1. **Принципы объектно-ориентированного программирования SOLID и STUPID.**

SOLID —

• S (Single Responsibility Principle) — Принцип единственной ответственности: класс должен отвечать за одну задачу.

• O (Open/Closed Principle) — Принцип открытости/закрытости: программные сущности должны быть открыты для расширения, но закрыты для модификации.

• L (Liskov Substitution Principle) — Принцип подстановки Лисков: объекты программы должны быть заменяемы на объекты их подтипов

• I (Interface Segregation Principle) — Принцип разделения интерфейса: Вместо общего интерфейса лучше делить его на более конкретные. Интерфейсы должны создаваться под конкретных клиентов.

• D (Dependency Inversion Principle) — Принцип инверсии зависимостей: объекты должны зависеть от абстракции, а не от других модулей. Например есть программа с модулями X Y Z. Каждый модуль должен зависеть от абстракции. А не X-Y Z или как то так.

STUPID — это антоним SOLID и включает в себя принципы, которые следует избегать:

• S (Single Point of Failure) — Единая точка отказа.

• T (Tightly Coupled) — Плотная связь между компонентами.

• U (Untestability) — Невозможность тестирования.

• P (Premature Optimization) — Преждевременная оптимизация.

• I (Inflexibility) — Негибкость.

• D (Duplication) — Дублирование кода.

1. **Класс Object. Реализация его методов по умолчанию. Для чего нужны эти методы.**

Класс Object является корневым классом в Java и предоставляет базовые методы, доступные для всех объектов. Основные методы:

• equals(Object obj) — сравнивает два объекта на равенство.

• hashCode() — возвращает хэш-код объекта.

• toString() — возвращает строковое представление объекта.

• getClass() — возвращает класс объекта.

• clone() — создает и возвращает копию объекта.

• finalize() — вызывается перед сборкой мусора.

Если **x.equals(y) == true**, то обязательно **hashcode(x) == hashcode(y)**

2) Если **hashcode(x) == hashcode(y)**, то не обязательно **x.equals(y) == true**

1. **Простое и множественное наследование. Особенности реализации наследования в Java.**

В Java поддерживается простое наследование, что означает, что класс может наследовать только один класс. Это предотвращает проблемы, связанные с "ромбовидным" наследованием. Однако Java поддерживает множественное наследование интерфейсов, что позволяет классу реализовывать несколько интерфейсов.

1. **Понятие абстрактного класса. Модификатор abstract.**

Абстрактный класс — это класс, который не может быть инстанцирован и может содержать абстрактные методы (но реализацию сделать можно все равно). Модификатор abstract используется для определения таких классов и методов. Абстрактные классы служат основой для других классов и могут содержать как абстрактные, так и обычные методы.

1. **Понятие интерфейса. Реализация интерфейсов в Java. Отличие интерфейсов от абстрактных классов.**

Интерфейс в Java — это контракт, который определяет набор методов, которые класс должен реализовать. Интерфейсы могут содержать методы с реализацией (default и static).

Отличия:

• Класс может наследоваться только от одного класса, но реализовать множество интерфейсов.

• Интерфейсы не могут иметь состояния (поля), кроме static final полей.

• Абстрактный класс может содержать как абстрактные, так и конкретные методы, тогда как интерфейс в основном определяет только абстрактные методы (до Java 8).

1. **Модификаторы default, static и private для методов интерфейса.**

default: позволяет добавить реализацию метода в интерфейс. Классы, реализующие интерфейс, могут использовать этот метод без необходимости его переопределять.

• static: позволяет определить статические методы в интерфейсе, которые могут быть вызваны без создания экземпляра класса.

• private: можно определять private методы внутри интерфейсов для использования в других методах интерфейса.

1. **Перечисляемый тип данных (enum) в Java. Особенности реализации и использования.**

Перечисляемый тип (enum) в Java используется для определения фиксированного набора констант. Он позволяет создавать типы с ограниченным набором значений, что улучшает читаемость и безопасность кода. Перечисления могут иметь поля, методы и конструкторы. Наследоваться нельзя

**values()**: возвращает массив из всех хранящихся в Enum значений:

**ordinal()**: возвращает индекс константы.

**valueOf()**: возвращает объект Enum, соответствующий переданному имени:

1. **Тип запись (record) в Java. Особенности использования.**

Записи (record) были введены в Java 14 как способ создания простых классов с неизменяемыми свойствами. Они автоматически предоставляют реализацию методов equals(), hashCode(), и toString().

Пример:

public record Point(int x, int y) {}

* Записи не могут расширять любой класс, хотя они могут реализовывать интерфейсы.
* Записи не могут быть абстрактными.
* Записи неявно являются final; они не могут быть унаследованы
* Вы можете объявить дополнительные поля в теле записи, но только если они статичны

1. **Методы и поля с модификаторами static и final.**

static: статические поля и методы принадлежат классу, а не экземпляру класса. Они могут быть вызваны без создания объекта класса.

• final: финальные поля не могут быть изменены после их инициализации, а финальные методы не могут быть переопределены в подклассах.

1. **Перегрузка и переопределение методов.**

пон

1. **Обработка исключительных ситуаций, три типа исключений.**

Java поддерживает обработку исключений через конструкции try-catch-finally. Три типа исключений:

1. Checked Exceptions: Исключения, которые должны быть обработаны или объявлены в сигнатуре метода (например, IOException).

2. Unchecked Exceptions: Исключения времени выполнения, которые не требуют обязательной обработки (например, NullPointerException).

3. Errors: Серьезные проблемы, которые обычно невозможно обработать (например, OutOfMemoryError).

1. **Стандартный массив и динамический массив (ArrayList). Основные различия. Как реализован ArrayList. Сложности временные и по памяти у массива и ArrayList**

Стандартный массив имеет фиксированный размер, который устанавливается при создании массива. Динамический массив (ArrayList) может изменять свой размер автоматически при добавлении или удалении элементов

Основные различия:

• Массивы имеют фиксированный размер; ArrayList может изменяться.

• Массивы могут хранить примитивные типы; ArrayList хранит только объекты.

Временные сложности:

• Добавление элемента в массив: O(n) (при необходимости копирования).

• Добавление элемента в ArrayList: O(1) амортизированное время (в случае увеличения размера O(n)).

1. **Вложенные, локальные и анонимные классы.**

Вложенные классы: классы, определенные внутри другого класса. Они имеют доступ к членам внешнего класса.

• Локальные классы: классы, определенные внутри метода. Они могут использовать переменные метода только если они являются финальными или эффективными финальными.

• Анонимные классы: классы без имени, создаваемые на месте при создании экземпляра класса или интерфейса.

1. **- Лямбда в Java; Во что в байткоде превращаются лябмды? Как написать?**

Лямбда-выражения позволяют записывать анонимные функции более компактно и читаемо. Они представляют собой реализацию функционального интерфейса.

Пример лямбда-выражения:

Runnable r = () -> System.out.println("Hello");

В байткоде лямбда-преобразования обычно преобразуются в вызовы методов статического класса (методы фабрики), реализующего функциональный интерфейс.

1. **паттерн билдер, синглтон**

не обязательно

1. **- Effective final переменные**

Переменная считается "эффективно финальной", если она не изменяется после своей инициализации, даже если она не объявлена с модификатором final. Это позволяет использовать такие переменные внутри лямбда-выражений и анонимных классов без необходимости явного указания final.