https://almtools.ldschurch.org/fhconfluence/pages/viewpage.action?spaceKey=Product&title=Core+Skills+-+Java+-+Apprentice

**Tasks:**

**Core Java**

* Describe the lifecycle of an object instance in Java and how garbage collection works
* Java runtime loads the class
* Reads it into memory along with static initializers
* When it is time for the object to die, Java removes the object from the memory and its internal reference to it.
* Java runtime calls the garbage collector and destroys the object when it is no longer in use.
* Describe how the basic data types are represented in memory (boolean, int, long, String, array of ints, array of Objects, class with fields)
* Boolean: has two values: true (1) or false (0).
* Int: whole number such as 2, 4, 8. Default value is 0. 32-bit signed. It is one of the eight primitive data types (byte, short, int, long, float, double, Boolean, char)
* Long: 64-bit signed. Bigger than int.
* String: It is a sequence of characters. Default value is null.
* array of ints: A sequence of ints.
* array of Objects: A sequence of references to objects.
* class with fields: A class has static fields that belong to the class itself. Initialized class object can access them.
* Write an application to find out how many total characters can be held in a list of strings before you run out of memory 🡪 Main.java
* Compare and contrast StringBuffer and StringBuilder and when to use each
* StringBuffer is mutable, synchronized, and thread safe.
* StringBuilder is mutable, but not synchronized, but faster.
* Compare/contrast use of ArrayList / LinkedList / HashMap / HashSet / TreeSet
* ArrayList: A set of objects positioned in a certain place in the array.
* LinkedList: A set of objects connected/linked in sequence. Can’t be accessed by a position.
* HashMap: Allows non-duplicate key + multiple null values.
* HashSet: Doesn’t allow duplicate elements. OK: hashMap.add(“James”); The second hashSet.add(“James”); doesn’t work.
* TreeSet: Duplicate values are not allowed.
* Write an application to read a file with 10k lines of text, and output another file with the lines in sorted order ([sample file](https://almtools.ldschurch.org/fhconfluence/download/attachments/21405828/lorem.txt?version=1&modificationDate=1412196796847&api=v2))
* See readFileInAndSortByFirstWord() in readFileInAndOutputInSortedOrder.java
* Write an application to read a file with 10k lines of text, and output another file with the lines in reverse sorted order
* See readFileInAndReverseSortByFirstWord() in readFileInAndOutputInSortedOrder.java
* Write code to show exception handling including examples of checked, unchecked, and Error exceptions
* Checked vs unchecked – Exceptions that are not handled specifically using try-catch in the code are unchecked meaning that they are not checked at the compile time. They are the ones under Error and RuntimeException classes. Everything else is checked. If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception (from Java doc)
* See xxxx sample
* Write your own enum type.  Describe when you would use it.
* An *enum type* is a special data type that enables for a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. Common examples include compass directions (values of NORTH, SOUTH, EAST, and WEST) and the days of the week. Because they are constants, the names of an enum type's fields are in uppercase letters. (from Java doc)
* See example (EnumTypeSample.java)…

**Working with Methods, Encapsulation and Inheritance**

* Show how to use a common piece of logic from two different classes, in three different ways: 1) by composition, 2) by inheritance, and 3) by static method calls, discuss the tradeoffs
* for example: two different classes that write a message to a file, one in XML, one in line-oriented text, but both need to reuse logic to open the file in the same way
* ---🡪 **Composition** is different form inheritance; inheritance is a “is-a” relationship whereas composition is a “has-a” relationship. In the sample come XmlWriter and TextWriter are similar and extended from the same class, Writer, but they do have a same method name, write(), and have different components and parts.
* --🡪 **Example (MyInheritance.java)**
* Create and **overload** constructors -- Create a class that has 4 fields and construct the class with variations of one required field and the others are optional.  Use constructor chaining as an example.
* ---🡪 **Example (OverloadExample.java)**
* Apply encapsulation principles to a class -- Show an example of good **encapsulation**.  Show a bad example of encapsulation and explain why.  Additionally explain access modifiers and how they can be used as part of the class encapsulation.
* Good encapsulation hides class fields from public so that the values are not directly accessed by getters and setters.
* Bad encapsulation lets others access and change field values without any restrictions.
* **🡪 Example: EncapsulationSample.java**
* Determine the effect upon object references and primitive values when they are passed into methods that change the values -- Create a method 3 parameters, one is parameter is pass by value, one is passed by reference and one with the keyword final.  Explain each and what the effects in side the method that changes each one.
* First parameter – primitive
* Second parameter – Object, original value of the object doesn’t change even if the reference gets changed.
* Third parameter – Because it is final, the reference can’t be changed. But the value can get changed.
* **-🡪 Example: ParameterPassing.java**
* Write code to show how access modifiers work: private, protected, and public, talk about why you would use each of these.
* Public modified means that it is visible from everywhere.
* Protected modifier means its members can be accessed within its own package.
* Private modifier means it can be visible only within its own class.
* **-🡪 Example: \AccessControl\ MyMain.java**
* Write code to show how virtual method invocation lets one implementation be swapped for another.
* -🡪 **Example: \JavaBadge\..\VirtualMethodSample\VirtualMethod**
* Write code that uses the instanceof operator and show how casting works.
* **🡪 Example: \JavaBadge\..\InstanceOfSample\**
* Show how to override a method in a subclass, talk about plusses and minuses in doing so.
* Show how to overload constructors and methods, talk about plusses and minuses in doing so.

**Library**

* Write an application that uses the slf4j logging library directly (can also choose log4j if you want)
* Do the following:
  + configure the logging using an accepted department log statement format (see [Application Logging (getting data into Splunk)](https://almtools.ldschurch.org/fhconfluence/pages/viewpage.action?pageId=16712403))
  + log at different logging levels (error, warn, info, debug), to see the effect of the default logging level setting
  + turn on DEBUG in the logging config to show DEBUG output
  + configure logging to go to **both** the console and a log file

<https://almtools.ldschurch.org/fhconfluence/display/Product/Java+Badge+Implementation>

https://almtools.ldschurch.org/fhconfluence/pages/viewpage.action?spaceKey=Product&title=Java+Badge+Implementation

<https://almtools.ldschurch.org/fhconfluence/pages/viewpage.action?spaceKey=~roondog&title=Java+Badge+-+Apprentice>