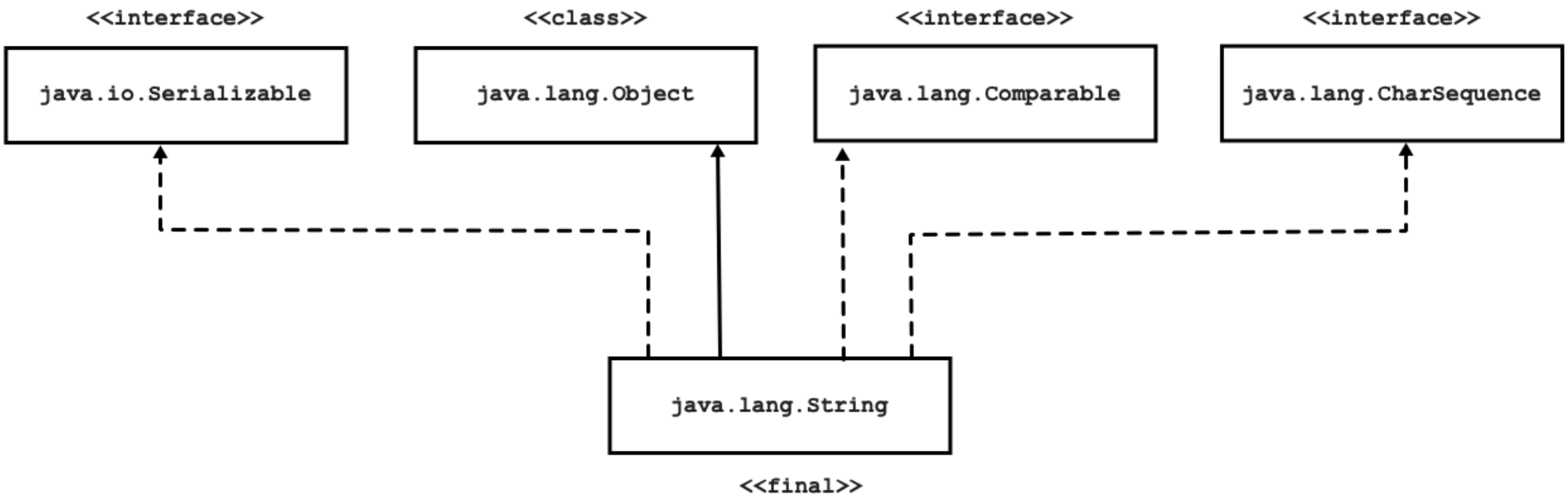
String Handling

- In Java, String is collection of character objects which do not end with null character.
- To manipulate String, we can use below classes:
    - java.lang.String
    - java.lang.StringBuffer
    - java.lang.StringBuilder
    - java.util.StringTokenizer
    - java.util.regex.Matcher
    - java.util.regex.Pattern

