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 3 objetos da classe anterior e escreva esses objetos num ficheiro. Use ObjecStreams.

b) – Construa um programa para ler o ficheiro criado no exercício anterior supondo que não sabe quantos objectos estão no ficheiro.

6 – Construa um programa que escreva para um ficheiro uma sequência de linha de texto, introduzidas pelo utilizador.

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

8 – Construa um programa que construa um array de valores aleatórios do tipo int e os escreva num ficheiro.

9 - Construa um programa que leia o ficheiro do exercício 8.

→ A classe InetAddress

10 – O programa abaixo permite-lhe obter o nome da máquina onde está a trabalhar. Implemente-o e explore a classe InetAddress.

```
import java.net.*;
public class GetName {
  public static void main (String args[]) throws Exception{
    InetAddress host = null;
    host = InetAddress.getLocalHost();
    System.out.println(host.getHostName());
    }
 }
```

11 – O programa abaixo permite-lhe obter o endereço IP da máquina onde está a trabalhar. Implemente-o e estude o seu comportamento.

```
import java.net.*;
public class GetIP {
  public static void main (String args[]) throws Exception{
    InetAddress host = null;
    host = InetAddress.getLocalHost();
    byte ip [] = host.getAddress();
    for ( int i= 0 ; i < ip.length; i++){
      if (i>0) System.out.print(".");
      System.out.print(ip[i] & 0xff);
    }
    System.out.println();
  }  }
```

12 – Pretende-se que o programa abaixo, dado um determinado IP, nos diga qual o nome da máquina correspondente. Complete-o.

```
import java.net.*;
import java.io.*;
public class IPtoName {
 public static void main (String args[] ) throws IOException{
          String s=" ";
          char c;
       System.out.print("IP address? ");
          while ( (c=(char)System.in.read()) != 10)
                    s+=c;
          s=s.trim();
       InetAddress address =null;
       try {
        < exercício >
       }
       catch (UnknownHostException e){
         System.out.println("IP malformed ");
       }
     }}
```

13 – Construa um programa que dado o nome de uma máquina, nos diga qual o IP correspondente.