Object-Oriented Programming
August 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. The problem consists in programming in Java a class `CenteredSegment` suited for representing *centered segments* on the real line. A centered segment is characterized by a *center* c and a *diameter* d, and contains all the real values that belong to the closed interval $[c - d, c + d]$. The parameters c and d are assumed to be integers, with $d \geq 0$. For instance, the centered segment generated from $c = 3$ and $d = 4$ contains all real numbers r such that $-1 \leq r \leq 7$. Choosing $c = 5$ and $d = 0$ yields a centered segment that only contains the value 5. In addition, each instance of `CenteredSegment` has a *serial number* of type `long`. This serial number is unique, meaning that two distinct instances of `CenteredSegment` always have different serial numbers.

   The class `CenteredSegment` must have the following features:

   - Instantiating `CenteredSegment` takes as arguments values for the center c and the diameter d of the centered segment to be created. This operation automatically assigns a serial number. Instances of `CenteredSegment` are immutable. It should be possible to read their serial number.
   - Instances of `CenteredSegment` must be clonable and comparable to each other for equality. Two centered segments are considered to be equal if they contain exactly the same real values, regardless of their serial number.
   - It should be possible to check whether a centered segment is a subset of another, i.e., whether all real values contained in the former also belong to the latter.
   - It must be possible to check whether a given real number belongs or not to a centered segment.
   - In case of any error, a dedicated exception should be thrown.

   Notes:

   - 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.
   - If you need to compute the absolute value of an integer number `v`, you can use the class method `int Math.abs(int v)` available in the Java class library.

2. Assuming that the class `CenteredSegment` of Problem 1 has already been programmed in some application, and cannot be modified anymore, one wishes to define a subclass `RationalCenteredSegment` of this class. The features and behavior of instances of `RationalCenteredSegment` are similar to those of instances of `CenteredSegment`, with the only difference that their associated center and diameter can be rational rather than integer numbers.

   (a) How would you implement in Java the class `RationalCenteredSegment`?
   Notes:

   - You are not asked to provide all details of this implementation, but only to describe how you would organize it.
   - You can assume that the source code of the application already defines a class for representing rational numbers.

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

3. (a) How does a class differ from an object? Can a class contain variables? methods? constructors? locks?

   (b) The Java implementation of generics relies on *type erasure*. What does it mean? And why does it make impossible to have arrays with elements of a generic type?

   (c) By which mechanism(s) can the Java compiler locate the source files that define the classes contained in a given package?

4. A concurrent application needs a class `Gate` for controlling the access to a shared resource. This class has the following features:

   - It admits a constructor taking as argument a strictly positive integer number t representing a threshold number of tasks. This threshold cannot be modified after instantiation.
   - It defines a method `void access()` that will be invoked prior to accessing the shared resource. This method counts the total number of times n that it has been invoked for the relevant instance of `Gate`, and blocks the caller whenever $n < t$. If $n \geq t$, then all the tasks previously blocked on this instance of `Gate` are unblocked.
   - It defines a method `void reset()` that resets to zero the invocation count n used by `void access()`. Additionally, if there are tasks blocked on this instance of `Gate`, all of them are unblocked by this operation.

   You are asked to provide a Java implementation of `Gate`.