1. **What is the difference between String, StringBuilder and StringBuffer?**
   1. String is immutable, but StringBuilder/StringBuffer are mutable (eg. by append() method);
   2. StringBuilder is faster than StringBuffer;
   3. StringBuilder is non-synchronized, which means StringBuilder is not thread safe. StringBuffer is synchronized, which means StringBuffe is thread safe.

2. **String s1 = "Jack"; String s2 = new String("Jack"). Check: s1 == s2? s1.equals(s2)?**
   s1 will be in the String constants pool, and s2 will be in the heap. That is:
   s1 == s2 will return false since "==" be used to compare the addresses of two objects.
   s1.equals(s2) will return true since equals() method be used to compare the values of two objects.

3. **Write "main" method**
   public static void main(String[] args) {}

4. **What is multiple inheritance? Does Java support it?**
   Multiple inheritance means one class (subclass) acquires the properties from two or more classes (superclasses), which is not supported in java but java support implements from multiple interfaces.

5. **Difference between Abstract class and Interface**
   1. Abstract class does not support multiple inheritance, but interface can do multiple implements.
   2. Abstract class has its constructor while interface doesn't have it.
   3. Abstract class can have abstract method and non-abstract method, while interface can only have abstract method.

6. **What is marker interface? Given an example of maker interface**
   Marker interface is an interface without any variable or method in it. Serializable is an example of marker interface.

7. **Write Java program to explain static and dynamic polymorphism**

```
public class PhoneFactory {
    public PhoneFactory() {
        System.out.println("welcome!");
    }

    pubic void makingPhone(type Apple) {
        System.out.println("making iphone…");
    }
```

```
        /* overload makingPhone() method here, and it will be finished at compiling time, so it is
static */
        pubic void makingPhone(type Samsung) {
                System.out.println("making Samsung…");
        }

}

public class MakingIphoneX extends PhoneFactory {
        /* override makingPhone() method here, and it will be finished at running time, so it is
dynamic */
        @Override
        public void makingPhone(type Apple) {
                System.out.println("IphoneX is coming soon…");
        }
}
```

8. **Write Java program to create a Singleton design pattern**

```
public class Singleton implements Serializable, Cloneable {
        private static Singleton instance = new Singleton();

        private Singleton() {}

        public static synchronized Singleton getInstance() {
                if (instance == null) {
                        instance = new Singleton();
                }
                return instance;
        }

        @Override
        public Object clone() throws CloneNotSupportedException {
                throw new CloneNotSupportedException();
        }

        protected Object readResolve() {
                return instance;
        }
}
```

9. **Write Java program to create a Factory design pattern**

```
public interface Phone {
        void make();
}
```

```
public class iPhone implement Phone {
        @Override
        void make() {
                System.out.println("making iPhone…");
        }
}

public class Samsung implement Phone {
        @Override
        void make() {
                System.out.println("making Samsung…");
        }
}


public class PhoneFactory {
        public Phone makePhone(String type) {
                if (type == null) {
                        return null;
                }
                else if (type.equals("Iphone")) {
                        return new iPhone();
                }
                else if (type.equals("Samsung")) {
                        return new Samsung();
                }
                return null;
        }
}
```

10. **Difference between ArrayList and LinkedList and their time complexity in insert and read operation**
    ArrayList has O(n) time complexity for arbitrary indices of add/remove, but O(1) for the operation at the end of the list. LinkedList has O(n) time complexity for arbitrary indices of add/remove, but O(1) for operations at end/beginning of the List. Both are not thread safe.

11. **If you define a customized "Student" class and you want to save it to a hashmap as map key, what do you need to do in the Student class?**
    The equals() and hashcode() methods must be overridden in Student class before the hashmap be used to restore a Student object as map key.

