Final Study Topics
CSC/CPE 203, Fall 2018

Objects
- == vs. .equals()
- vs. primitive types
  - and wrapper classes (java.lang.Integer, java.lang.Double, ...)
  - autoboxing, auto-unboxing

fields (data members)
- static
- instance (non-static)
- final: Value can only be set once

Methods
- abstract
- overriding
  - Calling superclass version
- final: Cannot be overridden

Interfaces
- Introduce a type
- Cannot be directly instantiated
- Cannot define fields
- Usually shouldn't define method implementations
  - (except for "default implementations", as seen in library code)
- A class may implement multiple interfaces
- An interface may extend another interface

Abstract classes
- Introduce a type
- may contain abstract methods
- Cannot be directly instantiated

Concrete classes
- Introduce a type
- Can be directly instantiated
- Can be subclassed
- constructor / calling superclass constructor

public / private / protected

final classes
- Cannot be subclassed

Polymorphism

- Overloading methods (Methods with the same names but different parameters , aka "Ad Hoc Polymorphism”)
- Normal ("Subtype Polymorphism")
- Generic ("Parametric Polymorphism")

Type

- Of a variable/field (“lval”, static type)
- Of an expression (“rval”, static type)
- Of an object ("dynamic type" or "runtime type")
- supertype
- subtype
- instanceof
- downcast

HashMap<K, V>

- Using HashMap to have only one object for a given key
  - (one "canonical" object)

equals/hash code

- Mostly useful for objects that serve as map keys
- Rules for constructing: equals
  - instanceof/downcast
  - check if other is null
  - java.util.Objects.equals()
  - @Override
- Rules for constructing: hashCode()
  - @Override
  - Prime numbers
  - Objects.hash()

Single Responsibility Principle

- "only one reason to change"
- "High cohesion, low coupling"
- "Encapsulation"

Open/Closed Principle

- Open for extension, closed for modification

Design by Contract

- Preconditions, postconditions, invariants

Liskov Substitution Principle (LSP)

- "Strong behavioral subtyping"
- "If S is a [Liskov-substitutable] subtype of T, then objects of type T may be replaced by objects of type S without altering any of the desirable properties of the program"
    - i.e. the things you can do with objects of type S may not be a subset of the things you can do with objects of type T
    - Venn diagram
- A subtype may weaken the preconditions and strengthen the postconditions

Interface Segregation Principle

- "Clients should not be forced to depend on methods they do not use"
- Role Interfaces

Dependency Inversion Principle

- You code shouldn't depend on a concrete detail that you might want to change later
- ex: Graphy's graph generator means Graphy's drawing part doesn't depend on the coordinate system
- ex: Streams mean you don't depend on how the elements in a collection are visited. Since you don't, you can pick a library function that visits them in parallel.

Comparator<T>

UML diagrams

- Indicating a subtype
- inheriting a class (is-a)
- implementing an interface (is-a)
- fields
- methods
- aggregation (has-a)
- other relationships (indicated with a line and, optionally, a label)

Design Patterns

- Strategy
    - e.g. PathingAlgorithm
- Template Method
    - Don't be afraid to introduce new abstract protected methods!
    - e.g. "occupantBlocks()" in OreBlob vs. Miner
- Visitor – understanding, but not memorization
    - e.g. Graphy: Painting shapes, printing shapes, Android vs. Desktop Java

Exceptions

- Generating an exception
- Stack backtrace
- catch
- finally

Functional interfaces

- Lambdas

Generics

- Generic classes
- Generic methods
- Wildcards (?, ? extends T, ? super T), PECS

LSP and Generics:

- Is Square a LSP subtype of Rectangle?
    - Can you create a Java Square class that is a subclass of a Java Rectangle class
    - Can you assign an object of type Square to a variable of type Rectangle?
- Is Miner a LSP subtype of Entity?
- Is List<Miner> a LSP subtype of List<Entity>?
    - Can you assign an object of type List<Miner> to a variable of type List<Entity>?
    - Can you assign an object of type List<Entity> to a variable of type List<Miner>?
- List<? extends Entity>:
    - Can you say "Entity e = list.get(0)"?
    - Can you say "list.put(e)"?
- List<? super Entity>
    - Can you say "Entity e = list.get(0)"?
    - Can you say "list.put(e)"?

Map/Reduce and Streams

- List.stream(), List.parallelStream()
- Stream.filter()
- Stream.map()
- Stream.collect()
- Collectors.toList()