CISC124 Exam Study Notes

History Of Java

- Language was first developed by James Gosling at Sun Microsystems in 1991
- He was designing a language called “Oak” for “Green Project”
- Green Project envisaged the centralized control of many processor-based devices in home
- “Oak” was designed to be a robust, efficient language with maximum profitability to different processors
- Green Project flopped

- In early 90s, Internet was blossoming. Web pages had to do more than just display static text and graphics
- Needed dynamic and interactive content
- But web pages are viewed on a wide variety of platforms, from Mac’s to Unix to IBM-PCs
- So any page-embedded language would need to run on all platforms
- Needed a **robust, compact, and multiplatform** language, so Oak was dusted off and call it something racy, like Java!

- In 1994, Sun demonstrated the use of Java in small bundles of code embedded in a web page called **applets**
- Netscape browsers started supporting applets in 1995, starting Java’s rise to fame
- Sun programmers continued to develop a code base for the language, adding many libraries
- They showed that Java could be used for more than just applets, and that full-blown application could be written in this high-level language

Recent History

- Early 2010, Oracle acquired Sun Microsystems
- Oracle seems to be sticking with their promise as demonstrated with recent released of Java 7 and 8
- The second “incarnation” of JavaFX has also been released replacing Swing

How Java Works

- Java standard (the syntax) is identical for all platforms
- A **compiler** (part of JDK sometimes called javac.exe) which is designed to run on your development platform, compiles your **source code** (*.java file) to a **byte code** file (*.cass file)
- The byte code file is **platform-independent** and is the thing you attach to your web page as an applet
- Every browser written or every platform and OS can have an embedded code processor called **JVM** (Java Virtual Machine) built in

- The JVM takes the byte code and executes it by generating the machine code that will be recognized by the platform that is running the browser
- Later, people took the JVM out of browser so they could run stand-alone Java applications. This is the **JRE** (Java Runtime Engine) (java.exe)
- The concept of write once, run anywhere is very appealing

- However, all IDE’s must use the appropriate JDK in the background
- Two components of JDK are: **javac.exe and java.exe**
- javac.exe is the byte code compiler and java.exe is the JRE which executes the byte code file

- Compilation is the process of converting the *.java file to a *.class file (the byte code file) this is done by calling javac.exe in the background, and supplying that program with all the required command line parameters

Java.exe Program:

- accepts the byte code file
- links in any required libraries
- creates executable code in memory
- converts it to machine language
- and sends it to the CPU

- The java.exe program must know the right machine language commands for just the type of CPU it is designed for

OOP in Java

- A class or "object definition" or an "object" consists of **instance variables and or methods**
- By convention, instance variables are all declared before the methods
- In Java, a **class** is an **Object**, and an **Object** is a **class**
- Code and attributes cannot be defined outside of a class
- The only code that can exist outside a method are attributes or other ("inner") class definitions

Attributes

- Also called **class variables** or an **instance variables** or **fields**
- Declared within a class at the same level at the method declaration
- These variables are known to all methods within a class (their "scope")
- You can control their privacy and the way they are stored in memory

Access Modifiers

public: means the attribute or method is available to any external class (as well as inside the class)

private: means that the attribute or method, the "member" is only available inside the class which it is declared

protected: means the member is only public to classes in the same package as the class in which the member is declared

static (First Pass)

- static means different things depending on where it is used
- public static members are available outside the class without the need to instantiate the class
- Any static member remains in memory until the program is complete
- Since main is static, it can only invoke other static methods when they the in the same class

Attribute Declaration

Syntax:

[private|public] [static] [final] type attributeName [= literalValue];

- Note that the type part is not optional — this is why java is a "declarative" language
- And a variable cannot change its type later ("static typing")
- You cannot use a variable unless you have declared it first

Variable Declaration
- Declaring a variable inside a method gives that variable the SCOPE of just inside the method, not outside the method
- Generally, a variable is only available inside the block {…} in which it is declared