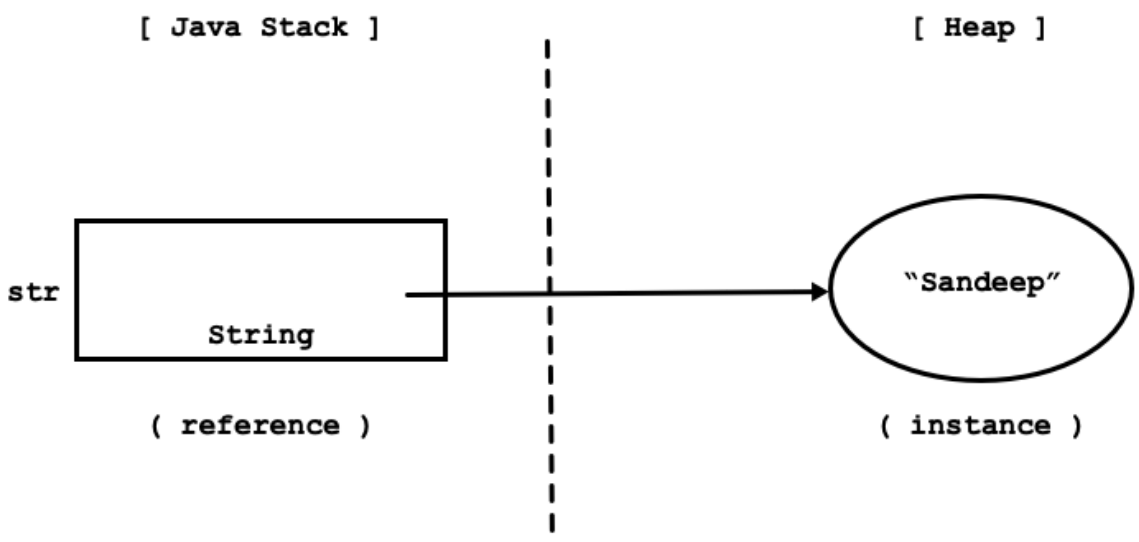
String



- String is not a built in type in Java. It is a final class declared in java.lang package. Hence it is considered as a non primitive type.
- It implements Serializable, Comparable and CharSequence interface.
- We can create instance of String using new operator as well as without new operator.
    - Code Snippet 1:

```
String str = new String("Sandeep");
//String str  => String reference
//new String("Sandeep"); => String instance
```

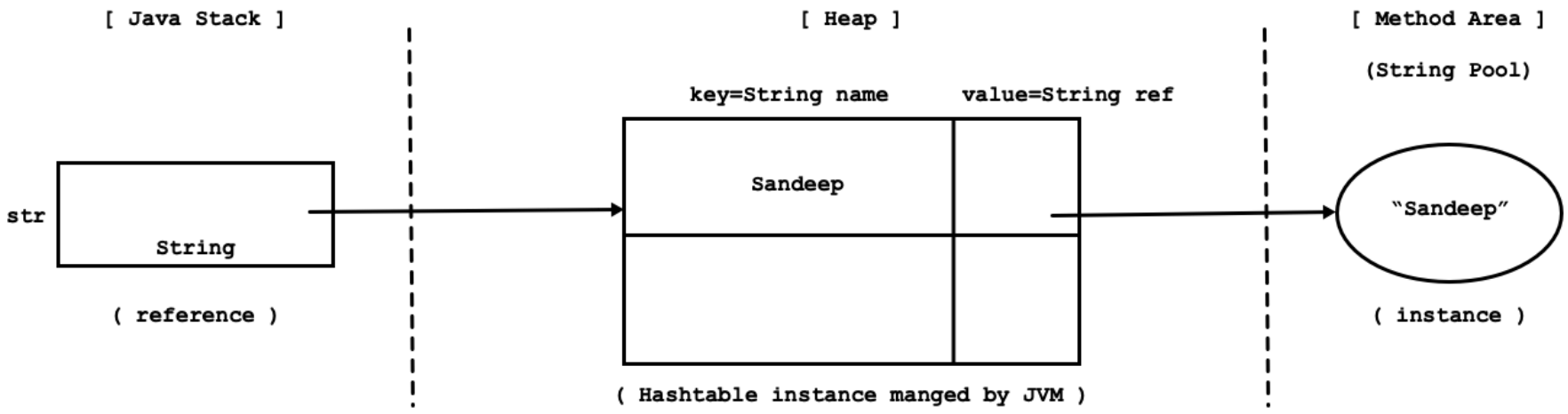
        - If we create instance of String using new operator then it gets space on Heap. Consider below image:



    - Code Snippet 2:

```
String str = "Sandeep";
//String str => String reference
//"Sandeep" => String literal
```

- If we create instance of String without new operator then it gets space on String literal/constant pool on method area. Consider below image:



- Let us see how String literals are implicitly interpreted by the compiler:

```
String str = "Sandeep";

//is equivalent to:

 char data[] = {'S', 'a', 'n', 'd', 'e', 'e', 'P'};
 String str = new String(data);
```

- In Java, to concatenate strings, we can use either + operator or concat method. Consider below code:
  - Code Snippet 1:

