Winter 2019, CSE8B
Yingjun Cao

CSE 8B Final Review

This review doc intends to include concept summary from the lecture slides and cover important questions on past exams. It prepares you for the final exam (hopefully) and will help your interview on Java as well (in the future). *Created by M., feel free to collaborate and add more content.*

Official Review Materials

- Past exams and quizzes

Important Staff for Final

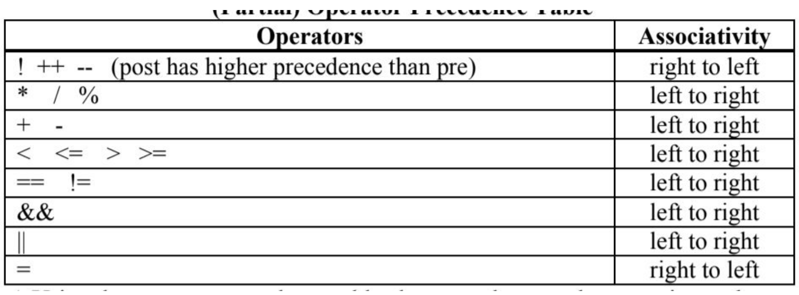
This section includes typical questions and important concepts summary. Read this section then you don't need to review anything else.

Basic Programming

- Variable Types *(FA 18, Q1)*
    - **Primitive** types: boolean, char (16 bit), int (32 bits), long, double (64 bits); these values exist on **stack**.
    - **Object**: reference exits on **stack**, points to the actual object stored on **heap**.
        - *Note: String, Arrays are all objects.*
    - Casting:
        - Auto upcasting: char is an int (ASCII), int is a double, etc.
        - Explicit downcasting: (int) charc

- **Scope** of variables *(FA 18, Q1)*
    - Primitive values (int, double, char, etc) passed in to a function is **LOCAL** to the function. The changes to these primitive values are local.
    - Reference to an object passed in to a function: changes to the reference would change the actual object.

- Operations (*FA 18, Q3)*
    - Post increment / Pre increment: x++ evalues to be the old value of x, ++x evaluates to be the new value of x as x + 1.
    - Operator precedence



(Partial) Operator Precedence Table

Operators	Associativity
! ++ -- (post has higher precedence than pre)	right to left
* / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
||	left to right
=	right to left

- Automatic type conversion:
    - Int divided by int is still an int (1 / 2 = 0 but 1 / 2.0 = 0.5).
    - Adding chars together result in int. (char a = ‘a’; char b = ‘b’; ‘a’ + ‘b’ = ASCII sum; BUT special case: ++‘a’ is ‘b’).

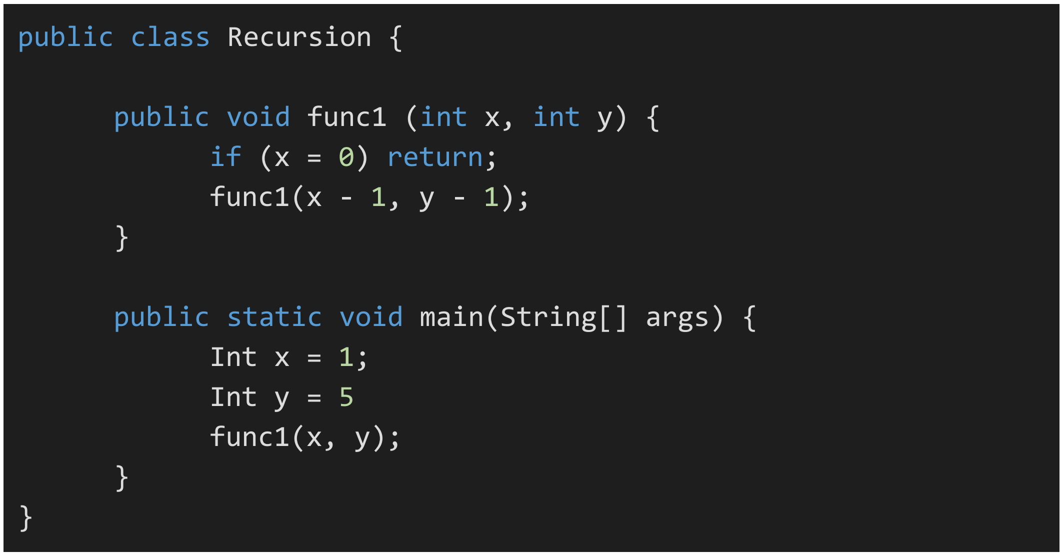
- Arrays / ArrayLists *(WI 18, Q5)*
    - Arrays are objects. **Fixed size**. (length as attribute, access by [index]).
        - Two-dimensional arrays: store reference to arrays inside an array.
    - ArrayLIst are also objects. **Variable size**, dynamic allocate memory. (add, remove, insert, size)

