Submit your answers to question 1-4 on paper. Submit answers only, not the questions. Submit question 5 as a Github repository. Please notify me if anyone attempts to copy your work.

1. Give your own example of two object references a and b that use classes in the Java API but *not* String, such that `a.equals(b)` is true, but `a==b` returns false.

2. (a) The java.lang.Number class has a public constructor, but if we write "new Number()" it is an error. Why?

(b) What is *another class* in the Java API that behaves like this (public constructor but cannot create instances)?

3. (a) How can you create a LocalDate object for the date 25 February 2017. Write Java code.

(b) Can we invoke "new LocalDate()" or "new LocalDate(*some params*)"? What does this tell you about the constructors for LocalDate?

(c) Read the Java API for LocalDate and look at the table of methods. How can you recognize *factory methods?* Just by looking at the table of methods, you can quickly identify which methods might be used to create LocalDate objects. How can you recognize these methods?

4. Give a "real world" example of a collection (or many things) where it would be appropriate to model the real world collection (in code) using each of these collection types:

(a) Set

(b) List

(c) Stack

(d) Map - and explain what are the keys and values

For each example, explain how the *semantics* of the collection matches your real-world example. Don't use the examples in the lecture slides.

5. Find at least 2 principles or guidance for writing good code that you think are helpful and can be applied by students in OOP. Focus on principles and guidance related to coding that the students in OOP can understand and apply; avoid principles related to design and architecture. There is some overlap between coding and design, though.

Create a Github project named **`codeguide`** that contains a README.md and a src/ directory containing source code exercises in applying the principles.

README.md - should be neatly formatted so it contains sections, with one section for each guideline or principle. Each guideline or principle should contain:

a) a name or phrase that summarizes the principle, such as "don't repeat yourself" (use this as the section header)

b) a full description

c) at least one complete example of how to apply it. Please use Markdown formatting for code. Surround code blocks with 3 backticks (` ` `).

d) a *short* code-improvement exercise for other students to do. Put the code in the src/ directory of the repo and link to it. The code should contain enough comments to tell the students what to do..

e) at least one reference. Include specific URL if there is one.

You can find these guidelines/principles in a book on good coding or search the Internet. The guidelines often have short names to help people remember them, such as the examples below.

Useful references:

Code Complete, 2E by Steve McConnell.

Clean Code by Robert Martin.

Effective Java by Joshua Block.

http://www.unclebob.com, Robert Martin's blog.

Examples

1. General Guidelines

Write code for the maintainer.

Don't Repeat Yourself

Consistently follow a coding convention (coding standard)

Avoid magic numbers

2. Principles related to Refactoring

Introduce an explanatory variable (that explains the purpose of an expression)

Replace expression with a method (when the expression is complex or is reused)

3. Related to design of methods

A function should do just one thing

Avoid (unwanted) side effects. O-O methods do have side effects that are by design. list.add(x) or date.setMonth(m) change the object (a side effect) but that is the purpose of the method.

Command-Query Separation Principle

Prefer Exceptions to Returning Error Codes

Return results, don't print them

Use an enumeration instead of int *codes* (e.g, small = 1, medium = 2, large = 3)