**Annotation**

**Annotation is syntactic methadata that can be added to class method variables and method argument**

**Java Collection** :

1. List : List of things
2. Set : List of unique things
3. Map: List of things with key identifier

**List:**

Java.util.List is an interface that extends interface collection

ArrayList is a class that implements List interface

ArrayList:

1. Naturally ordered. First element goes to index 0
2. Fast Iteration . Slow insertion

LinkedList:

1. Naturally ordered by index First element goes to Index 0
2. Elements are double linked together
3. The best choice to implement Stack and Queue

Vector:

Same as List but it is thread safe

How to sort the list

1. By Comparator and comparable of model and using Collection.sort();
2. JDK 8,Lumbda sort() list
3. Using JDK 8 SortedList from JDK FX
4. With Java 8 and streaming

list.stream().sorted(Comparator.comparing(Student::getAge).reversed())

**Comparable and Comparator :**

Comparator : is Multi sorting sequence Collection.sort(my list,new Comprator())

Comparable : is Single sorting sequence , Collection.sort(my list) – Natural ordering of objects

**Set:**

HashSet(class): none-ordered. Can accept one Null element, faster

LinkedHashSet(class): ordered version of HashSet. Very good for iteration order. First inserted element is 0

TreeSet(class): ordere d. First inserted element the first acceccesed, sorted, slow

EnumSet: is a set which contains instance of specific enum type

It has lots of static factory method to create an instance of EnumSet e.g. EnumSet.of(...).

**EnumSet**<**Color**> yellow **=** **EnumSet**.of(**Color**.**RED**, **Color**.**GREEN**);

SortedSet (interface): is natural sorted SortedSet<String> sortedSet = new TreeSet<>();

Sorting Sets:

SortedList:

If natural sorting element : 🡪 (by default implement Comparable such as Long, String)

Comparator :

We use TreeSet and SortedSet;

Compratore my comprator = new Comparator();

SortedSet set = new TreeSet(mycomparator);

Set.add

**Map**

HashMap: not ordered is fast for

Treemap : Sorted by key for natural order, slower

LinkedHashMap: ordered , fast insertion slow iteration

HashTable: thread safe of hashMap

Collections.synchronizedList is how we make the collection synchronized

Java Pattern:

1. **Façade** : face of building
   1. A facade is an object that provides a simplified interface to a larger body of code, such as a [class library](https://en.wikipedia.org/wiki/Class_library)
   2. **describles a higher level interface that makes the subsystem**
   3. Make a complex subsystem easier to use
   4. Reduce dependencies
2. **Session façade**;
   1. Group multiple subsystem facade by representing the central access to entire subsystem
3. **Singleton** :

It is a class that instantiate once per class loader per JVM

1. **Factory**

Returns several instance of subclasses while calling services is not aware of actual instantiation

Class Female extends Person

Class Male extends Person

Person female = new Female();

1. **Abstract Factory** 
   1. 1 hierarchy above factory. Group several factory classes to return several different type of sublclasses

Class Female extends Person

Class Male extends Person

Class Bmw extends Car

Class Honda extens Car

Abstarct factory return an instance of female,Bmw,Honda

1. **Proxy:**
   1. Represents functionality of anther class

Interface Log {

Void printLog(String message);

}

Pubic class ProcessLog implement Log {

Void printLog(String message) {

Log.info(message);

}

}

Public class ProxyLog implement Log{

Private ProcessLog processLog;

Public void printLog(String message) {

processLog.printlog(message);

}

}

1. **Strategy pattern:**

Manage algoritems,relationship and responsibility between objects

The Strategy pattern is to be used where you want to choose the algorithm to use at runtime

Public interface CompressStrategy {

Void compressFile(File file);

}

Public class ZipFile implements CompressStrategy {

Void CompressFile (File file) {

// zip the given file

}

}

Public class RareFile implement CompressStrategy {

Void compressFile() {

// rare the givenFile

}

}

Public class CompessContext {

Private CompressStrategy compressStartegy;  
 Public CompessContext (CompressStartegy cs) {  
 this.compressStrategy = cs;   
 }

Public void createArchive(File file) {

This.compressStrategy.compressFile(file);

}

}

Public class Test {

Public static void main () {  
 CompressContext context = new ComressContext(new ZipCompress());

Context. createArchive(file);

}

}

1. **Decorating design :**

Attach responsibilities to an object at run time , Inheritance attach responsibility at compile time

File file = new File (“c:\temp\dd.txt”);  
BufferOutPutStream out = new BufferOutPutSream(file);

1. **Decorating design :**

Composite design pattern allows you to have a tree structure and ask each node in the tree structure to perform a task

Thread programming

Thread:

Process is an execution of program.   
 Thread is a single sequence of within a process

To take advantage of full CPU power we use multithreading

Concurrency:

Ability to run several part of programs in parallel

By extending Thread class or implement runnable

In Spring : ThreadPoolTaskExecutor will execute the thread task.

**The spring component that implement runnable should be @Scope(“prototype”)**

Spring also has Callable interface when we need to return an object back.

