1 Что такое class и в чем отличие от object?

В Java, class и object - это два ключевых понятия, связанных с объектно-ориентированным программированием (ООП).

Class (класс):

Class представляет собой шаблон или описание, по которому создаются объекты. Это абстрактное представление сущности, которая определяет состояние (поля) и поведение (методы) объектов этого класса. Класс определяет, какие данные и методы будут доступны у созданных на его основе объектов.
Класс можно рассматривать как тип данных, определяющий, как выглядит и ведет себя объект конкретного типа.
Пример класса в Java:

```
public class Person {

    String name;
    int age;


    public void sayHello() {
        System.out.println("Здравствуйте! Меня зовут " + name + " и мне " + age + " лет.");
    }
}
```

Object (объект) - это экземпляр класса, созданный на основе определенного класса.
Объект представляет конкретное сущность, имеющую свое состояние (поля) и поведение (методы), определенное в классе.

Каждый объект является отдельным экземпляром класса, и у каждого объекта свои уникальные значения полей.
Пример создания объекта класса Person:

```
public class Main {
    public static void main(String[] args) {

        Person person1 = new Person();
        person1.name = "Виктория";
        person1.age = 42;

        person1.sayHello();
    }
}
```

Проще говоря, class представляет абстрактное описание сущности, а object - это конкретный экземпляр этой сущности, созданный на основе класса. Объекты используются для работы с данными и вызова методов, определенных в классах.

2. Для чего нужны packages и что такое import?

В Java, packages (пакеты) и import (импорт) - это механизмы для организации и структурирования кода, а также для управления видимостью классов и ресурсов между различными частями программы.

Package - это механизм группировки связанных классов, интерфейсов, и других ресурсов в единую иерархию.
Пакеты позволяют организовать классы и ресурсы в логические иерархии, что делает код более упорядоченным и обеспечивает более эффективное управление большими проектами.
Название пакетов обычно соответствует структуре директорий на файловой системе, где пакеты представлены в виде папок.
Пример объявления пакета в Java:

```
package com.example.myproject;
```

Import - это ключевое слово, используемое для указания, что классы или другие ресурсы из определенного пакета должны быть доступны в текущем файле кода.
Когда вы хотите использовать класс из определенного пакета в своем коде, вы должны импортировать этот класс. Это позволяет использовать простые имена классов вместо полных путей к классам.
Примеры импорта классов в Java:

```
import java.util.ArrayList;
import java.util.List;
import com.example.myproject.SomeClass;
```

При использовании класса, который не находится в базовом пакете Java (java.lang) или текущем пакете, требуется явно указать импорт. Но классы из базового пакета java.lang импортировать не обязательно, так как они доступны по умолчанию.

Пример использования импортированных классов:

```
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add("Виктория");
        names.add("Пётр");

        for (String name : names) {
            System.out.println(name);
        }
    }
}
```

Таким образом, packages помогают организовать код, а import позволяет получить доступ к классам и ресурсам из определенных пакетов в вашем коде.

3. Как следует называть имена классов и методов? Где можно найти документацию?

Следование общепринятым соглашениям по именованию классов и методов помогает сделать ваш код более читаемым и понятным для других разработчиков. Вот некоторые общие рекомендации по именованию классов и методов в Java:

Имена классов:

Имя класса должно начинаться с заглавной буквы и использовать "CamelCase" (также известный как "PascalCase") для разделения слов, например: MyClass, EmployeeDetails, StudentRecord.
Имя класса должно быть существительным и лучше всего описывать сущность или объект, которым класс представляет.
Имена методов:

Имя метода должно начинаться с маленькой буквы и также использовать "camelCase" для разделения слов, например: calculateSalary, getEmployeeName, processData.
Имя метода должно быть глаголом или глагольной фразой, которая описывает выполняемое действие.

