Inheritance

1. Lack of multiple inheritance.
2. Hard to trace hierarchy.
3. Can superclass changes affect subclass behavior? Favor composition over inheritance if possible.
4. Good – reuse code. Open – closed principle.

**Exception Handling**

1. Know try, catch, finally, throw, throws
2. Exceptions can be caught with try – catch or passed up the stack and declare that method throws the exception.
3. Understand hierarchy – Throwable, Error, Exception, RuntimeException as subclass of Exception
4. Checked vs. unchecked exceptions
5. Custom exceptions
6. Multiple catch blocks – order is importance
7. Multiple catch blocks that are chained with “|”
8. Ways to use super class constructors – ex: get the root cause of an exception if we throw a different exception in a catch block.
9. Do not swallow exceptions – empty catch blocks

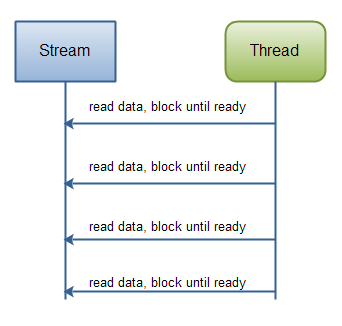
**Threads**

1. Multithreading, multitasking,
2. Thread class, Runnable interface, run method in both
3. Thread life cycle – thread likes to be in runnable state then run
4. start method calls run method to start this process.
5. Thread methods – among them join, start, run, currentThread many others

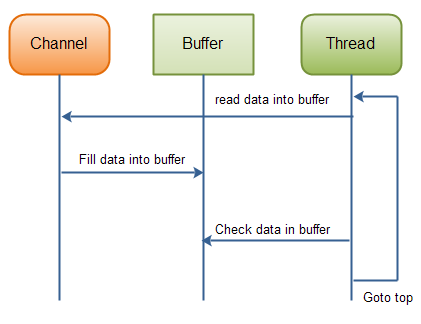
wait, notify, notifyAll from Object and synchronization of objects, methods

1. Thread constructors and use of overloaded constructors

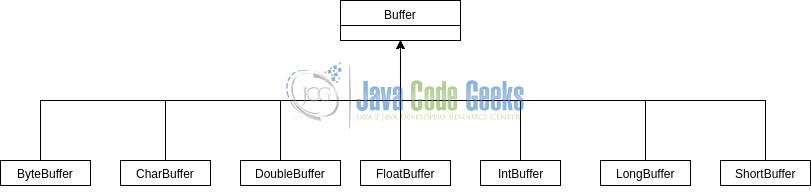
**IO**

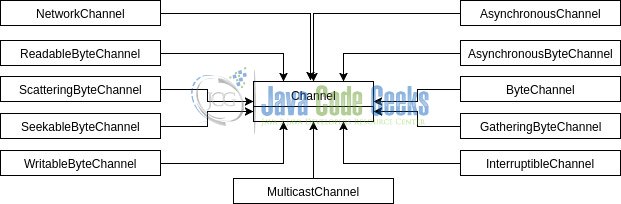
1. IO related to streams – Reader, Writer, InputStream, OutputStream top of hierarchy for character and byte data respectively.
2. Buffering improves performance.
3. Nice IO API based on Decorator design pattern.
4. Classes specialize in certain functionality and Java types like DataInputStream, ObjectInputStream and related output streams, etc.
5. Streams and Threads – blocking for IO
6. 

NIO – non-blocking IO. Thread does not have to wait for data to read and write as with IO. Non-blocking mode enables a thread to request reading data from a channel, and only get what is currently available, or nothing at all, if no data is currently available. Rather than remain blocked until data becomes available for reading, the thread can go on with something else. So this is different than IO.



**Channels and Buffer Hierarchy**





**Java 8 Features**

1. Lambdas – like anonymous inner classes. Lambdas can be substituted for functional interfaces (interfaces with just one abstract method). Functional interfaces can have static, default methods and methods related to Object class.
2. Method references
3. Streams – All Collection types can use stream() method to use the Stream methods. Know about intermediate and terminal Stream methods.
4. Do some research on java.time packages and sub-packages.
5. Optional

First Section Test Core Java 200 pts. total

1. 10/25 OOPS and core Java may be some log4j2/JUnit
2. Algorithm coding problem – one random problem of many
3. OOP Java Coding – one random problem of many
4. Bash Coding – one random problem of many

Important points:

1. You should review the read and readarray command in Bash Linux. Please understand the structure and syntax you need to access elements of an array, how to manipulate data, etc.
2. The algorithm coding problem will be like typical HackerRank problems you have seen. The OOP coding problem will test your knowledge of general coding, class structure, methods, variables, inheritance, polymorphism, encapsulation and general principles.
3. The Bash Linux coding problem will test your knowledge of Linux commands. Know about pipes, reading variables and arrays, sed and/or awk, and this should be similar to what we covered in class.
4. The MCQ’s cover your general knowledge of the subject. Many of the questions are coding questions and you must pick the correct answer.
5. Understand bit AND & and bit OR | operators and how they work with binary numbers 0, 1. Convert hexadecimal numbers to binary to work with these operators. For example to break down an int in Java that begins with 0x like: int j = 0x33, equal to 51, we could write in binary by getting the value of each of the hex digits (3, 3) and writing it as a binary number using 4 columns. In binary, 3 = 0011. So 0x33 = 00110011 = 51 in decimal.
6. We have seen we can create an instance of a class by using the “new” keyword. Design patterns are code structures in general that solve problems that we can get if we write poorly designed code. We know about abstraction and how we implement something from the user or client. Check the NumberFormat class for example and look at the getInstance() method. This returns an instance but the way it is created is not revealed. If you check some of the basic design patterns and what pattern actually is related to “building”, what pattern does this class seem to be demonstrating?