- **Strings** / StringBuilder *(WI17, Q5)*
    - Strings are **immutable**. Any operations on the String object won’t change the actual object, but instead returns a new String object.
        - E.g. str = “CSE”. Then str.concate(“ 8B”) returns “CSE 8B” while str still equals “CSE”
    - Compare two strings: **str.equals(other)**.
    - StringBuilders **can be changed.**

<u>Recursion</u>

- Definition: a function that calls itself directly or indirectly.

- **Question Type 1: Trace the recursive calls** *(FA 18, Q4*)
    - Stack trace: Each process (an executing program) has its own stack. Each function call creates its own *activation record* on the stack (The colored area). A function’s local variables are stored in the corresponding *activation record* of the function. When a function returns, its *activation record* is gone.
    - E.g. Suppose for the following program:



```
public class Recursion {

      public void func1 (int x, int y) {
            if (x = 0) return;
            func1(x - 1, y - 1);
      }

      public static void main(String[] args) {
            Int x = 1;
            Int y = 5
            func1(x, y);
      }
}
```

- Below as the stack area of this process.

Second call to Func1 by Func1:	*Return Address to First Call to Func1*
	X = 0
	Y = 4
First call Func1 by Main:	*Return Address to Main*
	X = 1
	Y = 5
Main:	
	X = 1
	Y = 5

- **SHOULD** know for 8B: when you are solving questions about recursion / multiple function calling, draw **separate space** for each different function call's local variables.
    - If a recursion doesn't terminate (wrong/no base case), it keeps creating new activation records, and thus overflow the stack.

- **Question Type 2: Design a recursive algorithm**
    - E.g. Sum Digits

- **Base case:** reduce the size of problem to 0 or 1. Find the expected output.
    - When size == 0, return 0.
- **Recursive case:** reduce the size of the problem by 1 to have the same problem. Assume we could solve the size (n - 1) problem, find out how to get the final result based on the solution of size(n - 1).
    - Reduced problem: sum of previous digits as sumDigits(digits from 0 to length - 1).
    - From reduced to current: the last digit + sum of previous digits.

Object Oriented Programming

- Object Oriented Programming (*Good for your interviews)*
    - Encapsulation / data hiding: both data/state and methods/behavior
    - Inheritance: inheritance of implementation (extends) and interface (implements)
    - Polymorphism: at runtime select the appropriate behavior based on the actual type of object referenced (dynamic method resolution)

- Class and Objects
    - Class: user-defined datatypes.
    - Objects: variables of those types. Their property is defined with instance variables, behavior is defined with instance methods.
    - Static variables are shared by all instances (objects) of this class. Can be accessed by an instance (obj.staticVar) or by class name (ClassName.staticVar)

- Method
    - Non-static/Static: non-static methods must be called by an instance
    - Override/Overload
        - Overload methods: exist in the **same class**, **same method name**, but different **method signature (parameters)**.
        - Overridden methods: exits in an inheritance relationship (by hierarchy), **same signature**.

- **Inheritance**

    - Provides an is-a relationship, from superclass to derived class, using the keyword *extends*. Subclass inherit everything from superclass except **private methods**, while parents' private instance variable are accessible through public super class methods.
        - Superclass specify commonality while subclass add attributes and behaviors specific to sub type.

