**Ответы на вопросы со слайда**

1. ООП – объектно-ориентированное программирование. Суть его заключается в создании классов и взаимодействия их экземпляров(объектов). Этот метод программирования позволяет структурировать программный код и создавать иерархию наследования.
2. ООП было создано для развития процедурного программирования, где программа разделена на функции. Но т.к. они не были связаны, а связь и их -взаимодействие были нужны, в последствии данный подход был изобретён.
3. Класс – это функции, типы характеристики какой-либо структуры. По которой можно сделать экземпляр, который будет иметь определённые характеристики, и выполнять созданные для него функции.
4. Экземпляром класса называется объект, который имеет определённые характеристики класса и поведение.
5. Свойства класса - это отдельные его атрибуты, которые могут быть определены сеттерами или конструктором класса. После чего объект будет иметь свои индивидуальные характеристики.
6. Методы класса - это внутриклассовые функции, которые могут выполнять определённый алгоритм для изменения характеристик объекта класса или для их отображения.
7. Модификаторы доступа – это ключевые слова, задающие доступ для классов, методов, переменных при наследовании или взаимодействии классов и их объектов
8. Public – общедоступный
9. Private – доступен только внутри самого класса, т.е. только в том классе в котором он присутствует
10. Protected- доступен внутри самого класса, а также может быть доступен в дочерних классах.
11. Интерфейс – это структура, помогающая объединить методы и правила взаимодействия элементов системы, другими словами интерфейсы позволяют взаимодействовать с объектом, класс которого наследует интерфейс с функциями для взаимодействия с классом.
12. Абстрактный класс – это класс в котором не реализован ни один метод, но они существуют, как и характеристики и конструкторы класса. Для того чтобы в отдельных реализациях класса при наследовании можно было использовать разные алгоритмы.
13. Статический метод – это метод, который существует независимо от того, создан ли какой-либо экземпляр класса. И взаимодействуют только со статическими переменными класса.
14. 3 основных принципа в ООП.
15. Инкапсуляция – это принцип в ООП, который помогает скрыть данные от внешнего воздействия, или наоборот дать доступ для взаимодействия.
16. Наследование - это принцип в ООП, который позволяет родительскими классами быть основой для дочерних, т.е. дочерние классы при наследовании перенимают свойства и поведение своего родителя
17. Полиморфизм – это принцип в ООП, который позволяет одному и тому же действию (функции, методу) выполняться разными способами. Т.е. при наследовании трёх классов от одного старшего, можно в трёх же этих классах реализовать алгоритм конкретной функции по-разному.

**Сравнение ООП в Python и Java**

|  |  |  |
| --- | --- | --- |
|  | Python | Java |
| Создание класса | В Python можно объявить класс где угодно и когда угодно | Имя исходного Java-файла должно соответствовать имени хранящегося в нем класса. Каждый Java-файл может содержать только один публичный класс. |
| Объявление и инициализация | Нужно объявлять и определять атрибуты внутри метода класса **init**(), который является аналогом конструктора в Java. Указывая перед именем переменных ключевое слово self, означает, что это атрибуты. Каждый экземпляр класса получает свою копию. Все переменные в Python не типизированы, и атрибуты не являются исключением.  Обращение к переменной класса происходит по следующему синтаксису:  Имя файла, содержащего класс (без расширения .py)  Точка  Имя класса  Точка  Имя переменной | В Java нужно объявлять атрибуты (с указанием их типа) внутри класса, но за пределами всех методов. Перед тем, как использовать атрибуты класса, нужно их определить. |
| Публичные и приватные | В Python всё – публичное  Вместо приватных переменных в Python имеется понятие непубличных (non-public) переменных экземпляра класса. Все переменные, названия которых начинаются с одинарного подчеркивания, считаются непубличными. Это соглашение об именах нисколько не мешает нам обратиться к переменной напрямую.  Для того, чтобы скрыть атрибут, используется двойное подчеркивание в начале названия переменной. Когда Python видит такую переменную, он автоматически меняет ее название, чтобы затруднить к ней прямой доступ. Однако, этот механизм всё равно не мешает нам обратиться к ней. | Java управляет доступом к методам и атрибутам, различая публичные и приватные данные.  Таким образом ограничивается доступ к ним извне. Чтобы предоставить доступ к приватным атрибутам, нужно объявлять публичные методы, которые устанавливают или получают эти данные |
| Управление доступом | Можно получить доступ к атрибутам напрямую. Поскольку всё – публичное, можно достучаться к чему угодно, когда угодно и откуда угодно. Мы можем получать и устанавливать значения атрибутов напрямую, обращаясь по их имени. В Python возможно даже удалять атрибуты, что немыслимо в Java  Однако бывает и так, что нужно контролировать доступ к атрибутам. В таком случае нам на помощь приходят Python-свойства (properties).  В Python свойства обеспечивают управляемый доступ к атрибутам класса при помощи декораторов (decorators). Используя свойства, объявляются функции в питоновских классах подобно геттерам и сеттерам в Java (бонусом идет удаление атрибутов). | В Java получается доступ к приватным атрибутам при помощи сеттеров (setters) и геттеров (getters) |
| self и this | Ключевое слово self служит аналогичной цели: обращение к членам-атрибутам, но в отличие от Java, оно обязательно.  Каждый self либо создает, либо обращается к атрибуту. Если пропустить его, то Python просто создаст локальную переменную вместо атрибута | Класс ссылается сам на себя, используя ключевое слово this. this подразумевается в Java-коде. Его в принципе даже необязательно писать, кроме случаев, когда имена переменных совпадают |
| Методы и функции | Можно вызывать функции из любого места видимости. Если функция не содержит ссылки на self, что означает, что это глобальная функция, а не функция класса. Она не сможет изменять или сохранять какие-нибудь данные какого-либо класса, но может использовать локальные и глобальные переменные. | Каждая написанная строчка на Java принадлежит какому-нибудь классу. Функции не существуют за пределами класса, и по определению все Java-функции — это методы. На Java ближе всего к чистой функции находится статичный метод |
| Наследование | Поддерживает множественное наследование, то есть создание класса более чем от одного родителя | Поддерживает только одиночное наследование, что означает, что классы в Java могут наследовать данные и поведение только от одного родительского класса. Зато в Java возможно наследование от множества интерфейсов. Интерфейсы обеспечивают группу связанных методов, которые нужно реализовать, позволяя дочерним классам вести себя сходным образом. |
| Типы данных и полиморфизм | Вместо идентификации объектов по типу, Python проверяет их поведение. | Каждый класс и каждый интерфейс в Java имеет тип. Следовательно, если два Java-объекта реализуют один и тот же интерфейс, считается, что они имеют один и тот же тип по отношению к этому интерфейсу. С помощью этого механизма можно взаимозаменяемо использовать различные классы, в чем и заключается полиморфизм. |
| Дефолтные методы | В Python подобный функционал обеспечивается набором так называемых магических методов. Каждый Python-класс наследует эти методы, и можно, переопределив их, изменить их поведение.    В Python для строкового представления объекта имеется два метода: repr() и str(). Однозначное представление объекта возвращается методом repr(), в то время как str() возвращает его в удобочитаемом виде. Это примерно как hashcode() и toString() в Java. | Все классы в Java имеют своим предком класс Object, который содержит определенный набор методов и передает их своим потомкам. Потомки могут эти методы либо переопределять, либо использовать по умолчанию. toString() возвращает строковое представление объекта. По умолчанию это имя класса и адрес в памяти. Этот метод вызывается автоматически, когда объект передается в качестве параметра в метод, требующий строковый аргумент, например, System.out.println(): |