protected void finalize() throws Throwable :

It is a predefined non static method of Object class.

Garbage Collector automatically call this method just before an object is eligible for garbage collection to perform clean-up activity.

Here clean-up activity means closing the resources associated with that object like file connection, database connection, network connection and so on we can say resource de-allocation.

Note :- a) JVM calls finalize method only one per object.

b) This method is deprecated from java 9V.

c) Calling finalize() method on a particular object is an indication that Object is eligible for GC

```
package com.ravi.finalize;

record Product(Integer id, String name)
{
    @Override
    public void finalize()
    {
        System.out.println("Product object is eligible for GC");
    }
}

public class FinalizeDemo
{
    public static void main(String[] args) throws InterruptedException
    {
        Product p1 = new Product(111, "Laptop");
        System.out.println(p1);

        p1 = null;

        System.gc();

        Thread.sleep(3000);

        System.out.println(p1);

    }

}
```

** What is the difference between final, finally and finalize()

final :

It is used to provide some kind of restriction, by using final modifier we can declare our class, Method and Field/variable as a final.

finally :

If we open any resourse as a part of try block then it should compulsory closed inside finally block, [If We close the resourses inside try block, It is always Risky] for resourse handling purpose.

finalize() :

It is a method of Object class, We should override this method in the corresponding class to perform cleanup activity, JVM will automatically call this method just before object destruction.

StringBuffer :-

While working with String class the drawback is memory consumption is very high because it is immutable so whenever we want to perform some operation on the existing String Object, a new String object will be created.

In order to solve the problem of immutability as well as high memory consumption, java software people has introdued a separate class called StringBuffer available in java.lang package from 1.0 onwards.

StringBuffer is a mutable class so we can modify the existing String hence automatically the memory consumption will be low but we have some performance issue because almost all the methods of StringBuffer class are synchronized so at a time only one thread can access the method of StringBuffer hence it is Thread-safe.

In order to solve this performance issue problem java software people has introduced StringBuilder class from 1.5v onwards.

StringBuilder :-

It is a predefined class available in java.lang packge. It is also mutable class. The only difference between StringBuffer and StringBuilder is, almost all the methods of StringBuffer are synchronized where as, all the methods of StringBuilder are non-synchronized hence performance wise StringBuilder is more better than StringBuffer.

Both the classes are sharing same API so, method name, return type, parameter list all are same.

**** What is the difference between String, StringBuffer and StringBuilder.

	String	StringBuffer	StringBuilder
Version	1.0 version	1.0 version	1.5 version
Object	Immutable and thread-safe	mutable	mutable
Storage	SCP (HEAP Memory)	HEAP MEMORY	HEAP MEMORY
Memory consumption	High	Less	Less
ThreadSafe	Threadsafe	Threadsafe	Not Threadsafe
Performance	Slow	Fast as compare to String	Fast as compare to StringBuffer
Application Area	No Frequently change in data	Data is changing frequently	Data is changing frequently

Note : All the immutable objects are bydefault Thread-safe

Program on StringTokenizer class

```
package com.ravi.finalize;

import java.util.StringTokenizer;

public class StringTokenizerDemo
{
    public static void main(String[] args)
    {
        String str = "Hyderabad is an IT city";

        StringTokenizer st = new StringTokenizer(str, "a");

        System.out.println("Total number of tokens :"+st.countTokens());

        while(st.hasMoreTokens())
        {
            System.out.println(st.nextToken());
        }

    }

}
```

public int countTokens() : Will provide number of Tokens
public boolean hasMoreTokens() : Will verify token is available or not
public String nextToken() : Will provide the token in String format

---

```
//String, StringBuffer and StringBuilder Objects comparison
public class Test25
{
public static void main(String args[])
    {
        StringBuilder sb1=new StringBuilder("Data"); //mutable
        sb1.append("Base");
        System.out.println(sb1);

        StringBuffer sb2=new StringBuffer("Data"); //mutable
        sb2.append("Base");
        System.out.println(sb2);

        String sb3 = new String("Data");  //immutable
       sb3.concat("Base");
      System.out.println(sb3);
    }
}
//public StringBuffer insert(int position, String str)
//Based on the index position we can insert the String
public class Test27
{
public static void main(String args[])
    {
        StringBuffer sb1=new StringBuffer("Hello");
        sb1.insert(2,"JSE");
        System.out.println(sb1);  //HeJSEllo

        StringBuilder sb2=new StringBuilder("Hello");
        sb2.insert(2,"JEE");
        System.out.println(sb2); //HeJEEllo
    }
}
//public AbstractStringBuilder reverse()
//Used to reverse the given String
class Test28
{
    public static void main(String[] args)
    {
        StringBuffer sb1=new StringBuffer("Hello");
        sb1.reverse();
        System.out.println(sb1); //olleH

        StringBuilder sb2=new StringBuilder("Java");
        sb2.reverse();
        System.out.println(sb2);  //avaJ
        }
}
//Program to demonstrate the performance of StringBuffer and StringBuilder classes.
public class Test29
{
    public static void main(String []args)
    {
      long startTime = System.currentTimeMillis();

        StringBuffer buffer = new StringBuffer("Java");

        for(int i=1; i<=1000000; i++)
        {
            buffer.append("Technology");
        }

        long endTime = System.currentTimeMillis();

        System.out.println("Total time taken bt StringBuffer class :"+(endTime - startTime)+" ms");


      startTime = System.currentTimeMillis();

        StringBuilder builder = new StringBuilder("Java");

        for(int i=1; i<=1000000; i++)
        {
            builder.append("Technology");
        }

        endTime = System.currentTimeMillis();

        System.out.println("Total time taken bt StringBuilder class :"+(endTime - startTime)+" ms");


    }
}
```