- Scopes
    - *Public:* accessible by **any java code**. Public methods and constructors define the public interface
    - *Private:* access is permitted only in code **within the class** the member is declared. Instance variables are typically private. Private methods are often utility methods.
    - *Protected:* access is permitted from within **same package** AND from within a **subclass** of the class
    - *None:* friendly-access / package access / default access: access is permitted only from within the **same package**.

- Abstract class & Interface
    - Abstract class: used as a superclass for inheritance and polymorphism and provides inheritance of both interface and implementation.
        - Can't be instantiated and any class with at least one abstract method has to be declared *abstract*.
    - Interface: used in place of abstract class when no default implementation to inherit, with keyword *implements*. Only contains *public abstract* and *public static final.* (If *static final*, assumed to be *public static final*).
    - Single inheritance of implementation, multiple inheritance of interface. (*Q9.5, WI18, explanation here)*

- **POLYMORPHISM** (The MOST important part in 8B)

    - **Compile Time**
        - Check if right to left is **is-a** relationship
        - Based on the **type of reference**, resolves the methods signature by going up the inheritance hierarchy.
        - Once found the method signature, emit the method body.
    - **Runtime**
        - Find the **actual object**, execute the code for the **method signature** determined at compile time.
        - Start by looking at the most specific class to check if method signature exists, if not, go up the inheritance hierarchy to find any.

    - Special Case: *This/Super (Q7, FA18)*
        - This refers to the calling object (at runtime, the actual object)
        - Super refers to the relative superclass: the parent of the current class (not the parent of the actual *this* object).

    - **Casting**

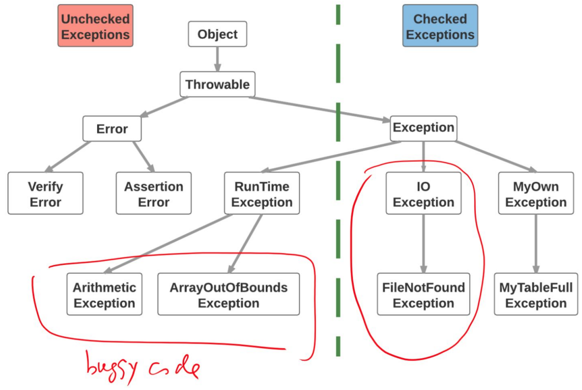
    - A subclass is automatically a superclass, but a superclass may not be a subclass.
        - E.g. A Student is a Person but a Person may not be a Student
    - **Compile Time**: superclass can be downcasted as long as the types are in a hierarchy relationship; **Run Time**: check the actual object is compatible with the type casts (**is-a)**.
        - E.g. At compile time, if p has type Person, (Student) p is valid, but (Computer) p is not valid. At runtime, we check if the actual object is the type being casted.
    - Special Case: casting of an interface. (*Q9.5, WI18, explanation here*)

- **Constructor** *(Q6, FA18)*
    - Inside out initialization: the first line is either this(args$^{\text{opt}}$) or super(args$^{\text{opt}}$), or the compiler insert **super()** to guarantee that grandparents/parents are initialized before the current child.
    - Compile error if referring to any instance variables or methods in the class or put *this(), super()* other than the 1st line.
    - For 8B, **insert the super() calls** in questions before you start.

- **Equals**
    - Check null and instanceof
    - Check the super's equals methods
    - Compare the two object's fields

Exception Handling *(Q12, FA18; Q14, WI17)*

- Exception: an unexpected event that occurs during the runtime of a program.
- Error: serious problems that a reasonable application should not catch.
- Exceptions and Error are all subclasses of Throwable.
- Exception has many subclasses for specific exception: ArithmeticException, IOException, etc.