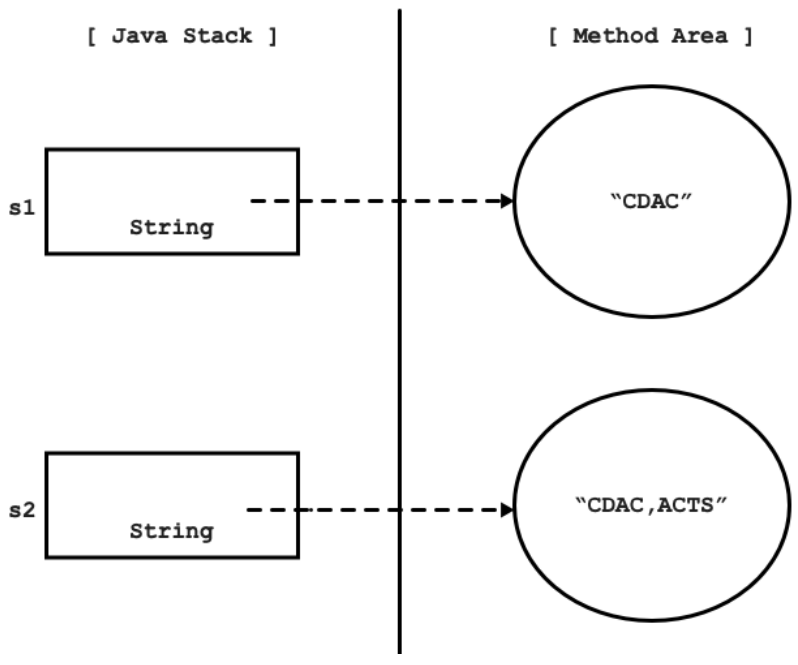
```
String s1 = "Hello";
String s2 = " World!!";
String s3 = s1.concat( s2 );
```

  - Code Snippet 2:

```
String s1 = "Hello";
String s2 = " World!!";
String s3 = s1 + s2;
```

- String objects are immutable. It means that, if we try to modify String then JVM create new instance of String. Consider below example:

```
String s1 = "CDAC";
String s2 = s1.concat(",ACTS");
System.out.println( s1 == s2 ); //false
```



- In Java, strings are implemented using UNICDE encoding.

String Constructors

- public String()

```
String str = new String( );
```

- public String(byte[] bytes)

```
byte[] bytes = new byte[ ]{ 83, 97, 110, 100, 101, 101, 112 };
String str = new String( bytes );
```

- public String(char[] value)

```
char data[] = {'S', 'a', 'n', 'd', 'e', 'e', 'P'};
String str = new String( data );
```

- public String(String original)

```
String str = new String("Sandeep");
```

- public String(StringBuffer buffer)

```
StringBuffer buffer = new StringBuffer("Sandeep");
String str = new String( buffer );
```

- public String(StringBuilder builder)

```
StringBuilder builder = new StringBuilder( "Sandeep" );
String str = new String( builder );
```

