**1. Понятие класса и объекта. Статический и нестатический контекст класса. Члены класса. Методы, поля, конструкторы, блоки инициализации. Ключевые слова abstract и final**

Класс — это шаблон или описание, по которому создаются объекты. Объект — это конкретный экземпляр класса, обладающий своими значениями полей и доступом к методам. Статический контекст связан с самим классом, а нестатический — с конкретным объектом. В статическом контексте нельзя напрямую обращаться к нестатическим членам, так как они принадлежат конкретным экземплярам.

Члены класса делятся на:

* **Поля (переменные)** — данные, которые хранят состояние объекта или класса.
* **Методы** — функции, определяющие поведение объекта.
* **Конструкторы** — специальные методы, которые вызываются при создании объекта и инициализируют его.
* **Блоки инициализации** — участки кода, выполняющиеся при создании объекта или загрузке класса (есть статические и нестатические).

Ключевое слово abstract помечает класс или метод как абстрактный. Абстрактный класс нельзя создать напрямую, а абстрактный метод не имеет реализации — его должны реализовать подклассы.

Ключевое слово final запрещает дальнейшие изменения:

* final-класс нельзя унаследовать.
* final-метод нельзя переопределить.
* final-переменной можно присвоить значение только один раз.

**2. Основополагающие принципы ООП. Инкапсуляция. Средства реализации инкапсуляции. Модификаторы доступа**

Есть два распространенных понятия — инкапсуляция и сокрытие. И под словом «инкапсуляция» авторы понимают то одно, то другое.

Изначальное значение слова «инкапсуляция» в программировании — объединение данных и методов работы с этими данными в одной упаковке («капсуле»). В Java в роли упаковки-капсулы выступает класс. Класс содержит в себе и данные (поля класса), и методы для работы с этими данными.

Сокрытие реализации осуществляется через простой и удобный интерфейс.

* private — доступен только внутри текущего класса.
* default (отсутствие модификатора) — доступен внутри пакета.
* protected — доступен внутри пакета и в наследниках.
* public — доступен отовсюду.

Доступ к приватным полям обычно обеспечивается через методы **геттеры** и **сеттеры**.

**3. Основополагающие принципы ООП. Наследование. Управление наследованием**

Наследование позволяет создавать новый класс на основе существующего, заимствуя его поля и методы. Это помогает переиспользовать код и организовывать иерархии.

Управлять наследованием можно:

* Ограничивая его через final (запрет наследования).
* Используя protected для ограничения доступа к членам.
* Переопределяя методы с помощью аннотации @Override, чтобы изменить или дополнить поведение родительского класса.

**4. Основополагающие принципы ООП. Полиморфизм. Средства реализации полиморфизма**

Полиморфизм позволяет работать с объектами разных классов через один интерфейс или базовый класс. Поведение объекта определяется его реальным типом.

В Java полиморфизм реализуется:

* **Переопределением методов** — когда подкласс изменяет поведение метода базового класса.
* **Перегрузкой методов** — создание методов с одинаковым именем, но разными параметрами.
* **Работой с интерфейсами и абстрактными классами** — объекты разных типов могут обрабатываться единообразно.

**5. Понятие класса и интерфейса: абстрактные классы, абстрактные методы. Отличие абстрактного класса от интерфейса**

Абстрактный класс — это класс, содержащий абстрактные методы без реализации, а также может содержать реализованные методы и поля. Интерфейс — это контракт, который описывает поведение, но не хранит состояния (до Java 8). В интерфейсе все методы по умолчанию абстрактные (без реализации), начиная с Java 8, можно добавлять default и static методы.

Отличие:

* Класс можно наследовать только от одного абстрактного класса, но реализовать несколько интерфейсов.
* Абстрактный класс может содержать поля и конструкторы, интерфейс до Java 8 — только методы без реализации.
* Интерфейс определяет только **что** должно быть сделано, абстрактный класс может частично описать **как**.

**6. Интерфейсы: определение, реализация, наследование**

Интерфейс — это описание набора методов без их реализации. Он определяет, какие действия должен поддерживать класс, но не указывает, как они выполняются. Класс реализует интерфейс с помощью ключевого слова, после чего обязан предоставить конкретную реализацию всех его методов.

Главная особенность интерфейсов — возможность множественной реализации. Один класс может реализовывать сразу несколько интерфейсов, в отличие от наследования классов, которое в Java возможно только от одного родителя. Интерфейсы также могут наследоваться друг от друга, объединяя методы нескольких интерфейсов в один, расширяя его возможности.

**7. Дженерики: Определение, реализация, примеры**

Дженерики — это способ создания обобщённых классов, интерфейсов и методов, которые могут работать с разными типами данных, не теряя при этом проверки типов во время компиляции. Они позволяют заранее указать, с каким типом будет работать структура, например коллекция, что избавляет от необходимости приведения типов и повышает надёжность кода.

Дженерики делают код универсальным и переиспользуемым. Вместо создания отдельной версии класса или метода для каждого типа, можно использовать один обобщённый вариант с параметром типа. Это упрощает структуру программы, уменьшает количество ошибок и делает код более понятным.

Из примеров можно сделать метод, который будет работать с различными математическими типами, такими как Int или Double одновременно. Также бывают и более сложные примеры, когда мы можем подать туда разные собственные классы, и на этом может строится логика.

**8. Интерфейсы Comparator и Comparable. Описание, различия, примеры использования**

Comparable — интерфейс, который реализуется самим классом и определяет естественный порядок объектов. В нем реализуется метод compareTo.

Пример:

public class Person implements Comparable<Person> {

    public int age;

    public int compareTo(Person other) {

        return this.age - other.age;

    }

}

Comparator — это отдельный объект-компаратор, который задает порядок сортировки для объектов. Используется, когда требуется сортировка по разным критериям.

Пример:

Comparator<Person> byName = (a, b) -> a.name.compareTo(b.name);

Отличие:

* Comparable задаёт **естественный порядок**, Comparator — **внешний, настраиваемый**.

**9. Коллекции типа List. Описание, представители. Механизм работы, различия реализаций**

List — упорядоченная коллекция элементов, допускающая дублирование.

Основные реализации:

* ArrayList — основан на массиве, быстрый доступ по индексу, медленная вставка/удаление из середины.
* LinkedList — основан на двусвязном списке, быстрая вставка/удаление, медленный доступ по индексу.

Отличаются они внутренним устройством: ArrayList хранит элементы в массиве, а LinkedList в связанных между собой узлах.

**10. Коллекции типа Map. Описание, представители. Механизм работы, различия реализаций**

Map — коллекция пар "ключ-значение", где ключи уникальны.

Основные реализации:

* HashMap — основан на хеш-таблице, быстрые операции, порядок не гарантирован.
* LinkedHashMap — сохраняет порядок вставки.
* TreeMap — отсортированная по ключам коллекция на основе красно-чёрного дерева.

Отличаются они по принципам хранения:

* HashMap использует хеш-функцию для быстрого поиска.
* LinkedHashMap добавляет связанный список для сохранения порядка.

TreeMap хранит элементы в отсортированном порядке по ключам.