1. Java syntax and review

2. Static members

3. Abstraction barrier

1. Java syntax and review

- Don't forget semicolons
- Balance the curly braces and the parentheses
- Make sure to return the proper data type
- Declare local variables within the method or subclass that is using them
- Write javadocs/comments for all methods, classes, and variables
- Know the differences between public, private, protected, and default and where to use any of them.
- Know when to use method headers such as abstract, static, final
- Double check all syntax

2. Static Members

- Methods and variables that belong to the class itself rather than an instance of a class.
- Static members are useful when manipulating instances of other classes without actually using a specific instance of the class itself
- Do not use static on methods that require variables from a specific instance of the class.

3. Abstraction Barrier

- The barrier between the person implementing the code and the person utilizing the code
- It exists to hide the implementation from the person using the code
- This is useful if the implementor decides to change the way the code is implemented without changing what the code does
- The implementor is free to change the implementation without the user being affected
- When testing, it is important to test for the overall effect of the code and not test the implementation of the code (Black-Box testing).

[www.ccs.neu.edu/course/cs3500f13/cs3500f13jys/recipe.txt]

4. Recipe for implementing an immutable ADT that is specified by an algebraic specification.

One abstract class with its methods
Basic creators have their own class
Static methods that call Constructors and helper methods
Override methods are public (toString, equals, hashCode)

5. Abstraction mechanisms

Procedural abstraction
Parametrization
Specification
Data abstraction (ADT) - data type and methods that specify behavior for it
Type hierarchy - super/sub types

6. Procedural abstraction

- the procedure by which abstractions are implemented
- specification: header with inputs/names for parameters and an input/output description (requires/modifies/effects clauses)
- properties: minimality, generality, simplicity
- total (behavior specified for all legal inputs) vs. partial procedures
- benefits: locality and modifiability

7. Data abstraction

look for similarities between data types and move to a parent class
→ make a new abstract data type
→ avoids repitition of code

Ex. Shapes

Purpose: efficiency, non repitition + easier modification clarity.

8. Iteration abstraction

- Do not need to know the structure of the data.
- always be able to use next, hasNext ()
- can always traverse through the ~~data~~ data

Ex. BST

purpose: abstraction barrier, ease of use, removed confusion

9. Testing

- writing tests to ensure functionality and that the code ~~does what~~ works against specifications + finding bugs

Ex: Black box + white box

- make sure to get all cases + conditions according to the spec
- multiple tests per case to check for special cases

Ex: 2 next calls to an iterator ⇒ expect to be 2 places ahead

★ INTEGRAL PART ★

- write tests as you go!!!

10. equals, hashCode, toString

- equals - two objects are behaviorally equivalent. Mutable objects are equal only if they are the same objects, such types can inherit equals from Object. Immutable objects are equals if they have the same state; immutable types must implement equals themselves.
- hashCode - indicates that if two objects are equivalent according to the equals method hashCode should produce the same value for them.
- toString - should return a string showing the type in current state of its object. It don not be inherited from Object.

11. Factory method pattern

Purpose: Isolate clients from representation of data type

① static methods

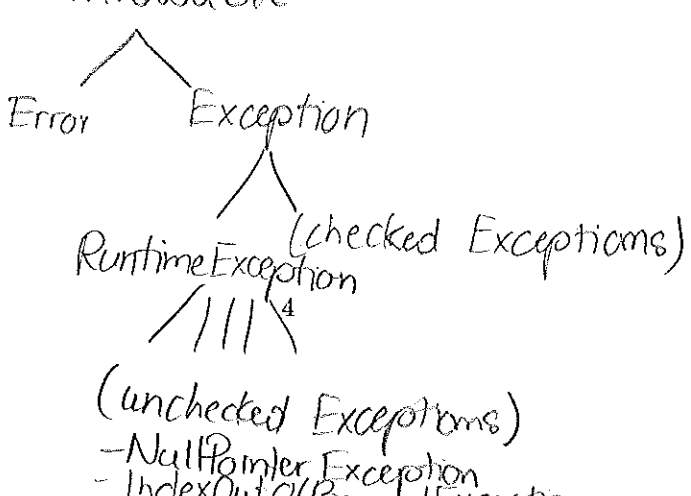
② factory objects

12. Exceptions - It allows a procedure to terminate normally on exceptional

two types :- checked - must be listed in the header

- unchecked

- you can hande an Exception by using a try-catch block.



13. Designing testing harness

- Designing a testing framework such that the designer can easily run tests of all parts of the program under various conditions and be notified of the status/validity of all tests

14. Writing algebraic specifications

- Defining the set and functionality of all methods that can be called on a specific data type.

