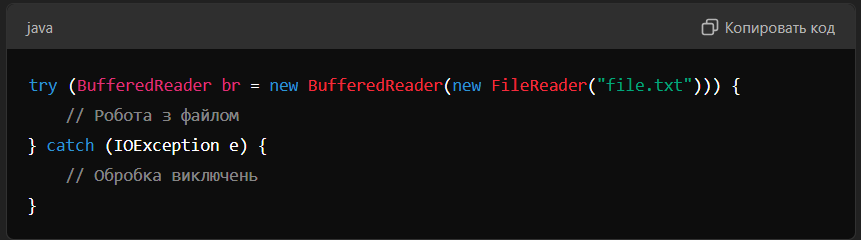
1. **Що таке потоки вводу-виводу?**

Потоки вводу-виводу (I/O streams) — це абстракції, які дозволяють програмам читати дані з джерел (ввід) і записувати дані до приймачів (вивід). Вони забезпечують послідовний доступ до даних, незалежно від їх фізичного розташування (файли, мережеві з'єднання, пам'ять тощо).

1. **Що таке блокуючий і неблокуючий ввід-вивід?**
   * **Блокуючий ввід-вивід**: Операції вводу-виводу, які призупиняють виконання програми до завершення операції. Наприклад, якщо програма читає дані з мережі, вона буде чекати, поки дані не стануть доступними.
   * **Неблокуючий ввід-вивід**: Операції, які дозволяють програмі продовжувати виконання, навіть якщо операція вводу-виводу ще не завершена. Це досягається через асинхронне програмування або використання багатопотоковості.
2. **Чим InputStream відрізняється від Reader?**
   * **InputStream**: Клас, призначений для читання байтових даних (8-бітових). Використовується для роботи з двійковими файлами, зображеннями, аудіо тощо.
   * **Reader**: Абстрактний клас для читання символьних даних (16-бітових Unicode символів). Підходить для роботи з текстовими файлами, де важливе кодування символів.
3. **Якщо потрібно обробити велику кількість байтів у файлі, як це краще всього робити з точки зору швидкодії? Як це краще НЕ робити?**
   * **Рекомендовано**: Використовувати буферизовані потоки, такі як BufferedInputStream або BufferedReader. Вони зменшують кількість фізичних доступів до диска, зберігаючи дані у внутрішньому буфері.
   * **Не рекомендується**: Читати файл по одному байту або символу без використання буферизації. Це значно сповільнює роботу програми через часті звернення до диска.
4. **Як працює конструкція try-with-resources? Яким чином слід використовувати цю конструкцію при роботі з файлами?**

Конструкція try-with-resources автоматично закриває ресурси після завершення блоку try. Це забезпечує коректне звільнення ресурсів і запобігає витокам пам'яті.

**Приклад використання з файлом:**

У цьому прикладі BufferedReader буде автоматично закритий після завершення блоку try, навіть якщо виникне виключення.

1. **Для чого потрібні методи flush() та close()?**
   * **flush()**: Очищує буфер потоку, примусово записуючи всі дані з буфера до цільового пристрою або файлу. Корисно, коли потрібно забезпечити, що всі дані були записані до закриття потоку.
   * **close()**: Закриває потік і звільняє всі пов'язані з ним системні ресурси. Після виклику цього методу подальші операції з потоком можуть викликати виключення.