1. Java programs (byte code) can be executed (at least) in four different ways: interpretation, ordinary compilation, Just-In-Time (JIT) compilation, or directly on a Java-processor.
   a. Give examples on a situation where ordinary compilation would be better than the alternatives. Explain, why each alternative would be worse.
      For example, some small method is executed only once, or some small device does not have a Java compiler. Compilation might take more time than interpretation once.
   b. Give similar examples and explanations on when the other alternatives would be best choice.
      Compilation: same (time critical) program is executed many times.
      JIT compilation: Some large program which contains many modules, and only a few of them are ever needed at the same time.
      Java processor: Any small device that is (almost) only used to execute Java-programs.
   c. What would be best alternative to execute (Java) programming project at school? Why?
      Java byte code interpreter, because execution speed is not an issue. Program is developed until it works, and then not needed any more.
   d. What processes are needed at execution time, when your Java program MyProg is executed with 1) interpretation, 2) ordinary compilation, 3) JIT compilation, 4) Java processor?
      1) Java byte code interpreter, 2) compiled and linked MyProg in machine language, 3) Java byte code interpreter, Java compiler, dynamic linker, 4) MyProg in Java byte code
   e. When would you use C# instead of Java? What do C# and Java have in common? How do they differ?
      Microsoft C# is meant for JIT compilation. Both C# and Java have similar "byte code". The MCIL bytecode for C# is used also for other languages but C# in Microsoft environment (e.g., C, Java),
   f. What kind of programming language is Scala? What does it have to do with Java?
      General purpose high level programming language, that is first compiled to Java byte code, and then executed on JVM. Scala supports both object based (e.g. Java) and functional (e.g., Haskell) programming models.

2. Subroutines vs. macros, literals vs. constants
   a. Give an example on a situation where routine XYZ would be better to implement as a macro instead of as a subroutine. Explain. Give an example?
      XYZ is a piece of code that can appear anywhere and it is used often. It does not need context, just code is enough. It can be so short that implementing it as a subroutine is not efficient. For example, for a code segment of just a few machine instructions it is not cost effective to implement a subroutine call and return structures, which would take much more to execute than the work itself. You may also want to use name parameters which are usually not allowed in subroutines.
      Example: subroutine call prelude and prolog.
   b. Give an example on a situation where routine XYZ would be better to implement as a subroutine instead of as a macro. Explain. Give an example?
      XYZ is commonly used, it may be long, its operation may depend on context, it may have call-by-value or call-by-reference parameters. With macros, the semantics of formal parameters may vary wildly based on actual parameters used.
      Example: 1000 instruction long routine, that is invoked 500 times.
   c. Give an example on a situation where value X would be better to implement as a literal instead
of a constant in an instruction. Explain. Give an example?
X might be so large that is does not fit into instruction constant field. X may be of type (e.g.,
float, long int, char) that is not supported as instruction constants. Maybe the value of X is not
known at compilation time, but it is computed at execution time.
Example: floating point value, or any variable value.

d. Give an example on a situation where value X would be better to implement as a constant in
instruction instead of a literal. Explain. Give an example?
X is small constant, that does not have any larger semantic meaning.
For example number 1 that is used to increment a counter.
If it would be implement as a literal, you would need an extra memory reference
every time it is used. A smart compiler would probably try to place it in a register,
but it would be kind of a waste for a register.
Example: constant that does not change from installation to another. Number of months in a
year.

3. Literals, variables, parameters
a. How does using a literal differ from using a (global) variable? Advantages? Disadvantages?
Literals are usually stored in specific area of memory, called literal area. You can easily specify
restrictions on the use of that area. For example, you can make it read-only. Literals are global,
and they can be used everywhere. Some systems allow undefined literals, whose values will then
be only defined when (if) they are used the first time.

b. What danger is there if one could write to literal area?
Assume that you have the value of pi as a literal in memory, its value is 3.14159265 and
somebody changes it to 3.15. Next time somebody uses pi, the result will be incorrect and it will
be very difficult to locate the source of the error.

c. How does using a literal differ from using the constant field in an instruction?
Literals are located in memory, and their use is somewhat more restricted and slower than if the
data was stored in the instruction constant field. Any value can be implemented as literal, but
only small number of data types are accepted in the instruction constant field.
Some systems allow undefined literals, whose values will then be only defined when (if) they
are used the first time. The instruction constant field values must be defined at compilation time.
Literal value is in memory only once and in one place, whereas a constant stored in instruction
constant field is stored in memory in different places in different machine instructions.

d. Macros use usually call-by-name parameters instead of call-by-value or call-by-reference
parameters. Could you use call-by-value or call-by-reference parameters in macros?
Macros are processed at the beginning of the compilation, or in fact even in stage 0 before the
compilation. The call-by-reference concept is associated to execution time memory addresses
which do not even exist at compilation time.

