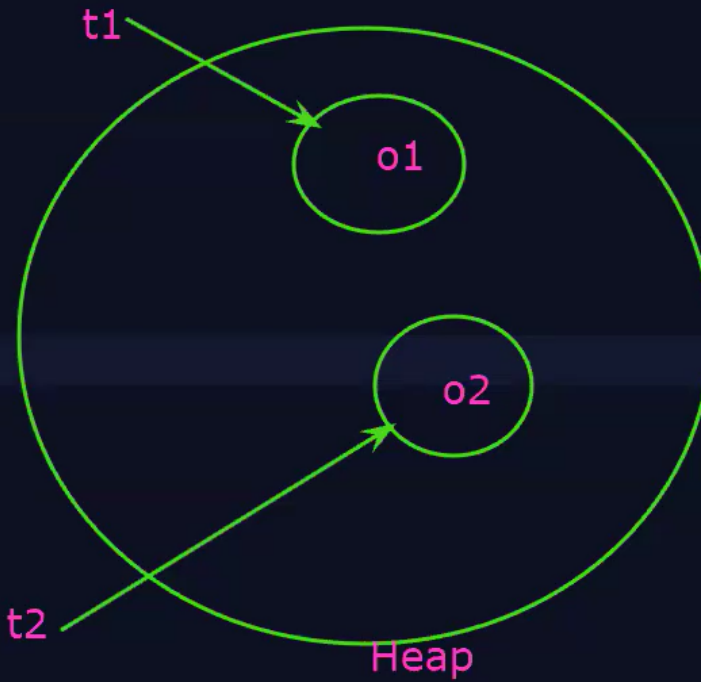
Done
DownCastingDemo

```
syntax:
-keyword 'super'
-super():implicitly call superclass constructor
-super cannot be used inside the static methods
-A subclass cannot access private members of the superclass using super keyword


'final' Keyword:constant
-------------------------
-It is used to restrict modifications in three context:
1. Final variable
    -value cannot be changed (constant)
    -final variable is a constant that cannot be reassigned after initialization

2. Final Method
    -cannot be overriden
3. Final class
    -cannnot be inherited
-----------------------------
4. Final parameter:
    -Method parameters cannot be changed
5. Blank Final Variable
```

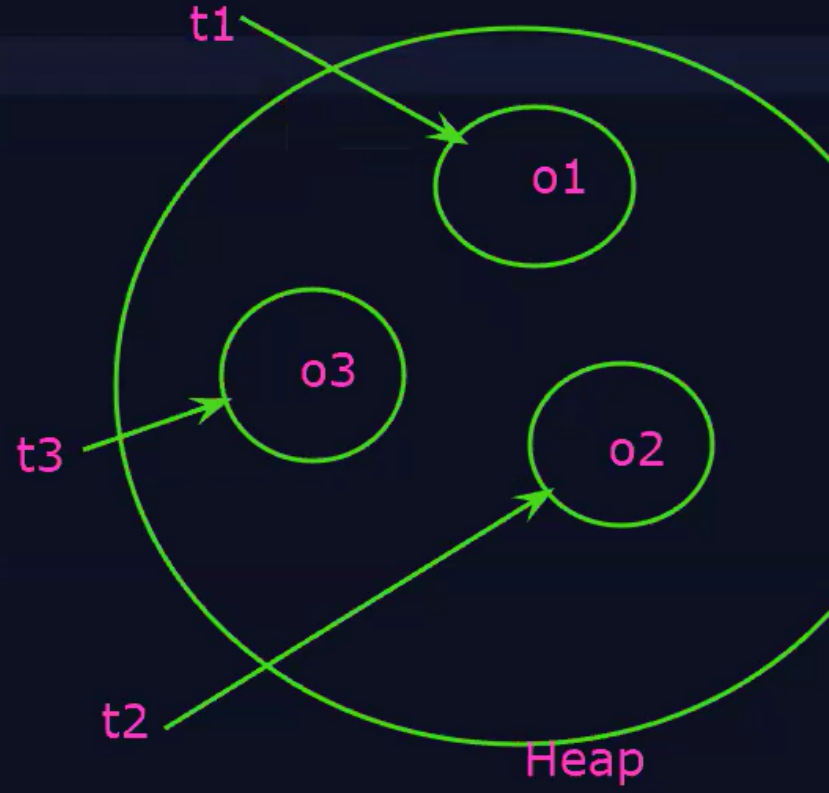
```
instanceof keyword: operator
-----------------------------
-It is used to check whether an object is an instance of a specific class or
subclass.
-result=> boolean (true/false)

Upcasting:
---------
-Assigning a subclass object to a superclass reference
-Allows a subclass object to be treated as asuperclass object
-It enables overriding and dynamic binding(dynamic method dispatch)
-imporoves code flexibility and reusability.


Downcasting:
---------------
-Converting a superclass reference back into subclass reference
-It enables calling subclass specific methods
```

```
1398
1399  Garbage Collection: JVM
1400  --------------------
1401  -Automatic memory management
1402  -GC : Garbage Collector : to free up heap memory
1403  -It works on objects that are no longer references
1404
1405  Request GC methods:(2 ways)
1406  -------------------
1407  1. System.gc();//Method 1 : REquest GC to run
1408  2. Runtime.getRuntime().gc();// Method 2: Way to request GC
1409
1410
1411  Ways to make object eligble for Garbage Collection:
1412  ---------------------------------------------------
1413  1. Nulling a Reference variable:
1414
1415  Test t1 = new Test();
1416  Test t2 = new Test();
1417
1418  t1 = null;//Object is now eligible for GC
```