String Methods

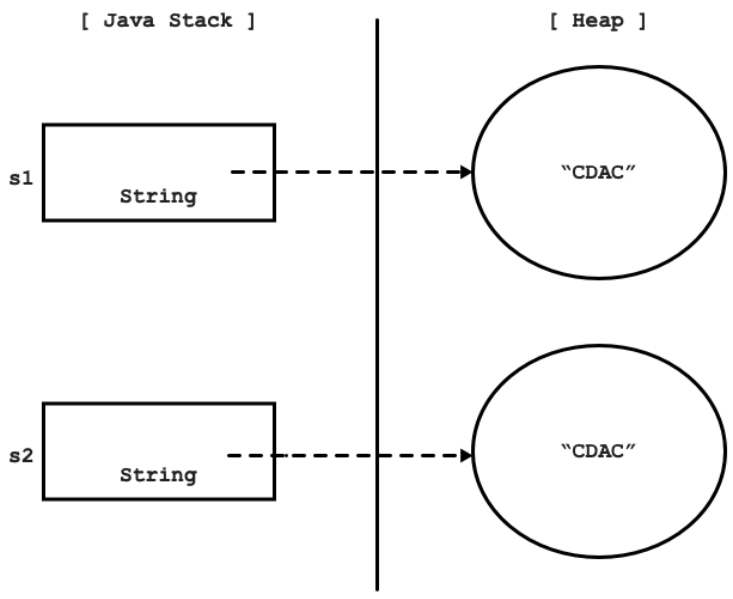
- public char charAt(int index)
- public int compareTo(String anotherString)
- public String concat(String str)
- public boolean contains(CharSequence s)
- public boolean endsWith(String suffix)
- public boolean equals(Object anObject)
- public boolean equalsIgnoreCase(String anotherString)
- public static String format(String format, Object... args)
- public byte[] getBytes()
- public int indexOf(int ch)
- public int lastIndexOf(int ch)
- public int indexOf(String str)
- public int lastIndexOf(String str)
- public String intern()
- public int length()
- public boolean isEmpty()
- public String[] split(String regex)
- public String substring(int beginIndex)
- public String substring(int beginIndex, int endIndex)
- public char[] toCharArray()
- public String toLowerCase()
- public String toUpperCase()
- public String trim()
- public static String valueOf(Object obj)

String Twisters

- Code Snippet 1

```
String s1 = new String("CDAC");
String s2 = new String("CDAC");
if( s1 == s2 )
```

```
    System.out.println("Equal");
  else
    System.out.println("Not Equal");
  //Output: Not Equal
```

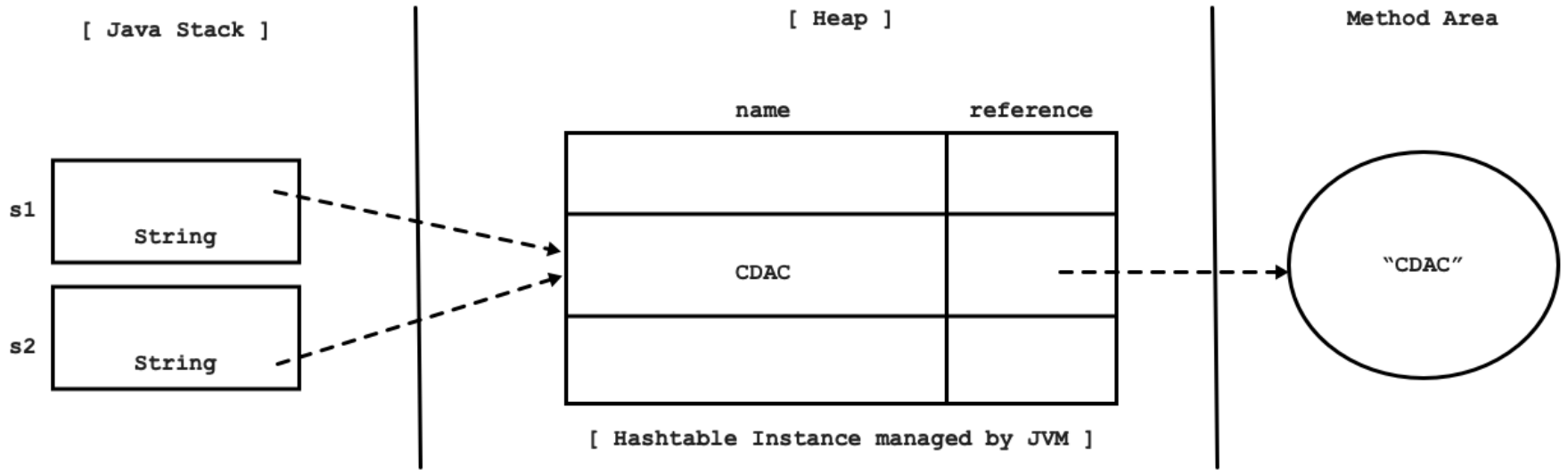


- Code Snippet 2

```
  String s1 = new String("CDAC");
  String s2 = new String("CDAC");
  if( s1.equals( s2 ) )
    System.out.println("Equal");
  else
    System.out.println("Not Equal");
  //Output: Equal
```