Method Declaration ("Header")
Syntax:
[privatelpublic] [static] [final] returnType methodName ([parameterList]) {…}
- If main invokes methods or uses attributes in the same class as itself then those attributes and method must also be declared static
- A method must have a returnType
- The returnType can be any single Object or a primitive type
- If the method does not return anything, then the keyword void is used instead
- parameterList provides a means of passing items, or parameters into a method
- it can consist of one or many parameters, separated by commas

Main Method
- For the JVM to run an application, it must know where to start
- By design the starting point is always the execution of the main method
- The JVM expects the main method to be declared exactly as shown — the only thing you can change is the name of the String array, called args above

Methods - the return Statement
- Also important: Unless the return type is void, the method must contain at least one return statement. A void method can also use a return statement without anything after it, as a means of termination of the method
- A method always stops (terminates) when a return statement is encountered

Primitive Types in Java
char byte short int long float double boolean

- Everything else in Java is an Object
- Object construction often involves the **data abstraction** of several primitive type attributes

Integer Primitive Types
- Byte short int long
- For Byte: from -128 to 127 inclusive (1 byte)
- For Short: from -32768 to 32767 inclusive (2 bytes)
- For int: from -21347483648 to 2147483647 inclusive (4 bytes)
- For long: from -9223372036854775808 to 9223372036854775807 inclusive (8 bytes)
- A "byte" is 8 bits, where a "bit" is either a 1 or 0

Number Ranges
- Memory limitations and the system used by Java (two's complement) to store numbers determines the actual numbers
- The Wrapper classes can be used to provide the values

Real Primitive Types
- Also called “Floating Point” Types
- float or double
- For float: (4 bytes) roughly
- For double: (8 bytes)

Character Primitive Type
- char
- From ‘\u0000\ to ‘\uffff’ inclusive that is from 0 to 65535 (base 10) or 0 to ffff (base 16 r hexadecimal)
- A variable of the char type represented a Unicode character, can also be represented as ‘a’ or ‘8’ etc

Boolean Primitive Type
- Boolean is either true or false

String Objects
- String’s are not primitive data types, but are **objects**

Array Declaration
- **new** is always involved with the instantiation of an Object
- Arrays are Objects in Java
- An array literal is just a set of same-type values separated by commas in a set of { }
- To get the size of array use the .length attribute
- Array indices range from 0 to .length - 1
- Cannot dynamically size an array in Java
- In Java, you cannot use pointers to access array elements only the indices
- You can never obtain a memory address in Java
- The first array element is at index 0
- In Java, you will get an immediate error and your program will halt if you try to go beyond the bounds of an array

Literal Values
- Integers for ex: 12 -142 0 333444891
- If you write these kinds of numbers into your program, Java will assume them to be of type “int” and store them accordingly
- If you want them to be type “long” then you must append a “L” to the number

43L 99938475L -22223487L

- Real or “Floating Point” numbers: 4.5 -1.0 3.457E-10
- These literals are assumed to be of “double” type
- If you want them to be stored as “float” types, append an “F”
- Ex. 3.45F

Binary, Octal and Hex Literals

- Use the prefix (0b or 0B) in front of the numbers to get a binary literal
- Octal, just use —0
- Hex use 0x
- You can also use the underscore _ in-between digits to aid in visualizing number
- System.out.println() by default displays the numbers in base 10
- Use printf to show the numbers in another codex or base

Legal Variable Names

- Java names may contain any number of letters, numbers and underscore characters but they must begin with a letter

Standard Java Naming Conventions:

- Names beginning with lowercase letters are variables or methods
- Names beginning with uppercase letters are class names
- Successive words within a name are capitalized (Camel Case)
- Names in capital letters are constants

Variable Declaration:

- Java may prevent you from using variables that are not initialized
- So it is often good practice to initialize your variables before use

Constant Attribute Declaration

Syntax:

[private|public] [static] final type ATTRIBUTE_NAME = literal_value;

- The java keyword **final** can be used to make sure a variable value is no longer “variable”
- Usually they are declared **public static**
- The value must be assigned

- Java will not allow your program to change a constant’s value once it has been declared
- Note that constant names are all in upper-case by convention

Type Casting

- When a value of one type is stored into a variable of another type
- Casting of primitive types in one direction is automatic, you do not have to deliberately or “explicitily” cast
- A value to the left can be assigned to a variable to the right without explicit casting:]

byte > short > int > long > float> double

Casting Operator

- If you really want to cast in the other direction, then you must make an explicit cast

int anotherVar = (int)345.892;

is legal, the “(int)” part of the statement casts the double to an int

- Note how numbers are truncated not rounded!

