SE INTERVIEW QUESTIONS

https://www.edureka.co/blog/interview-questions/java-interview-questions/

Java is pass by value.

When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object

In **Java**, a **static** member is a member of a class that isn't associated with an instance of a class. Instead, **the member belongs to the class itself.**

STATIC - https://www.geeksforgeeks.org/static-keyword-java/

STATIC - When a member (block, variable, method, nested class) is declared static, it can be accessed before any objects of its class are created, and without reference to any object

-**Static Classes** - Static classes are used as inner classes so you **do not have to initiate the inner class**.

```
Class OuterClass {
        Static String message =”Suck it”;

        Static class NestedStaticClass {
                Public void printMessage() {
                        System.out.println(message);
                }
        }
        Public class InnerClass {
                Public void display() {
                        System.out.println(message);
                }
        }
}
Class Main {
        Public static void tacos() {
                System.out.println(“tacos”);
        }
        public static void main(String args[]){

                // create instance of nested Static class
                 OuterClass.NestedStaticClass printer = new OuterClass.NestedStaticClass();
                printer.printMessage();
                 tacos();

                // In order to create instance of Inner class we need an Outer class
                // instance. Create Outer class instance for creating non-static nested class
```

```
        OuterClass outer = new OuterClass();
        OuterClass.InnerClass inner  = outer.new InnerClass();
        inner.display();
    }
}
```

-**Static methods** are the methods in Java that can be called without creating an object of class

-**Static variable** - When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level.

You cannot use the static keyword with a class unless it is an inner class.
A static inner class is a nested class which is a static member of the outer class.
It can be accessed without instantiating the outer class, using other static members.
Just like static members, a static nested class does not have access to the instance variables and methods of the outer class.

-what are static variables, classes, and methods used for? Why use them/when to use them? If you can pass static variables to other functions and classes, how would you do that?

Exception vs Errors - https://www.geeksforgeeks.org/errors-v-s-exceptions-in-java/

Try Catch Finally - https://www.geeksforgeeks.org/flow-control-in-try-catch-finally-in-java/

Error : An Error "indicates serious problems that a reasonable application should not try to catch.
Errors are the conditions which cannot get recovered by any handling techniques. It surely cause termination of the program abnormally

Ex: Runtime can be a reason for error:

```
public static void test(int i) {
        if (i == 0)
            return;
        else {
            test(i++);
        }
}
```

```
Exception in thread "main" java.lang.StackOverflowError
    at StackOverflow.test(ErrorEg.java:7)
```

An **Exception** "indicates conditions that a reasonable application might want to catch."

Exceptions are the conditions that occur at runtime and may cause the termination of program. But they are recoverable using try, catch and throw keywords.

```
int a = 5, b = 0;
// Attempting to divide by zero
try {
    int c = a / b;
}
catch (ArithmeticException e) {
```

```
   e.printStackTrace();
}
System.out.println(“Tacos --”);

This will print:
java.lang.ArithmeticException: / by zero
        at Main.main(Main.java:62)
Tacos --
// it will not terminate the program
```

Try Catch FINALLY - The finally block ***always*** executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling — it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

-When to use error handling exceptions. Why use them? What happens to the program if you do not use error handling? What happens to your program if you don’t use error handling for REST or importing information/data?

Final - https://www.javatpoint.com/final-keyword

The **final keyword** is for variable, method and class

Final Variable - If you make any variable as final, you cannot change the value of final variable(It will be constant).

If you change the value - Output:Compile Time Error

Final Method - If you make any method as final, you cannot override it.

```
class Bike{
  final void run(){System.out.println("running");}
}
class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

Output:Compile Time Error

Final Class - If you make any class as final, you cannot extend it.

```
final class Bike{}
 class Honda1 extends Bike{
   void run(){System.out.println("running safely with 100kmph");}
```

```
  public static void main(String args[]){
  Honda1 honda= new Honda();
  honda.run();
  }
 }
```

Override - https://www.geeksforgeeks.org/overriding-in-java/

-**Override method** - When a method in a subclass has the **same name, same parameters and same return type(**or sub-type) **as a method in its super-class**, then the method in the subclass is said to override the method in the super-class. Use @Override above the subclass method when overriding **Declare the parent method as final if you do not want it to be overriden**. Ex:

```
Class ParentClass {
        Final void show() {   System.out.println("Overriding is not allowed");   }
        // if we want child to override it:
        // void show() {        System.out.println("Overriding is allowed");   }
}
Class ChildClass extends ParentClass {
        Void show() {   System.out.println("Child class cannot override this method");  }
        // @Override
        // Void show() {   System.out.println("Child class override this method");  }
}
```

Prints out:

```
13: error: show() in Child cannot override show() in Parent
   void show() {  }
        ^
  overridden method is final
```

If we call super.show(); in child overriden child method, the parent method will be shown. Ex:

```
Class ChildClass extends ParentClass {
        @Override
        Void show() {
                super.show();
                System.out.println("Child class can override this method");
        }
}
Class Main {
        Public static void main(String[] args) {
                Parent obj = new Child();
                obj.show();
        }
}
```

method signature - is part of the **method** declaration. It's the combination of the **method name and the parameter list**

Overload method - allows different methods to have the same name, but different signatures where the signature can **differ by the number of input parameters** or **type of input parameters** or **both**. Overloading is related to compile-time (or static) polymorphism.

-Four principles of Object Oriented Programming -

(Encapsulation, Data Abstraction, Inheritance, and Polymorphism)

Encapsulation - hiding of data implementation by restricting access to public methods. private instance variable and public accessor methods.

-**Abstraction** - **Hiding** the **internal implementation** of the feature and **only showing the functionality** to the users. i.e. what it works (showing), how it works (hiding). Both abstract class and interface are used for abstraction.

Abstraction - Abstract means a concept or an Idea which is not associated with any particular instance
Using abstract class/Interface we express the intent of the class rather than the actual implementation
https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/
Applying abstraction means that each object should **only** expose a high-level mechanism for using it.

Interface vs Abstract Class

Methods - Interface can have only abstract methods.
Methods - Abstract class can have abstract and non-abstract methods.
Variables - Interface - In a Java**,** are by default final.
Variables - **Abstract class** may contain non-final variables.

USE

Interface - It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.
Interface - Interfaces specify what a class must do and not how. It is the blueprint of the class.
Abstraction is a process of hiding the implementation details and showing only how or what it works (showing), how it works (hiding)

```
Abstract class Shape {
        Abstract void area() { }    }
Square extends Shape {
        @override
        area() { }    }
```

```
public interface MusicList {
        public int getNumChannels();
        public float getSampleRate();
        public int getNumSamples();
}
public class MusicLinkedList implements MusicList {
        @Override
        public int getNumChannels() { }
}
```

Inheritance - parent & child classes

polymorphism - gives a way to use a class exactly like its parent so there's no confusion with mixing types. But each child class keeps its own methods as they are.

This typically happens by defining a (parent) interface to be reused. It outlines a bunch of common methods. Then, each child class implements its own version of these methods.

Polymorphism - It means one name many forms. It is further of **two types — static and dynamic. Static polymorphism** is achieved using **method overloading** and **dynamic polymorphism** using **method overriding**. It is closely related to inheritance. We can write a code that works on the superclass, and it will work with any subclass type as well.

Polymorphism - being able to assign a different meaning or usage to something in different contexts.
-To allow an entity such as a variable, a function, or an object to have more than one form.

polymorphism gives a way to use a class exactly like its parent so there's no confusion with mixing types. But each child class keeps its own methods as they are.

-What is a hashmap and when do you use it? What happens on the Java end of a hash map/Explain hashing?
-Name some JUnit Annotations. When do you use them and why?
-Talk about how you connected your Spring application to your SQL database. What did you use and why?
-If given two strings, how would you return only the unique chars in N+N
-Git commands - push to master
-Difference between linked list and array. Iteration, access.

Java Primitive Data Types:

- **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive)
- **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).
- **int**: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -2^{31} and a maximum value of $2^{31}-1$.
- **long**: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$.
- **float**: The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the Floating-Point Types, Formats, and Values section of the Java Language
- **double**: The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the Floating-Point Types, Formats, and Values section of the Java Language Specification.
- **boolean**: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
- **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

Dynamic: Java is a dynamic programming language. It supports dynamic loading of classes which means classes are loaded on demand.

Java is pass by value

Final keyword is used with the class to make sure that any other class can't extend it.

Access Modifiers: Access Modifiers are the keywords which are used for set accessibility to classes, methods, and other members. In Java, these are the four access modifiers:
Public: can be accessed by any class or method.
Protected: Classes or methods that are protected can be accessed by the class of the same package, or by the sub-class of the parent class, or within the same class.
Default: Default is accessible within the package only. All the classes, methods, and variables are of default when the public, protected, or private are not used.
Private: Classes, methods, and variables which are defined as private can be accessed within the class only.

Enum - field consists of fixed sets of constants

What is the difference between equals() and == in Java?
Equals() method is defined in Object class in Java and **used for checking equality of two objects** defined by business logic.
"==" or equality used to compare primitives and objects.

An object has: state, behavior, and identity.

Q18. What are the main concepts of OOPs in Java?

1. **Inheritance**: Inheritance is a process where one class acquires the properties of another.
2. **Encapsulation**: a mechanism of wrapping up the data and code together as a single unit.
3. **Abstraction**: hiding the implementation details from the user and only providing the functionality to the users.
4. **Polymorphism**: the ability of a variable, function or object to take multiple forms.

instance variable - a variable which is bounded to its object itself.

Stack Vs Heap
Stack memory only contains local primitive and reference variables to objects in heap space.
Heap - Whenever an object is created, it's always stored in the Heap space.

this() - the current instance of a class
super() - the current instance of a parent/base class

Strings are immutable

Array vs ArrayList
ArrayList - Can contain values of different data types
Arrays - Need to specify the index in order to add data

Constructors - block of code which is used to initialize an object.

Method Overriding - to “Change” existing behavior of the method.

Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.

Throw vs Throws - https://beginnersbook.com/2013/04/difference-between-throw-and-throws-in-java/
Throws clause is used to **declare an exception**, which means it works **similar to the try-catch block**.
On the other hand **throw** keyword is used to **throw an exception** explicitly.
If we see syntax wise than **throw** is followed by an instance of Exception class and **throws** is followed by exception class names.

For example:
throw new ArithmeticException("Arithmetic Exception");
And
throws ArithmeticException;

Throw keyword is used in the method body to **throw an exceptio**n,
while **throws** is used in method signature to **declare the exceptions** that can occur in the statements present in the method.

```
void myMethod() {
  try {
    //throwing arithmetic exception using throw
    throw new ArithmeticException("Something went wrong!!");
  }
  catch (Exception exp) {
    System.out.println("Error: "+exp.getMessage());
  }
}
```

```
//Declaring arithmetic exception using throws
void sample() throws ArithmeticException{
  //Statements
}
```