Примеры:

```
public class Employee {

    public void calculateSalary(int hoursWorked, double hourlyRate) {

    }
}
```

Документация:
Для более подробной информации о стандартах именования в Java и других рекомендациях по стилю кода, можно обратиться к официальным руководствам по кодированию:

Oracle Code Conventions for the Java Programming Language:
https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html

Google Java Style Guide:
https://google.github.io/styleguide/javaguide.html

Code Conventions for the Java Programming Language (by Sun Microsystems):
https://www.oracle.com/technetwork/java/codeconvtoc-136057.html

IntelliJ IDEA Code Style for Java (от JetBrains):
https://www.jetbrains.com/help/idea/code-style-java.html

Eclipse Code Conventions (от Eclipse Foundation):
https://www.eclipse.org/projects/project.php?id=eclipse

4. Есть ли ограничение по количеству конструкторов в классе?

В языке Java нет жестких ограничений на количество конструкторов в классе. Мы можем определить столько конструкторов, сколько нам необходимо, но существуют некоторые правила, которые следует учитывать:

Уникальность параметров: Каждый конструктор должен иметь уникальный список параметров. Java не позволяет определить несколько конструкторов с одинаковыми типами и порядком параметров.

Конструкторы по умолчанию: Если мы не определяем ни одного конструктора, Java автоматически предоставит "конструктор по умолчанию" (без параметров). Он создается автоматически только в том случае, если у класса нет ни одного конструктора.

Перегрузка конструкторов: В Java допускается перегрузка конструкторов, т.е. определение нескольких конструкторов с различными списками параметров. Это позволяет создавать объекты с различными наборами значений аргументов.

Примеры:

```
public class MyClass {

    public MyClass() {

    }

    public MyClass(int value) {

    }

    public MyClass(int value, String name) {

    }
}
```

Таким образом, количество конструкторов в классе ограничивается только нашими потребностями и логикой приложения. Мы можем создавать конструкторы для различных сценариев создания объектов в нашем классе.

5. Что будет, если не указать ни одного конструктора в классе?

Если не указать ни одного конструктора в классе, то Java автоматически создаст для этого класса конструктор по умолчанию (default constructor). Конструктор по умолчанию не принимает аргументов и выполняет пустое тело. Если в классе не определены другие конструкторы, то этот конструктор по умолчанию позволит создавать объекты этого класса без передачи аргументов.

Пример конструктора по умолчанию:

```
public class MyClass {

    public MyClass() {

    }


}
```

Следует знать, что как только мы явно определим хотя бы один конструктор с аргументами, конструктор по умолчанию больше не будет автоматически создаваться.

6. Для чего нужен this?

В Java ключевое слово this используется для ссылки на текущий объект (экземпляр класса), внутри которого оно вызывается. Оно позволяет различать локальные переменные и поля класса с одинаковыми именами, а также передавать текущий объект в другие методы.

this имеет следующие основные применения:

Различие между полями класса и локальными переменными: Если у нас есть переменные с одинаковым именем в классе и в методе, используя this, можно явно указать, что нужно обратиться к полю класса, а не к локальной переменной.
Пример:

```
public class MyClass {
    private int number;

    public void setNumber(int number) {
        this.number = number;
    }
}
```

Передача текущего объекта в другие методы: Когда мы вызываем метод изнутри объекта, мы можем передать сам объект в качестве аргумента в другие методы.
Пример:

```
public class MyClass {
    private int value;

    public void someMethod() {
        anotherMethod(this);
    }

    public void anotherMethod(MyClass obj) {

        obj.setValue(42);
    }

    public void setValue(int value) {
        this.value = value;
    }
}
```

В этом примере this используется для передачи текущего объекта MyClass в метод anotherMethod(), где его можно использовать для установки значения поля value.

Использование this помогает уточнить, на какой объект мы ссылаемся, и избежать путаницы между полями класса и локальными переменными.

7. Что такое инкапсуляция и какие способы его достижения существуют в java?

Инкапсуляция - это один из четырех основных принципов объектно-ориентированного программирования (ООП) и представляет собой механизм, позволяющий объединить данные (переменные) и методы, работающие с этими данными, в единый объект. Основная идея инкапсуляции заключается в том, что данные объекта должны быть скрыты от внешнего доступа, и можно получать к ним доступ только через определенные методы (геттеры и сеттеры), что обеспечивает контроль доступа к данным и защищает их от неправильного использования или изменения.

