**1 Задание**

Сериализация – запись объекта в поток

Десериализация – чтение объекта из потока При сериализации и десериализации мы выбираем, какие данные объекта надо записать и какие надо считать

Сериализация означает преобразование объекта в поток байтов (обычно в формат, который можно сохранить или передать по сети), так что он может быть восстановлен позже. Это полезно, когда вам нужно сохранить состояние объекта или передать его через сеть или сохранить в файл.

Десериализация - это процесс восстановления объекта из потока байтов. Это означает, что данные, которые были сериализованы, могут быть прочитаны и преобразованы обратно в объекты в памяти.

**2 Задание**

1. Открытие потока: Сначала необходимо открыть поток для чтения или записи данных. Это может быть файловый поток, поток сокета, поток ввода-вывода в памяти и так далее. В зависимости от языка программирования и среды выполнения это может быть реализовано разными способами.

2. Чтение или запись данных: После открытия потока можно читать данные из него или записывать данные в него в соответствии с вашими потребностями. Например, при чтении файлов это могут быть последовательные чтения байтов или чтение строк из файла. При записи файлов это может быть запись байтов или строк в файл.

3. Обработка данных: После чтения данных или записи данных обычно выполняется некоторая обработка данных в соответствии с логикой вашей программы. Это может включать в себя анализ и преобразование данных, фильтрацию, сортировку или любую другую манипуляцию данных.

4. Закрытие потока: После того как все операции с потоком завершены, важно закрыть его, чтобы освободить ресурсы и убедиться, что все данные правильно сохранены. Это особенно важно для файловых потоков и других ресурсоемких операций ввода-вывода.

5. Обработка исключений (при необходимости): Во время работы с потоками могут возникать ошибки, такие как ошибки ввода-вывода или проблемы с доступом к ресурсам. Поэтому важно предусмотреть обработку исключений для обработки этих ситуаций и выполнения соответствующих действий для восстановления или уведомления об ошибке.

Поток работает 1 (один) раз! • Создан, но не запущен • Запущен (isAlive - true) • Приостановлен • например, во время методов wait, sleep, join • Закончил работу (isAlive – false)

**3 Задание**

В языке Java различают два вида потоков:

байтовые потоки. Это аналог потоков двоичных данных, позволяющих компактно сохранять информацию;

символьные потоки. Это потоки, которые представлены удобным способом (для людей) кодирования информации в виде понятных текстовых символов. Во многих языках программирования символьные потоки ассоциируются с текстовым форматом представления информации.

**4 Задание** IO (ввод-вывод) и NIO (новый ввод-вывод) - это два различных подхода к обработке ввода-вывода в Java. Вот их основные отличия:

1. IO (ввод-вывод):

- Это традиционный, блокирующий подход к вводу-выводу в Java, в основе которого лежат классы, такие как InputStream и OutputStream, а также Reader и Writer.

- При использовании IO поток блокируется во время операции ввода-вывода, что означает, что программа приостанавливает свою работу, пока операция не будет завершена.

- Каждая операция ввода-вывода обрабатывается отдельным потоком, что может привести к низкой эффективности в системах с большим числом одновременных операций ввода-вывода.

2. NIO (новый ввод-вывод):

- NIO был введен в Java 1.4 и представляет собой альтернативный, неблокирующий подход к вводу-выводу.

- Основными классами в NIO являются ByteBuffer, Channel и Selector.

- В NIO операции ввода-вывода могут быть неблокирующими, что позволяет программе продолжать работу во время ожидания завершения операции ввода-вывода.

- Селекторы (Selector) позволяют одному потоку управлять несколькими каналами ввода-вывода, что может привести к более эффективному использованию ресурсов в системах с большим числом одновременных операций ввода-вывода.

В целом, NIO часто предпочтительнее, чем традиционный IO, в ситуациях, когда требуется обрабатывать много операций ввода-вывода одновременно, или когда требуется высокая производительность ввода-вывода. Однако использование NIO может быть сложнее для понимания и реализации, чем традиционный IO.