1. **“Disassemble” `public static void main(String args[])`.**

   One of the popular java interview questions for freshers, and very easy one.

   - `public` is an access modifier. We use it to specify the access to this method. Here modifier is “public”, so any Class has the access to this method.
   - `static`. This Java keyword means that we use this method without creating a new Object of a Class.
   - `Void` is the return type of the method. It means that the method doesn’t return any value.
   - `main` is the name of the method. JVM “knows” it as an entry point to an application (it should have a particular signature). `Main` is a method where the main execution occurs.
   - `String args[]`. This is the parameter passed to the main method. Here we have the arguments of String type that your Java application accepts when you run it. You can type them on the terminal.

2. **What is the difference between `equals()` and `==` ?**

   First, “`==`” is an operator whereas `equals()` is a method. We use `==` operator for reference comparison (or address comparison) and `equals()` method for content comparison. It means that `==` checks if both objects point to the same memory location while `equals()` compares values in the objects.

3. **Can we execute a program without `main()` method?**

   Many java basic interview questions are really easy. Like this one. So short answer is: yes, we can. For example we can do it using static block.

   You can use static block to initialize the static data member. It is executed before the `main` method, at the time of class loading.

```
class Example{
 Static{
System.out.println("static block is invoked");
}
  public static void main(String args[]){
   System.out.println("Now main method");
  }
}
```

   The output is:

```
static block is invoked
Now main method
```

   What about total main method absence? If you try to run an ordinary class without the main method at all, you got the next error: Main method not found in class Test, please define the main method as: public static void main (String [] args) or a JavaFX application class must extend javafx.application.Application. The error itself says that if this is a JavaFX application and the class is inherited from javafx.application.Application, then it is possible.

4. **What is `immutable` object? Can you create `immutable` object?**

   You can't modify objects of an `immutable` class after their creation. So once you create them, you can't change them. If you try to modify `Immutable` object you get a new object (clone) and change this clone while creation.

   A good example is `String`, it is `immutable` in Java. That means you cannot change the object itself, but you can change the reference to the object.

5. **How many objects are created in the following code?**

   One of java technical interview questions that substitutes #4.

```
      String s1="Hello";
      String s2="Hello";
      String s3="Hello";
```

   The answer is "only one" because Java has a String Pool. When we create a String object using the new() operator, it creates a new object in heap memory. If we use String literal syntax, like in our example, it may return an existing object from the String pool, if it already exists.

6. **How many objects are created in the following code?**

```
      String s = new String("Hello");
```

   There are 2 objects. One is in string constant pool and the other in heap.

7. **What is Difference Between `String`, `StringBuilder` And `StringBuffer` Classes in Java ?**

   There is one of the leader in top java interview questions.

   First of all `String` is an Immutable class. That means you can't modify its content once created. While `StringBuffer` and `StringBuilder` are mutable classes, so you can change them later. If we change content of `String` object, it creates a new string therefore it doesn't modify the original one. That's why the performance with `StringBuffer` is better than with `String`.

   The main difference between `StringBuffer` and `StringBuilder` that `StringBuffer`'s methods are synchronized while `StringBuilder`'s are not.

8. **Is there any difference in `String` that was created using literal and with `new()` operator?**

   It is. If we create string with `new()` operator it appears in heap and not added to the string pool. If you create `String` using literal it is created in `String` pool which exists in Perm area of heap.

9. **Can you override `private` or `static` method in Java?**

One of java tricky interview questions for rookies. You really can’t override `private` or `static` method in Java.

You can’t override the `private` methods because the scope of the private access specifier is only within the class. When you are going to override something, we should have parent and child class. If method of superclass is `private`, child class can’t use it, and the methods in child class will be treated as new methods (not overridden).

`Static` methods also cannot be overridden, because `static` methods are the part of the Class itself, and not a part of any object of the class. Sure you can declare same `static` method with same signature in child classes, but again, they will be treated as new methods.

10. **Difference between `Abstract Class` and `Interface`**

One of the popular java developer interview questions that bears on OOP principles. First of all, in Java `interface` defines a behavior and `abstract class` creates hierarchy.

Abstract class	Interface
It is possible to have a method body (non-abstract methods) in abstract class	Interface can have only abstract methods. In Java 8 or newer, it became possible to define default methods and implement them directly in the interface. Also, Interfaces in Java 8 could have static methods.
Instance variables can be in abstract class	An interface can’t have instance variables.
Constructors are allowed	The interface can’t have any constructor.
Static methods are allowed	Static methods are not allowed
Class can have only one abstract parent	One interface may implement different classes
The abstract class may provide the implementation of the interface.	The Interface can't provide the implementation of the abstract class.
An abstract class is allowed to extend the other Java class and implement multiple Java interfaces.	An interface is allowed to extend the other Java interface only.
A Java abstract class can have private and protected class members	Members of a Java interface are public by default

11. **Can we declare `static` variables and methods in an `abstract` class?**

    Yes, it is possible to declare `static` variables and methods in `abstract` method. There is no requirement to make an object to access the static context. So we are allowed to access the static context declared inside the `abstract` class by using the name of the `abstract` class.