4. What does program mystery.k91 do? How does it do it?
Copy multiply instruction G into add instruction K; in this way you can modify code as data. Real
systems do not usually allow this, at least in user-mode, because read and execute the code segment.
In this case, the instruction K ("ADD R1, Z") becomes copy if instruction G ("MUL R1, Y").
Program now does two multiplications by Y and prints the value of X*Y*Y.

And what about program mystery3.k91? How does it do it?
It changes the opcode (DIV) of instruction H into 17 (ADD), using bit manipulation and bit mask
(MSK).
First you use the bitmask (MSK) and and-operation to clear the opcode, because in bits 24-31 from
right are all zeroes in MSK. The new opcode 17 is then loaded into R5, and move left to its proper
place for an opcode. Finally it is combined with the rest of the instruction H bits with an or-operation.
How could you use this type of programming? What problems are there with this type of programming?
You could some inner loop structure by removing branches that would be taken the same way every time anyway. Or, you could compile Java byte code into machine code, and then execute it. This type of approach might be useful, if you save much more time in resulting code execution than you use to modify the code. Debugging becomes a major problem now. This is only theoretical for normal Java programs, because you can not modify the code at execution time.

5. Assume that you are encoding a ttk-91 simulator with Titokone. Your simulator reads machine language ttk-91 code and emulates the execution with simulated instructions, one instruction at a time.
   a. How would you define the simulated ttk-91 structures (registers, memory) in your program?
      Simulated general registers are in array SimR[8], simulated PC, IR, TR in variables simPC, simIR, simTR, and simulated memory is array simMem[512].
      Instruction fields are in variables simOpcode, SimRj, simM, simRi, simAddr.
   b. How would you code (with which ttk-91 instructions) the fetch-phase of the fetch-execute cycle?
      
      ```
      load r1, simPC ; instruction fetch
      load r2, simMem[r1]
      store r2, simIR
      add r1, =1 ; increment PC
      store r1, simPC
      ```
   c. How would you code breaking one machine instruction into its fields? (Hint: use bit masks with and, shl, shr-instructions)
      ```
      load r1, simIR
      load r2, =255 ; 8 bit mask ; OPER
      shl r2, =24 ; move it to field opcode
      and r2, r1 ; opcode alone
      shr r2, =24 ; move it to right edge
      store r2, simOpcode
      load r2, =7 ; 3 bit mask ; Rj
      shl r2, =21 ; move it to field Rj
      and r2, r1
      shr r2, =21
      store r2, simRj
      load r2, =3 ; 2 bit mask ; M
      shl r2, =18 ; move it to field M
      and r2, r1
      shr r2, =18
      store r2, simM
      load r2, =7 ; 3 bit mask ; Ri
      shl r2, =16 ; move it field Ri
      and r2, r1
      shr r2, =16
      store r2, simRi
      shl r1, =16 ; ADDR
      shr r1, =16 ; fill with zeroes
      store r1, simAddr
      ```
   d. How would you implement fetching the second operand to TR?
      ```
      load r1, simAddr ; compute 2nd operand to r1
      load r2, simRi ; is Ri==0
      jzer r2, noRi ; do not use Ri
      add r1, simR(r2) ; add value of register Ri
      noRj load r2, simM ; assume mode ok (0-2)
      jzer r2, okTR
      load r1, simMem(r1) ; normal memory reference (read)
      ```
sub r2, =1
jzer r2, okTR
load r1, simMem(r1) ; indirect mem ref (read)
okTR store r1, simTR

e. How would you code the execution phase of instruction "add r2,r3"?
You may assume that the second operand value is already in TR.

load r2, simOpcode ; test if add instruction
compr r2, =17
jnequ no_add

load r1, simRj ; index of the 1st operand register (=2)
load r5, simR(r1) ; value of simulated simuloidun register r2
add r5, simTR ; add 2nd operand value
store r5,simR(r1) ; store result into simulated register r2

no_add NOP

f. How would you code the execution phase of instruction "jump loop"?
You may assume that the second operand value is already in TR.

load r2, simOpcode ; test if jump instruction
comp r2, =32
jnequ no_jump

load r1, simTR ; TR has value of symbol loop from ADDR-field
store r1, simPC

no_jump NOP