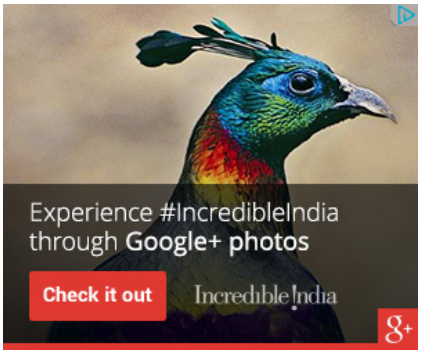
SATURDAY, MARCH 10, 2012

10 points on finalize method in Java - Tutorial Example



finalize method in java is a special method much like main method in java. `finalize()` is called before Garbage collector reclaim the Object, its last chance for any object to perform cleanup activity i.e. releasing any system resources held, closing connection if open etc. Main issue with finalize method in java is its not guaranteed by JLS that it will be called by Garbage collector or exactly when it will be called, for example an object may wait indefinitely after becoming eligible for garbage collection and before its finalize() method gets called. similarly even after finalize gets called its not guaranteed it will be immediately collected. Because of above reason it make no sense to use finalize method for releasing critical resources or perform any time critical activity inside finalize. It may work in development in one JVM but may not work in other JVM. In this java tutorial we will see some **important points about finalize method in Java**, *How to use finalize method*, what to do and what not to do inside finalize in Java.

what is finalize method in Java - Tutorial Example

1) `finalize()` method is defined in `java.lang.Object` class, which means it available to all the classes for sake of overriding. finalize method is defined as protected which leads to a popular core java question "**Why finalize is declared protected instead of public**"? well I don't know the exact reason its falls in same category of questions like why java doesn't support multiple inheritance which can only be answer accurately by designers of Java. any way making finalize protected looks good in terms of following rule of encapsulation which starts with least restrictive access modifier like private but making finalize private prevents it from being overridden in subclass as you can not override `private` methods, so making it `protected` is next obvious choice.

2) One of the most important point of finalize method is that its *not automatically chained like constructors*. If you are overriding finalize method than its your responsibility to call `finalize()` method of super-class, **if you forgot to call then finalize of super class will never be called**. so it becomes critical to remember this and provide an opportunity to finalize of super class to perform cleanup. Best way to call super class finalize method is to call them in `finally` block as shown in below example. this will granted that finalize of parent class will be called in all condition except when JVM exits:

```
@Override
   protected void finalize() throws Throwable {
       try{
           System.out.println("Finalize of Sub Class");
           //release resources, perform cleanup ;
       }catch(Throwable t){
           throw t;
       }finally{
           System.out.println("Calling finalize of Super Class");
           super.finalize();
       }

   }
```

3) finalize method is called by **garbage collection thread** before collecting object and if not intended to be called like normal method.

4) finalize gets called only once by GC thread, if object revive itself from finalize method than **finalize will not be called again**.

5) Any Exception thrown by finalize method is ignored by GC thread and it will not be propagated further, in fact I doubt if you find any trace of it.

6) There is one way you can guarantee running of finalize method by calling `System.runFinalization()` and `Runtime.getRuntime().runFinalization()`. These methods ensures that JVM call finalize() method of all object which are eligible for garbage collection and whose finalize has not yet called.

Alternative of finalize method for cleanup.

So far its seems we are suggesting not to use finalize method because of its non guaranteed behavior but than `what is alternative of releasing resource`, performing cleanup because there is no destructor in Java. Having a method

Recommended Reading

- 5 books to master object oriented and Java design patterns
- 5 Good books to learn Spring Framework
- 5 Must read jQuery books for Programmers
- 9 Must read Java Programming Books for Developers
- Which programming book to buy, if given $100 to spend

Subscribe To This Blog Free

Posts

Comments

Follow Us

Follow @javinpaul 3,781 followers

Javarevisited on Follow

+3,949

like `close()` or `destroy()` make much sense for releasing resources held by classes. In fact JDK library follows this. if you look at `java.io` package which is a great example of acquiring system resource like **file descriptor** for opening file, offers `close()` method for opening stream and `close()` for closing it. in fact its one of the best practice to call close method from finally block in java. Only caveat with this approach is its not automatic, client has to do the cleanup and if client forgot to do cleanup there are chances of resources getting leaked, which again suggest us that we could probably give another chance to finalize method. You will be pleased to know that Java 7 has added automatic resource management feature which takes care of closing all resource opened inside try block automatically, leaving no chance of manual release and leakage.

When to use finalize method in Java

As last paragraph pointed out that there are certain cases where *overriding finalize make sense*, like an ultimate last attempt to cleanup the resource. If a Java class is made to held resource like input output devices, JDBC connection than you should override finalize and call its `close()` method from finalize. though there is no guarantee that it will run and release the resource timely best part is we are not relying on it. It just another last attempt to release the resource which most likely have been already released due to client calling `close()` method. This technique is heavily used inside Java Development library. look at below example of finalize method from FileInputStream.java

```
protected void finalize() throws IOException {
        if ((fd != null) &&  (fd != FileDescriptor.in)) {

            /*
             * Finalize should not release the FileDescriptor if another
             * stream is still using it. If the user directly invokes
             * close() then the FileDescriptor is also released.
             */
            runningFinalize.set(Boolean.TRUE);
            try {
                close();
            } finally {
                runningFinalize.set(Boolean.FALSE);
            }
        }

    }
```

What not to do in finalize method in Java

trusting finalize method for releasing critical resource is biggest mistake java programmer can made. suppose instead of relying on `close()` method to release file descriptor, you rely on finalize to relapse it for you. Since there is no guaranteed when finalize method will run you could effectively lock hundreds of file-descriptor of earlier opened file or socket and there is high chance that your application will ran out of file-descriptor and not able to open any new file. Its best to use finalize as last attempt to do cleanup but **never use finalize as first or only attempt**.

That's all on **finalize method in Java**. as you have seen there are quite lot of *specifics about finalize method* which java programmer should remember before using finalize in java. In one liner don't do time critical task on finalize method but use finalize with caution.

Other **Java tutorials** you may like

How SubString method work internally in java
How to parse XML file in Java using DOM parser
How to convert Enum to String in java with Example
10 Best practices to follow while writing code comments in Programming
10 Design principle Java programmer should know
10 JVM options Java developer should know
How to avoid deadlock in Java – Code Example

Posted by Javin Paul at 2:12 AM

Labels: core java , core java interview question
Location: United States

Recent Posts

How Maven find dependency JARs while building Java Project

Blog Archive