FORTECH.

https://www.investigatii.md/uploads/resurse/Clean_Code.pdf

Namings

1. The main rule for naming is that it should reveal its intent from the start.
2. Classes should be named with singular nouns or group of nouns that are not general (don't use: data, info, processor, builder...) and methods with verbs or verb phrases.
3. Also the team should pick one word per concept (for example all members should use "update" for db changes) and keep its meaning for consistency. Problem domain name should be used when the code has more to do with the domain instead of the solution.
4. For solution code you should choose more well-known technical names (accountVisitor-> JobQueue).
5. Make the naming relevant to the context ({street, number, zipCode} could be renamed to {addressStreet, addressNumber, addressZipCode}, if we also have state which is not part of the address).
6. Always change the name if it seems to you that you can find a better one which is more readable and searchable.
7. Avoid disinformation.
8. Beware of using names that have only small differences from another in the same class/package. Also beware of different names with the same meaning.
9. Use explanatory variable

Methods

10. Methods should be short, maximum 20 lines.
11. Each method should do only one thing and should only do something or respond to something.
12. Statements in each method should be part of the same level of abstraction.
13. The stepdown rule : We read the code from up to down, so every method should be followed by those at the next level of abstraction.
14. Function arguments would ideally not be present, but more than 3 arguments should never be present.
15. Constructors also should have maximum 3 arguments.
16. Constructors should contain just the arguments that are strictly necessary for the creation of the object.
17. Function arguments should be in order of importance.
18. Avoid passing Boolean as arguments.
19. Methods should not be used as a train wreck (b.getC().getD().getE().doThing()), instead you can split them up into variables.
20. Keep all of the operations in a method at the same level of abstraction (you shall not have db-related terms in the upper layers like Controller).
21. For the methods with one argument it should be used for **(monadic forms)** :
    - Asking a question about that argument (Boolean)
    - Making some changes to that argument and returning it
    - Events, when setting the current state

SOLID Principles

22. A class should have one and only one reason to change, meaning that a class should have only one job (**S**RP).
23. Objects or entities should be open for extension, but closed for modification (**O**CP).
24. Base classes should be replaceable with their sub-classes (**L**SP).
25. A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use (**I**SP).
26. Entities must depend on abstractions not on concretions (**D**IP).

Comments

27. Avoid using comments because you can't realistically maintain them and this can result inconsistencies between code and comments.
28. Code written well with fewer comments its better than a lot of comments.
29. Always delete commented code.
30. Eliminate dead code.
31. Java-doc Comments should be used only for Interfaces.

Tests

32. Tests should be readable and simple.
33. Every function in tests should have tested one concept and should have one assert.

Classes

34. A class is not a data structure, but a template for a real-life object.
35. Switch and if/else statements should be buried in a low-level class where they should be used only once to create polymorphic objects.
36. Prefer polymorphism over if / else or switch / case
37. Always avoid null if it's possible, use instead Optional.
38. Avoid static.
39. Avoid getters and setters.
40. Don’t inherit constants, use instead Enums or Static imports.
41. Avoid long import list by using wildcards.

42. Small number of instance variables.

Understandably:

43. Avoid encodings.
44. Prefer dedicated value objects to primitive type.
45. Avoid logical dependency. Don't write methods which works correctly depending on something else in the same class.
46. Avoid negative conditionals.
47. Always apply the Scouts Rule: “Always leave the campground cleaner than you found it.”
48. Don't use *magic numbers* in the middle of a block of code without explanation.
49. Extract hard-coded Strings and numbers into constants with explanatory name.
50. Prefer solution that doesn't require duplicate code.
51. Try to use common design patterns, such as DTO, DAO, MVC .

Structure:

52. Separate concepts vertically.
53. Declare variables close to their usage.
54. Dependent or similar functions should be close.
55. Don't break indentation.