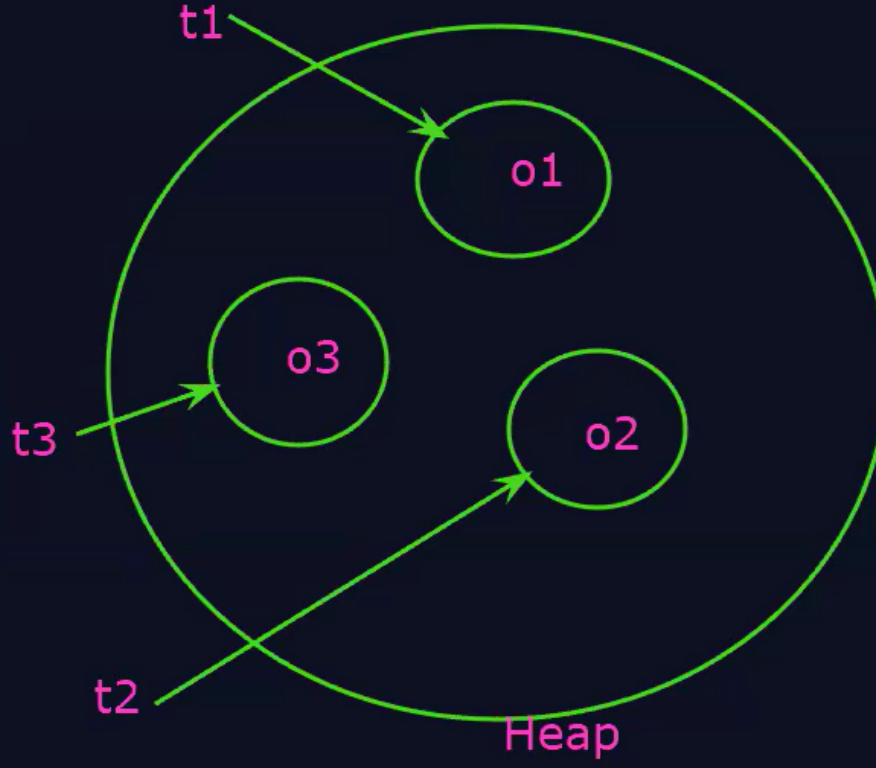
```
1405  Request GC methods:(2 ways)
1406  --------------------
1407  1. System.gc();//Method 1 : REquest GC to run
1408  2. Runtime.getRuntime().gc();// Method 2: Way to request GC
1409
1410
1411  Ways to make object eligble for Garbage Collection:
1412  ------------------------------------------------
1413  1. Nulling a Reference variable:
1414
1415  Test t1 = new Test();
1416  Test t2 = new Test();
1417
1418  t1 = null;//Object is now eligible for GC
1419  t2 = null;
1420
1421  2. Re-Assigning a Reference Variable:
1422
1423  Test t1 = new Test();
1424  Test t2 = new Test();
1425  Test t3 = new Test();
1426
```



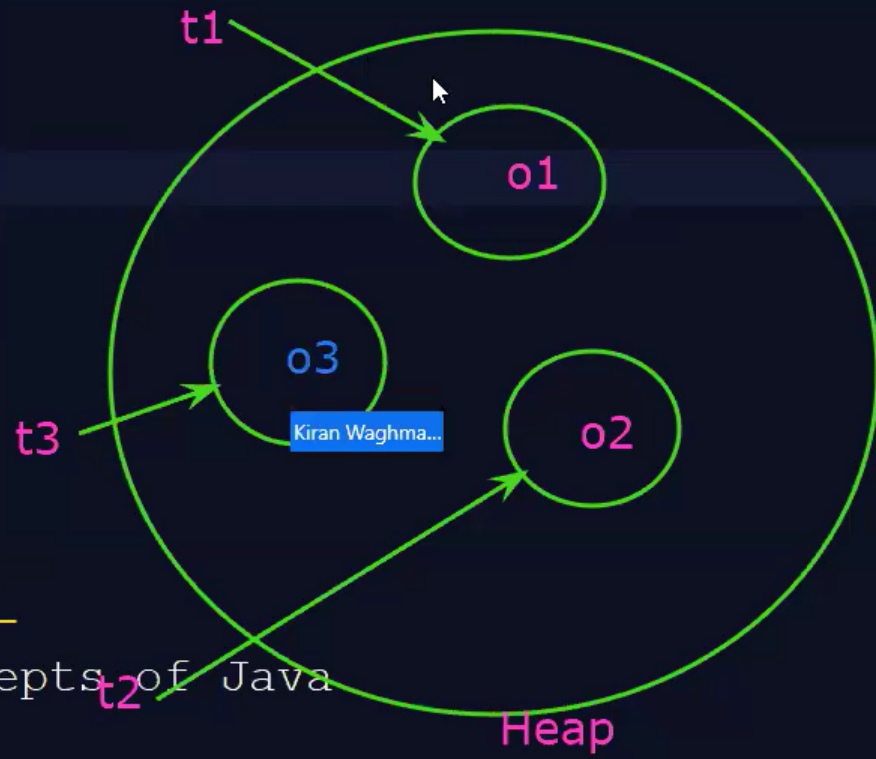
```
Ways to make object eligble for Garbage Collection:
-----------------------------------------------------
1. Nulling a Reference variable:

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

t1 = null;//Object is now eligible for GC
t2 = null;

2. Re-Assigning a Reference Variable:

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

t1=t2;
t3=t2;

3. Island of Isolation:

Test t1 = new Test();
```



```
2. Re-Assigning a Reference Variable:

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

t1=t2;
t3=t2;

3. Island of Isolation:

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

t1.obj1 = t2.obj2;
t2.obj2 = t3.obj3;
t3.obj3 = t1.obj1;

t1=null;
t1=null;
t3=null;
```



```
3. Island of Isolation:

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

t1.obj1 = t2.obj2;
t2.obj2 = t3.obj3;
t3.obj3 = t1.obj1;

t1=null;
t1=null;
t3=null;


1. 80%:100GB: threshold
2. Memory Full
3. Depends on his mood

----------------------------------------
Day 10: Object Oriented Programming Concepts of Java
Date : 13-March-2025
```



java.awt.image	Provides classes for creating and modifyin
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing *beans* -- components based on the JavaBeans™ architecture.
java.beans.beancontext	Provides classes and interfaces relating to bean context.
java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.lang.annotation	Provides library support for the Java programming language annotation facility.
java.lang.instrument	Provides services that allow Java programming language agents to instrument programs running on the JVM.
java.lang.invoke	The `java.lang.invoke` package contains dynamic language support provided directly by the Java core class libraries and virtual machine.
java.lang.management	Provides the management interfaces for monitoring and management of the Java virtual machine and other

