Object-Oriented Programming
June 2021

Notes or documents of any kind forbidden. Duration: 2h30. Please answer the questions on separate sheets labeled with your name, section, and student ID.

---

1. The problem consists in programming in Java a class `DateNotBiss` suited for representing a date in a non-bissextile year. In other words, an instance of this class is characterized by a *month* m such that $1 \leq m \leq 12$, and a *day* d such that $1 \leq d \leq \ell_m$, where ℓ_m is the length of month m. The lengths of the twelve months are respectively equal to 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30 and 31.

   The class `DateNotBiss` should satisfy the following requirements:

   - It must be possible to instantiate a date given a month m and a day d.
   - It must be possible to print a date on standard output (in the format of your choice).
   - It must be possible to modify a date by adding a given positive or negative number of days. For exemple, adding -10 days to June 7th should yield May 28th. This operation fails if the resulting date does not belong to the same year, for example, if one adds 5 days to December 28th.
   - Instances of this class must be clonable, comparable to each other, and serializable. It must be possible to manipulate them simultaneously from separate threads.
   - In case of any error, a dedicated exception should be thrown.

   Note: You are free to implement any additional classes required by your solution.

2. (a) How would you define a subclass `DateBiss` of `DateNotBiss` suited for representing dates in bissextile years (i.e., years with 29 days in February)? (You do not need to fully program `DateBiss`; it is sufficient to explain what you would do.)

   (b) Which application of inheritance did you use in your answer to (a)? Is the substitution principle satisfied? (Justify your answer.)

3. The following interface is defined in the source code of a program:

```
public interface Action
{
  void operation();
}
```

You are asked to program a class `ExecutionMachine` containing a single public class method `void execute(Action a, int n)`. This method should acquire the lock of the object referenced by `a`, then run successively `n` times `a.operation()` in a newly created thread (or not at all if $\texttt{n} \leq 0$), and then release the lock of the object.