12. **Explain how HashMap works internally**

There are lots of buckets/bins, when we call put() method, java will hash the key and generate a hash code, calculate the index, and get the index will determine which bucket will hold this key-value pair. the java will call equals() method to check all keys in bucket, if matched, the exist key-value will be replaced, otherwise, the new key-value will be inserted in.
When we call get() method, Firstly, it finds the hashcode of the key to locate the bucket, Secondly, it scans the nodes, calling the equals method to find if the node's key equals the current key. If end of the list is reached, the key is decided to not exist, and null will be returned.

13. **If you want to remove one element from an arraylist, can you use for loop? If not, what will you use?**
No we can't, but we can use Iterator.

14. **What is checked and unchecked exception? Given each one an example**
Checked exception as known as compile exception because checked exception is checked by the compiler at compile time, SQLException is one of checked exception.
Unchecked exception including Error and RuntimeException, they will be checked at runtime. IndexOutofBoundException is one of unchecked exception.

15. **What are the two ways to create a java thread?**
Extends Thread class or implements Runnable interface.

16. **What is an ExecutorSerivce? Given an example of how you have used ExecutorService.**
ExecutorService helps to maintain thread pool
ExecutorService excutor = Executor.newCachedTreadPool();
use *invokeAll()* to start all the threads return a list of Future objects holding their status and results.
use *shutdown()* to shut down manully.

17. **Given an ArrayList including all US states, write a program using Java 8 stream and lambda to return all states starting with letter "N", for example, New Jersey, New York etc.**

```
public ArrayList<String> states(ArrayList<String> states) {
     ArrayList<String> Nstate = states.stream().filter(s -> s.charAt(0).equals("N")).findAny().orElse(null);
    return Nstate;
}
```

18. **What is functional interface?**
An interface contains one and only one abstract method is called functional interface, and it has an annotation *@FunctionalInterface*.

19. **Given an "Employee" table including columns: id, name, department, salary. Write a SQL to find the total salary for each department. Your result should 1) include two columns with column name as: Department and TotalSalary; 2) Only show results with total salary more than 2000; 3) sorted in descending order according to department.**

```
SELECT SUM(salary) AS TotalSalary, department AS Department
FROM Employee
GROUP BY department
HAVING TotalSalary > 2000
ORDER BY 2 DESC;
```

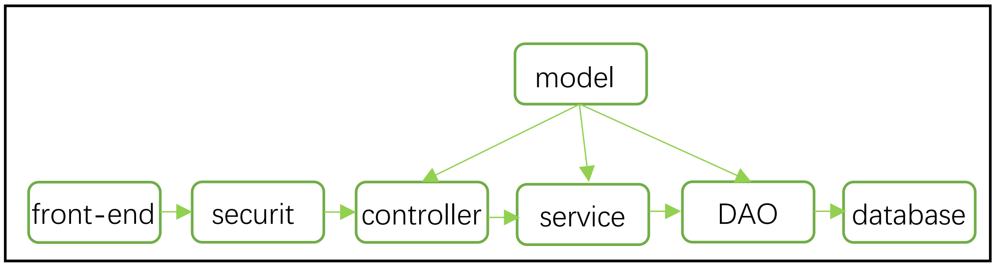
20. **Difference between Final, Finally and Finalize**

Final means it cannot be changed. If final be used to decorate a class, this class cannot be inherited; if final be used to decorate a method, this method cannot be override; if final be used before a variable, so this variable cannot be modified.
Finally be used in try...catch block, and finally part will always be executed eventually
Finalize be used in garbage collection.

21. **Difference between git rebase and git merge**

git rebase rewrite the master branch which means it doesn't need commit
git merge to merge a branch to the master branch and it need commit.

22. **Draw a diagram to show the data flow chain in enterprise project, your diagram show includes: front end, database, dao, model, service, controller and security.**



23. **Write a program to reverse a string.**

```
public String reverse(String str) {
     char[] chars = str.toCharArray();
    int left = 0, right = str.length() – 1;
    while (left < right) {
        char temp = chars[left];
        chars[left] = chars[right];
        chars[right] = temp;
        left++;
        right--;
    }
    return new String(chars);}
```