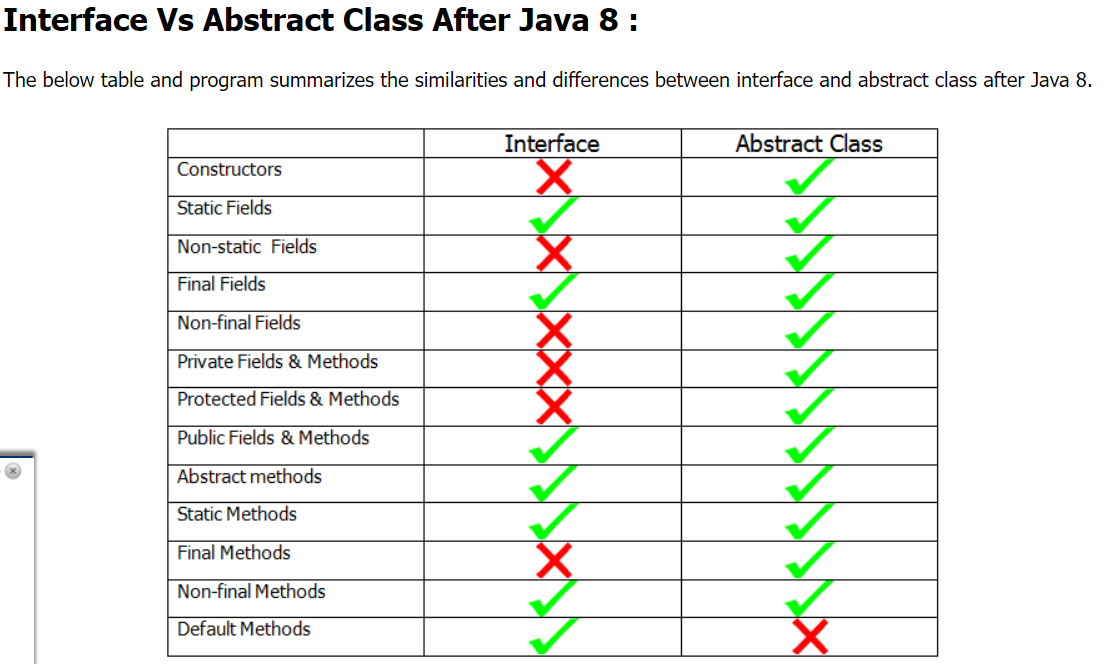
**Difference between Abstract class vs Interface  in Java 8**

Prima facia, **in Java 8, an interface looks like an abstract class** and one can reason about, *can we use an interface with default methods in place of an abstract class in Java?*  
  
Well, I believe they are for two different purposes and we will learn more once we start using Java 8 regularly, but following the semantics difference between abstract class and interface with default method will guide you further :  
  
1) Abstract classes are **classes**, so they are not restricted to other restrictions of the interface in Java, like abstract class can have the **state**, but you cannot have the state on the interface in Java.  
  
2) Another semantic difference between an interface with default methods and an abstract class is that you **can define constructors inside an abstract class**, but you cannot define constructors inside an interface in Java.  
  
In reality, [default](http://www.java67.com/2017/08/default-methods-in-interface-multiple.html) or [defender methods](http://javarevisited.blogspot.sg/2014/07/default-defender-or-extension-method-of-Java8-example-tutorial.html#axzz4pp42TeHu) are introduced to maintain backward compatibility and the same time making Collection API more suitable to be used inside key Java 8 features like [lambda expressions](https://javarevisited.blogspot.sg/2014/02/10-example-of-lambda-expressions-in-java8.html).  
  
Without adding default methods, it wasn't possible to declare any new method on the existing interface in Java without breaking all classes which implement it, but because of the default method, you can now better evolve your API.  
  
  
Read more: <https://www.java67.com/2017/08/difference-between-abstract-class-and-interface-in-java8.html#ixzz7ACmuo6j1>



what method cannot be overridden and basic rules of overriding

* A method declared final cannot be overridden.
* A method declared static cannot be overridden but can be re-declared.
* If a method cannot be inherited, then it cannot be overridden.
* Constructors cannot be overridden.
* **We should override Parent class abstract methods in Child classes to provide**

**implementation.**

**. While overriding we can't reduce the scope of access modifier.**

**private < default < protected < public**

**Rule: While overriding if the child class method throws any checked exception**

**compulsory the parent class method should throw the same checked exception or**

**its parent otherwise we will get compile time error.**

# Thread Safety and how to achieve it in Java

When multiple threads are working on the same data, and the value of our data is changing, that scenario is not thread-safe and we will get inconsistent results. When a thread is already working on an object and preventing another thread on working on the same object, this process is called Thread-Safety.

### **How to achieve Thread Safety**

1. Using [Synchronization](https://www.geeksforgeeks.org/synchronized-in-java/).
2. Using [Volatile Keyword](https://www.geeksforgeeks.org/volatile-keyword-in-java/).
3. Using [Atomic Variable](https://www.geeksforgeeks.org/atomic-variables-in-java-with-examples/).
4. Using [Final Keyword](https://www.geeksforgeeks.org/final-keyword-java/).

| **First Level Cache** | **Second Level Cache** |
| --- | --- |
| This is local to the Session object and cannot be shared between multiple sessions. | This cache is maintained at the Se  ssionFactory level and shared among all sessions in Hibernate. |
| This cache is enabled by default and there is no way to disable it. | This is disabled by default, but we can enable it through configuration. |
| The first level cache is available only until the session is open, once the session is closed, the first level cache is destroyed. | The second-level cache is available through the application’s life cycle, it is only destroyed and recreated when an application is restarted. |
|  |  |