Object-Oriented Programming
June 2025

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. You are asked to program in Java a class `Pair<T>` suited for representing pairs (v_1, v_2) of values v_1, v_2 of type `T`. This class must have the following features:

   - Instantiating `Pair<T>` takes as arguments two references v_1, v_2 of type `T`, which cannot be null.
   - Instances of `Pair<T>` must be clonable, comparable to each other, and serializable. Two pairs are considered to be equal if they are composed of the same references v_1 and v_2, regardless of their order. In other words, pairs (v_1, v_2) and (v_2, v_1) are equal. The cloning operation must be shallow.
   - The class overrides the method `String toString()` with its own implementation, that generates a character string of the form "(s_1, s_2)", where s_1 and s_2 are the strings respectively obtained by invoking `toString()` on the components v_1 and v_2 of the pair.

     For example, a pair constructed from two instances of `java.lang.Integer` representing respectively 6 and 2025 should produce either `"(6, 2025)"` or `"(2025, 6)"`.
   - 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. For a specific application, one needs to define a subclass `DistinctPair<T>` of `Pair<T>`, that only allows to construct pairs (v_1, v_2) such that $v_1 \neq v_2$, i.e., such that the objects referenced by v_1 and v_2 are not considered to be equal.

   (a) Give a Java implementation of `DistinctPair<T>`, consistent with your answer to Problem 1.

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

3. (a) What is the purpose of the visibility of a constructor? Enumerate the possible choices for this visibility in Java, and describe briefly the semantics of each of them.

   (b) Explain the limited form of multiple inheritance allowed by the Java language. In this context, how does a Java interface differ from an abstract class?

   (c) Why does the Java language impose to declare checked exceptions thrown by methods and constructors? Why isn't there a similar obligation for runtime exceptions?

   (d) In Java, what is a lock, and how can it be created? Are locks associated to objects or classes?

4. A safety management application relies on a class `Room` for ensuring that the maximum occupancy of a room is never exceeded. This class has the following features:

   - It admits a constructor taking as argument an integer number representing the maximum number of people that can be present at any time in the instantiated room. This room is initially empty.
   - It defines two methods `void enter()` and `void leave()` that are invoked, from different execution threads, by all people when they (respectively) enter and leave the room.
   - When someone tries to enter a room that has reached its maximum occupancy, the method `void enter()` should block, until space has been made available by someone else leaving the room.
   - If the method `void leave()` is invoked when the room is empty, it must trigger the checked exception `RoomException`. This exception must also be thrown by the constructor if it receives an invalid argument.

   You are asked to provide a Java implementation of `Room`, assuming that the class `RoomException` is already available.