- To handle exception, use *try, catch, finally*

```
public class ExceptionExample {
      public static void main (String[] args) {
          try {
              int x = 3 / 0;
          }
          catch (ArrayIndexOutOfBound e) {
              System.out.println("Index");
              throw e;
          }
          catch (ArithemticException e) {
             System.out.println("math");
             throw e;
          }
          finally {
             System.out.println("Finally here");
          }
          System.out.println("No exception caught and thrown");
      }
}
```

  - Catch block executes **if and only if** a corresponding exception (of a subclass of the corresponding exception) occurred and no catch block above has caught that exception.
  - Finally **always executes** after try (if normal behavior) or catch (if an exception occurs).
  - The code below finally executes **if and only if**
    - The code executes normally **or**
    - An exception is caught but not thrown.

GUI & EventHandler

- Javafx
  - All javafx applications **extends** from Application and overrides **public void start(Stage** primaryStage**).**
  - Levels: stage → scene (container of all items in the scene) → helper class that organizes the layout (Border, HBox, etc) → component (TextField, etc.)

- Event Handling
  - **Event**: treated as an object in java. Many subclasses specifying different types of events.

        - E.g. KeyEvent, ActionEvent, etc. Refer to the question statement in Final
    - **EventHandler**:
        - Event Handlers for a specific type of event **implements EventHandler <TYPE of Exception>** interface.
        - Event handler overrides the **public void handle(EventType e)** method.

<u>Vim / Unix Commands</u>

File related	Write(save) a file	:w
	Quit a file	:q
	Force quit without saving	:!q
	Find a specific pattern	/pattern
Edit	Append	a
	Delete entire line	dd
	Copy	y
	Paste	p
	Move around (left, down, up, right)	h, j, k, l
	Undo	u
	Auto-indent a file	gg=G
Directory	Make a new directory	mkdir
	Change directory	cd
	Remove a directory	rm -r
	Move or rename a file	mv

- Examples
    - Delete three words: 3dw

Winter 2019, CSE8B
Yingjun Cao

Bullet Points Summary

Lec 2 - Basic Coding

- Coding habit
    - Test cases → Design on Paper → Test on Paper → Write code and Debug
    - Test cases first, implementation second
    - Test in a separate file
    - Backtrace to locate problem

- Evaluating Expressions
    - Strictly from left to right
    - Post/pre increment and decrement

- Short Circuiting
    - &&, ||

Lec 3 / 4 - Arrays

- Initialize arrays with new int[n] → on the stack, all default 0.
- Reference to array

- 2-D array: each array index stores an array.
    - Int [][] arr = new int[i][j]
    - Assign value each entry of array by loops.

- Array of primitives vs. Array of reference

- Coding style

Lec5 - String / StringBuilder / ArrayList

- String
    - Immutable **objects**
    - Return new string if method “changes” the string
    - E.g. new String(“blbabla”) vs. “blabla”
    - **==:** point to the same object
    - **Equals:** check if the data is the same

- Chars as **16 bit values**
    - Caution: local variable when replace chars.

- StringBuilder
  - Mutable objects
  - Can be changed via method calls

- ArrayList
  - Store any type of **object** (Integer instead of int, e.g.)
  - Dynamically change size
    - ArrayList <Type> name = new ArrayList<Type>()
    - name.add()
  - Methods like add, remove, get, size, etc

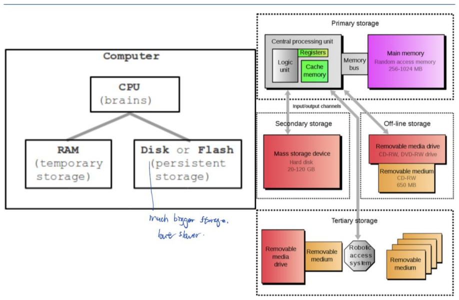
- Casting
  - Double, float, int, char, short, boolean
  - “Char is int, but int is not char”

Lec6 - Array / ArrayList

- Array**s** class include many useful static methods
- Import Arrays from the java library
- Sort() by the alphabetic order, note upper and lowercase.