- Code Snippet 3

```
  String s1 = "CDAC";
  String s2 = "CDAC";
  if( s1 == s2 )
    System.out.println("Equal");
  else
    System.out.println("Not Equal");
  //Output: Equal
```

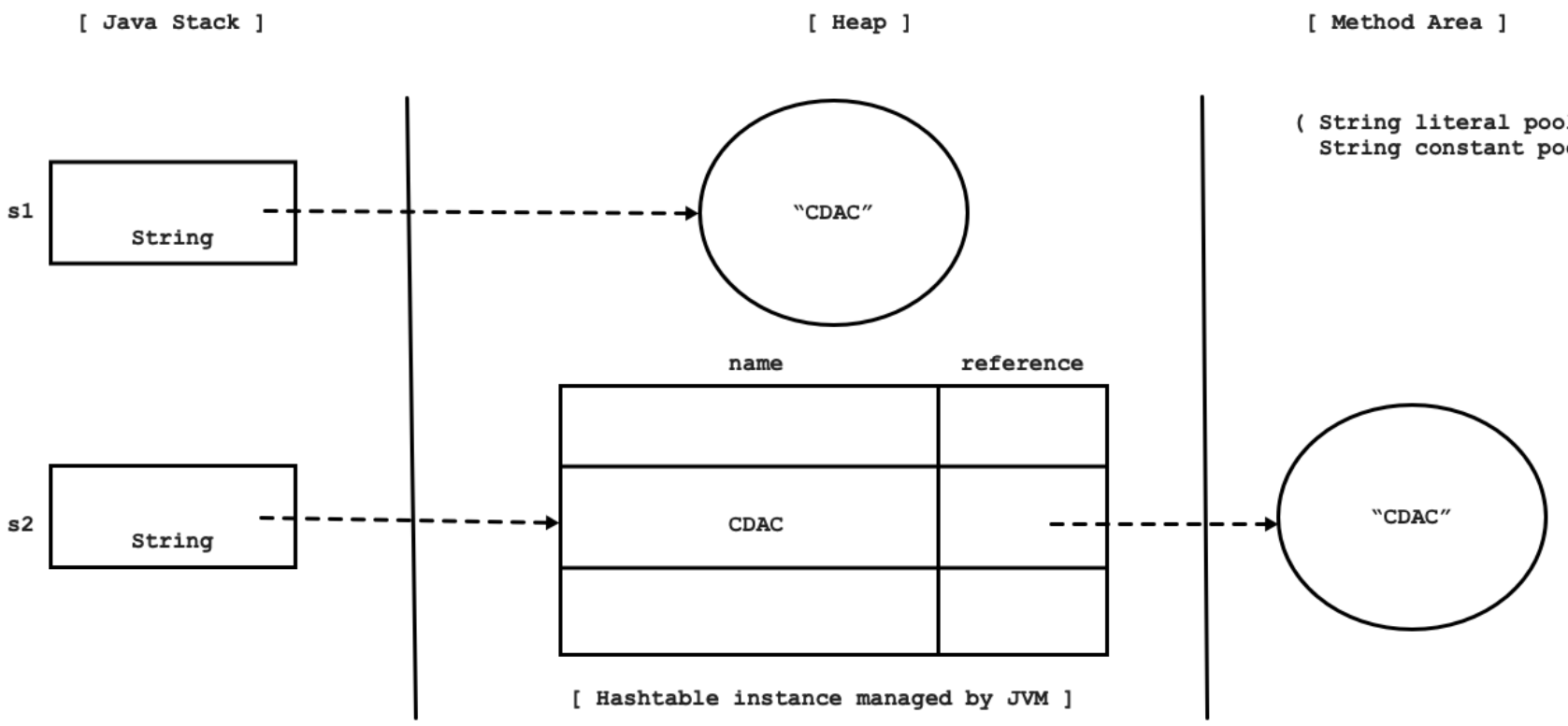


- Code Snippet 4

```
  String s1 = "CDAC";
  String s2 = "CDAC";
  if( s1.equals( s2 ) )
    System.out.println("Equal");
  else
    System.out.println("Not Equal");
  //Output: Equal
```