Standard Ed. 8

All Classes All Profiles

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom

All Classes

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractAnnotationValueVisitor8
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeCont

java.text.spi	Service provider classes for the classes in the java.text package.
java.time	The main API for dates, times, instants, and durations.
java.time.chrono	Generic API for calendar systems other than the default ISO.
java.time.format	Provides classes to print and parse dates and times.
java.time.temporal	Access to date and time using fields and units, and date time adjusters.
java.time.zone	Support for time-zones and their rules.
java.util	Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).
java.util.concurrent	Utility classes commonly useful in concurrent programming.
java.util.concurrent.atomic	A small toolkit of classes that support lock-free thread-safe programming on single variables.
java.util.concurrent.locks	Interfaces and classes providing a framework for locking and waiting for conditions that is distinct from built-in synchronization and monitors.
java.util.function	*Functional interfaces* provide target types for lambda

finalize

```
protected void finalize()
                throws Throwable
```

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.

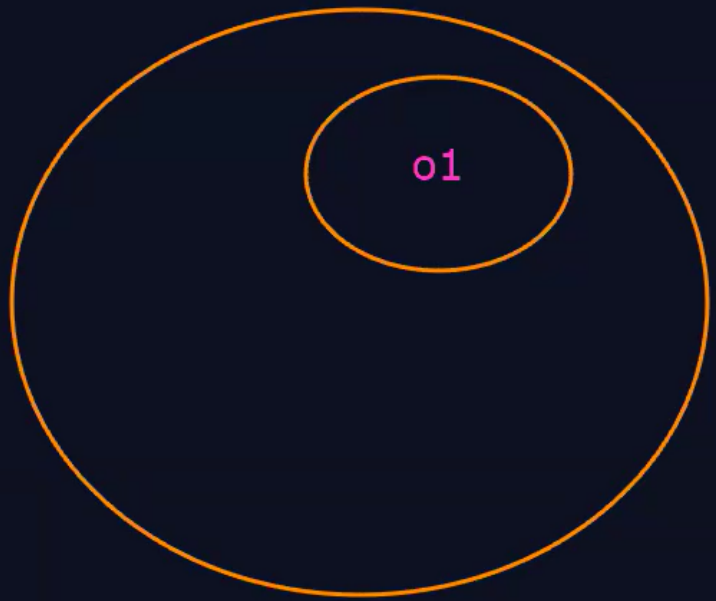
The general contract of `finalize` is that it is invoked if and when the Java™ virtual machine has determined that there is no longer any means by which this object can be accessed by any thread that has not yet died, except as a result of an action taken by the finalization of some other object or class 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.

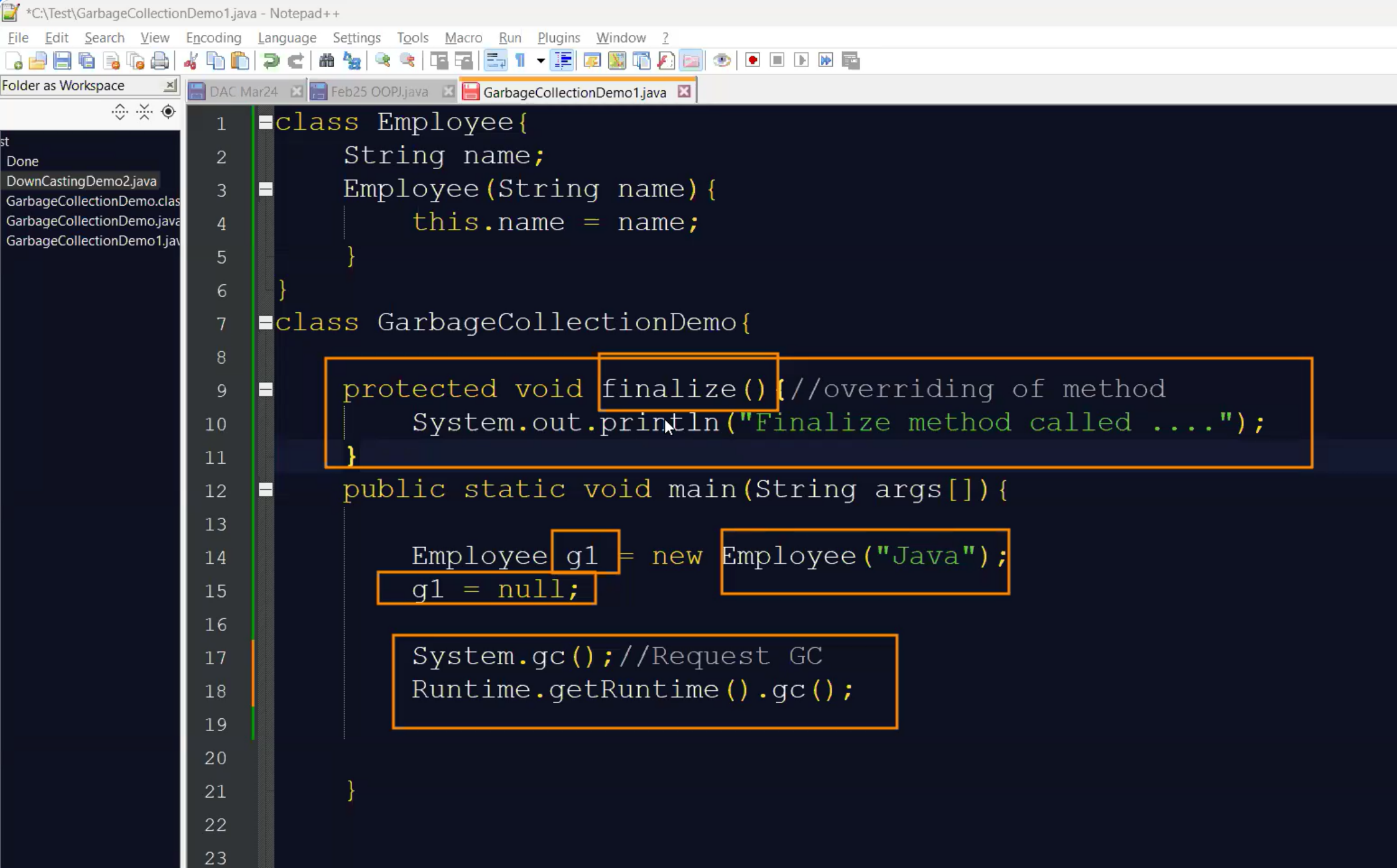
The `finalize` method of class `Object` performs no special action; it simply returns normally. Subclasses of `Object` may override this definition.