12. **What types of memory areas are allocated by JVM?**

    Class Area stores perclass structures, for example, runtime constant pool, fields, method datas, and all the code for methods.

    Heap is a runtime data area where memory is allocated to the objects.

    Stack stores frames. It contains local variables and partial results, and takes part in method invocation and return. Every thread has a private JVM stack, created at the same time as the thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

    Program Counter Register contains an address of the Java virtual machine instruction currently being executed.

    Native Method Stack contains all the native methods that are used in the application.

13. **Why is multiple inheritance isn't allowed in java?**

    It would be really complicated. Imagine three classes `A`, `B`, and `C` and `C` inherits `A` and `B`. Now, `A` and `B` classes have the same method and you call it from a child class object... Which one? `A`'s or `B`'s? Here we have ambiguity.

    if you try to inherit two classes Java renders compile time error.

14. **Can we overload the `main()` method?**

    Sure, we are allowed to have many `main` methods in a Java program by using method overloading. Try it out!

15. **Can we declare a constructor as `final`?**

    Nope. A constructor can't be declared as a `final` because it can't be inherited. So it is senseless to declare constructors as `final`. However, if you try to do it, Java compiler throws you an error.

16. **Can we declare an interface as `final`?**

    No, we can't do this. An interface can't be `final` because the interface should be implemented by some class according to its definition. Therefore, there is no sense to make an interface `final`. However, if you try to do so, the compiler will show an error.

17. **What is the difference between `static binding` and `dynamic binding`?**

    The `binding` which can be resolved at compile time by compiler is called `static` or early binding. `Binding` of all the `static`, `private` and `final` methods is done at compile-time.

In `Dynamic binding` compiler can't choose a method to be called. Overriding is a perfect example of `dynamic binding`. In overriding both parent and child classes have same method.

```
Static Binding
class Cat{
 private void talk()
{System.out.println("cat is mewing...");
}

 public static void main(String args[]){
  Cat cat=new Cat();
  cat.talk();
 }
}
Dynamic Binding
class Animal{
 void talk(){
System.out.println("animal is talking...");
}
}

class Cat extends Animal{
 void talk(){
System.out.println("cat is talking...");
}
 public static void main(String args[]){
  Animal animal=new Cat();
  animal.talk();
 }
}
```

18. **How to create a read-only class in Java?**

You can do it by making all of the class' fields private. The read-only class has only getter methods which return the private property of the class to the `main` method. You are not able to modify this property, the reason is the lack of setters method.

```
public class HockeyPlayer{
private String team ="Maple leaf";
public String getTeam(){
return team;
}
}
```

19. **How to create a write-only class in Java?**

Again, you should make all of the class' fields `private`. Now, your write-only class should have only setter methods and no getters. Therefore, we can't read the properties of the class.

```
public class HockeyPlayer{
private String team;
public void setTeam(String college){
this.team = team;
}
}
```

20. **Each `try` block must be followed by a `catch` block, mustn't it?**

Nope. It is not a necessity. Each-`try` block can be without a `catch` block. It could be followed by either a catchblock or a finally block or even without them at all.

```
public class Main{
     public static void main(String []args){
        try{
            int variable = 1;
            System.out.println(variable/0);
        }
        finally
        {
            System.out.println("the other part of the program...");
        }
     }
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
the other part of the program...
```

One more example:

```
class Main {
        public static void main(String[] args) throws IOException {
            try(InputStreamReader inputStreamReader = new InputStream
                BufferedReader reader = new BufferedReader(inputStrea
                System.out.println("test");
            }
        }
    }
```

Output:

```
test
```

P.S.: Before Java 8 methods in interfaces could have been abstract only. In Java 8 or newer, it became possible to define default methods and implement them directly in the interface.

21. **What is the difference between `throw` and `throws` keywords?**

    `Throws` is used to declare an exception, so it works similar to the `try-catch` block. `Throw` keyword is used to throw an exception explicitly from a method or any other block of code.

    `Throw` is followed by an instance of `Exception` class and throws is followed by exception class names.

    `Throw` is used in the method body to throw an exception. `Throws` is used in a method signature to declare the exceptions that may occur in the statements present in the method.

    It is allowed to throw one exception at a time but you can handle multiple exceptions by declaring them using `throw` keyword.You can declare multiple exceptions, e.g., `public void method()throws IOException`, `SQLException`.