For future practical sessions, test the following command on a faculty computer:

```
docker run alpine ls
```

Prefer an external PDF viewer to the one integrated in firefox.

The learning outcome of this practical session is to practice Java Streams operations with simple collectors and to acquire a new development skill that consists in writing code with a REPL (Read-Evaluate-Print Loop, jshell, a debugger-like tool).

The first step consists in using Java 16 or higher to benefit from jshell, which allows interactive development. This form of development is especially relevant when prototyping. Code can be copy-pasted and multi-lines instructions can be achieved with Alt-<Enter>.

To facilitate the development further, make sure all interesting methods are available directly with the following two static imports.

```java
import static java.util.Comparator.comparing;
```

Finally, open your navigator on the official pages documenting the methods used in this practical session.

1 Traders and Transactions Classes

```java
public class Trader {
    private final String name;
    private final String city;
    public Trader(String n, String c) {
        this.name = n;
        this.city = c;
    }
    public String getName() {
        return this.name;
    }
    public String getCity() {
        return this.city;
    }
    public String toString() {
        return "Trader: " + this.name + " in " + this.city;
    }
}
public class Transaction {
    private final Trader trader;
    private final int year;
    private final int value;
    public Transaction(Trader trader, int year, int value) {
        this.trader = trader;
    }
```
public Trader getTrader() {
    return this.trader;
}

public int getYear() {
    return this.year;
}

public int getValue() {
    return this.value;
}

public String toString() {
    return "{" + this.trader + ", " + "year: " + this.year + ", " + "value: " + this.value + "}";
}

2 Data Set

Trader raoul = new Trader("Raoul", "Cambridge");
Trader mario = new Trader("Mario", "Milan");
Trader alan = new Trader("Alan", "Cambridge");
Trader brian = new Trader("Brian", "Cambridge");
List<Transaction> transactions = Arrays.asList(
    new Transaction(brian, 2011, 400),
    new Transaction(brian, 2012, 300),
    new Transaction(raoul, 2012, 1000),
    new Transaction(raoul, 2011, 400),
    new Transaction(mario, 2012, 710),
    new Transaction(mario, 2012, 700),
    new Transaction(alan, 2012, 950)
);

3 Requests

Using only the variable transactions:

1. Finds all transactions in 2011 and sort by value (small to high).
2. What are all the unique cities where the traders work?
3. Returns a string of all traders’ names sorted alphabetically.
4. Are any traders based in Milan?
5. What’s the highest value of all the transactions?
6. Prints all unique pair of traders’ names as a list of strings such that the first trader has once made a transaction with lower value than a different second one. For instance, we keep the pair Brian/Raoul because there is at least one transaction from Brian with lower value than at least one transaction from Raoul. However, we discard the pair Alan/Mario because all transactions from Alan have a higher value than all transactions from Mario.
7. Finds all traders from Cambridge and sort them by name.

8. Finds all transactions’ values from the traders living in Cambridge.

9. Finds the transaction with the smallest value.

10. Find all traders that have increased their transactions values over time.