# Вопросы:

1. Понятие репозитория проекта. Структура проекта.
2. Автоматизация бизнес-процессов.
3. Понятие артефакта и протокола проекта. Состав перечня артефактов и протоколов проекта.
4. Организация работы команды в системе контроля версий.
5. Особенности системы контроля версий Git.
6. Система контроля версий Git: создание проекта, локального и удаленного репозитория, фиксация и отправка коммитов.
7. Система контроля версий Git: клонирование удаленного репозитория, извлечение и получение изменений, совместная параллельная разработка, работа с ветвями.
8. Понятие отладки программных продуктов.
9. Инструменты отладки программных продуктов.
10. Понятие ручного и автоматизированного тестирования программных продуктов.
11. Методы и средства организации тестирования программных продуктов.

# 12. Тестирование интерфейса пользователя программных продуктов.

Тестирование интерфейса пользователя (UI) сосредоточено на проверке графического интерфейса (GUI), через который пользователь взаимодействует с программным продуктом. Его основная цель — убедиться, что интерфейс удобен, интуитивно понятен и работает без ошибок.

**Основные аспекты тестирования интерфейса:**

1. **Проверка элементов интерфейса:**
   * Наличие и корректное отображение всех элементов (кнопки, поля ввода, выпадающие меню, изображения).
   * Правильное расположение элементов на экране.
   * Адаптация интерфейса для различных устройств (например, ПК, планшетов, смартфонов).
2. **Тестирование взаимодействия:**
   * Проверка, правильно ли работают события, связанные с элементами (например, нажатия на кнопки, заполнение форм).
   * Оценка отзывчивости интерфейса (время отклика на действия пользователя).
3. **Проверка эргономики и удобства использования:**
   * Является ли интерфейс понятным для пользователя.
   * Отсутствие перегруженности элементов.
4. **Тестирование доступности:**
   * Возможность использовать интерфейс людям с ограниченными возможностями (например, поддержка экранных читалок, управление клавиатурой).

**Виды тестирования интерфейса:**

* **Ручное тестирование:** Тестировщик сам взаимодействует с приложением, проверяя каждый элемент.
* **Автоматизированное тестирование:** Используются инструменты, которые симулируют действия пользователя.

Тестирование интерфейса пользователя (UI) направлено на проверку внешнего вида и функциональности интерфейса, с которым взаимодействует конечный пользователь. Цель — убедиться, что пользовательский интерфейс интуитивно понятен, удобен, не имеет ошибок, и все элементы работают корректно. Примеры аспектов тестирования интерфейса:

* Проверка отображения всех элементов (кнопки, текстовые поля, меню).
* Проверка правильности работы событий (клики, ввод текста).
* Тестирование адаптивности интерфейса для разных устройств и разрешений экрана.

# 13. Функциональное тестирование программных продуктов.

Функциональное тестирование — это процесс проверки работоспособности программного продукта в соответствии с заданными требованиями. Оно фокусируется на том, чтобы подтвердить, что каждая функция системы работает правильно.

**Цели функционального тестирования:**

1. Проверить соответствие системы заявленным функциональным требованиям.
2. Убедиться, что система корректно обрабатывает вводимые данные.
3. Гарантировать правильность результатов выполнения функций.

**Методы функционального тестирования:**

* **Позитивное тестирование:** Проверка, что система работает корректно с ожидаемыми данными.
* **Негативное тестирование:** Проверка поведения системы с неверными или неожиданными данными.
* **Smoke-тестирование:** Быстрая проверка основных функций системы.
* **Регрессионное тестирование:** Убедиться, что внесенные изменения не повлияли на ранее работавшие функции.

**Этапы функционального тестирования:**

1. Определение требований, которые необходимо протестировать.
2. Создание тестовых сценариев.
3. Выполнение тестов.
4. Сравнение фактического результата с ожидаемым.
5. Документирование результатов.

**Примеры проверяемых функций:**

* Вход в систему.
* Добавление, удаление и изменение записей.
* Обработка платежей.

Функциональное тестирование проверяет, правильно ли работает функциональность приложения в соответствии с его требованиями. Оно фокусируется на том, чтобы убедиться, что все функции системы работают как ожидается:

* Проверка выполнения основных операций (например, отправка формы, загрузка данных).
* Проверка корректности результатов, получаемых при взаимодействии с приложением.

