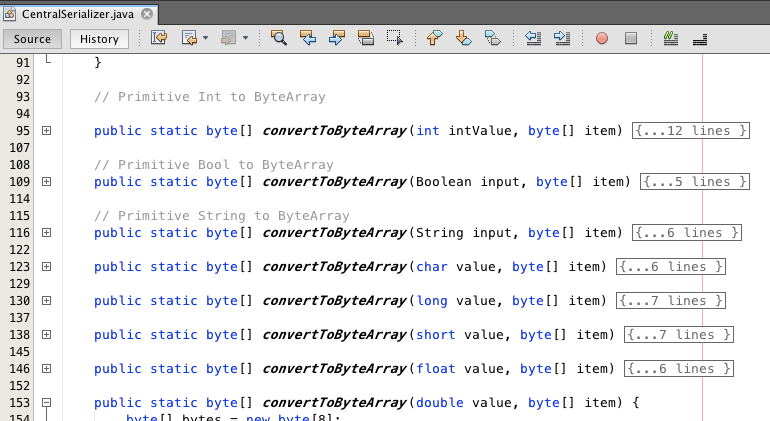
# Centralizing

The “object of primitive type to a sequence of bytes” and vice-versa methods are created in a single class is to avoid the need for every class to have these methods. They are static (they receive an object and return bytes, or vice versa), they can simply be concentrated in a static class (no instances) and invoked from anywhere.

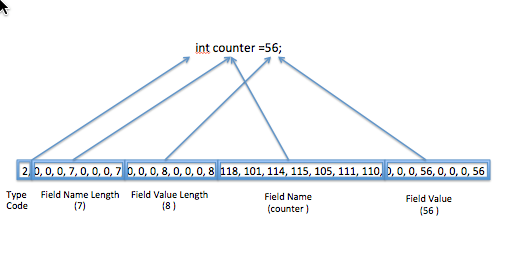
The methods to serialize can receive a primitive object (integer, Boolean, etc) and an array of bytes, returning the array of bytes with the serialized object’s bytes appended. Therefore, each serialization method grows the byte array.

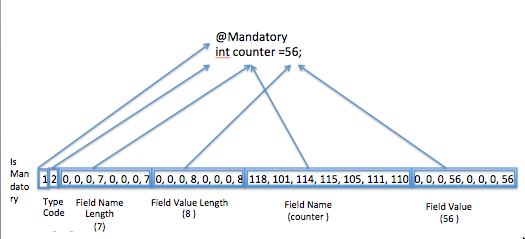


In this Class

“serializePrimitive” method serialize primitive with following algorithm

Example:

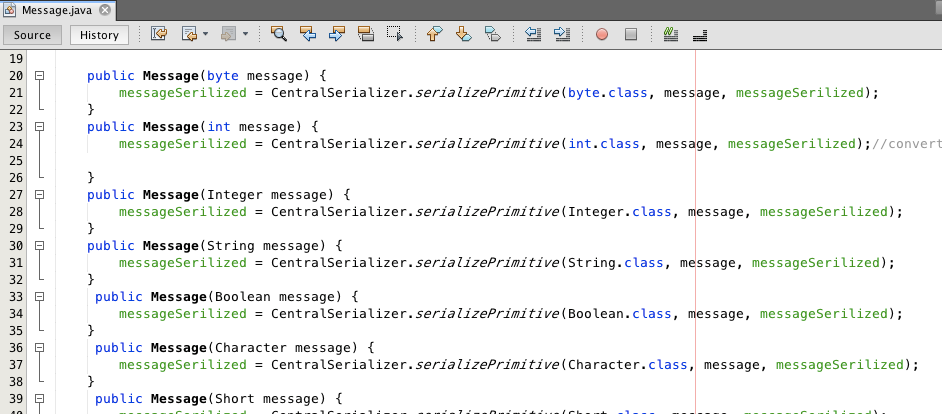




# Message

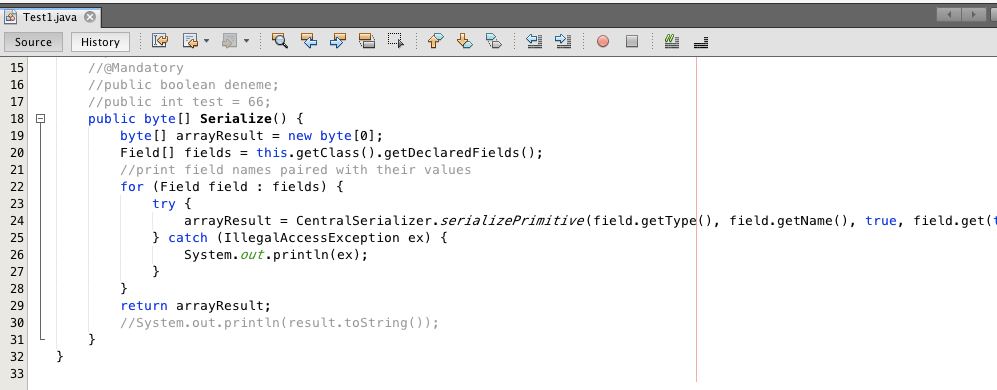
When the user wants to send a message, that message is an instance of a class that has a method that knows how to serialize it, by invoking the serialization methods of the static class for each of its variables.

For each of primitive type message and non-primitive type messages, the Message Class knows how serialize the object.



