# Object-Oriented Programming
# June 2026

---

*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 IPv4 protocol defines an *IP address* as a 32-bit number split into four 8-bit components, written in decimal notation from the most to the least significant one, and separated by dots. For instance, one of the main web servers of the University has the IP address 139.165.51.107.

   An *IP prefix* is mostly defined as an IP address, but with an additional field written after a slash ('/'), indicating how many bits of its value are relevant. The relevant bits are the most significant ones, and the other bits are ignored. Every IP address that matches all relevant bits is said to *belong* to the prefix. For instance the prefix 139.165.0.0/18 indicates that only the first 18 bits of the IP address 139.165.0.0 are relevant, which means that all addresses from 139.165.0.0 to 139.165.63.255, and in particular 139.165.51.107, belong to it.

   (a) You are asked to program in Java two classes `IPAddress` and `IPPrefix` suited for representing, respectively, IP addresses and prefixes. These classes must have the following features:

   - It must be possible to instantiate an IP address by providing its four components, and an IP prefix by providing either its four components or an IP address, together with the number of relevant bits.
   - Instances of `IPAddress` and `IPPrefix` must be immutable, clonable, serializable, and comparable to each other for equality. Two IP addresses are equal if their components are pairwise equal. Two IP prefixes are equal if the sets of IP addresses that belong to them are identical.
   - It must be possible to check whether a given IP address belongs or not to a given IP prefix.
   - Calling `System.out.println` with an IP address or prefix as argument must display this address or prefix in the notation defined above.
   - 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.

   (b) Which application of inheritance, if any, did you use in your answer to problem (a)? Is the substitution principle satisfied? (Justify your answer.) If you did not use inheritance, explain why.

2. (a) What is the encapsulation principle? Why is it essential to object-oriented programming?

   (b) Explain the limited form of multiple inheritance allowed by Java.

   (c) How does the Java compiler determine whether a particular exception is a checked one?

   (d) What is the main advantage of defining generic classes?

3. The Java source code of an application contains the following fragment:

```
public interface Computation
{
  ComputationResult compute();
  void handle(ComputationResult r);
}
```

   In classes that implements this interface, the method `compute()` is assumed to perform a computation that may last from a few seconds to several minutes, and `handle(r)` processes the result `r` of such a computation, which is a fast operation.

   You are asked to program in Java a class `ConcurrentComputation` with the following features:

   - This class admits a single constructor that accepts as arguments a reference `c` of type `Computation` and a strictly positive integer number `n`.
   - Instantiating this class has the effect of creating and starting `n`+1 threads, such that:
     - `n` threads repeatedly call `c.compute()` in order to perform concurrent computations. Each time that such a computation terminates, the returned result is communicated to the (`n` + 1)-th thread.
     - The (`n`+1)-th thread continuously waits for the result `r` of a computation carried out by another thread to be received, and then processes this result by calling `c.handle(r)`.
     - The instantiation operation terminates after the `n` + 1 threads have been created and started.

   Note: You are free to implement any additional classes required by your solution, but you are not required to implement `ComputationResult` or any class implementing `Computation`.