Java Interview Questions

1. Java Basic

1.1 Why is Java a platform independent language?

Java is a cross-platform language. The Java complier complies the code and make the code convert to platform-independent byte code. And the byte code runs in JVM. JVM can run in mutiple systems.

1.2 What is JDK, JRE, JVM?

JDK: Java development kit. It is a whole package containing a lot of tools, like complier, JRE and JVM. Complier makes the Java code to byte code.

JRE: An environment to run Java code. JRE includes a lot of java class libraries and JVM.

JVM: A virtual machine, class file runs in this virtual mechine and virtual mechine explain the code to the operation system.

1.3 Why is Java not a pure object oriented language?

Because Java has primitive data type (byte, boolean, char, short, int, float, long, and double)

1.4 Difference between Heap and Stack Memory in java. And how java utilizes this.

Heap: Heap is the place that store the instance. That means when we create a object, this object will store in heap . And heap is unique in JVM. So different thread can use the share object that stored in heap.

Stack: Every thread has an independent stack. Every thread controls its own stack. Stack is used to store the temporary data and the data.

Different:

1. Heap is unique in JVM. Stack is not. Every thread has a stack.
2. Heap stores instance. Stack store temporary data.

1.5 What do you understand by an instance variable and a local variable?

Instance variable can be used in every place in a class. Local variable can only be used in a function or an instruction, the other functions in class can not use this variable.

1.6 What are the default values assigned to variables and instances in java?

There is not default values in instances and variable. (except the primitive data type) We should instance the object and give the object values, otherwise the construction function has default value.

1.7 What do you mean by data encapsulation?

Make the data is private. We can only use the function to access the variable in object, such as get and set function.

1.8 What is different between equals() and '=='?

We use the '==' to compare the values. But OOP's everything is object. The value is inside of object. So we can not just compare two object by '==' even though their valuas are same. Because object is stored by different ways in memory. Objects' memory is different, '==' will return false. We should use equals() to compare the data inside of object. Equals() only compares the values.

1.9 What is different between overloading and overriding?

Overloading means a object can has many functions using the same name, but these function's input has to be different. Construction function is a sample.

Overriding means the son class or the implement class override a function. This function has to have same name, same input and same return. But we can change the function logic.

1.10 Explain the final keyword.

a variable with final can not be changed. a function with final can not be overridden. a class with final can not extend.

1.11 Can static function be overloaded or overridden?

static function can be overloaded. but can not be overridden.

1.12 Why String is final?

1. We can use String pool. If a string value is created. We can just use this string in string pool without creating.
2. Thread-safe. Synchronized.
3. String is always as key in HashMap or other hash collections. Immutable is good to calculate the hash value.

1.13 String, StringBuilder, StringBuffer

String is immutable. StringBuilder and StringBuffer are variable. StringBuffer is Thread-safe, StringBuilder is not.

If we want to change the string value, we should use StringBuilder and StringBuffer.

1.14 What is different between abstract class and interface?

Abstract class: abstract class can not be made a instance. abstract class can only be extended and implement the function in the son class. If there is a abstract function in a class, this class has to be abstract class.

Interface: interface's function is public by default.

Different :

1. a class can only extend one abtract class, but a class can implement mutiple interface.
2. Abstract is the 'IS-A' relationship. Interface is the 'LIKE-a' relationship.
3. If we want a lot of unrelated class use one function, we should use interface, such as compareTo()
4. If we want control some related class use one function, we should use abstract class.
5. Most of the time, we should use the interface. It is convenient and simple.

6. The loosely coupled nature of composition is preferable over the tightly coupled nature of inheritance.

2. Java Thread

2.1 How to create a thread?

Extend Thread

Implement Runnable (recommanded use runnable)

Implement Callable (be like Runnable, but has return value)

2.2 What is singleton? How to code a singleton class?

Singleton class means no matter how many times this class is created. This class will always return a same object, which means this object is unique in JVM. And Spring framework uses singleton by default.

Double Checked Locking

```
public class Singleton {
    private volatile static Singleton uniqueSingleton;

    private Singleton() {
    }

    public Singleton getInstance() {
        if (uniqueSingleton == null) {
            synchronized (Singleton.class) {
                if (null == uniqueSingleton) {
                    uniqueSingleton = new Singleton();    // error
                }
            }
        }
        return uniqueSingleton;
    }
}
```

2.3 How does an exception propagate in the code?

When an exception happened. If this exception happened in a try-catch block. Program will run the catch block code. If not, the exception will propagate to the function which used this function until this exception finds a try-catch block. If exception still does not find the try-catch block in main-function. The program will be terminated.