В Java существуют следующие способы достижения инкапсуляции:

Использование модификаторов доступа: В Java есть модификаторы доступа, такие как private, protected, public и отсутствие модификатора (package-private). С помощью этих модификаторов можно управлять видимостью данных и методов класса для других классов. Используя private, можно ограничить доступ к данным только внутри самого класса. protected дает доступ к данным класса внутри класса и его подклассов. public позволяет получить доступ к данным из любого места в программе. Если не указать модификатор доступа, применяется уровень доступа по умолчанию, когда данные доступны только в пределах пакета.

Геттеры и сеттеры (accessor и mutator methods): Геттеры предоставляют доступ к приватным данным из других классов, а сеттеры позволяют изменять значения приватных данных из других классов. Это позволяет контролировать доступ к данным и предоставить возможность проверки или обработки значений перед их установкой или получением.

Пример:

```
public class MyClass {
    private int myData;

    public int getMyData() {
        return myData;
    }

    public void setMyData(int value) {
        myData = value;
    }
}
```

Классы без методов, содержащие только приватные поля: Это практика, при которой класс может содержать только приватные поля без методов, позволяющая управлять данными и обеспечивая доступ к ним только через геттеры и сеттеры.

Использование интерфейсов: Интерфейсы позволяют определить абстрактные методы без реализации. Классы могут реализовывать интерфейсы и предоставлять свою собственную реализацию методов. Такой подход также способствует инкапсуляции, поскольку классы могут скрывать детали своей реализации и предоставлять только общий интерфейс.

Правильное использование инкапсуляции помогает создавать более надежные и поддерживаемые программы, а также облегчает изменения внутренней реализации класса, не затрагивая код, который использует этот класс.

8. Какие модификаторы доступа существуют? коротко опишите каждый из них

В Java существуют четыре модификатора доступа:

public: Полный доступ из любой части программы. Используется для предоставления общего интерфейса, доступного для всех.

protected: Доступен внутри своего пакета и в подклассах. Используется для предоставления доступа к методам и полям внутри иерархии наследования.

default: Доступен только внутри своего пакета. Используется для ограничения видимости в пределах пакета.

private: Доступен только внутри своего класса. Используется для скрытия деталей реализации и предоставления контролируемого доступа к данным и методам класса.

9. Для чего нужны геттеры и сеттеры?

Геттеры (get-методы) и сеттеры (set-методы) являются часто используемым паттерном в объектно-ориентированном программировании и служат для обеспечения инкапсуляции данных в классах. Они предоставляют публичные методы для доступа и установки значений приватных полей объекта.

Геттеры (get-методы):

Используются для получения значений приватных полей объекта.
Обычно именуются в формате getИмяПоля(), например, getName(), getAge().
Позволяют получить доступ к значениям полей, не изменяя их.
Сеттеры (set-методы):

Используются для установки значений приватных полей объекта.
Обычно именуются в формате setИмяПоля(), например, setName(), setAge().
Позволяют изменить значения полей объекта, обеспечивая контроль над изменениями.
Использование геттеров и сеттеров вместо напрямую обращения к полям класса имеет несколько преимуществ:

Инкапсуляция: Геттеры и сеттеры позволяют скрыть внутреннюю реализацию класса и обеспечить контролируемый доступ к полям. Это упрощает изменение внутренней структуры класса, не затрагивая внешний код, который использует эти методы.

Контроль доступа: Геттеры и сеттеры позволяют установить ограничения на доступ к полям. Например, в сеттере можно добавить проверки на допустимость значения, что обеспечивает более надежную работу с данными.

Код-безопасность: Используя геттеры и сеттеры, можно управлять доступом к данным, что помогает избежать ошибок и неправильного использования данных в других частях программы.

Изменение поведения: При необходимости изменить логику доступа к полям (например, добавить логирование, кэширование и т.д.), это можно сделать внутри геттеров и сеттеров, не затрагивая остальной код, использующий класс.