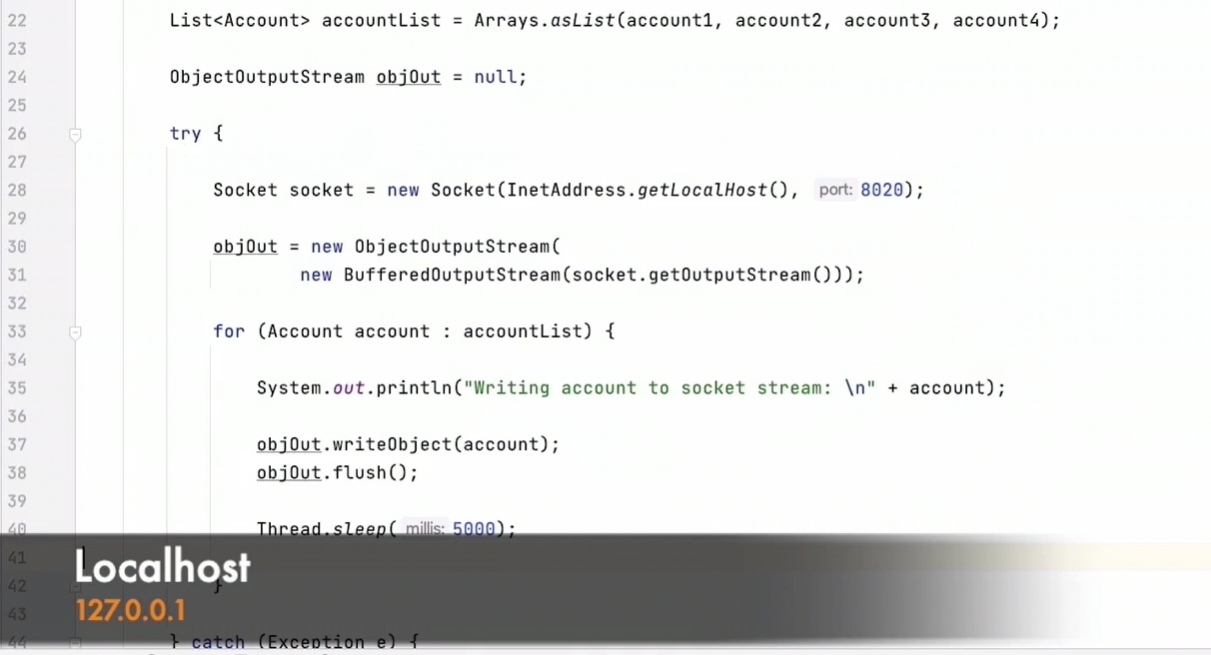
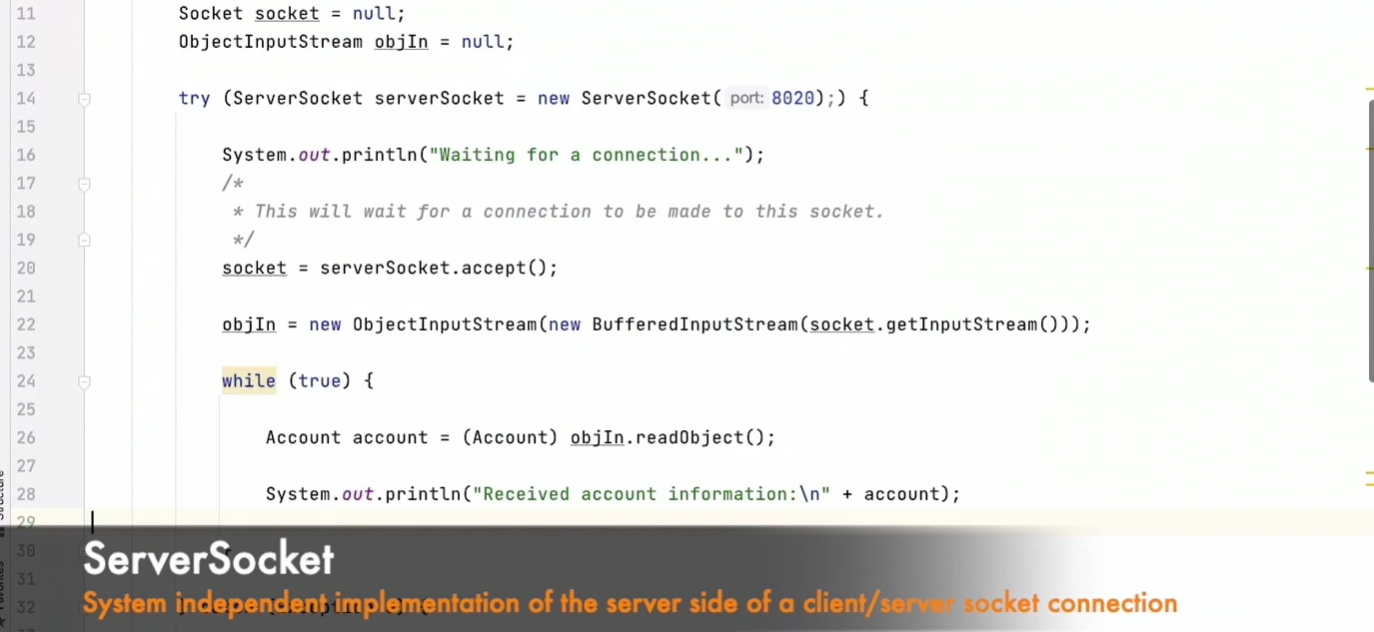
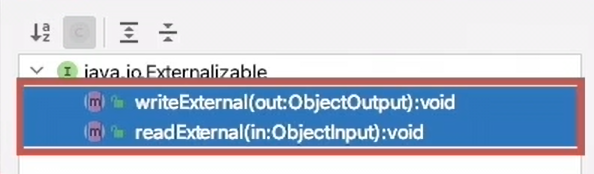
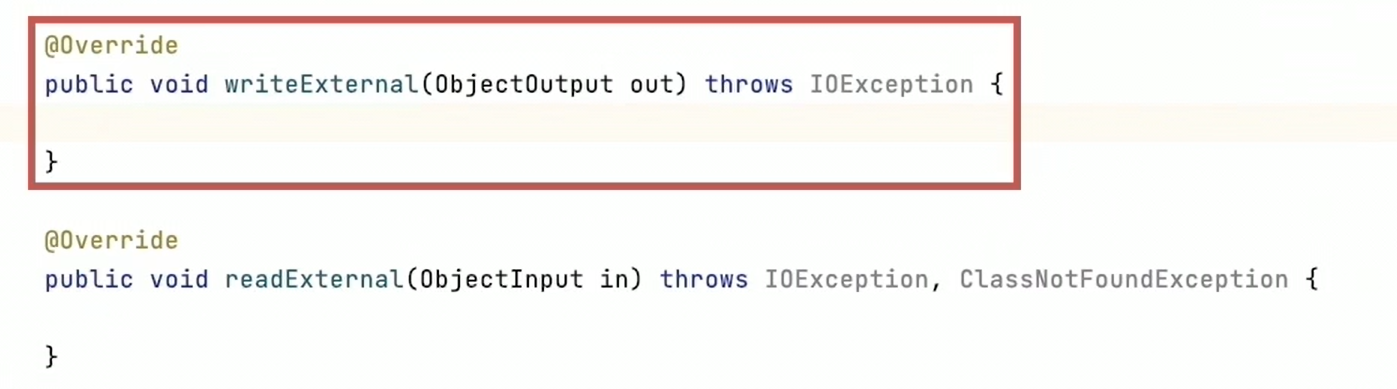
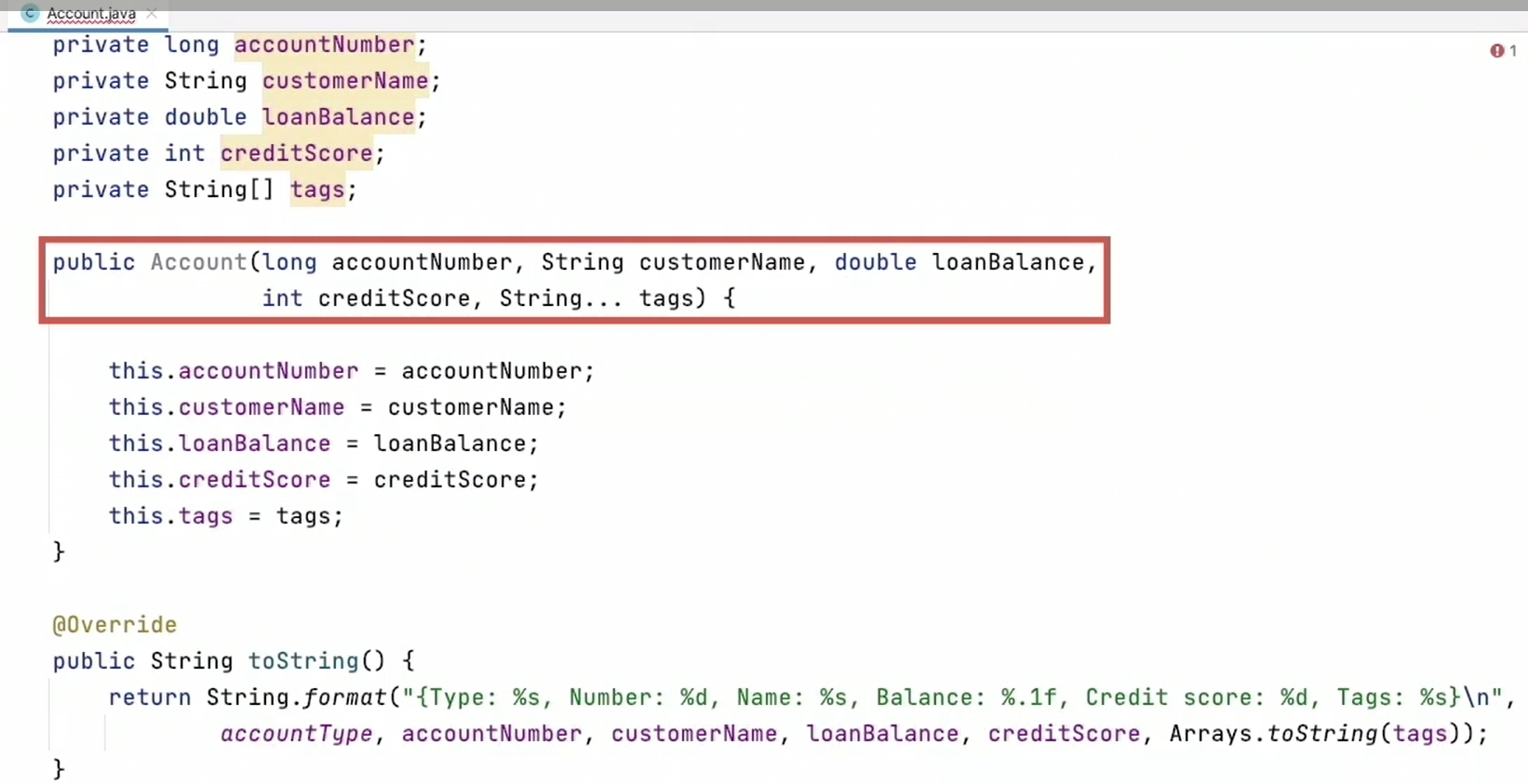
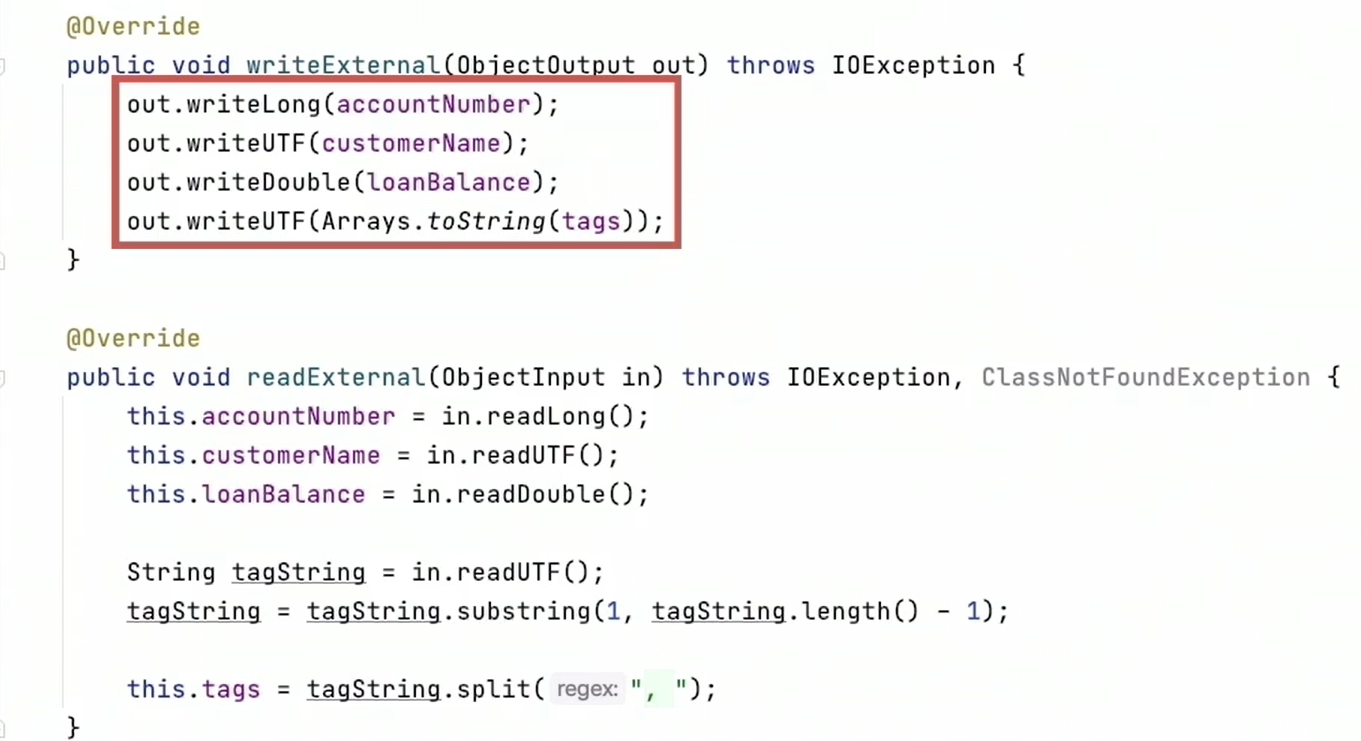
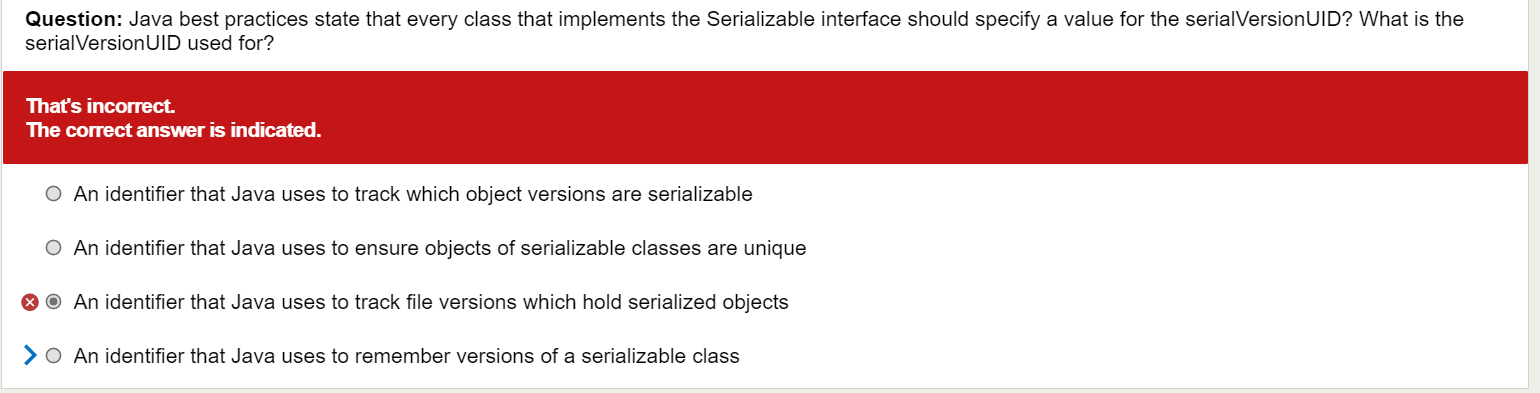
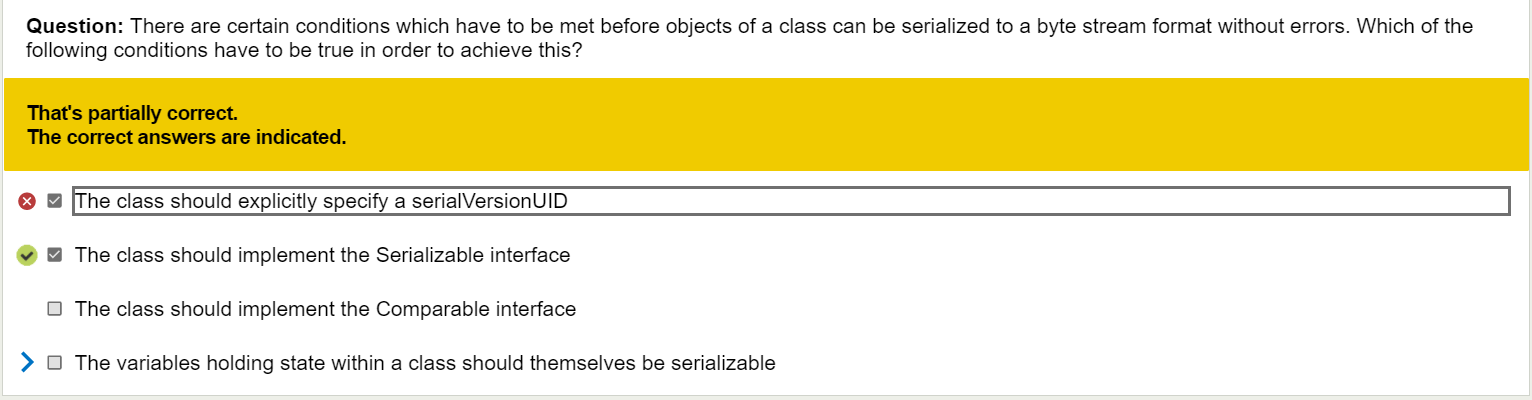
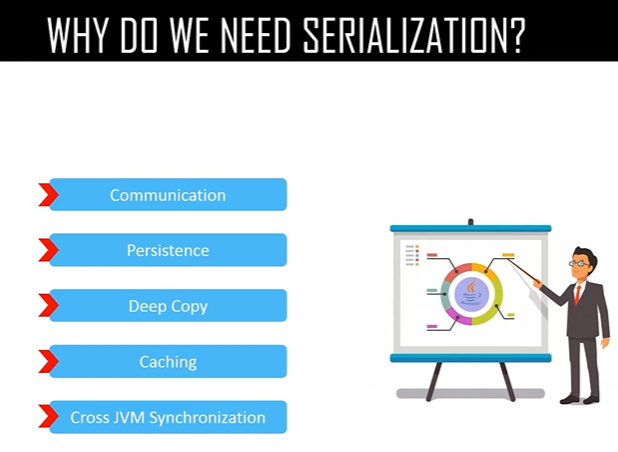
* NotSerializableException if class is not Serializable because they cannot be converted in byte stream.
* ClassNotFoundException is reading object is not compatible with class object.
* InvalidClassException if serialVersionUID mismatch.
* Adding new field means your class can still read older object and no need to change serialVersionUID – Backward Compatible.
* If you change object’s type, then it is backward incompatible, and we must change serialVersionUID.