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-

st
Done
DownCastingDemo2.java
GarbageCollectionDemo.java

```
2  class GarbageCollectionDemo{
3
4      protected void finalize(){
5          System.out.println("Finalize method called ....");
6      }
7      public static void main(String args[]){
8
9          GarbageCollectionDemo g1 = new GarbageCollectionDemo();
10         g1 = null;
11
12         System.gc();//Request GC
13
14
15     }
16
17                          g1
18 }
19
20
```




*C:\Test\GarbageCollectionDemo1.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Folder as Workspace
DAC Mar24
Feb25 OOPJ.java
GarbageCollectionDemo1.java
Done
DownCastingDemo2.java
GarbageCollectionDemo.clas
GarbageCollectionDemo.java
GarbageCollectionDemo1.jav
class Employee{
String name;
Employee(String name){
this.name = name;
}
}
class GarbageCollectionDemo{
protected void finalize(){//overriding of method
System.out.println("Finalize method called ....");
}
public static void main(String args[]){
Employee g1 = new Employee("Java");
g1 = null;
System.gc();//Request GC
Runtime.getRuntime().gc();
}

*E:\CDAC\K10_Feb25\Feb25 OOPJ\Feb25 OOPJ.java - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Folder as Workspace

DAC Mar24 | Feb25 OOPJ.java | GarbageCollectionDemo1.java

Done
DownCastingDemo2.java
GarbageCollectionDemo.clas
GarbageCollectionDemo.java
GarbageCollectionDemo1.jav

```
1393  ---------------
1394  -Converting a superclass reference back into subclass reference
1395  -It enables calling subclass specific methods
1396  Parent p = new Parent();
1397  child d = (child) p;
1398
1399  Garbage Collection: JVM
1400  ----------------------
1401  -Automatic memory management
1402  -GC : Garbage Collector : to free up heap memory
1403  -It works on objects that are no longer references
1404
1405  Request GC methods:(2 ways)
1406  ---------------------
1407  1. System.gc();//Method 1 : REquest GC to run
1408  2. Runtime.getRuntime().gc();// Method 2: Way to request GC
1409
1410
1411  Ways to make object eligble for Garbage Collection:
1412  ---------------------------------------------------
1413  1. Nulling a Reference variable:
1414
1415  Test t1 = new Test();
```

finalize

```
protected void finalize()
                throws Throwable
```

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.

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-

```java
class Employee{
    String name;
    Employee(String name){
        this.name = name;
    }
}
class GarbageCollectionDemo1{

    protected void finalize(){//overriding of method
        //System.out.println("Finalize method called ....");
    }
    public static void main(String args[]){

        Employee g1 = new Employee("Java");
        g1 = null;

        System.gc();//Request GC : JVM
        Runtime.getRuntime().gc();


    }
```

java.lang

Interfaces

Appendable
AutoCloseable
CharSequence
Cloneable
Comparable
Iterable
Readable
Runnable
Thread.UncaughtExceptionHandl

Classes

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

`Serializable, CharSequence, Comparable<String>`

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

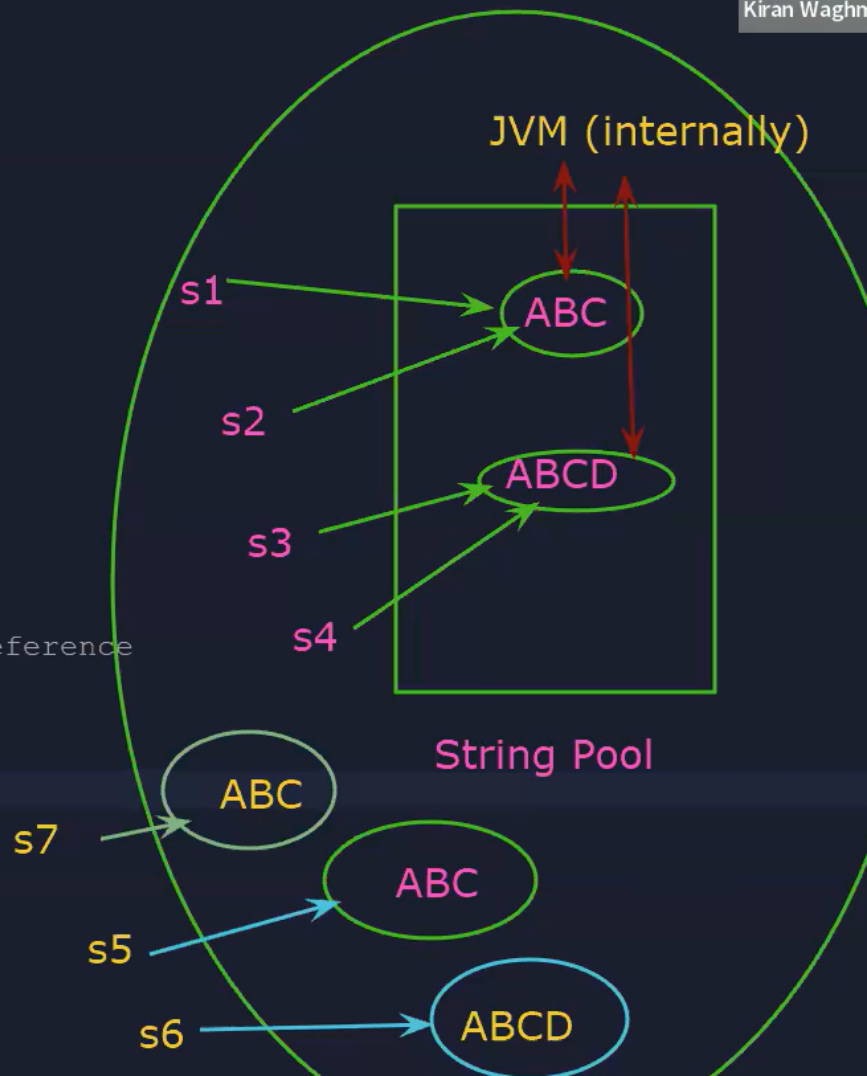
Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