# 14. Тестирование интеграции программных продуктов.

Тестирование интеграции проверяет взаимодействие между различными модулями системы или внешними системами. Основная цель — выявить проблемы на стыке между модулями, которые могут возникнуть из-за различий в их реализации или способе взаимодействия.

**Виды интеграционного тестирования:**

1. **Инкрементальное тестирование:**
   * **Нисходящее:** Сначала тестируются модули высокого уровня, к которым постепенно добавляются модули низкого уровня.
   * **Восходящее:** Сначала тестируются модули низкого уровня, а затем соединяются с модулями высокого уровня.
2. **Большой взрыв:** Все модули интегрируются одновременно, а затем тестируется вся система целиком.
3. **Тестирование API:** Проверка взаимодействия между системами через их интерфейсы.

**Ключевые аспекты:**

* Корректность передачи данных между модулями.
* Управление ошибками при передаче данных.
* Производительность взаимодействия.

**Примеры интеграции:**

* Взаимодействие клиентской части (например, веб-приложения) с серверной частью.
* Работа программы с базой данных.
* Интеграция внешних API, например, платежных систем.

Тестирование интеграции проверяет, как взаимодействуют разные части системы или разные системы между собой. Оно направлено на выявление проблем, возникающих при соединении различных модулей, например, интеграция базы данных с сервером или взаимодействие разных сервисов через API.

* Проверка корректности обмена данными между модулями.
* Проверка устойчивости системы при нагрузке.

# 15. Инструментарии анализа качества программных продуктов в среде разработки.

Анализ качества программного обеспечения позволяет выявить потенциальные проблемы и улучшить его структуру, производительность и безопасность. Современные среды разработки предоставляют множество инструментов, которые помогают в этой задаче.

**Основные цели анализа качества:**

1. Выявить дефекты на ранних этапах разработки.
2. Оптимизировать производительность кода.
3. Обеспечить соответствие стандартам кодирования.
4. Повысить читаемость и поддержку кода.

**Инструменты анализа качества:**

1. **Статический анализатор кода:**
   * Проверяет код без его выполнения.
   * Инструменты: PMD, Checkstyle, FindBugs, SpotBugs.
2. **Динамический анализатор:**
   * Оценивает производительность и поведение приложения во время выполнения.
   * Инструменты: VisualVM, JProfiler.
3. **Платформы для управления качеством:**
   * SonarQube — предоставляет детальные отчеты о качестве кода, указывает на проблемы и предлагает их решение.
4. **Инструменты тестирования:**
   * JUnit, TestNG — для автоматизации тестирования.
5. **Средства контроля версий:**
   * GitHub, GitLab — для отслеживания изменений и обеспечения совместной работы.

**Преимущества использования инструментов анализа:**

* Экономия времени за счет автоматического поиска ошибок.
* Снижение числа дефектов в финальной версии продукта.
* Улучшение качества и читаемости кода.

Для анализа качества программных продуктов существуют различные инструменты, которые помогают в выявлении ошибок, проверке стиля кода, производительности и безопасности:

* **JUnit** — для юнит-тестирования.
* **SonarQube** — для анализа кода на качество, поддержку стандартов кодирования и выявление ошибок.
* **JProfiler** или **VisualVM** — для анализа производительности приложения.
* **FindBugs** — для нахождения багов в коде.
* **PMD** — для анализа стиля кода.

# 16. Понятие исключительной ситуации. Обработка исключительных ситуаций.

**Исключительная ситуация** — это событие, которое нарушает нормальный поток выполнения программы. Она может возникнуть из-за ошибок (например, деление на ноль, обращение к несуществующему файлу) или других непредвиденных обстоятельств.

**Типы исключений:**

1. **Проверяемые исключения (Checked Exceptions):**
   * Должны быть обработаны программой.
   * Примеры: IOException, SQLException.
2. **Непроверяемые исключения (Unchecked Exceptions):**
   * Возникают во время выполнения и не требуют обязательной обработки.
   * Примеры: NullPointerException, ArithmeticException.
3. **Ошибки (Errors):**
   * Обычно связаны с проблемами в виртуальной машине Java.
   * Примеры: OutOfMemoryError.

**Механизм обработки исключений:**

* **try:** Блок, в котором может возникнуть исключение.
* **catch:** Блок, в котором исключение перехватывается и обрабатывается.
* **finally:** Блок, который выполняется всегда (даже если исключение не возникло).
* **throw:** Используется для явного выброса исключения.
* **throws:** Указывает, какие исключения может выбросить метод.

