Question 1

For embedded software, it is important that the software is dependable and robust (fault tolerant). Firstly, the garbage collector is used because it simplifies dynamic memory management and eliminates memory leaks that could be caused by not using the garbage collector. Built in exceptions in Java are used since unlike other low-level programs will not continue execution of the program until the exception is handled. Casting is an important mechanism used in embedded software since they tend to have very strict limitations (i.e. no integers) so casting becomes a useful tool to circumvent that.

JVM's bytecode verifier is an important mechanism for embedded systems because it prevents software from interfering with other operations on the machine. Also, JIT is used to speed up the compilation and execution time. However, is the embedded system does not have a lot of memory, JIT should be avoided since it takes up additional memory space.

Encapsulation is also used, in order to keep data safe. Ex: in the APDU class, there are only getters for the private attributes because we do not want them to be modified once they are initialized. Delegation is also used in this class, since the 4 cases are fairly similar, there was no need to repeat the constructor code multiple times, so having them all call a general constructor (case 4) was the best way to handle this. Abstract classes should be avoided in embedded systems since we want everything to be very specific in order to avoid breaking the software as much as possible.

This are all the mechanisms I could think of but note this is not a complete list. Also, it is important to note that I am not very familiar with embedded software so listing mechanisms that are not relevant in an embedded context is difficult. I simply highlighted what I included and what should be avoided.

Note

In theory, if the code I made works and is not the most efficient method, it is trivial to make it more efficient (just need more time) since the logic is correct.
Note2: Unit tests were not included for the TLV methods.

Assumptions

I have made some assumptions while coding due to some stuff being unclear.
They are the following:

1. Besides the android UI, I will not be using any libraries.
2. The header values and the data are all signed (hence using the byte datatype)
3. Le and Lc are both shorts since it is specified that the APDU cannot be greater than 250 bytes, so a short will suffice. In a normal APDU there is up to 65 540 bytes (header+data), so this solution does not work for that scenario. A possible method is to use char data type to represent Le and Lc since char is 16-but signed value.
4. Since constructors cannot return values, returning the APDU as byte array will be done as a method.
5. Since it does not specify that the byte array is the only attribute of the APDU object, I have added a data array attribute in order to make encryption and decryption easier.

6. TLV tags will be assumed to be 1 byte long since “define your type of tag” was specified. The length octet will also be assumed to always be in a “definite short” form.
7. It was fairly unclear what methods I needed to create in the TLV class, so I chose to make the following methods based on what is said in the document:
    a. Find Value = find the contents (value) based on a single TLV input (excluding nested tags).
    b. Tag present = returns the class, P/C and the tag number of the TLV being passed. (nested tag will not be included).
    c. Parser = given a byte array stream, it will create a 2D array holding each TLV found in the stream (does not include nested tags).