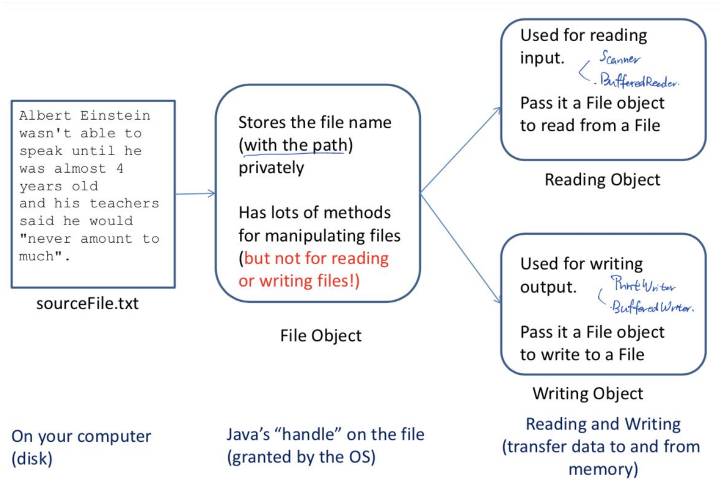
Lec7 - File Input / Output / BufferReader

- Computer structure



much bigger storage.
but slower.

- Workflow:



Albert Einstein
wasn't able to
speak until he
was almost 4
years old
and his teachers
said he would
"never amount to
much".

sourceFile.txt

Stores the file name (with the path) privately

Has lots of methods for manipulating files (but not for reading or writing files!)

File Object

Used for reading input.
Scanner
BufferedReader.

Pass it a File object to read from a File

Reading Object

Used for writing output.
PrintWriter
BufferedWriter.

Pass it a File object to write to a File

Writing Object

On your computer (disk)

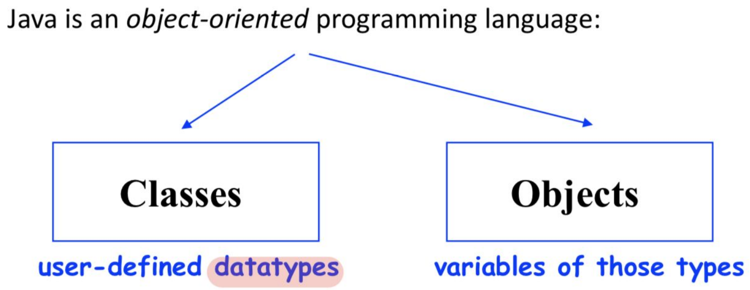
Java's "handle" on the file (granted by the OS)

Reading and Writing (transfer data to and from memory)

- Reading
    - Scanner (formatted)
        - hasNextLine(), nextLine()
    - BufferedReader (byte by byte)
- Writing
    - PrintWriter (formated, immediate write)
    - BufferedWrite (byte by byte, buffered write)

- I/O exception

Lec 8 - Class and Objects



Java is an *object-oriented* programming language:

Classes

Objects

user-defined datatypes

variables of those types

- Objects
    - Property: instance variables
    - Behavior: instance methods (constructor as a special kind of method)

- Method
    - Calling static or non-static methods
    - This (as the calling object)

Lec 9 / 10 - Inheritance and Scope

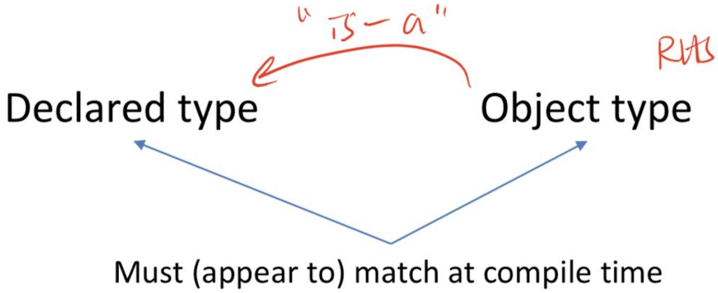
- Keyword: **extends**

- Support single-rooted inheritance
    - Provides an **is-a** relationship
    - Superclass/base class → derived/subclass

- Inherited everything except: **private methods**
- Private instance variable are accessible through public supper class methods

- Override: signature doesn't change, the method is from super class
- Overload: multiple methods in a class share the same name but different parameters

- Variable types and objects types



RHS is a subclass of LHS.

