Guidelines for successful Assignments 2 & 3 submissions

Below are some very basic practices to adhere to in your code:

1. **No magic number** – if you need one, make it a constant.
2. **Avoid code duplication** – in the last homework students came up with very specific methods (e.g., getVelocityEastBound, getVelocityWestBound), hopefully to promote abstraction. This is a good goal; the problem was that these methods replicated the same code with only tiny changes. **Please promote both abstraction and code reuse.**
3. **Provide specific assumptions** (in Javadoc) regarding your method functionality. It is recommended to **check the preconditions at the beginning of ANY non-private method**.
4. To promote encapsulation and abstraction, **create your own exceptions** (preferably extend RuntimeException)
5. **Any major type (class) should have an interface.**
6. **Include ONLY necessary operations for your client in your API.**
7. **Any helper methods should be NON-public.**
8. **Prefer setting your compile-time type as interface** (e.g., List and not ArrayList, Map and not HashMap).
9. When using existing APIs – **read documentation and understand it.** You might be asked to explain it.
10. When getting **warnings** from IntelliJ and/or Maven – **understand them and try to resolve.** You might be asked to explain it.
11. **Test as you code** – do NOT wait till you created 20 classes with 20 methods.
12. **Come up with diverse tests** - code coverage test verifies which lines are executed while testing. Thus, if you have some rare conditions that you check in your code, you should come up with appropriate tests.
13. Come up with **UML/Class diagrams** of your design. You can analyze dependencies between your modules and these will be handy, when presenting your design.
14. **Tell a story in your programming** (pick up meaningful names, throw meaningful messages and provide comprehensive Javadoc)
15. **Keep this story simple – KISS principle** (e.g., do not create inheritance relationships, unless they promote code reuse) https://en.wikipedia.org/wiki/KISS_principle
16. **Keep you code clean and maintainable with 10-50-500 Rule –** avoid *Monolithic Code*, *Spaghetti Code*

    10: No package can have more than 10 classes.
    50: No method can have more than 50 lines of code.
    500: No class can have more than 500 lines of code.
17. **SOLID Class Design Principles -** SOLID is an acronym for design principles coined by Robert Martin (http://en.wikipedia.org/wiki/SOLID_%28object-oriented_design%29):

Design Principle	Description
Single responsibility principle	**A class should have one and only one task/responsibility. More than one task per class, it leads to confusion.**

Open/closed principle	Focus more on extending the software entities rather than modifying them.
Liskov substitution principle	It should be possible to substitute the derived class with base class (will be covered in the next lecture)
Interface segregation principle	**Like Single Responsibility Principle, but applicable to interfaces: each interface should be responsible for a specific task. Do not include unnecessary methods.**
Dependency inversion principle	**Depend upon Abstractions - but not on concretions. This means that each module should be separated from other using an abstract layer which binds them together (**do NOT create strong dependencies between classes – if you need such → you need a 3d party class).

Please analyze your assignment with respect to the above principles. If you violate any of the three highlighted in BOLD principles above, you would like to reconsider your design.

18. **Usage of Design Patterns (starting from Assignment 4 an on…) –** design patterns will be covered in the upcoming two lectures.
    ****
19. **Be self-critical –** leave your assignment aside for a couple of hours, afterwards try to explain to yourself what you did. Does it make sense?
20. **Practice presenting your design in 6-7 minutes.** Rest will be devoted to questions.
21. **Document ideas:**
    Always include the following in your writeup: explain the requirements you implemented, briefly explain your design (you may want to include a UML diagram) and testing (especially corner cases), mention assumptions properly.

GOOD LUCK