```java
package com.hdfc.collections;

//if class is final , it cannot extended
public final class FinalTest {

    //Variable 'value' might not have been initialized
    private final int value;
    public FinalTest(){
        this.value=10;
    }


    private final static String CONSTANT ="CONSTANT";
    private final static String ANTHER_CONSTANT_VALUE;
    static {
        ANTHER_CONSTANT_VALUE="some value";
    }


    public final void testMethod(final String value){

        //Cannot assign a value to final variable 'x'
       final int x =10;

       //Cannot assign a value to final variable 'value'
        //value="new String";
    }
}
```

```java
package com.hdfc.collections;

public class FinallyTest {


    public static void main(String[] args) {

        try {
            int a = 1;
            int b = 0;
            int c = a / b;
            System.out.println("try");
            //db = connect();
           // File open;
        }catch (Exception e){
            System.out.println("catch");
            //failure log;
            //execption log;
        }finally {
            System.out.println("finally");
            //close connection;
            //close file;
        }
    }
}
```

```java
package com.hdfc.collections;

public class FinalizeTest extends Object {

    private String name;

    public FinalizeTest(String name){
        this.name=name;
    }

    @Deprecated()
    public void test(){
        //
    }
    public String test1(){
        String value1 ="garbage String 1";
        String value2 ="garbage String 1";
        return "John";
    }


    /**
     * Called by the garbage collector on an object when garbage collection determines that there
are no more references to the object. A subclass overrides the finalize method to dispose of system
resources or to perform other cleanup.
     * When running in a Java virtual machine in which finalization has been disabled or removed,
the garbage collector will never call finalize(). In a Java virtual machine in which finalization is
enabled, the garbage collector might call finalize only after an indefinite delay.
     * The general contract of finalize is that it is invoked if and when the Java virtual machine
has determined that there is no longer any means by which this object can be accessed by any thread
that has not yet died, except as a result of an action taken by the finalization of some other
object or class which is ready to be finalized. The finalize method may take any action, including
making this object available again to other threads; the usual purpose of finalize, however, is to
perform cleanup actions before the object is irrevocably discarded. For example, the finalize method
for an object that represents an input/output connection might perform explicit I/O transactions to
break the connection before the object is permanently discarded.
     * The finalize method of class Object performs no special action; it simply returns normally.
Subclasses of Object may override this definition.
     * The Java programming language does not guarantee which thread will invoke the finalize method
for any given object. It is guaranteed, however, that the thread that invokes finalize will not be
holding any user-visible synchronization locks when finalize is invoked. If an uncaught exception is
thrown by the finalize method, the exception is ignored and finalization of that object terminates.
     * After the finalize method has been invoked for an object, no further action is taken until
the Java virtual machine has again determined that there is no longer any means by which this object
can be accessed by any thread that has not yet died, including possible actions by other objects or
classes which are ready to be finalized, at which point the object may be discarded.
     * The finalize method is never invoked more than once by a Java virtual machine for any given
object.
     * Any exception thrown by the finalize method causes the finalization of this object to be
halted, but is otherwise ignored.
     * Deprecated
     * Finalization is deprecated and subject to removal in a future release. The use of
finalization can lead to problems with security, performance, and reliability. See JEP 421  for
discussion and alternatives.
     * Subclasses that override finalize to perform cleanup should use alternative cleanup
mechanisms and remove the finalize method. Use java.lang.ref.Cleaner and
java.lang.ref.PhantomReference as safer ways to release resources when an object becomes
unreachable. Alternatively, add a close method to explicitly release resources, and implement
AutoCloseable to enable use of the try-with-resources statement.
     * This method will remain in place until finalizers have been removed from most existing code.
     * Throws:
     * Throwable - the Exception raised by this method
     * API Note:
     * Classes that embed non-heap resources have many options for cleanup of those resources. The
class must ensure that the lifetime of each instance is longer than that of any resource it embeds.
java.lang.ref.Reference.reachabilityFence can be used to ensure that objects remain reachable while
resources embedded in the object are in use.
     * A subclass should avoid overriding the finalize method unless the subclass embeds non-heap
resources that must be cleaned up before the instance is collected. Finalizer invocations are not
automatically chained, unlike constructors. If a subclass overrides finalize it must invoke the
superclass finalizer explicitly. To guard against exceptions prematurely terminating the finalize
chain, the subclass should use a try-finally block to ensure super.finalize() is always invoked. For
example,
     *   @Override protected void finalize() throws Throwable {      try {           ... // cleanup
subclass state      } finally {           super.finalize();      }
     * @throws Throwable
     */
    @Override
    protected void finalize() throws Throwable {
        //not suggested to use this method
        //this method will be called by Garbage collection
        System.out.println("finalize");
        //to clean the resource
    }

    public static void main(String[] args) throws InterruptedException {

        FinalizeTest object=   new FinalizeTest("test");
        object=null;


        //System.exit(0); //Terminates the currently running Java Virtual Machine

        //to start Garbage Collection
        System.gc();
        ///Runtime.getRuntime().gc();
        Thread.sleep(5000);

        System.out.println(Runtime.getRuntime().availableProcessors());
        System.out.println(Runtime.getRuntime().freeMemory()); //bytes
        System.out.println(Runtime.getRuntime().maxMemory());//bytes
        System.out.println(Runtime.getRuntime().totalMemory());//bytes

    }
}
```

```
package com.hdfc.collections;

public class Test {

    Test i;

    public static void main(String[] args) {

        Test t1 = new Test();
        Test t2 = new Test();
        Test t3 = new Test();

        t1.i = t2;
        t2.i=t3;
        t3.i = t1;

        //OutOfMemoeryError
        for (int i=0; i< Integer.MAX_VALUE; i ++){
            Test t = new Test();
            System.out.println(i);
        }


    }
}
```