1. Сериализация объекта - возможность “сохранить” (записать) текущее состояние объекта в байты. Десериализация - перевод из байтов обратно в объект. Важно! Объект, который может сериализоваться должен быть помечен маркерным интерфейсом (который не требует реализации никаких методов) **Serializable**!
2. Создание класса, наследующего **Thread**: Для начала, нужно создать новый класс, который наследуется от **java.lang.Thread**. В этом классе необходимо переопределить метод run(), который будет содержать логику, которую вы хотите выполнить в отдельном потоке. Либо имплементировать интерфейс **Runnable** и реализовать метод **run**() - этот способ более удобный тк в случае наследования **Thread** класс будет ограничен от остальных наследованный из-за отсутствия множественного наследования
3. Запуск потока: После создания класса потока, нужно создать экземпляр этого класса и вызвать метод **start**(). Метод запускает поток, вызывая метод **run**() в новом потоке исполнения.

3) В Java существуют два основных типа потоков: пользовательские и демоны. Пользовательские потоки создаются для выполнения конкретных задач и продолжают работу до тех пор, пока их задача не будет выполнена или пока приложение не прервет их работу. Они используются для задач, связанных с пользовательским интерфейсом или требующих активного взаимодействия с пользователем. Демоны - служат поддержкой пользовательским потокам, выполняя задачи в фоновом режиме. Они не мешают завершению JVM, даже если продолжают работать, и обычно заняты вспомогательными задачами, например, управлением памятью, не удерживая программу от завершения.

4) Java IO ориентируется на потоки, что подразумевает чтение и запись данных по одному байту за раз, в то время как Java NIO предпочитает работу с буферами, что позволяет более гибко и быстро обрабатывать данные. Хотя Java IO поддерживает многопоточность, она использует блокирующий режим, где каждый поток должен дождаться завершения операции ввода-вывода, что может привести к простою при высокой нагрузке. Java NIO, с другой стороны, предлагает более продвинутые возможности для работы с многопоточностью благодаря поддержке неблокирующего ввода-вывода и асинхронных операций, что позволяет нескольким потокам одновременно обрабатывать данные без ожидания завершения каждой операции ввода-вывода. Это делает NIO более эффективной для сетевых приложений и приложений, работающих с большими объемами данных, благодаря улучшенной производительности и эффективности использования системных ресурсов.