What Makes an Expression?

What are all the components available to a programmer to use to put a line of code togher?

- variables
- literal values
- keywords
- operators
- method invocations
- Punctuation

Arithmetic Operators

- The standard binary arithmetic operators in Java are:
  Addition, Subtraction, Multiplication, Division, Mod (%)
- All these operators apply to all numeric primitive data types
- All require values on both sides of the operator (why they are called binary operators)
- Java does not have ** or ^ (exponentiation), have to use **Math.pow(x,y)**

Integer Arithmetic

- Arithmetic operators between integers produce integer results by truncating the answer, fractional parts are discarded
- If you have an integer on both sides of an arithmetic operator, the result will be an integer

Mixed Type Arithmetic Expressions

- Suppose you have a "mixed type" expression involving an arithmetic operator
- To evaluate the expression, Java will cast one side to match the other
- For ex: if one side is an int and the other side is a double, the int will be automatically cast to a double before the operation takes place

Strings and the "+" Operator

- Not only can "+" operate on numeric values, but it can also handle String's on either or both sides
- If one side is not a String, it will be changed to one, and then it will be concatenated to the String on the other side:

Ex: 4 + "you" stores as "4you"
"apples" + "oranges" + 9 + 9 stores as "applesoranges99"
3+7+"little piggies" stores as "10 little piggies"

- Expressions are evaluated from left to right unless precedence rules apply

Unary Arithmetic Operators

- Unary operators include "+" and "-" where -aNum negates the value produced by aNum
- They also include the increment (++) and decrement (- -) operators which increase or decease an integer value by 1
- **Pre increment** and **Pre decrement** operators appear **before** a variable. They increment or decrement the value of the variable before it is used in the expression
- **Post increment** and **Post decrement** operators appear after a variable. They increment or decrement the value of the variable after it is used in the expression
- It is better to keep increment and decrement operators out of larger expressions

Assignment Operators
= set equals to
***=** multiply and set equal to
/= divide and set equal to
-= subtract and set equal to
+= add and set equal to

- An assignment statement in Java has form: varName = expression;
- The expression is evaluated, and the result of the expression is stored in the specified variable

Logical Binary Operators
- Return either true or false

== equals to
!= not equals to
> greater than < less than
>= greater than or equal to
<= less than or equal to
&, && logical "And"
|, || logical "Or"

- The only unary logical operator is "!"
- Called the "not" operator
- It reverse the logical value of a boolean

Aside: "|" or "||"
- A single | or & ALWAYS evaluates both sides of the expression whether it is necessary to not
- && stops if the left side is false, || stops if the left side is true

Precedence Rules
- Operator precedence rules determine which operations take place in what order:
  - Unary operators and casting are done first
  - Then *, /, %
  - Then +. -
  - Then >, <, >=, <=
  - Then ==, !=
  - then &, &&, |, ||
  - Then =, *=, /=, -=, +=
- Use "()" to control order of operations as the expression inside ( ) will be evaluated before stuff outside of ()

Expressions
- Expressions are combinations of variables, literal values, operators, keywords, method calls
- What are all the components available to a programmer to use to put a line of code together?
  - variables, literals values, keywords, operators, method invocations, punctuation

Method Invocations
Three Aspects to Consider:

1. Naming the method
2. Providing argument(s) or not
3. Doing something with the return value or not

1. Naming the Method

- if the method is in the same class, then just use the name of the method

(the compiler assumes the existence of the object reference to the current object as this. or it assumes the name of the current object, when static)

- If the method is not the same class you must first identify the object owning the method and obtain the method using the dot operator
- If the method is declared as static, then you can invoke the method without instantiating the class that contains the method