**Преимущества обработки исключений:**

* Позволяет избежать сбоев в программе.
* Помогает разделить логику программы и обработку ошибок.
* Улучшает читаемость и структуру кода.

Исключительная ситуация (или исключение) — это событие, которое прерывает обычный поток выполнения программы из-за ошибок или необычных ситуаций, таких как деление на ноль, попытка доступа к несуществующему файлу, и т. д. Обработка исключений позволяет избежать сбоев в программе и реализовать механизмы для корректного завершения работы.

В Java для обработки исключений используются конструкции try, catch, finally.

1. 17. Методы и способы идентификации сбоев и ошибок.
2. 18. Ошибки системных компонентов и способы их выявления.
3. 19. Перечень документов о результатах тестирования.
4. 20. Процедура проведения инспекции кода модулей проекта.
5. 21. Обработка исключений. Объявление исключений. Привести пример (представить код).
6. 22. Обработка исключений. Выбрасывание исключений. Привести пример (представить код).
7. 23. Обработка исключений. Перехват исключений. Привести пример (представить код).
8. 24. Обработка исключений. Получение информации об исключениях. Привести пример (представить код).
9. 25. Обработка исключений. Когда следует использовать исключения. Привести пример (представить код).

# Практические задания:

1. Разработать проект: вывод сообщения в терминал. Разместить проект на удаленном репозитории системы контроля версий. Внести изменения в проект: добавить тест, изменения в проекте реализовать отдельной ветке. Ветку разместить на удаленном репозитории проекта. С помощью инструментов веб-версии системы контроля версий объединить ветки. Изменения загрузить в разрабатываемый проект на локальном репозитории.
2. Выполнить исправления в opensource или другом (не вашем) проекте, размещенном на удаленном репозитории системы контроля версий.
3. Обработка исключительных ситуаций. Разработать программу, которая запрашивает у пользователя два целых числа и отображает их сумму. В случае некорректного ввода необходимо его обработать и повторно запросить у пользователя числа.
4. Обработка исключительных ситуаций. С помощью двух массивов (String[] months = {"январь", …}; int[] dom = {31,… };) разработать программу, которая предложит пользователю ввести целое число от 1 до 12, а затем отобразит месяц и количество дней, соответствующие этому целому числу. Если пользователь вводит недопустимое число, то программа должна отображать Недопустимое число с помощью перехвата исключения.
5. Обработка исключительных ситуаций. Разработать программу ввода числа и отображения соответствующего месяца. Необходимо предотвратить ввод пользователем любого числа, кроме целого.
6. Обработка исключительных ситуаций. Определить класс Triangle для задания треугольника с тремя сторонами. В треугольнике сумма длин любых двух сторон больше длины третьей стороны. Класс Triangle должен удовлетворять этому правилу. Создать класс IllegalTriangleException и конструктор класса Triangle, чтобы выбросить объект типа IllegalTriangleException, если треугольник создан со сторонами, нарушающими это правило.
7. Обработка исключительных ситуаций. Создать класс с двумя целочисленными полями: числителем и знаменателем. Конструктор этого класса создает дроби при заданных значениях, но выбрасывает исключение при нулевом знаменателе. Для этого дополнительно создать класс проверяемого исключения, объекты которого выбрасываются, если знаменатель объекта равен 0, а также написать тестовый класс, который проверяет эти классы.
8. Разработать метод, которому передается коллекция объектов типа ArrayList, а возвращается коллекция ArrayList, но уже без дубликатов. Необходимо использовать метод contains() интерфейса List.
9. Реализовать алгоритм линейного поиска элемента в массиве. При нахождении элемента необходимо вернуть его позицию в массиве. Если элемент не найден, то вернуть -1.

10. Реализовать поиск наибольшего элемента в массиве с помощью метода compareTo() интерфейса Comparable. Определите класс Circle с полем radius и найдите наибольший элемент в массиве экземпляров этого класса.

11. Реализуйте поиск наибольшего элемента в двумерном массиве с помощью метода compareTo() интерфейса Comparable.

12. Разработать обобщенный метод для сортировки массива объектов типа Comparable. Для проверки этого метода разработать программу, которая сортирует массив целых чисел, массив чисел типа double, массив символов и массив строк.

13. Реализовать класс Cat. У класса должны быть реализованы следующие приватные методы:

− sleep() – выводит на экран «Sleep»

− meow() – выводит на экран «Meow»

− eat() – выводит на экран «Eat» И публичный метод:

− status() – вызывает один из приватных методов случайным образом.

