**Solid Principles:** refer Baeldung

**Access Modifiers:**

* Public: everywhere
* Private: only within class
* Protected: within pkg, outside pkg only in subclasses.
* Default: only within pkg.

**Abstract Class vs. Interface:** extends/implements

* A class can inherit one class, can implement multiple interfaces i.e. interface helps to achieve **multiple inheritance**.

Interface Rules:

* [constants variables](https://www.baeldung.com/java-final)
* [abstract methods](https://www.baeldung.com/java-abstract-class)
* [static methods](https://www.baeldung.com/java-static-default-methods)- can be given implementation after java 8
* we can’t instantiate interfaces directly
* an interface can be empty, with no methods or variables in it
* we can’t use the *final* word in the interface definition, as it will result in a compiler error
* all interface declarations should have the *public* or default access modifier; the *abstract* modifier will be added automatically by the compiler
* an interface method can’t be *protected* or *final*
* up until Java 9, interface methods could not be *private*; however, **Java 9** introduced the possibility to define [**private methods in interfaces**](https://www.baeldung.com/java-interface-private-methods)
* interface variables are *public*, *static*, and *final* by definition; we’re not allowed to change their visibility

**Multithreading:**

* Extends Thread Class, override Run method
* **Volatile Keyword**- improves **Visibilty**- change made by one thread should be immediately visible to other , though it doesn’t ensure **atomicity**
* **synchronised** keyword-

-Instance methods, Static methods & code blocks

- Ensures both visibility and atomicity- hard lock.

**Exception Handling:**

* Try, catch, finally
* Throw() from try {}
* Throws with method declaration- overriding method also need to declare
* Checked(FileNotFound, IOException, SQLException) , Unchecked (NullPointer, ArithmeticException) Exception : checked must be declared as throws or should be handled in try-catch

**Serialisation:**

* Object->stream-> use case to transfer object over the network.
* **Serial Version UID** : The Serialization runtime associates a version number with each Serializable class called a SerialVersionUID, which is used during Deserialization to verify that sender and receiver of a serialized object have loaded classes for that object which are compatible with respect to serialization
  + --user may give or can be generated default also --- UID Mismatch in serialization and deserialization -> InvalidClassException
* **Transient** Keyword: When applied to a variable, it instructs the Java Virtual Machine (JVM) to exclude that variable from the serialization process.
* **Static Variables:** These variables are not serialized, So during deserialization static variable value will loaded from the class.(Current value will be loaded.)

**String :**

* String is a literal as well as Class
* String Constant pool: dediacte space in heap for String Constants.

-- String s1=”ABC”;

-- String s2=”ABC”; ->>> s1 & s2 point to same literal in SCP, no new space allocated.

* == Compares values, .equals() compares references

Eg: String s3=new String (“ABC”);

String s4= new String (“ABC”);

s1==s2 ->> false;

s3==s4->> false

s3.equals(s4)->> true

s3.equals(s2)->> true

**Static Keyword:**

* **declare a field static, exactly a single copy of that field is created and shared among all instances of that class.**
* **static variables are stored in the heap memory.**
* **access static fields without object initialization.**
* *static* methods in Java are resolved at compile time. Since method overriding is part of Runtime Polymorphism,***static* methods can’t be overridden.**
* **Abstract methods can’t be static.**
* *static* methods**can’t use *this* or *super* keywords.**
* instance methods can directly access both instance methods and instance variables
* instance methods can also access *static* variables and *static* methods directly
* *static* methods can access all *static* variables and other *static* methods
* ***static* methods can’t access instance variables and instance methods directly.** They need some object reference to do so.

**Final Keyword:**

* Can be class, variable or method
* Immutability
* Final class can’t be inherited, method cant overridden, variables cant be reassigned.

**API Stream:** refer Geeks

* Java 8 feature – to handle & process collections
* Method chaining, clean code
* .map(), .filter(), .sorted()
* .collect(), .forEach(), .reduce()

**Collections:**

**Miscellaneous:**

Arrays.asList()

Arrays.sorted()

**for** (Map.Entry<String,Integer> mapElement : hm.entrySet()) -> hashmap traverse