# Common Design Patterns

# Common Design Patterns in Java

- Factory Pattern
  - Have used this already in homeworks
- Builder Pattern
  - Have seen this in the mockito / guava code from last lecture
- Delegate Pattern
  - Also known as composition
- Decorator Pattern
  - Particular form of Delegate Pattern
- Singleton Pattern
  - Careful! Often done in a non-Thread safe manner

# Factory Pattern

- A factory class decouples the user from the implementing classes.
- Used in creation of objects
  - Like a constructor but with additional flexibility
    - Saw this in the Factory objects you've created in homework assignments
  - Immutable objects can be cached more easily with the factory pattern
- Form of encapsulation
  - Can leverage the Factory pattern to allow construction of objects
- Example!

# Builder Pattern

- A builder is used to construct complicated objects in steps
- Extremely useful for creating immutable objects were immutability can only happen after a number of steps are performed
  - E.g.; see the builder for ImmutableCollection objects in Guava
- Composed of “chaining” methods which end in a final call to a method returning the type in question; usually named **build**()
- More complicated builder patterns can exist if certain steps should happen before others, this works by composing two or more builder classes together
- Example!

# Composite/Delegate Pattern

- Known generally as the composite pattern
- The delegate pattern is when desired functionality is “delegated” to another object which has already implemented the functionality
  - You’ve seen this when implementing the Multimap homework. The desired functionality was the Map interface and you delegated the implementation to a HashMap or TreeMap
- You can “compose” multiple delegates to create a very complicated set of functionality for a single class without that class needing to implement all the logic itself
  - This is typically done with interfaces. A class, Foo, implements one to many interfaces. The implementation of those interfaces is provided by Foo via other objects already implementing the interfaces.
- Example!

# Decorator Pattern

- The Decorator pattern is a dynamic way to add behavior to an entity at runtime by using the Delegate pattern
- Decorators all implement a common interface.  They then use the delegate pattern to combine/concatenate their work
- We've seen this already with the InputStream/OutputStream
  - BufferedInputStream can decorate a FileInputStream, both of which are InputStream classes.
- Example!

# Singleton Pattern

- Used when you want a single instance of class
  - one and only one object should exist within the JVM
- Easy to do unless you need to be thread-safe
- Example!
  - Non-thread safe
  - Thread safe

# Questions / Final Review

- Regarding Design patterns?
- Regarding Core Java?
- Questions on the Final?