Callable interface give us ability to have a return value from call() thread method.

Difference between Start() and run()

When program

execute starts() 🡪 thread,sart() a new Thread object is created and code inside run method is executed.

Execute run() 🡪 the code inside run method executes in the same thread. New new Thread object is created.

when program calls start() method a **new Thread**is created and code inside run() method is executed in new Thread while if you call run() method directly **no new Thread is created** and code inside run() will execute on **current Thread**

**Synchronizing**

It’s the Capability to control access of multiple threads to shared resource

Synchronization key can be applied in **Method level** or **block level** of Code

Method level : method itself is Synchronized

Block Level: block of the code within a method is synchronize.

**+**: Volotaile

It is lock-free way to achieve synchronization in Java.

When to use Volatile

We use when we operate atomic operation such as all primitive data type except long and double.

To ensure every thread see updated value for \_instance and the value is not cached

Specifying a variable as volatile tells the JVM that any threads using that variable are not allowed to cache that  
value at all.

What is atomic operation:

Read and write are doing in 1 step . long and Doouble are 64 bit therefore read and wirte are performed in 2 steps. Most OS are 32 bit

ThreadLocal:

Enables you to create variables that can only be read and written by the same thread. Thus,

Problem with Thread programming :

* Liveness failure: The program does not react anymore due to problems in the concurrent access of data, e.g. deadlocks.
* Safety failure: The program creates incorrect data.

To avoid deadlock:   
Dead luck does not happen because of requesting the lock. The way threads are request are requesting locks will cause deadlock.

public void method1() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

 synchronized (String.class) {

 System.out.println("Aquired lock on String.class object"); }

} }

 public void method2() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

 synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

}

}

}  
  
The sequence of synch method on object are the same.

**How to make thread safe:**

* Immutable objects are thread safe.
* Atomic operation are thread safe
* Final variables and read methods are thread safe
* Locking is thread safe

Use Synchronize keyword on method level or class level or use atomicInteger

public class Counter {

    private int count;

    AtomicInteger atomicCount = new AtomicInteger( 0 );

    /\*

     \* **This method thread-safe now because of locking and synchornization**

     \*/

    public synchronized int getCount(){

        return count++;

    }

    /\*

     \* **This method is thread-safe because count is incremented atomically**

     \*/

    public int getCountAtomically(){

        return atomicCount.incrementAndGet();

    }

Read more: <http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html#ixzz4wH8kTIkw>

**System class:**

It is built in final class that presents some predefine method such as println();

**Immutable class**

Immutable class cannot be changed after construction and immutable object reference cannot be changed either.

We have to make the class final and variables final and not perform cloning of objects in the getter method to return a copy of Obect and not returning actual object reference. No Setter method;

**Synchronized block VS method**

Difference is selection of lock on which critical section is locked.

**Synchronized method**   
 Depending on whether a method

is a static method or non static locks on🡪 either class level lock or object lock.

Never mix static and non static synchronized method in Java.

**Synchroized block**

reduces scope of lock

public class SycnronizationExample {

public synchronized void lockedByThis(){  
 System.out.println(" This synchronized method is locked by **current" instance of object** i.e. this")  
 }

public static synchronized void lockedByClassLock() {  
 System.out.println("This static synchronized method is locked by class level lock of this class i.e.

SychronizationExample.class");

}

public void lockedBySynchronizedBlock(){

System.err.println("This line is executed without locking");

Object obj = String.class; //class level lock of Stirng class

synchronized(obj){

System.out.println("synchronized block, **locked by lock represented** using obj variable");

}

}

Read more: http://www.java67.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html#ixzz5BA5tQ59u

**Clone**

It is the way of creating an exact copy of an object**.**

**Hibernate :**

**Difference between L1 and L2**

L1 is in session level and it is valid during open-close current session

L2 is in application level(Session-factory)

By default L1 is enable. in very first call the session is open. getId() then goes to L2

Multiple calls from within the same open session does not hit DB

**Difference between load and get**

1. Get hits the dana for every single call and retrieve the entire object including associations and if cannot find it return null
2. Load proxy the object and might not return association plus throws not found exception

**Difference between Flush() and Refresh()**

Session.flush() 🡪 synch an object with database

Session.refresh() 🡪 Re-read the object from DB after DML

**Objects in Hibernate  
Detached, Persistent and Transient**

[**http://javarevisited.blogspot.ca/2012/12/how-to-find-middle-element-of-linked-list-one-pass.html**](http://javarevisited.blogspot.ca/2012/12/how-to-find-middle-element-of-linked-list-one-pass.html)

**Reverse String in Java :**

1. Use String builder reverse
2. Convert to Byte array[]

byte [] strAsByteArray = input.getBytes();

1. Convert to character array

char[] try1 = input.toCharArray();

1. Convert to ArrayList

String input = “abcderfhte”

Char[] inputChar = input.toCharArray();

For (char i:inputChar) {

Array.add(i);

}

Collection.reverse(array)