1. ***InputStream и OutputStream:***

Эти потоки работают с байтами и предназначены для чтения и записи двоичных данных соответственно.

2. ***Reader и Writer:***

Эти потоки работают с символами и предназначены для чтения и записи текстовых данных. InputStream лучше использовать для двоичных файлов (например, изображений, музыки), а Reader — для текстовых файлов.

3. ***ObjectOutputStream:***

С помощью ObjectOutputStream мы можем записать объекты в файл, при условии, что они реализуют интерфейс Serializable.

4. ***BufferedInputStream/BufferedOutputStream:***

Буферизированные потоки увеличивают производительность путем сокращения числа обращений к физическому носителю данных за счет использования временного буфера.

5. ***Метод close():***

Этот метод необходимо вызывать для закрытия потоков и освобождения системных ресурсов, которые потоки занимали во время работы.

6. ***Try-with-resources:***

Это конструкция Java, которая гарантирует закрытие ресурсов (которые реализуют интерфейс AutoCloseable) после использования. Она предпочтительна, так как автоматизирует управление ресурсами и предотвращает утечки памяти.

7. ***Mark/reset:***

Метод mark() позволяет поставить метку в текущей позиции в потоке, а reset() возвращает поток к этой метке. Эти методы не поддерживаются всеми типами потоков.

8. ***InputStream в Reader:***

Да, InputStream можно преобразовать в Reader с помощью InputStreamReader, который превращает байтовый поток в символьный.

9. ***Считывание строки из файла:***

В этом случае лучше использовать Reader, например BufferedReader в сочетании с FileReader, так как они оптимизированы для чтения текста.

10. ***Запись строки в файл:***

Для записи строки в файл лучше использовать Writer, например BufferedWriter в сочетании с FileWriter, так как они обеспечивают эффективную запись текста.

11. ***Программа для работы с результатами игры:***

В приложении..