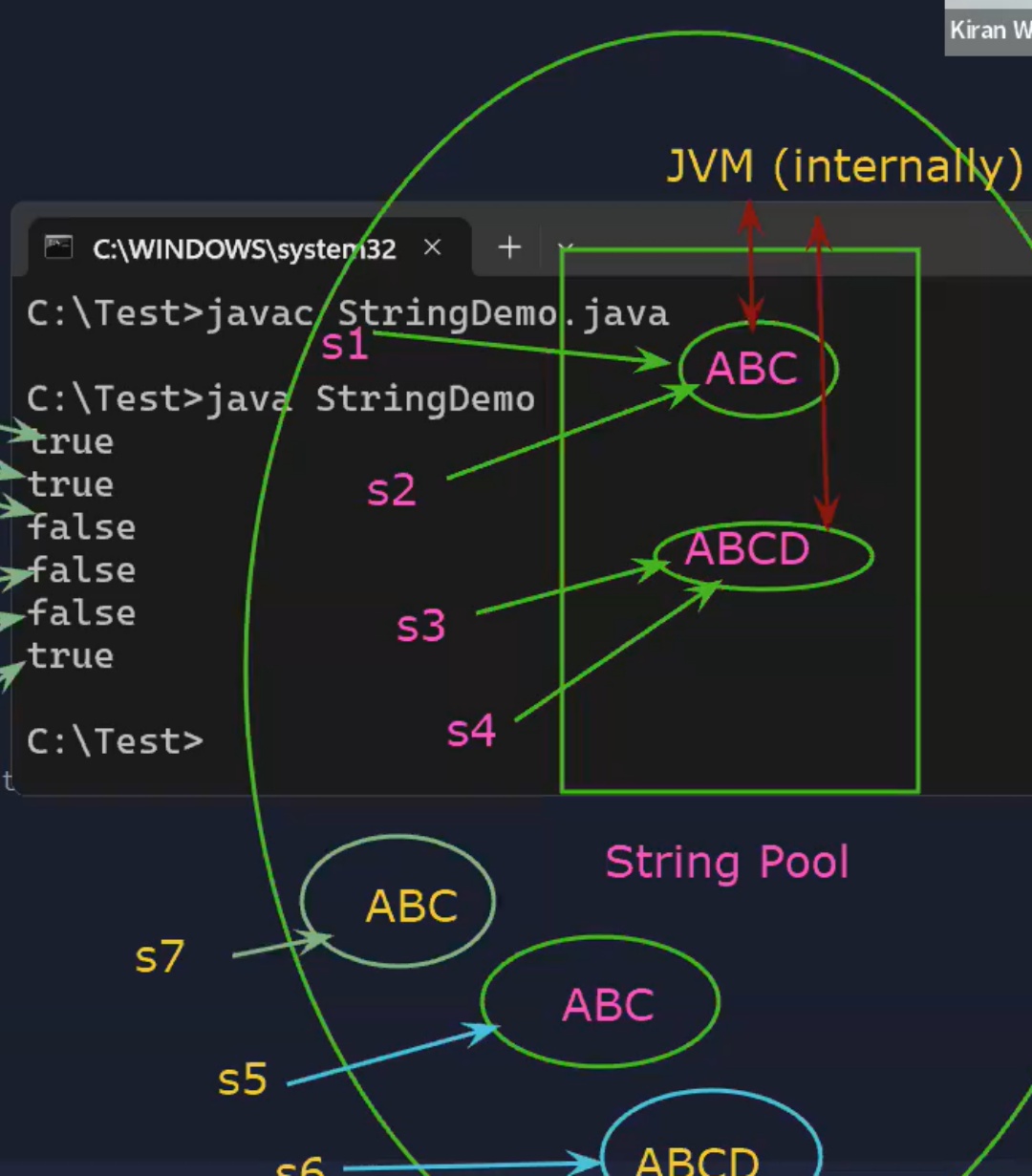
```
1460
1461  finalize(): method is called before an object is garbage collected to perform
      cleanup operations.
1462
1463  String:
1464  --------
1465
1466  - A string in Java is an object that represents a sequence of charaters
1467  - Package: import java.lang.String class
1468  - Immutable (cannot be changed after creation)
1469  - Implementes Serializable , Comparable and CharSequence interfaces.
1470  - String can be created using String literals and 'new' keyword.
```