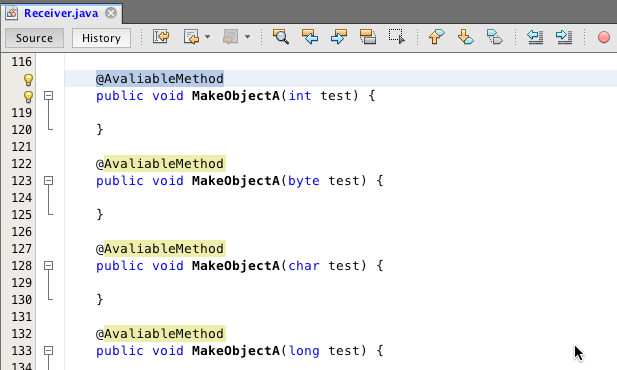
Non-primitive objects have serialize method for each of its variables by reflection.

This class is also responsible to create Base64 string.



# Receiver

Receiver has operation of services with @AvalibleMethod annotation.



These operations are used in compliance test. Receiver has also methods to create primitive and non- primitive methods to create object from byte array that are invoked when suitable operation is found in compliance test..

# @AvaliableMethod Annotation

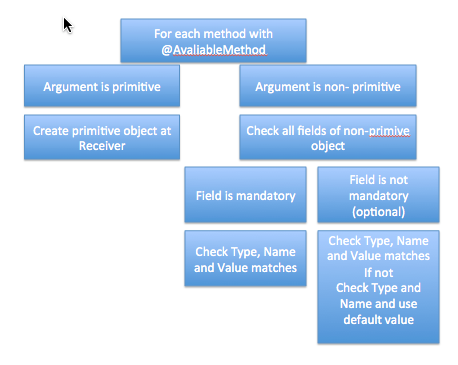
@AvalibleMethod annotation is used to get operation of services in Receiver

# @Mandatory Annotation

@Mandatory annotation is used to mark mandatory fields of operation argument.

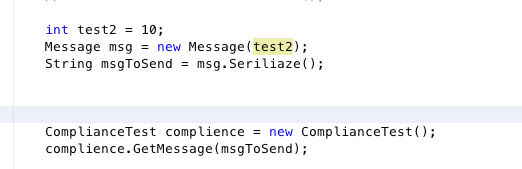
# Compliance Tests

Compliance is done at the binary level at ComplianceTest class with primitive components. Only the components that match are assigned to the formal argument of the operation. If it finds a suitable operation, it builds another array of bytes from both the message and the formal argument. Components that match use the message ones. Components in the formal argument that are not matched by any in the message use the ones already in the formal argument. Then, the platform invokes the operation to create object from array of bytes.

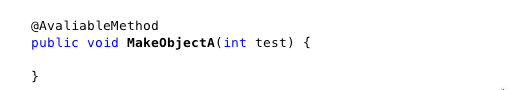


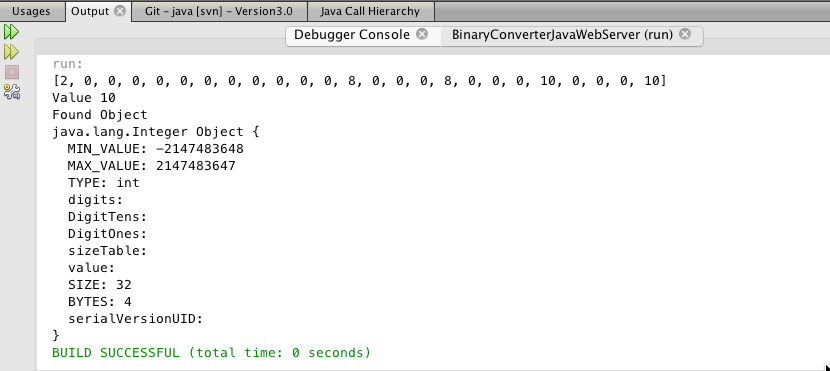
# Tests

## Message with Primitive object

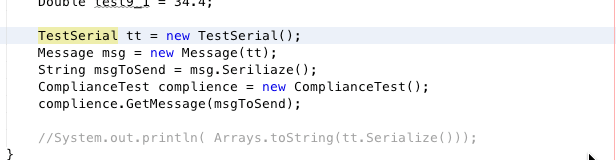


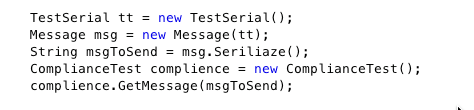
We have following method that has argument with the same type

So the result will be found

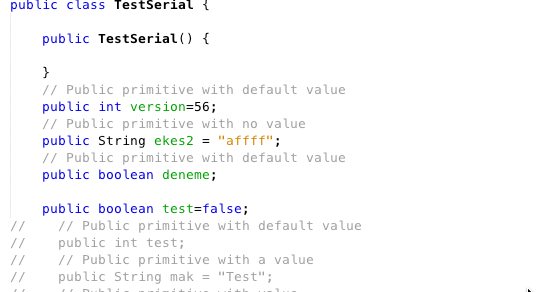


## Message with Non-Primitive object

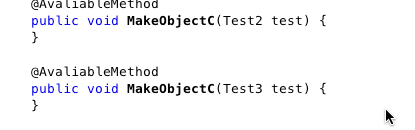




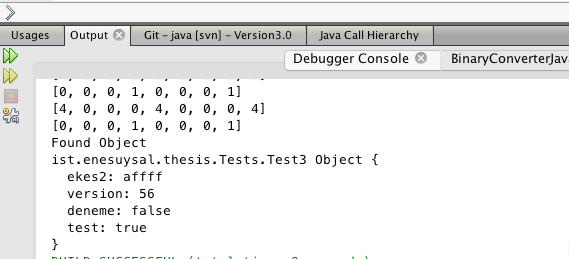
Here we have TestSerial class with following fields



and we have MakeObjectC method that has argument with Test3 class



The match will be found .



Because;

All mandatory fields matches with type, name and value and all non-mandatory fields are matches with type and name. if there would be more field in TestSerial then they would be ignored.