  + Generally, we do not change serialVersionUID that is wrong.
  + And it throws InvalidClassException due to incompatible Class object.
* transient not Serializable.
* Nested Class: if not implement Serializable then it will throw NotSerializableException.
  + All nested class must be Serializable.
* Socket: an endpoint of a two-way communication link between two application on the same network.
  + 
* 
* Externalizable : Custom serialization and deserialization of object. It is not marker interface. It fast compare with Serializable.
* 
* No argument construction is compulsory to implement Externalizable otherwise it throw InvalidClassException.
* 
* 
* 
* 
* 
* 
* Adding new field or method is a compatible change and changing class hierarchy or UN-implementing Serializable interface are non-compatible changes.
* To Prevent A Child Class From Being Serialized When It's Parent Class Already Implements Serializable Interface, implement writeObject() and readObject() method in the child class and throw NotSerializableException from those methods.
* If superclass implements Serializable - constructor is not called while if the superclass doesn't implement Serializable - constructor is called during DeSerialization process of child class.
* Use readResolve method to Prevent Deserialization Process Creating Another Instance Of Singleton Class.
* the serialization process can be customized. When an object is serialized, objectOutputStream.writeObject (to save this object) is invoked and when an object is read, ObjectInputStream.readObject () is invoked. What most people do not know is that Java Virtual Machine provides you with an option to define these methods as per your needs. Once this is done, these two methods will be invoked by the JVM instead of the application of the default serialization process. Classes that require special handling during the serialization and deserialization process must implement special methods with these exact signatures:
  + private void writeObject(java.io.ObjectOutputStream out)  
    throws IOException
  + private void readObject(java.io.ObjectInputStream in)  
    throws IOException, ClassNotFoundException;
  + private void readObjectNoData()  
    throws ObjectStreamException;

**Compatible Changes**

* Adding a new field will not affect serialization. The newly added field will be set to its default values when the object of an older version of the class is unmarshaled.
* Changes In access modifiers such as private, public, protected or default is compatible since they are not reflected in the serialized object stream.
* Changing a transient field to a non-transient field, static to non-static are compatible changes.
* Adding or removing writeObject()/readObject() methods.

**Incompatible Changes**

* Changing implementation from Serializable to Externalizable interface,
* Deleting an existing Serializable field,
* Changing non-transient field to transient, non-static to static,
* Changing the field type,
* Updating the class package.
* Modifying the writeObject() / readObject() method, we must not modify these methods, though adding or removing is a compatible change.