Getting into Practice: Input/Output

Exercice 1: Simulate the *ls* command:

We want to develop a Java program that simulates the "ls" command, which is used to list information about the files and directories in a given directory.

The user enters the full path to the directory, and then the list of files and directories contained in that directory is displayed. Display the type <DIR> for directories and <FILE> for files for each directory and file, as well as the permitted access modes 'r' if readable, 'w' if writable, and 'h' if it is a cache file.

Display example:

..\xampp\htdocs\tp1\index.php <FILE> rw-

..\xampp\htdocs\tp1\accueil.htm <FILE> rw-

..\xampp\htdocs\tp1\images < DIR > rw

Exercise 2: Objects Serialization in products.dat

The objective of this task is to use read and write files to save and re-read a collection of product-type objects.

- Create a Product class with the attributes id, name, brand, price, description, and number in stock. The product class must implement the Serializable interface.
- Create an IProduitMetier interface that will declare the methods for managing our Product entities. This interface contains the following methods:
  - public void add(): which allows you to add a product to the list.
  - public List<Product> getAll(): which loads the list of products from a file and returns them as a list.
  - public Product findById(long id): returns a product by id.
  - public void delete(long id): deletes a product by id.
  - public void saveAll(): saves all products to the file.
- Create a MetierProduitImpl class that implements the IMetier interface. This class contains an attribute that represents a list of products and an attribute that contains the file name for saving the products.

- Write an Application class containing the main method that offers the user the following menu in a while loop:

  1. Display the list of products.
  2. Search for a product by its ID.
  3. Add a new product to the list.
  4. Delete a product by ID.
  5. Save the products: this option saves the list of products to a file named products.dat.
  6. Exit this program.

Getting into Practice: Handling Exceptions

Exercise 1:

You will create a Calculator class with three methods that must handle common errors.

Tasks:

- Method divide(a, b):
  - Must return the result of a / b.
  - If b is zero, display: "Error: Division by zero not possible."
- Method convertToNumber(text):
  - Must convert a string to a number (e.g., "123" → 123).
  - If the string is not a valid number (e.g., "abc"), display: "Error: 'abc' is not a valid number"
- Calculate(operation, a, b) method:
  - Accepts: +, -, *, /
  - For any other operation, display: "Error: Operation '&' not supported"

Exercise 2:

1. Write the TooFastException class inheriting from the Exception class. The TooFastException class does not contain any attributes. The constructor takes an integer as a parameter and calls the super-constructor with the message "*This is an exception of type TooFastException. Speed involved:* " + the integer.

2. Write the Vehicle class, which has no attributes. The Vehicle class offers an empty constructor and contains a single testSpeed() method. The method testSpeed() takes an integer as a parameter and returns nothing. If the integer is greater than 90, the method throws a TropViteException.

3. Add a main to the Vehicle class. Create an object of type Vehicle and call the

testSpeed() method with two different values. Display the call stack of the exception.