Recordando POO:

→ Streams

Uma stream é uma abstracção que representa uma fonte genérica de entrada de dados ou um destino genérico para escrita de dados que é definida independentemente do dispositivo físico concreto. Todas as classes que implementam streams em Java são subclasses das classes abstractas

InputStream e **OutputStream** para streams de bytes

e das classes abstractas

Reader e **Writer** para streams de caracteres (texto).

→ Streams de caracteres

As subclasses de Writer têm que implementar os métodos definidos nesta classe abstracta, nomeadamente, write(String s), write (int c); write(char[] b); flush(), close(), ...
Analogamente as subclasses de Reader têm que implementar, entre outros, os métodos int read() int read(char[] c); close(), ...

→ As classes FileReader e FileWriter

Construtores: FileReader (File file); FileReader (String filename); ...
FileWriter (File file); FileWriter (String filename); ...

As classes FileReader e FileWriter permitem-nos respectivamente ler e escrever caracteres em objectos do tipo File.

No entanto a leitura e a escrita de um carácter de cada vez não é geralmente a forma mais eficiente de manipular ficheiros de texto. As classes BufferedReader e BufferedWriter possuem métodos para leitura e escrita linha a linha.

→ As classes BufferedReader e BufferedWriter

Construtores: BufferedReader (Reader in)
BufferedWriter (Writer out)

1– Teste a classe abaixo.

```
public class c1 {
  public static void main (String args[]){
    BufferedWriter bw;
```

```
    try {
     bw = new BufferedWriter ( new FileWriter ("d:\\My_work\\teste1.txt"));
     bw.write("1");
     bw.newLine();
     bw.write("2");
     bw.flush();
     bw.close();
    }
    catch (IOException e){
      System.out.println(e.getMessage());
    }
}}
```

A principal vantagem da classe BufferedWriter é que esta realiza escritas optimizadas sobre streams (de caracteres) através de um mecanismo de "buffering". Os dados vão sendo armazenados num buffer intermédio sendo a escrita na stream de destino apenas efectuada quando se atinge o máximo da capacidade do buffer.

Como vimos acima, o construtor da classe BufferedWriter recebe como argumento um objecto da classe Writer, o que significa que uma instância da classe BufferedWriter pode ser definida sobre qualquer subclasse da classe Writer, sempre que for necessário optimizar operações de escrita pouco eficientes.
Simetricamente uma classe BufferedReader pode ser definida sobre qualquer subclasse da classe Reader.

2 - Crie uma classe que leia o ficheiro teste1.txt.

→ A classe PrintWriter

Construtores:
PrintWriter(OutputStream out); PrintWriter(OutputStream out, boolean autoFlush)
PrintWriter(Writer out); PrintWriter(Writer out, boolean autoFlush)

As instâncias de PrintWriter podem ser criadas sobre qualquer subclasse de Writer e também sobre uma qualquer stream de bytes (subclasses de OutputStream)
Esta classe define os métodos print() e println() que recebem como parâmetro um valor de **qualquer** tipo simples.

3 – Teste a classe abaixo.

```
public class c2 {
  public static void main (String args[]){
    PrintWriter pw;
     try {
     pw = new PrintWriter ( new FileWriter ("d:\\My_work\\teste2.txt"));
     pw.println(2.31);
     pw.println(false);
     pw.print("X");
     pw.flush();
     pw.close();
     }
    catch (IOException e){
      System.out.println(e.getMessage());
    }
}
}
```

4 - Crie uma classe que leia o ficheiro teste2.txt.

→ Streams de bytes

As subclasses de OutputStream e InputStream implementam streams de bytes. Os métodos abstractos definidos na classe OutputStream são idênticos aos de Writer com a diferença de que em vez de caracteres aceitam bytes.

→ As classes ObjectInputStream e ObjectOutputStream

Construtores: ObjectInputStream(InputStream in)
ObjectOutputStream(OutputStream out)

Uma ObjectOutputStream permite armazenar objectos através do método writeObject() que implementa um algoritmo de serialização que garante que todas as referências cruzadas existentes entre instâncias de diferentes classes serão repostas aquando do processo de leitura dessas mesmas instâncias.

Para que se possam gravar instâncias de uma determinada classe numa ObjectOutputStream é necessário que a classe implemente a interface *Serializable*. Além disso, todas as variáveis dessa classe terão que ser também serializáveis. Isto significa que todas as variáveis de instância da classe devem por sua vez pertencer a classes serializáveis. Os tipos simples são por definição serializáveis, assim como o são os arrays e as instâncias das classes String, Vector ou ArrayList.

5 – Construa uma classe à sua escolha definindo os atributos e construindo os getters e setters de forma automática no netBeans ou procure no seu trabalho de outras disciplinas uma classe já feita.

a) - Construa um programa de teste que instancie vários objetos da classe anterior e escreva esses objetos num ficheiro. Use ObjecStreams.

b) – Construa um programa (pode usar o mesmo projecto mas use outro método main) para ler o ficheiro criado na alínea supondo que não sabe quantos objectos estão no ficheiro.

6 – Construa um programa que escreva para um ficheiro uma sequência de linhas de texto, introduzidas pelo utilizador (use ObjectStreams).

7 – Construa um programa que leia o ficheiro do exercício 6.

8 – Construa um programa que peça ao utilizador nomes de livros e os adicione a uma lista (objeto do tipo ArrayList). Quando sair do programa a lista de livros deve ser guardada num ficheiro de objectos.

9 – Modifique o programa anterior tal que: comece por ler o ficheiro que contém a sua lista de livros; o programa deverá permitir ao utilizador pesquisar se um nome está na lista de livros; listar todos os livros e adicionar novos livros.