2.4 Is it mandatory for a catch block to be followed after a try block?

No. Try block can only have final block. But most of the time. We should use try-catch.

2.5 Is finally block always executed?

Yes, unless we let the system terminate, such as system.exit()

2.6 Introduce the Thread Lifecycle

1. New (create but not start())
2. Runnable (already started, but JVM does not run this Thread)
3. Running (JVM runs this Thread)
4. Unrunnable
    a. Blocked （wait other Thread release lock）
    b. Waiting （wait other thread to call this thread）
5. Terminated

2.7 Thread Pool 7 Parameters

1. CorePoolSize: always active thread count
2. MaximumPoolSize: when Core Threads are all already used, and work queue is full. Thread Pool will create a new Thread. The maximum thread count number is less than MaximumPoolSize.
3. KeepAliveTime: when a thread is free and thread count is greater than CorePoolSize. If this Thread free time is greater than KeepAliveTime, this Thread will be killed.
4. Unit: KeepAliveTime's unit
5. WorkQueue: A job is comming, will add to this queue and wait for thread execute.
6. ThreadFactory: the factory which used to create thread.

7. Reject Handler: when work queue is full and Thread count is equal to MaximumPoolSize, new comming job will be reject.

3. Java Collection

3.1 What is the difference between LinkedList and ArrayList?

ArrayList is a group of data continuously stored in memory.

LinkedList is stored in memory without ordering, so we should use additional tag to reference the next node.

Difference:

1. ArrayList searches more quickly than LinkedList, because it continues, so we can just use the index to search in memory.
2. LinkedList adds and deletes data more quickly than ArrayList, because if we add or remove data in array, wo should re-order the arraylist. But we only need to change the index in LinkedList when we want to add or delete values.
3. Actually, most of the time, we should use ArrayList, because LinkedList uses more memory (about 4 times than ArrayList), so there is high GC stress and high memory used. Besides, if we just add or delete data at the end of the list, arraylist is quicker, because we do not need to move the other data.
4. The only way we should use LinkedList is to add or delete the first data in a list frequently. Actually we can just use ArrayDuequ.

3.2 How does the size of ArrayList grow dynamically? And also state how it is implemented internally.

Every time we insert a value into ArrayList. ArrayList will determine whether the list is full or not. If it is, the capacity will increase to 1.5 times, because it is = OldCapacity + OldCapacity/2 (which means when OldCapacity is odd, it will be 1.45 times, because odd / 2 will take floor.)

3.3 What is the difference between HashMap and HashTable?

1. HashTable is synchronized. HashMap is not.
2. HashMap allows null key and null value, HashTable does not allow the null key and null value.
3. If we want to use synchronized HashMap, we can use ConcurrentHashMap

3.4 Introduce HashMap

1. Data Instruction: ArrayList<Entry>, Entry is a Linkedlist, including hashcode, K - V and a **Entry<K,V> next** (next entry index)
2. When two keys' hashcode is the same, they will put into a same index in ArrayList, and use a linked to connect different entry.

   eg：

   a. There is entry1<key1, value1> and entry2<key2, value2>, key1.hashCode() == key2.hashCode().
   b. Map puts both of them, entry1 puts first.
   c. They will be in the same index in ArrayList<Entry>. And entry 2 will be the first index in the LinkedList, because it is the last put entry and the first index in the LinkedList. (**Head Insertion**)
3. But after JDK1.8, the LinkedList instruction has changed to **Red-Black** Tree **when the LinkedList size is greater than 8 or the ArrayList.size() is greater than 64**. Because in LinkedList, some extremly situation, all the key is the same hashcode, the search will be slow, because we need search the whole LinkedList. (**LinkedList search time complexity is O(N), Red-Black Tree search time complexity is O(logN)**)
4. Params (default)：
   a. Initialize ArrayList Capacity：16 (should be Two to the N   2^N)
   b. LOAD_FACTOR: 0.75 (75% if the ArrayList<Entry> 75% value is not null, We should double the ArrayList size)
   c. Threshold: 8 (if LinkedList greater than 8, LinkedList will become the Red-Black Tree)
   d. Max_Tree_Capacity: 64 (if ArrarList greater than 64, LinkedList will become the Red-Black Tree)
5. ConcurrentHashMap
   a. It is a synchronized HashMap. Because ConcurrentHashMap divides the ArrayList<Entry> into segments. And every single segment is locked when it has changed. So when two keys are not in one segment, they can be changed together. That improves performance. That makes ConcurrentHashMap synchronized but very quickly.