Object-Oriented Programming
August 2024

Notes or documents of any kind forbidden. Duration: 3 1/2h. Please answer the questions on separate sheets labeled with your name, section, and student ID.

---

1. A graphic application manipulates *patterns*, characterized by a rectangular matrix of Boolean values in which *true* denotes black pixels, and *false* white ones. For instance, the matrix

$$\begin{bmatrix} \top & \bot & \top \\ \bot & \top & \bot \\ \top & \bot & \top \end{bmatrix},$$

where $\top$ stands for *true* and $\bot$ for *false*, represents a pattern with 3 rows and 3 columns, that has the shape of an "X".

The problem consists in programming in Java a class `Pattern` for representing such patterns. This class must have the following features:

- Instantiating `Pattern` takes as arguments a number of rows $n \geq 1$ and a number of columns $m \geq 1$, and creates a corresponding pattern composed only of white pixels.
- It must be possible to turn a pixel black in a pattern, by giving its coordinates (i, j) with $1 \leq i \leq n$ and $1 \leq j \leq m$.
- It must be possible to check whether a pattern p_1 is *masked* or not by a pattern p_2, which means that the two patterns have the same dimensions, and for every pixel of p_1 that is black, the pixel of p_2 at the same coordinates is black as well.
- It must be possible to *concatenate* two patterns p_1 and p_2, provided that they share the same number of rows. The result of this operation is a new pattern obtained by placing the columns of p_2 immediately after the columns of p_1. For instance, the concatenation of the pattern given in the first example and of a pattern with 3 rows and 1 column entirely composed of white pixels is described by the matrix

$$\begin{bmatrix} \top & \bot & \top & \bot \\ \bot & \top & \bot & \bot \\ \top & \bot & \top & \bot \end{bmatrix}.$$

- Instances of `Pattern` must be clonable, comparable to each other, serializable, and thread-safe.
- Calling `System.out.println` on a pattern p must display its underlying matrix, in a text format of your choice.
- In case of any error, a dedicated exception should be thrown.

Note: You are free to implement any additional classes required by your solution, as well as to choose the interpretation of details that are not specified in this problem statement.

2. A new release of the application considered in Problem 1 needs to manage patterns composed of pixels that are not only black or white, but can take arbitrary colors.

   Assuming that you are free to organize the source code of the application as you wish, would you define the class `ColoredPattern` suited for this new notion of pattern as a subclass of `Pattern`, or would you organize differently the hierarchy between the classes of your program? Please answer the question by drawing a diagram showing the subclassing relation that you would define between your classes, and explaining which application of inheritance is used in each branch of this relation.

   Notes:

   - You are not asked to program explicitly the class `ColoredPattern`.
   - If your solution requires to modify your answer to Problem 1, it is sufficient to explain how you would perform this modification, without describing it in detail. In particular, you are not asked to adapt the mask check operation to colored patterns.

3. (a) Explain the effect of evaluating the Java expression `new Object[2][]`.

   (b) Write a Java class that can only be instantiated once. Any subsequent attempt to instantiate it must trigger an exception of your choice.

   (c) What are the differences between an abstract class and a Java interface?

   (d) In the context of Java generics, what is a bounded type parameter? When is such a parameter useful?

4. The problem consists in programming in Java a class `Concurrent` that defines a class method `multipleRun` such that:

   - this method accepts as arguments a reference r of type `java.lang.Runnable`, and an integer n;
   - when this method is invoked with a value of n such that $n \geq 1$, it creates n threads that all execute concurrently the method `run` of the object referenced by r. (Each thread executes this method only once.) If $n < 1$, then the method `multipleRun` does nothing.