyweight):\*\* Обеспечивает эффективное разделение, совместное использование и многократное использование мелких объектов.

- \*\*Цель:\*\* Минимизация использования памяти или вычислительных ресурсов путем совместного использования общих частей объектов.

18. \*\*Событийно-управляемое программирование:\*\*

- \*\*Событийно-управляемое программирование (Event-Driven Programming):\*\* Парадигма программирования, в которой выполнение программы определяется событиями, такими как действия пользователя, изменения внешних условий и т. д.

- \*\*Цель:\*\* Реагирование на события и выполнение соответствующих действий без явного контроля программой.

19. \*\*Основные свойства форм приложения Windows Forms среды разработки C#:\*\*

- \*\*Text:\*\* Текст заголовка формы.

- \*\*BackColor и ForeColor:\*\* Цвет фона и цвет текста.

- \*\*Size и Location:\*\* Размер и положение формы.

- \*\*Controls:\*\* Коллекция элементов управления, размещенных на форме.

- \*\*Name:\*\* Уникальное имя формы.

- \*\*Visible:\*\* Видимость формы.

- \*\*Enabled:\*\* Включение или выключение возможности взаимодействия с формой.

20. \*\*Методы оптимизации программного кода:\*\*

- \*\*Профилирование:\*\* Идентификация участков кода, занимающих больше всего времени выполнения.

- \*\*Оптимизация алгоритмов:\*\* Выбор или изменение алгоритмов для улучшения производительности.

- \*\*Ленивая загрузка (Lazy Loading):\*\* Загрузка данных или ресурсов только при необходимости.

- \*\*Кэширование:\*\* Сохранение результатов вычислений для повторного использования.

- \*\*Параллельное выполнение (Parallelization):\*\* Разделение задач для выполнения на нескольких ядрах процессора.

21. \*\*Цели и методы рефакторинга программного кода:\*\*

- \*\*Цели рефакторинга:\*\*

- Улучшение читаемости кода.

- Уменьшение дублирования кода.

- Улучшение структуры программы.

- Облегчение поддержки и дальнейшего развития.

- \*\*Методы рефакторинга:\*\*

- \*\*Выделение метода (Extract Method):\*\* Выделение повторяющегося фрагмента кода в отдельный метод.

- \*\*Переименование (Rename):\*\* Изменение имени переменных, методов или классов для улучшения читаемости.

- \*\*Удаление неиспользуемого кода (Remove Unused Code):\*\* Избавление от лишних или неиспользуемых элементов.

22. \*\*Правила разработки пользовательского интерфейса:\*\*

- \*\*Ясность и простота:\*\* Интерфейс должен быть легким в понимании и использовании.

- \*\*Согласованность:\*\* Все элементы интерфейса должны следовать общему стилю и визуальной концепции.

- \*\*Обратная связь:\*\* Пользователь должен получать информацию о результате своих действий.

- \*\*Эффективность:\*\* Минимизация количества шагов для достижения цели пользователя.

- \*\*Отзывчивость:\*\* Система должна реагировать на действия пользователя немедленно.

- \*\*Безопасность:\*\* Защита от нежелательных действий и обеспечение безопасности данных.

23. \*\*Арифметические операции языка C#:\*\*

- \*\*Сложение (+):\*\* Складывает два операнда.

- \*\*Вычитание (-):\*\* Вычитает правый операнд из левого.

- \*\*Умножение (\*):\*\* Умножает два операнда.

- \*\*Деление (/):\*\* Делит левый операнд на правый.

- \*\*Остаток от деления (%):\*\* Возвращает остаток от деления левого операнда на правый.

- \*\*Инкремент (++):\*\* Увеличивает операнд на 1.

- \*\*Декремент (--):\*\* Уменьшает операнд на 1.

24. \*\*Понятие массива. Виды массивов. Синтаксис массивов:\*\*

- \*\*Массив:\*\* Структура данных, содержащая элементы одного типа, расположенные в памяти последовательно.

- \*\*Виды массивов:\*\*

- \*Одномерные массивы:\* Массивы с одним измерением.

- \*Двумерные массивы:\* Массивы с двумя измерениями (матрицы).

- \*Jagged массивы:\* Массивы массивов, где каждый подмассив может иметь разную длину.

- \*\*Синтаксис массивов в C#:\*\*

- \*Одномерный массив:\* `int[] numbers = new int[5];`

- \*Двумерный массив:\* `int[,] matrix = new int[3, 3];`

- \*Jagged массив:\* `int[][] jaggedArray = new int[3][];`

25. \*\*Пространства имен, псевдонимы и статический импорт:\*\*

- \*\*Пространство имен (Namespace):\*\* Логическая группировка классов, интерфейсов, делегатов и других типов для организации кода.

- \*\*Псевдонимы (Alias):\*\* Возможность создать короткий идентификатор для длинного полного имени типа.

using ShortName = System.Collections.Generic.List<int>;

- \*\*Статический импорт:\*\* Возможность использования статических членов класса без указания полного имени класса.

using static System.Console;

Пример использования псевдонима и статического импорта:

using ShortName = System.Collections.Generic.List<int>;

class Program

{

static void Main()

{

ShortName list = new ShortName();

WriteLine("Hello, World!");

}

}