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

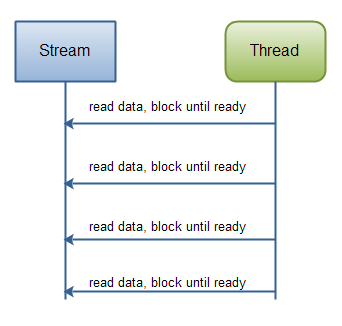
**Threads**

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

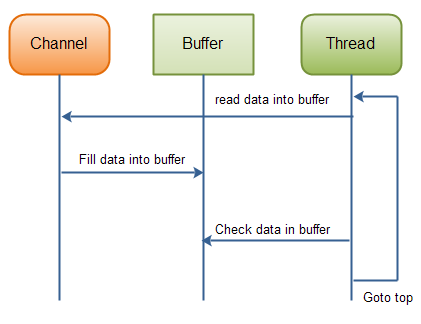
wait, notify, notifyAll from Object and synchronization of objects, methods

1. Thread constructors and use of overloaded constructors

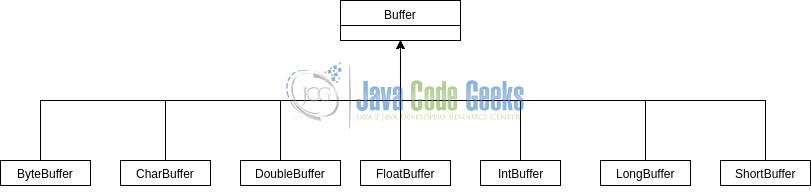
**IO**

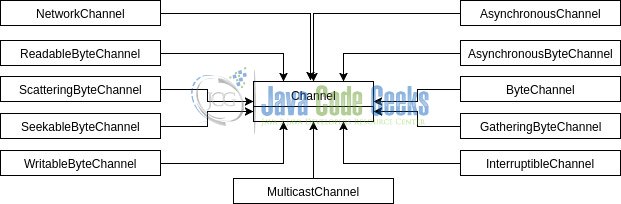
1. IO related to streams – Reader, Writer, InputStream, OutputStream top of hierarchy for character and byte data respectively.
2. Buffering improves performance.
3. Nice IO API based on Decorator design pattern.
4. Classes specialize in certain functionality and Java types like DataInputStream, ObjectInputStream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

**Logging and Testing**

1. Check Apache log4j site at <http://logging.apache.org/log4j/log4j-2.12.1/index.html>
2. Log4j2 primer reference from site above is not bad. There are many good links at this site and of course others on Internet and in O’Reilly, Percipio.
3. Summary for logging:
4. Make sure classpath (dependencies in Maven) have correct log4j2 dependencies core and api. Check mvnrepository.com or copy them. Update Maven project in Eclipse or other IDE. If we use Maven exclusively do not need this.
5. Create log4j2.xml or properties file. Put it where? src/main/resources. We can use one of many templates in tutorials, actual code, on Internet, etc. Then we can “tweak” it.
6. We can now use import statements to get any Logger LogManager, etc. in our Java files.
7. Create logger and start using log levels.
8. Summary for JUnit testing:
9. Make sure classpath (dependencies in Maven) have correct Junit Jupiter dependencies like junit-jupiter and junit-jupiter-engine. In addition the sure-fire-plugin is good for generating test reports in the target directory of Maven. Check mvnrepository.com or copy them. Update Maven project in Eclipse or other IDE.
10. Add tests in src/test/java in Maven project. The project package structure should be the same as your source packages in src/main/java. This is a best practice.
11. Each test must have a @Test annotation. All test methods except parameterized test methods (have examples in Teams but we did not discuss much) have void return types and no parameters.
12. Obviously there is more to testing but these steps start the process.

**Database and SQL**

1. Database concepts – normalization, cardinality, ERD, primary key, foreign key, data integrity, constraints,
2. SQL – DML, DDL, DCL
3. SQL – w3schools.com provides good outline and examples in database, tables.
4. JDBC

**Servlets and JSP**

1. Edit index.jsp to be like the ppt welcome file with two text fields and a button.
2. Add a servlet that is accessed by jsp file in #1.
3. Make sure your action attribute in JSP file is consistent with your name in the servlet (preceded by “/”).
4. Test it.
5. HTTP methods like GET, POST, PUT, DELETE, etc.
6. Session Handling – hidden form fields, cookies (not advised), URL rewriting, HttpSession. HTTP is stateless protocol.
7. Life cycle of servlets
8. Filters, Listeners (Listeners not emphasized but can be worthwhile)
9. JSPs are Java.
10. Tag libraries can be important here or other view technologies like Thymeleaf. TLD or Tag Library Descriptors.
11. Tomcat structure both embedded and actual on disk. Configure in Eclipse.
12. WAR files and structure, structure of WAR files.
13. 404 and 500 errors – meaning of these types of errors.
14. JSP tags – declarations, scriplets (try to avoid), directives, expressions.
15. JSP tags – other JSP tags. Don’t memorize but know how to look up and use if you need them.
16. Roles of servlets, JSPs, Data layer.
17. Things to expand and/or check:
18. Just start like we did with creating the welcome-file (index.jsp).
19. Then work on servlet. What methods will go in servlet? Can be very rough. Don’t worry if you overuse method at first.
20. How about another class to work on Model/Data? Can use knowledge of JDBC for this.
21. Then work on general interface for data so if we change RDBMS then we have to change very little code.
22. Create final client response page with results of data.
23. Use initial JSP page to get data in your query. Can you eventually move and use query in your properties file so you can just change it here?

**Section 2 test**

In general the test will definitely cover areas since Section 1 test. Some questions are very specific and we may not have hit all the details. So make sure you study the following.

1. The code questions will be on SQL, Servlets, JDBC. There should not be anything unusual for these questions.
2. The MCQ’s have some specific information. Understand some of the Servlet and JDBC API, along with recognizing some attributes and definitions for JSP tags. For example, what attributes are in <jsp:useBean /> tag? What is the purpose of this tag?
3. What are some main attributes in the JSP page directive (<%@ page…>) including isELIgnored?
4. Know the implicit objects in JSP. Examples: request – HttpServletRequest, out – JspWriter, session – HttpSession, …
5. Understand purpose of include directive ((<%@ include…>).
6. Know about i18n.
7. Investigate some of the core tag library tabs like the <c:url> tag.
8. Know about forwarding and redirecting.
9. Java Beans and POJOs.
10. Know about expressions, directives, declarations, scriplets in JSP pages.
11. Know servlet life cycle methods and service method in Servlet interface.
12. Know how certain objects in Java are mapped to the database such as java.sql.Timestamp.

**Spring core, JPA, Hibernate**

1. Good reference and practice for Spring core (other modules and demos from these modules, basic practice exercises and tutorials) from same site as building a spring boot application at spring.io: <https://docs.spring.io/spring-framework/docs/current/reference/html/core.html>
2. Easy straight forward guides to various modules: <https://spring.io/guides>
3. Spring Core tutorial: <https://java2blog.com/introduction-to-spring-framework/>
4. Good exercise when you get to JPA: <https://webdev.jhuep.com/~jcs/ejava-javaee/coursedocs/content/html/jpa-relationex-setup.html>
5. Tutorial and some basic documentation from Hibernate site: <https://docs.jboss.org/hibernate/orm/4.1/quickstart/en-US/html/ch01.html>
6. Many O’Reilly/Percipio videos and books – ex. Spring in Action 4th ed. <https://learning.oreilly.com/library/view/spring-in-action/9781617291203/>