Загрузить проект в отдельный репозиторий с названием, совпадающим с названием проекта.

14. Реализовать класс Student. У класса должны быть следующие приватные поля:

− String name – имя студента.

− String surname – фамилия студента.

− int[] grades – последние 10 оценок студента.

Их может быть меньше, но не может быть больше 10.

И следующие публичные методы:

− Getter и setter-методы для name.

− Getter и setter-методы для surname.

− Getter и setter-методы для grades.

− метод, добавляющий новую оценку в grades. Самая первая оценка должна быть удалена, новая должна сохраниться в конце массива (т.е. массив должен сдвинуться на 1 влево).

− метод, возвращающий средний балл студента (рассчитывается как среднее арифметическое от всех оценок в массиве grades).

Загрузить проект в отдельный репозиторий с названием, совпадающим с названием проекта.

15. Реализовать класс Student. У класса должны быть следующие приватные поля:

− String name – имя студента

− String surname – фамилия студента

− int[] grades – последние 10 оценок студента.

Необходимо реализовать класс StudentService. У класса должны быть реализованы следующие публичные методы:

− bestStudent() – принимает массив студентов (класс Student), возвращает лучшего студента (студент, который имеет самый высокий средний балл). Если таких несколько – вывести любого.

− sortBySurname() – принимает массив студентов (класс Student) и сортирует его по фамилии.

Загрузить проект в отдельный репозиторий с названием, совпадающим с названием проекта.

16. Напишите программу на Java, используя Java (IO или NIO), которая считывает содержимое текстового файла и выводит на экран количество строк в этом файле, а также самые длинные строки. Необходимо написать программу, которая принимает в качестве входного параметра путь к текстовому файлу. Программа должна следующие действия:

− Подсчитать общее количество строк в файле.

− Найти самые длинные строки в файле. Если есть несколько строк одинаковой максимальной длины, выведите все такие строки. 17. Разработать программу на Java, используя Java (IO или NIO), которая считывает содержимое двух текстовых файлов и создает третий файл, содержащий объединенное содержимое этих двух файлов. Программа должна считать содержимое этих файлов, а затем записать объединенное содержимое в третий файл. Очередность записи в третий файл не важна.

18. Создать функцию, для сложения двух чисел. Написать пять модульных тестов для проверки работоспособности данной функции. 19. Создать функцию, которая ищет наибольшее число в массиве. Написать пять модульных тестов для проверки работоспособности данной функции.

20. Создать функцию, для умножения двух чисел. Написать пять модульных тестов для проверки работоспособности данной функции. 21. Создать функцию нахождения максимального из трех чисел. Написать пять модульных тестов для проверки работоспособности данной функции.

22. Создать функцию, для определения количества букв в строке. Написать пять модульных тестов для проверки работоспособности данной функции. 23. Создать функцию, для определения количества цифр в строке. Написать пять модульных тестов для проверки работоспособности данной функции.

24. Создать функцию, для определения количества специальных символов в строке. Написать пять модульных тестов для проверки работоспособности данной функции.

25. Разработать класс для хранения данных о пользователе туристического агентства. Используя созданный класс, смоделировать работу приложения туристического агентства. Загрузить проект в отдельный репозиторий с названием, совпадающим с названием проекта.

26. Разработать класс для хранения данных о покупателе магазина ювелирных украшений. Используя созданный класс, смоделировать работу приложения магазина ювелирных украшений. Загрузить проект в отдельный репозиторий с названием, совпадающим с названием проекта.

27. Разработать класс для хранения данных о покупателе магазина спортивной одежды. Используя созданный класс, смоделировать работу приложения магазина спортивной одежды. Загрузить проект в отдельный репозиторий с названием, совпадающим с названием проекта.

28. Разработать класс для хранения данных о товаре в строительном магазине. Используя созданный класс, смоделировать работу приложения хранения данных о товаре в строительном магазине. Загрузить проект в отдельный репозиторий с названием, совпадающим с названием проекта.

29. Разработать класс для хранения данных о животном в питомнике для бездомных животных. Используя созданный класс, смоделировать работу приложения для хранения данных о животном в питомнике для бездомных животных. Загрузить проект в отдельный репозиторий с названием, совпадающим с названием проекта.

30. Разработать класс для хранения данных об автомобилях, находящихся в таксопарке. Используя созданный класс, смоделировать работу приложения. Загрузить проект в отдельный репозиторий с названием, совпадающим с названием проекта.