2. Providing Arguments for the Parameters

- If the method has been declared to accept arguments
- if the method does not have any parameters, then you must still use empty brackets when you invoke the method

3. Using a Return Value

- A non-void method will return something
- you can use that "something" in an expression, or just store it in a variable
- The method has declared the type of that "something"
- If the method was declared as void, you will not get a return value and you can only invoke the method by itself, not as part of an expression or an assignment statement

Punctuation - Java Whitespace

- Multiplace spaces are treated as one space
- Leading and trailing spaces are ignored
- Tabs and empty lines (line feeds) are ignored
- Line Continuation:
  - Long lines can be continued on the line below — break the line anywhere there is a space, but not in the middle of a string
  - For good stype, indent the line to show it is a continuation

- Comment: inline //, block /*…*/
- Comma , used in parameter lists and array literals
- Semicolon ; used to end a statement and with for loop syntax
- Colon : used with switch statements and "for each" loop
- Period or "dot operator" . used with objects to obtain members
- Also [ ] , ( ) and { }

Conditionals or "Selection Statements" : if - else statement

Abbreviated if Statement

- Uses the "ternary operator" - ?
- expression1 ? expression2 : expression 3
- Expression1 must evaluate to boolean
- expression2 (when true) and expression 3(when false) must evaluate to same type

Dangling else Problem
- It is not unusual to nest if statements inside each other
- One issue that can arise is the “Dangling else” problem
- Indentation might give a visual clue, but has no syntactic meaning
- The else should be associated with the closest if
- Use { } if necessary

Switch Statement
- The code to be run depends on which val# value matches expression
- If none match, the statements after the default: clause are run
- The expression and val# values (of “Case Labels”) must all be of the same integer (or char) type
- The break; statements make sure that following cases are not executed after a match has been made
- It is possible to do multiple cases on one line, but it is clumsy

Repetition or Using “Loops”
Java has: while, do/while, for, the for each loop

Loops (Misc)
- Don’t declare variables inside loops, as the repeated declaration process uses up time and memory necessarily
- There is not limit in Java to how many levels you can nest loops
- It is customary but not necessary, to use the variables i,j,k as loop counters when the counter has no intrinsic meaning

Other Java Keywords used with Loops
- The **continue** statement interrupts the execution of a loop and returns control to the top of the loop
- The **break** statement interrupts the execution of a loop, and transfers control to the first statement after the loop
- Only use these keywords when it makes your code easier to read
- Avoid the use of more than one break or continue inside a loop
- Overuse of break statements can lead to “spaghetti” code

Building Modular Code
- Easier to build, test, debug, modify and share
- Building objects just takes modular design to the next level

Designing Functions
- Functions are written o avoid repeating code so, make sure you only have one function to do the “one thing”
- Functions should be short: if you can satisfy all the other rules and the code still explains itself, then the function is short enough

Functions should only do one thing and do it well
- Keep all code within the function at the same level of abstraction
- Your program should be readable as a top-down narrative. The most abstract functions will be at the top of the program leading to the least abstract functions further down

- Use less than three parameters whenever possible
- Try to use parameters for input only
- Flag parameters are ugly, they are saying that the function does at least two things
- If needed, multiple parameters can be grouped into an object or list

- The function should not spawn any side effects
  - such as changing the contents of variables passed by reference
  - or spawning off another process that is at the same level of abstraction as function itself
- A function should either do something or answer something, not both

Some Useful Java Classes
- The classes defined in the java.lang package are automatically imported for you
- They include:
  The Wrapper classes, Math, Object, String, System, Thread

static Methods
- static attributes and methods are loaded once into memory and not garbaged collected until main is finished
- These methods will run faster the second time (and later) they are invoked
- Generally they are utility methods that do not depend on the values of a class' attributes
- static methods can be invoked without instantiation of the Object that owns them

- static methods and attributes are shared by all instances of a class, there is only one copy of these methods in memory
- A static method can only invoke other static methods in its own class — you can't have pieces of code disappearing from a static method in memory
- This is all done for reasons of ease of use and efficiency