- Code Snippet 5

```
String s1 = new String( "CDAC" );
String s2 = "CDAC";
if( s1 == s2 )
  System.out.println("Equal");
else
  System.out.println("Not Equal");
//Output: Not Equal
```

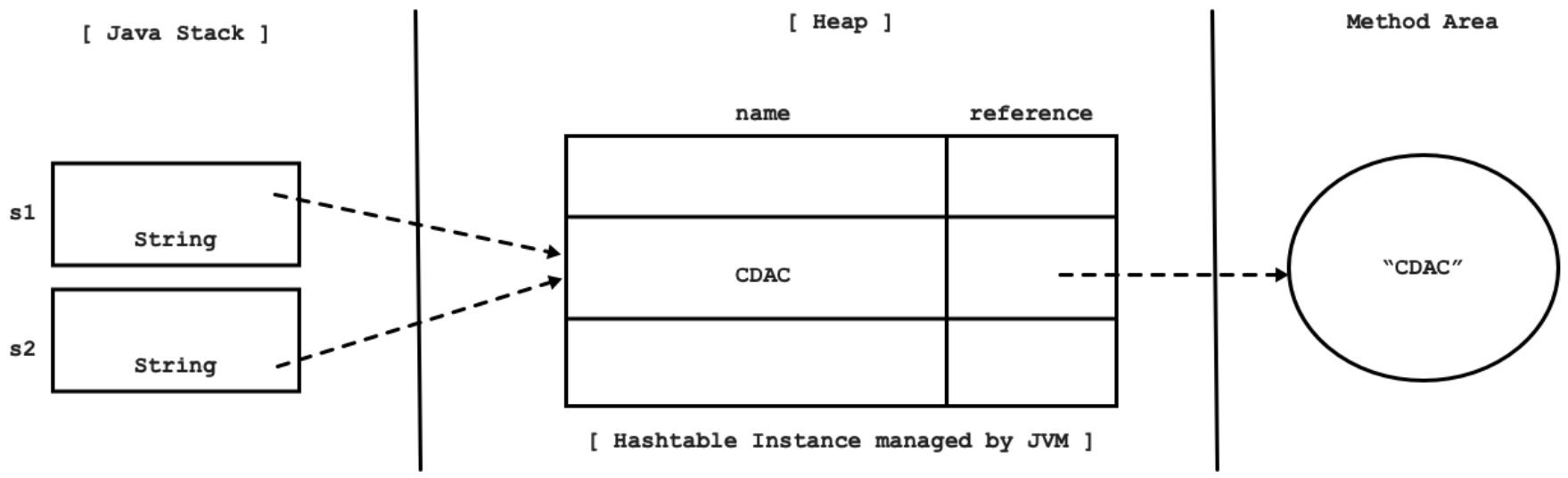


- Code Snippet 6

```
String s1 = new String( "CDAC" );
String s2 = "CDAC";
if( s1.equals( s2 ) )
  System.out.println("Equal");
else
  System.out.println("Not Equal");
//Output: Equal
```

- Code Snippet 7

```
String s1 = "CD"+"AC";
String s2 = "CDAC";
if( s1 == s2 )
  System.out.println("Equal");
else
  System.out.println("Not Equal");
//Output: Equal
```

