Вопросы для подготовки:

- Что такое сериализация?

Сериализация — это преобразование объекта или дерева объектов в какой-либо формат с тем, чтобы потом эти объекты можно было восстановить из этого формата. Используется, например, для сохранения состояния программы (то есть, некоторых её объектов) между запусками. Или для передачи данных между различными экземплярами программы (или различными программами), например, по сети.

Главная идея состоит в том, что сериализованный формат — набор байт или строка, которую можно легко сохранить на диск или передать другому процессу или, например, по сети, в отличие от самого объекта. А значит, задача сохранения объекта/группы объектов при этом сводится к простой задаче сохранения набора байт или строки.

JSON — один из популярных форматов для сериализации, он текстовый, легковесный и легко читается человеком.

- Что из себя представляет процесс сериализации/десериализации с использованием интерфейса Serializable?

В Java за процессы сериализации отвечает интерфейс Serializable. Этот интерфейс крайне прост: чтобы им пользоваться, не нужно реализовывать ни одного метода!

Интерфейс, у которого нет ни одного метода, выглядит странно :/ Зачем он нужен? Ответ на этот вопрос есть выше: только чтобы предоставить нужную информацию Java-машине.

В одной из прошлых лекций мы мельком упоминали интерфейсы-маркеры. Это специальные информативные интерфейсы, которые просто помечают наши классы дополнительной информацией, в будущем полезной для Java-машины. Никаких методов, которые нужно было бы имплементировать, у них нет. Так вот, Serializable — один из таких интерфейсов.

Еще один важный момент: переменная private static final long serialVersionUID, которую мы определили в классе. Зачем она нужна?

Это поле содержит уникальный идентификатор версии сериализованного класса.

При сериализации объекта сериализуются все объекты, на которые он ссылается в своих переменных экземпляра. И если те объекты тоже ссылаются на третьи объекты, они тоже сериализуются. И так до бесконечности. Все классы в этой цепочке должны быть Serializable, иначе их невозможно будет сериализовать и будет выброшено исключение.

Проблемы такого рода решаются в Java при помощи ключевого слова transient. Если добавить к полю класса это ключевое слово — значение этого поля не будет сериализовано.

- Как изменить стандартное поведение сериализации/десериализации?

Реализовать интерфейс java.io.Externalizable, который позволяет применение пользовательской логики сериализации. Способ сериализации и десериализации описывается в методах writeExternal() и readExternal(). Во время десериализации вызывается конструктор без параметров, а потом уже на созданном объекте вызывается метод readExternal.

Если у сериализуемого объекта реализован один из следующих методов, то механизм сериализации будет использовать его, а не метод по умолчанию :

writeObject() - запись объекта в поток;

readObject() - чтение объекта из потока;

writeReplace() - позволяет заменить себя экземпляром другого класса перед записью;

readResolve() - позволяет заменить на себя другой объект после чтения.

- Как исключить поля из сериализации?

Transient

- Какое влияние оказывают на сериализуемость модификаторы полей static и final?

При стандартной сериализации поля, имеющие модификатор static, не сериализуются. Соответственно, после десериализации это поле значения не меняет. При использовании реализации Externalizable сериализовать и десериализовать статическое поле можно, но не рекомендуется этого делать, т.к. это может сопровождаться трудноуловимыми ошибками.

Поля с модификатором final сериализуются как и обычные. За одним исключением – их невозможно десериализовать при использовании Externalizable, поскольку final поля должны быть инициализированы в конструкторе, а после этого в readExternal() изменить значение этого поля будет невозможно. Соответственно, если необходимо сериализовать объект с final полем необходимо использовать только стандартную сериализацию.

- Зачем нужно поле serialVersionUID в сериализации?

serialVersionUID используется для указания версии сериализованных данных.

Когда мы не объявляем serialVersionUID в нашем классе явно, среда выполнения Java делает это за нас, но этот процесс чувствителен ко многим метаданным класса включая количество полей, тип полей, модификаторы доступа полей, интерфейсов, которые реализованы в классе и пр.

Рекомендуется явно объявлять serialVersionUID т.к. при добавлении, удалении атрибутов класса динамически сгенерированное значение может измениться и в момент выполнения будет выброшено исключение InvalidClassException.

- Когда стоит изменять значение поля serialVersionUID?

serialVersionUID нужно изменять при внесении в класс несовместимых изменений, например при удалении какого-либо его атрибута.

- В чем особенность сериализации Singleton'ов?

Проблема в том что после десериализации мы получим другой объект. Таким образом, сериализация дает возможность создать Singleton еще раз, что недопустимо.

уществует два способа избежать этого:

1)явный запрет сериализации.

2) определение метода с сигнатурой (default/public/private/protected/) Object readResolve() throws ObjectStreamException, назначением которого станет возврат замещающего объекта вместо объекта, на котором он вызван.