- **Object-oriented Programming** (*very important)*
    - Encapsulation / data hiding: both data/state and methods/hehavior
    - Inheritance: inheritance of implementation (extends) and interface (implements)
    - Polymorphism: at runtime select the appropriate behavior based on teh actual type of object referenced (dynamic method resolution)

- **Scope keyword**
    - **Public:** accessible by any java code. Public methods and constructors define the public interface
    - **Private:** access is permitted only in code within the class the member is declared. Instance variables are typically private. Private methods are often utility methods.
    - **Protected:** access is permitted from within same package AND from within a subclass of the class
    - **None:** friendly-access / package access / default access: access is permitted only from within the same package.

- **Dynamic binding**:
    - Compile time checking, runtime checking

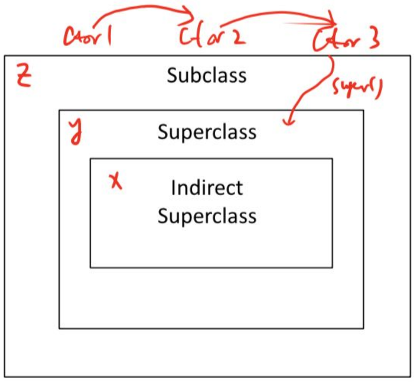
    - First look for methods in its own class, then super class.
    - Super and this

Lec 11 - Constructor

- **Initialize vars in a newly allocated array**

- Special method:
    - Same class name
    - No return type

- If a class has no constructor, the compiler automatically insert to the first line.

- Fist line
    - Either: this(args opt) or super(args opt)
    - Or default super()
    - 

- InstanceOf

LEC 12 - Constructors

- **Constructor**

    - Inside out initialization

    - First line
        - Either this($args_{opt}$) or super($args_{opt}$)
        - Else compiler inserts super () as first statement

    - Note: access modifier (*private* field of parent)

    - Compile Error
        - Refer to any instance variables or methods in this class

        - Put `this()` or `super()` other than the 1st line

- **Equals**

    - Operator: instanceof to test object type

    - Has-a Vs. Is-a relationship
        - Inheritance Vs. Composition

    - Check the inner states of objects

<u>LEC 13/14 - Polymorphism</u>

- Classes
    - Super: specify commonality
    - Sub: add attributes and behaviors specific to sub type
        - `Override`

- Any subclass **is-a** specialized type of the super class
    - Right to left **is-a**
    - Treat superclass types and subclass types similarly

- Procedure
    - Compile time:
        - method signature resolution
        - Knows only the **type of reference**
        - Emit code for that method signature it finds
        - Only look at the class definition

    - Runtime:
        - dynamic binding
        - Find **actual object**
        - Execute code for the **method signature** determined at the compile time in the **actual object**

- Casting

    - Only compiler's view of type, not actual object

    - Checking
        - Compile time: **explicit cast** for downcast
        - Runtime: check the actual object is compatible with the type cast (**? is-a**)

- Overloading vs. Overriding
    - Overload: same name, different signature in the **same class**
        - Resolves at compile time
    - Overriding: same signature in **type hierarchy**
        - Resolves at run time

LEC 15 - Abstract Class

- Anti-polymorphism
    - **Final**: class, method
- **Abstract Class**
    - Ideas
        - Used as superclass for inheritance and polymorphism
        - Provides mixture of
            - Inheritance of interface
            - Inheritance of implementation
    - Contains at least **1** abstract method
    - No instantiation
    - `Abstract, extend`
- **Interfaces**
    - Used in place of abstract class when no default implementation to inherit
    - Keyword: **interface** (interface is a type), **implements**
    - Only declare **public abstract** and **public static final**
- Single inheritance of implementation, multiple inheritance of interface

LEC 16/17 - Recursion

- Functions calling themselves
    - In every step, problem reduces to a problem of **exactly the same nature**
    - Until some specific (small, base) point where answer is easy and obvious