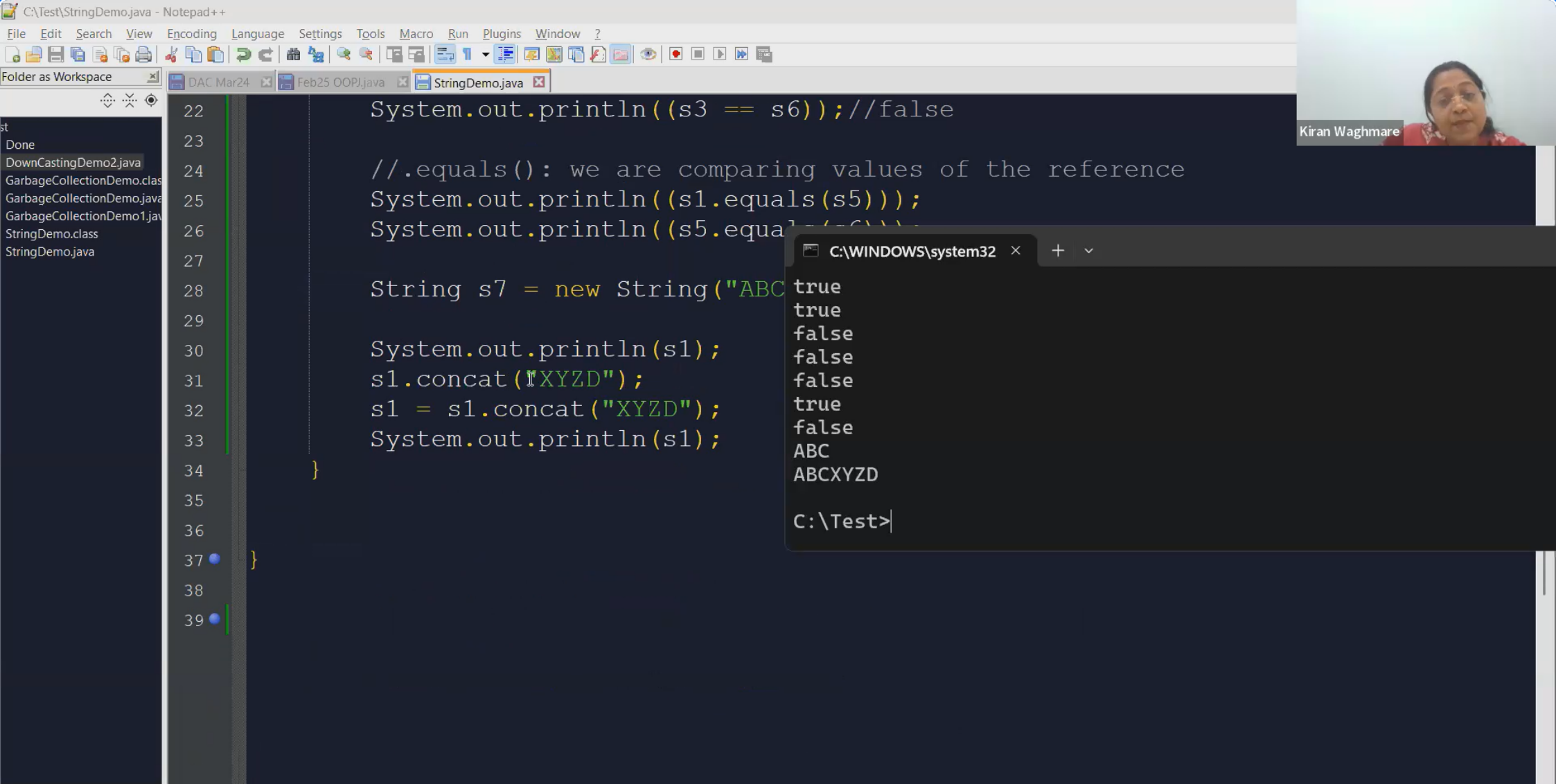
```java
public static void main(String args[]){
    //Method 1
    String s1 = "ABC";//String literal
    String s2 = "ABC";
    String s3 = "ABCD";
    String s4 = "ABCD";

    //== will compare reference content
    //and not the object content (values)
    System.out.println((s1 == s2));//true
    System.out.println((s3 == s4));
    System.out.println((s1 == s3));
    //Method 2
    String s5 = new String("ABC");
    String s6 = new String("ABCD");
    System.out.println((s5 == s6));//false
    System.out.println((s3 == s6));//false

    //.equals(): we are comparing values of the reference
    System.out.println((s1.equals(s5)));
    System.out.println((s5.equals(s6)));

    String s7 = new String("ABC");
}
}
```



```java
public static void main(String args[]){
    //Method 1 : String literals
    String s1 = "ABC";//String literal
    String s2 = "ABC";
    String s3 = "ABCD";
    String s4 = "ABCD";

    //== will compare reference content
    //and not the object content (values)
    System.out.println((s1 == s2));//true
    System.out.println((s3 == s4));
    System.out.println((s1 == s3));

    //Method 2
    String s5 = new String("ABC");
    String s6 = new String("ABCD");
    System.out.println((s5 == s6));//false
    System.out.println((s3 == s6));//false

    //.equals(): we are comparing values of t
    System.out.println((s1.equals(s5)));
    System.out.println((s5.equals(s6)));

    String s7 = new String("ABC");

    System.out.println(s1);
    s1.concat("XYZD");
    s1 = s1.concat("XYZD");
    System.out.println(s1);
}
```

```
C:\Test>javac StringDemo.java

C:\Test>java StringDemo
true
true
false
false
false
true

C:\Test>
```




C:\Test\StringDemo.java - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Folder as Workspace
DAC Mar24
Feb25 OOPJ.java
StringDemo.java
Done
DownCastingDemo2.java
GarbageCollectionDemo.clas
GarbageCollectionDemo.java
GarbageCollectionDemo1.jav
StringDemo.class
StringDemo.java
System.out.println((s3 == s6));//false
//.equals(): we are comparing values of the reference
System.out.println((s1.equals(s5)));
String s7 = new String("ABC
System.out.println(s1);
s1.concat("XYZD");
s1 = s1.concat("XYZD");
System.out.println(s1);
}
}
Kiran Waghmare
C:\WINDOWS\system32
true
true
false
false
false
true
false
ABC
ABCXYZD
C:\Test>

C:\Test\StringDemo.java - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Folder as Workspace

