Clean code techniques

1) Consider static factory methods instead of constructors
2) Use constructor chaining
3) Use Builder if you have constructor telescoping
4) Do not return null from method(Can return Collections.emptyList instead)
5) Do not return integers as error codes (-1, 0, 1) from methods. Throw exceptions instead.
6) Fewer (0-2) method arguments is better. Split method or pass object instead. No Boolean flags.
7) Fail fast and return early. If incoming variable into method can cause an exception, test the variable and throw exception at the top of the method.
8) Also check valid conditions at the top of the method and return a valid response if it doesn't meet the criteria. Should be consecutive if statements at the top of method, rather than nested ifs trying to check each thing as you go.
9) Refactor duplication. Make methods out of the duplicated code.
10) Boolean variables should start with "is".
11) Extract complex conditionals to Boolean methods.
12) Ternary conditions must be one line. Not nested.
13) Throw exceptions with a message → throw new IllegalArgumentException("Invalid age")
14) Use try-with for resources instead of try-catch-finally

Clean Classes

1) Class should be cohesive – elements belong together. Fields and methods all relate to the intention of the class. Methods are interrelated and use the class fields.
2) Adhere to SRP
3) Reduce coupling by using SRP and practicing cohesion. Use smaller classes because they depend on fewer other classes. Program to an interface. Maintain strong encapsulation (keep things as private as possible) and consider dependency injection.
4) Use principle of proximity. As methods are called, put the method declaration close to it.

Refactoring

1) Methods should be up to 10 lines. 10-20 lines is often ok. Consider refactoring at 20 lines.
2) Pass in whole object as a parameter instead of just its parts as primitives. Or you could create an object and put your primitives in that object.
3) Combine entities if you can and it makes sense.
4) If your class is too large, split out the functionality. Note that not all classes have to have member variables. They can just have methods.
5) Remove conditional complexity. Can be replaced with polymorphism or delegation and the strategy pattern.
6) Make sure subclasses don't inherit fields and methods that they don't need.
7) Don't use temporary class fields that are null most of the time. Or used by only one method in the class. Create a new class if necessary → return new AlgoCalculator(this).execute();
8) Look for code that is duplicated with just minor variations.

9) Avoid solution sprawl – where a solution is broken into multiple classes or places. An update should not require additional changes in multiple classes or modules (Shotgun Surgery).
10) Don't have a class that uses methods or accesses data of another class more than its own. You could move those methods into the class. Or implement new, well –encapsulated method and let the class call that method.
11) Don't let one class know about internal details of another class.
12) Don't let a class expose too many internal details about itself.
13) Don't call class getter chains. Put the first leg of the chain in a class method so you only have to change it in one place. Then call that method. Customer.getAddress.getCountry.toString() would become a public getCountry method on customer which returns this.getAddress.getCountry.toString().
14) Don't have a class where the only thing it does is delegate work to other classes. This excludes some design patterns like façade, proxy or adapter.