- Stack frames and space

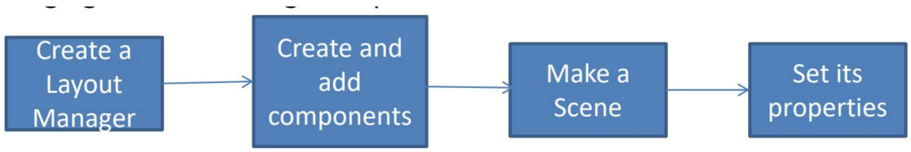
- Examples:
    - Search: linear, binary (sorted)
    - Fibonacci sequence
    - Find sums/averages

<u>LEC 18 - Graphics U/I</u>

- JavaFX (vs. flash, html5)

- Levels
    - Stage: top level component
    - Scene: container for all items in the scene
    - Border, HBox, …, Group: helper classes that govern where components appear in the scene

- Creating GUI



    - Layout manager: border pane, hbox, vbox, stack pane, flow pane
    - Items: text area, image viewer, shapes

<u>LEC 19 - GUI/Events</u>

- * color class constructor parameters

- Property Binding
    - object.property().bind(other.property())

<u>LEC 20 - Event Handling</u>

- Event

- Event Handling
    - As a separate class
    - Handler as an object with override “handle” method

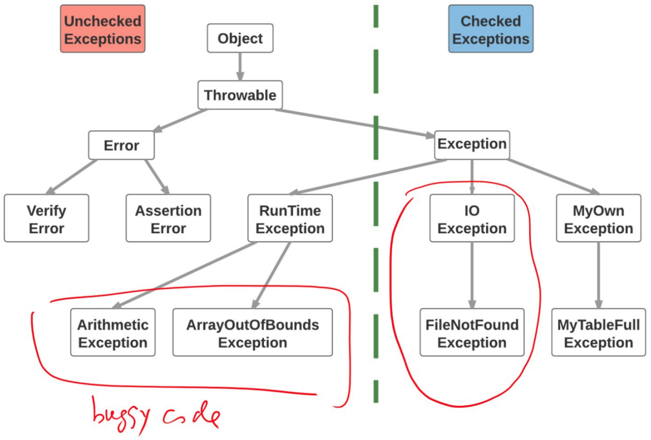
LEC 21 - Hashmaps / Sets

- Keys and value pairs
    - No order
- HashMap<String, Integer>
    - Put()
    - Get()
    - keySet()
    - contains()
- HashSet
    - No order
    - Add, delete, search fast

LEC 22 - Exception Handling

- Exception: unexpected behavior when running a program
    - Treated as an object in java
    - Many derived classes for different types of exception, including ArithmeticException, IOException, etc.

- Error handling:
    - Try, Catch, finally, throws
    - Exception type has to match



- Types
    - Error: JVM error (memory)
    - Checked exception: explicitly caught or thrown (sth unexpected)
    - Unchecked: no need to explicitly catch or thrown; normal course of execution

- Order of catch: sequential
    - To catch an inheritance relationship of exceptions, put the most derived class first.

- Quitting: final
- Definition
    - Process: A program in execution. Each with its own logical memory (Text, data - heap + static variables, stack)
    - Thread: a path in execution (each with its own stack divided from the process' stack).
        - Share resources (data, text)

- Why Thread:
    - Avoid blocking on background actions
    - Improve efficiency

- Threading vs. Parallel Programming
    - Threading: launch a thread to take care of something which may take awhile.
    - Parallel Programming: have more than one computing resources trying to divide up the task

- Parallel Execution
    - Kernel level thread: one core runs one thread at a time.
    - E.g. Divide the parallelizable parts by min{#threads, #cores} + serial parts

- Implementation
    - Interface: Runnable
        - Method: **public void run(){}**
        -
    - Interface: Thread
        - Method: start(), sleep(), join(), etc..
        - Use separate loop to start and join all threads. Each iteration, one thread start/join.

- Examples:
    - Parallel Print
    - Parallel computation

- Synchronization
    - Race condition: some operations may not be atomic.
    - Single core / multiple core.
    - Resolve:
        - Synchronized keyword for method
        - Lock
        - Semaphores

- Amdahl’s law