15. Abstraction function

- A (conceptual) function that defines a mapping of one or more "concrete" representations of data to an abstract representation of the same data

Updated information based on Professor feedback:

16. Rep Invariant
Anything in your structure that can't change. Also, a property that all legitimate objects satisfy. You have to follow it when you're programming your code, then check for it in your methods. You would also have a repOk method that would verify that a certain object is a legitimate version of your data structure.

Example: In Binary Search Trees, the rep invariant having all elements left of a given element are less than it, and all elements to the right are greater than it. Additionally, the left and right are also BSTs.

// 1: c -> Boolean
// The rep invariant c.q != null && all elements of c.q are integers

---

17. Iterator and Iterable
Any class that implements Iterator must implement the hasNext(), next(), and remove() methods. Contrarily, any class that implements the Iterable interface must have an iterator method, which returns an iterator as above.

An iterator returns a generator, which is an object that accesses all elements of a collection, then return those elements in order (either the natural order, or the order to give it to sort by). Additionally, by implementing the Iterable interface, you gain the ability to use for-each loops instead of solely using while loops in your code.

Per notes on the board:
iterator -> has a method iterator(), implements Iterable interface
generator -> implements iterator

---

18. Total Order
A total order on some set D is a binary relation R on D such that:
- R is transitive:
  meaning if xRy and yRz, then xRz
- R is antisymmetric:
  meaning for all x and y, if R(x,y) = R(y,x), then x and y must be equal
- R satisfies the law of trichotomy:
  meaning that exactly one of the following is true: x < y, x > y, or x = y

19. Binary search tree

Sorted by a comparator

Can have a left binary search tree and a right binary search tree

Has a value

Every value in the left binary search tree is "less than" the value, while is "less than" every value in the right binary search tree, as determined by the comparator

The left and right subtrees follow the same rules

20. Nested classes

- cannot be accessed from outside the parent class
- has its own class file upon compilation
- Cannot create instance from it
- can access everything from parent class
- static nested classes cannot access members of the outside class, but non-static ones can
-

21. Asymptotic notation

- characterizes functions according to their growth rates
- if $f(x)$ is $O(g(x))$, then $g(x)$ is "greater than" $f(x)$ as x approaches ∞

Mathematically, there is an $x_0 > 0$ and $x_1 > 0$ such that $f(x_0) < g(x_0)$ for all $x > x_0$

$$\lim_{x \to \infty} f(x) < \lim_{x \to \infty} g(x)$$

if $f(x)$ is $O(g(x))$ then $\lim_{x \to \infty} g(x) = \lim_{x \to \infty}$

$f(x)$ is $O(g(x))$ if there exists a c and n_0 such that $f(n) \leq c\,g(n)$ for all $n \geq n_0$

$f(x)$ is $\Omega(g(x))$ if there exists a c and n_0 such that $c\,f(n) \geq g(n)$ for all $n > n_0$

$f(x)$ is $\Theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$

22. Efficiency

$f(n)$ is $O(g(n))$ = $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$ (worst-case running time)

$f(n)$ is $\Omega(g(n))$ = $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$ (best-case running time)

$f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$ ~Average case running time

rule of thumb for order of n: count loops and inner loops.

ie.

```
for(int i=0, i< list.size(); i++){
    for(int j=0; j< list.size(); j++){
        ...
    }
}// Loop within a loop, O(n^2)
```

23. Debugging

System.out.println()

Debug boolean coupled with RepOK

Override to String method in debugging

write tests first

test small chunks of code

24. Mutability

Refers to the ability to mutate values in the object without generating a new instance

Impacts the implementation (equals and hash code)

Debugging should include checks of repOK to ensure mutable structure is consistently valid

able to use "this"

Can inherit methods from Object as two mutable objects are equal iff they are the same object

Cannot write algebraic specifications

Thaila Corrington
James Jarisch
Jake Van Kennett
stephanie Lee
Zach Webert
Chris Freeley

25. Java access modifiers

public — everything can access it

private — only w/in the class

protected — only w/in package or in any subclasses

default — only w/in package

26. Abstract data types

- independent of implementation
- defined by behavior
- example: IntSet
  - defining a class that represents a set of integers
  - Set can be represented as an array or ~~a~~ vector as long as the properties of the set remain true for the client

Set of data, operations, descriptions

27. Overriding vs. Overloading

Overriding — redefining a method that a parent class defines.

Overloading — ~~two~~ multiple methods w/ the same name w/ different argument types/ different # of arguments