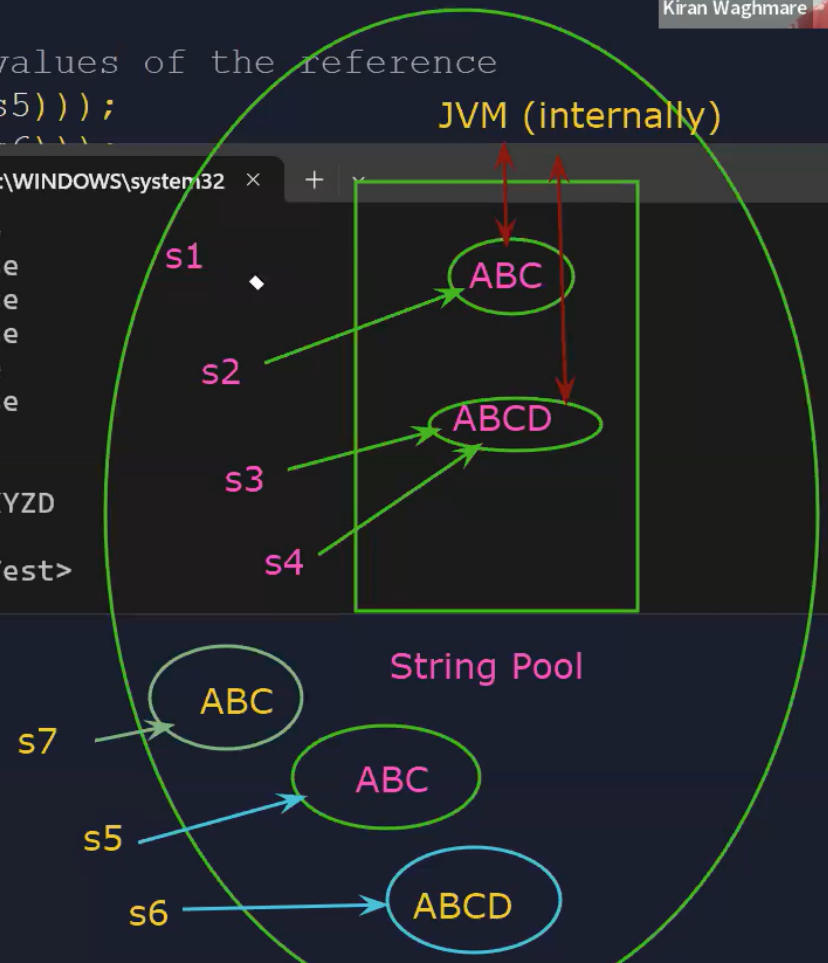
DAC Mar24 | Feb25 OOPJ.java | StringDemo.java

Done
DownCastingDemo2.java
GarbageCollectionDemo.clas
GarbageCollectionDemo.java
GarbageCollectionDemo1.jav
StringDemo.class
StringDemo.java

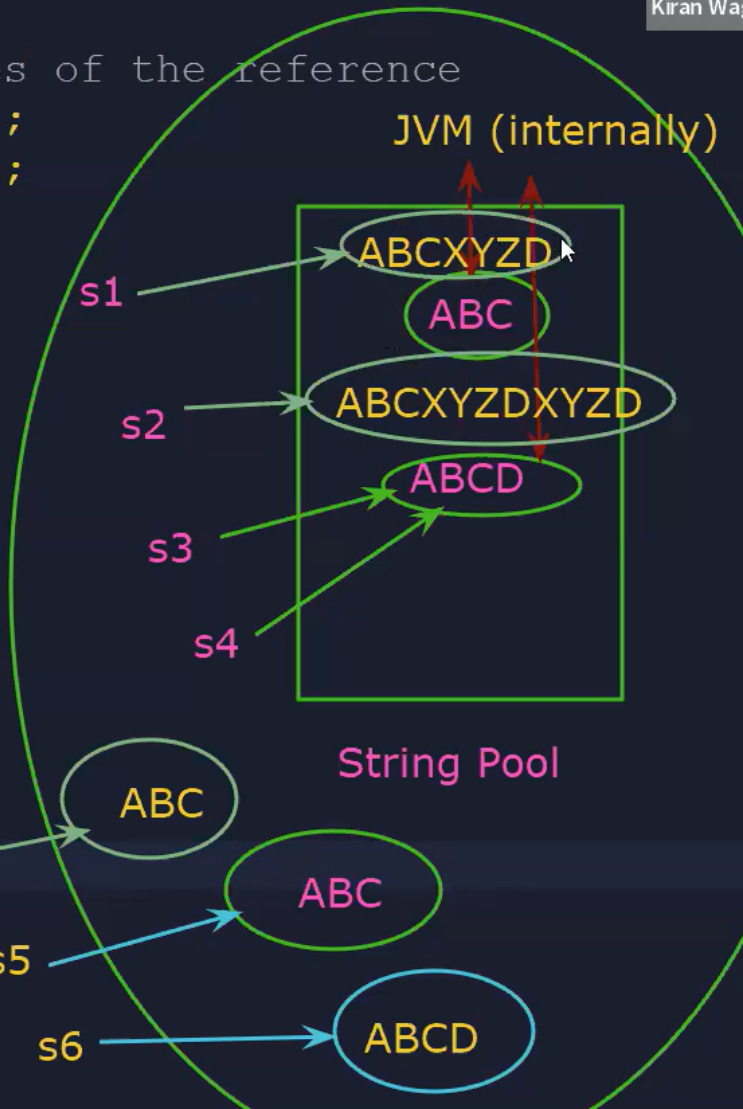
```
22          System.out.println((s3 == s6));//false
23
24          //.equals(): we are comparing values of the reference
25          System.out.println((s1.equals(s5)));
26          System.out.println((s5.equal
27
28          String s7 = new String("ABC
29
30          System.out.println(s1);
31          s1.concat("XYZD");
32          System.out.println(s1);
33          s1 = s1.concat("XYZD");
34          System.out.println(s1);
35      }
36
37
38  }
39
40
```

