Вопросы по теме Клонирование объектов:

1. Расскажите про клонирование объектов.
2. Какой способ клонирования предпочтительней?
3. Что такое сериализация?
4. Как сделать Java-класс сериализуемым?
5. В чем разница между интерфейсом Serializable и Externalizable в Java?
6. Что такое serialVersionUID? Что произойдет, если вы его не определите?
7. Какие методы используются в процессе сериализации и десериализации в Java?
8. При сериализации вы хотите, чтобы некоторые члены не сериализовались? Как этого добиться?
9. Что произойдет, если один из членов класса не реализует интерфейс Serializable?
10. Новый класс реализует интерфейс Serializable и наследуется от суперкласса, который его не реализует. В суперклассе есть переменные, которые наследует новый класс. Какое будет состояние у этих переменных после десериализации?
11. Можете ли вы настроить процесс сериализации или переопределить процесс сериализации по умолчанию в Java?
12. Новый класс не реализует интерфейс Serializable и наследуется от суперкласса, который его реализует. Как не допустить сериализации нового класса?
13. У вас есть класс, который вы сериализовали и сохранили, а затем изменили этот класс. Что произойдет, если вы десериализуете такой объект?

Клонирование объектов.

Существует три способа клонирования объектов:

1. переопределение метода clone() и реализация интерфейса Cloneable();
2. использование конструктора копирования;
3. использование механизма сериализации.

Метод clone().

При использовании метода clone() будет создана копия объекта, у которого вызвали этот метод. Копии будут присвоены значения полей объекта-образца. Если копируемый объект содержит ссылки на другие объекты, то эти ссылки будут скопированы, но дубликаты тех объектов не создадутся.

Существует нюанс при клонировании final полей. Если значение поля является примитивным или неизменяемым объектом, то проблем не возникнет. Но если значение поле изменяемый объект, то возникнет ошибка компилятора. Единственный выход при такой ситуации – отказаться от поля final (удалить этот модификатор).

Интерфейс Cloneable – маркерный интерфейс, не содержащий методов, предназначенный для того, чтобы указать Java-машине что класс, реализующий данный интерфейс, можно клонировать с помощью метода clone(). При вызове метода clone() Java проверяет, реализует ли класс интерфейс Cloneable и если не реализует, то выкидывает исключение CloneNotSupportedException.

Порядок использования «стандартной» технологии клонирования:

1. добавить интерфейс Cloneable;
2. переопределить метод clone(), вызвав стандартную реализацию или написав свою собственную реализацию.

Метод clone() нужно переопределять в любом случае, т.к. в классе Object он объявлен как

protected native Object clone() throws CloneNotSupportedException;

так что он доступен для вызова только классам из его пакета (java.lang.*) или классам-наследникам.

Конструктор копирования.

Конструктор, который принимает копируемый объект и создает на его основе копию. Этот метод клонирования предпочтительней, чем метод clone(). Он имеет ряд преимуществ:

1. полный контроль над процессом клонирования (поля указываются явно);
2. отсутствие ошибок наследования (не нужен интерфейс Cloneable);
3. можно клонировать final поля.

Сериализация.

Сериализация – процесс преобразования объекта в двоичный код с целью его дальнейшей передачи либо для сохранения текущего состояния объекта.

Десериализация – процесс восстановления объекта из двоичного кода.

Чтобы сделать класс сериализуемым необходимо добавить интерфейс Serializable. Это маркерный интерфейс. Для сериализации объекта нужно, чтобы все его поля поддерживали сериализацию, иначе будет брошено исключение NotSerializableException. Поля, помеченные ключевым словом transient и static поля не сериализуются.

Во время сериализации, на основе содержимого класса (поля, методы, порядок их объявления), неявно генерируется уникальный идентификатор версии сериализованного класса - private static final long **serialVersionUID** поле, которое предназначено для контроля версии объекта. Когда мы пытаемся провести десериализацию, значение serialVersionUID сравнивается со значением serialVersionUID класса в нашей программе. Если значения не совпадают, будет выброшено исключение InvalidClassException. Данное поле можно задать явно, указав нужное значение.

Сериализация выполняется с помощью класса **ObjectOutputStream** методом **writeObject**(). Десериализация выполняется с помощью класса **ObjectInputStream** методом **readObject**().

Интерфейс Externalizable – позволяет реализующему классу использовать механизм сериализации, был создан как аналог интерфейсу Serializable, в котором устранялись проблемы с производительностью (в ранних версиях Java (Java1.1, Java1.2) сериализация при большом количестве зависимостей была очень медленной). В современных версиях Java проблемы с производительностью устранены. Таким образом преимущества в использовании данного интерфейса нету, даже наоборот, интерфейс Externalizable налаживает ряд обязанностей и ограничений.

Особенности использования:

1. класс обязан иметь конструктор по умолчанию;
2. все классы-наследники, происходящие от класса, реализующего Externalizable, должны иметь конструкторы по умолчанию;
3. нужно реализовать два обязательных метода writeExternal() и readExternal();
4. static поля можно сериализовать, но не рекомендуется (можно словить ошибку, которую потом тяжело найти);
5. нельзя десериализовать final-переменные (эти поля инициализируются при вызове конструктора по умолчанию и после этого невозможно изменить их значение).

Вопрос: У вас есть класс, который вы сериализовали и сохранили, а затем изменили этот класс. Что произойдет, если вы десериализуете такой объект?

Ответ: Если было только добавление полей, то никаких проблем не возникнет. При десериализации такие поля будут игнорироваться. Если было изменение или удаление, то десериализация будет новозможной.

Вопрос: Можете ли вы настроить процесс сериализации или переопределить процесс сериализации по умолчанию в Java?

Ответ: Да, можно настроить сериализацию переопределив методы writeObject() и readObject().

Вопрос: Новый класс не реализует интерфейс Serializable и наследуется от суперкласса, который его реализует. Как не допустить сериализации нового класса?

Ответ: Если суперкласс реализует интерфейс Serializable, то и классу-наследнику этот интерфейс передастся, т.е. класс-наследник будет иметь возможность использовать механизм сериализации. Чтобы запретить сериализацию нового класса нужно переопределить методы writeObject() и readObject() принудительно бросив в них исключение NotSerializableException. Тогда при попытке сериализации будет выброшено данное исключение.

```
private void writeObject(ObjectOutputStream o) throws NotSerializableException {
    throw new NotSerializableException();   }
```

Вопрос: Новый класс реализует интерфейс Serializable и наследуется от суперкласса, который его не реализует. В суперклассе есть переменные, которые наследует новый класс. Что произойдет с данными переменными?

Ответ: Эти переменные будут функционировать так же, как если бы суперкласс реализовывал интерфейс Serializable.

Вопрос: Какой способ клонирования лучше? Метод clone(), конструктор копирования или сериализация?

Ответ: Все зависит от ситуации. Если нужно поверхностное копирование, без клонирования ссылочных переменных, то лучше использовать конструктор копирования. Если нужно глубокое копирование (все ссылочные переменные, вся иерархия наседования), или нам нужно передать состояние объекта за пределы программы, или сохранить состояние объекта, то лучше выбрать сериализацию.

Вопрос: В чем проблема сериализации паттерна Singleton?

Ответ: После десериализации мы получим другой объект (а объект должен быть таким же, по ссылке и по содержанию, как и до сериализации). Таким образом, сериализация дает возможность создать Singleton еще раз. Выход:

использовать шаблон enum singleton

```
public enum MySingleton {
    INSTANCE;
}
```

или определить в классе метод

```
private Object readResolve() throws ObjectStreamException {
    return getInstance();
}
```