C:\WINDOWS\system32

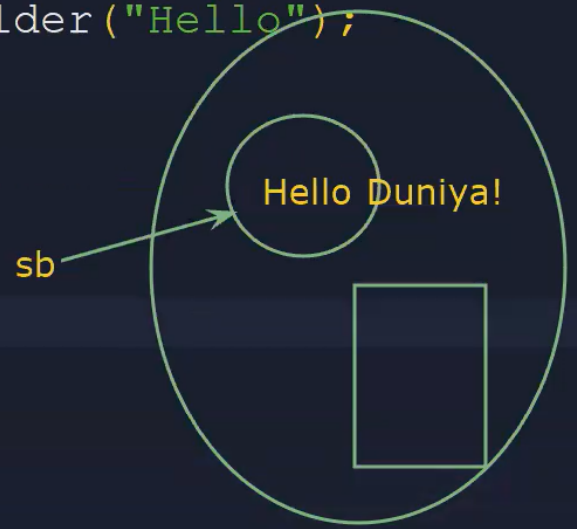
```
true
false
false
false
true
false
ABC
ABC
ABCXYZD

C:\Test>
```



```
22          System.out.println((s3 == s6));//false
23
24          //.equals(): we are comparing values of the reference
25          System.out.println((s1.equals(s5)));
26          System.out.println((s5.equals(s6)));
27
28          String s7 = new String("ABC");
29
30          System.out.println(s1);
31          s1.concat("XYZD");
32          System.out.println(s1);
33          s1 = s1.concat("XYZD");
34          System.out.println(s1);
35          s2 = s1.concat("XYZD");
36      }
37
38
39  }
40
41
```

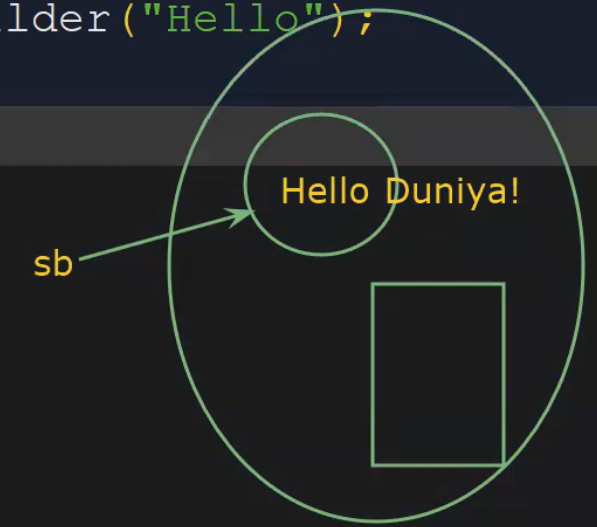


```
5    public static void main(String args[]){
6
7    StringBuffer sb = new StringBuffer("Hello");
8    System.out.println(sb);
9    sb.append("Duniya!");
10   System.out.println(sb);
11
12   StringBuilder sb1 = new StringBuilder("Hello");
13   System.out.println(sb1);
14   sb1.append("CDAC!");
15   System.out.println(sb1);
16   }
17
18
19 }
20
21
```



```
5    public static void main(String args[]){
6
7    StringBuffer sb = new StringBuffer("Hello");
8    System.out.println(sb);
9    sb.append("Duniya!");
10   System.out.println(sb);
11
12   StringBuilder sb1 = new StringBuilder("Hello");
13   System.out.println(sb1);
14   sb1.append
15   System.out
16   }
17
18
19 }
20
21
```

```
C:\WINDOWS\system32
4

C:\Test>javac StringDemo2.java

C:\Test>java StringDemo2
Hello
HelloDuniya!

C:\Test>javac StringDemo2.java

C:\Test>javac StringDemo2.java
```

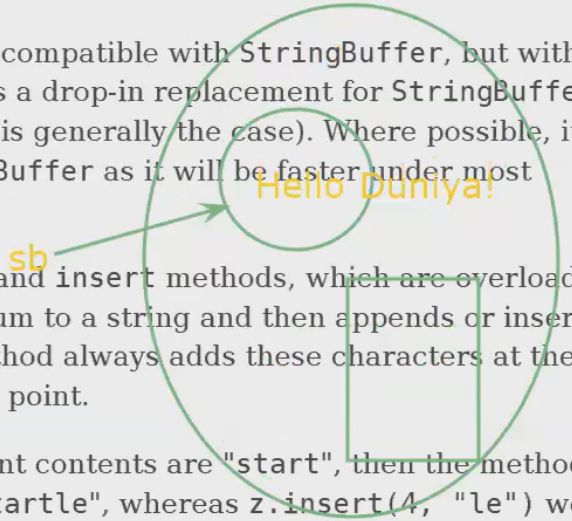


All Implemented Interfaces:

`Serializable, Appendable, CharSequence`

```
public final class StringBuilder
extends Object
implements Serializable, CharSequence
```

A mutable sequence of characters. This class provides an API compatible with `StringBuffer`, but with no guarantee of synchronization. This class is designed for use as a drop-in replacement for `StringBuffer` in places where the string buffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to `StringBuffer` as it will be faster under most implementations.



The principal operations on a `StringBuilder` are the `append` and `insert` methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string builder. The `append` method always adds these characters at the end of the builder; the `insert` method adds the characters at a specified point.

For example, if `z` refers to a string builder object whose current contents are `"start"`, then the method call `z.append("le")` would cause the string builder to contain `"startle"`, whereas `z.insert(4, "le")` would alter the string builder to contain `"starlet"`.

```
5   public static void main(String args[]){
6
7   StringBuffer sb = new StringBuffer("Hello");
8   System.out.println(sb);
9   sb.append("Duniya!");
10  System.out.println(sb);
11
12  StringBuilder sb1 = new StringBuilder("Hello");
13  System.out.println(sb1);
14  sb1.append("CDAC!");
15  System.out.println(sb1);
16
17  String s = "Hello";
18  System.out.println(s);
19  s.concat("Bhai!");
20  System.out.println(s);
21  s.concat("Bhai!");
22  System.out.println(s);
23  }
```