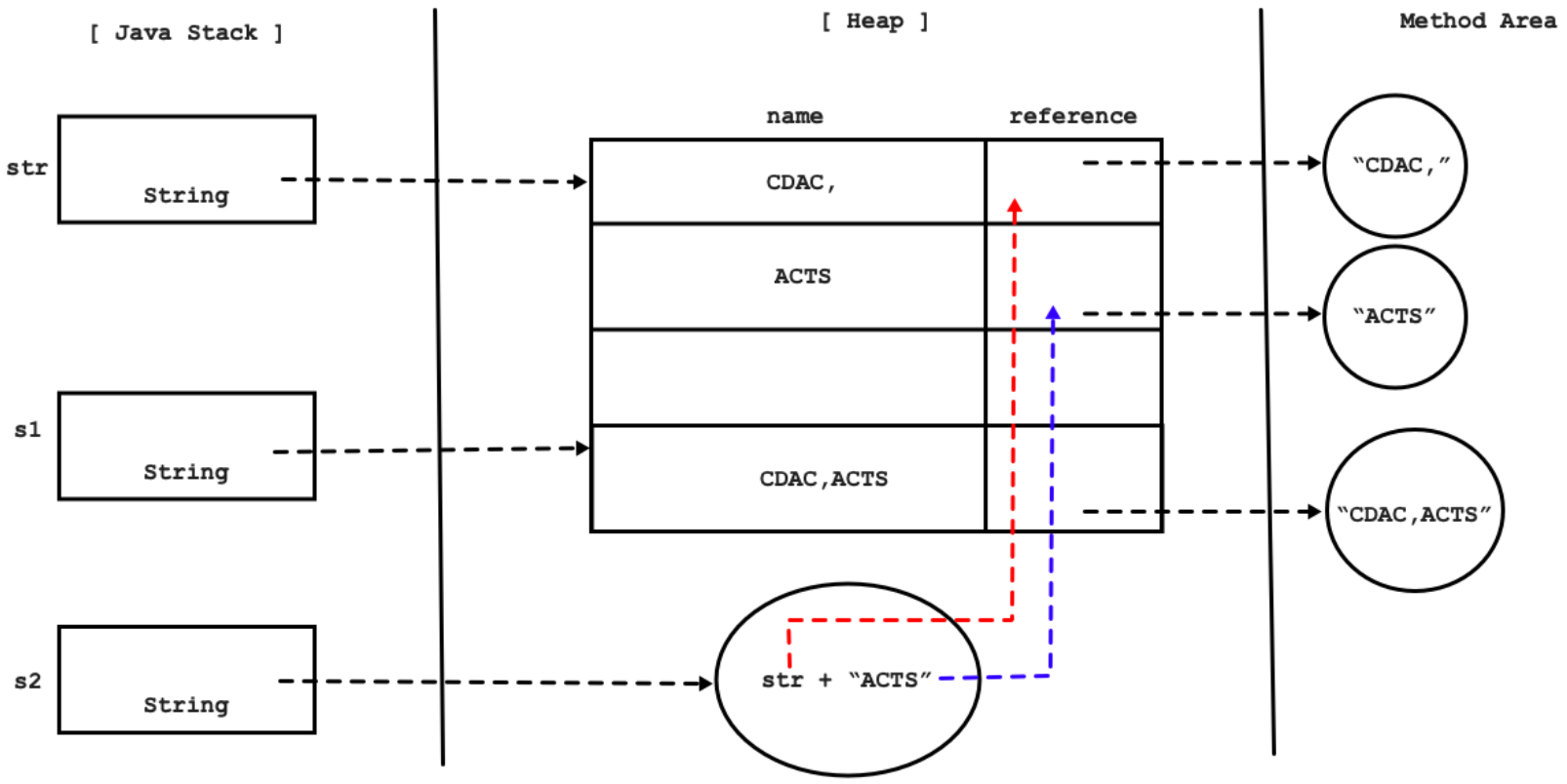


- Code Snippet 8

```
String s1 = "CD"+"AC";
String s2 = "CDAC";
if( s1.equals( s2 ) )
  System.out.println("Equal");
else
  System.out.println("Not Equal");
//Output: Equal
```

- Code Snippet 9

```
String str = "CDAC,";
String s1 = "CDAC,ACTS";
String s2 = str + "ACTS";
if( s1 == s2 )
  System.out.println("Equal");
else
  System.out.println("Not Equal");
//Output: Not Equal
```

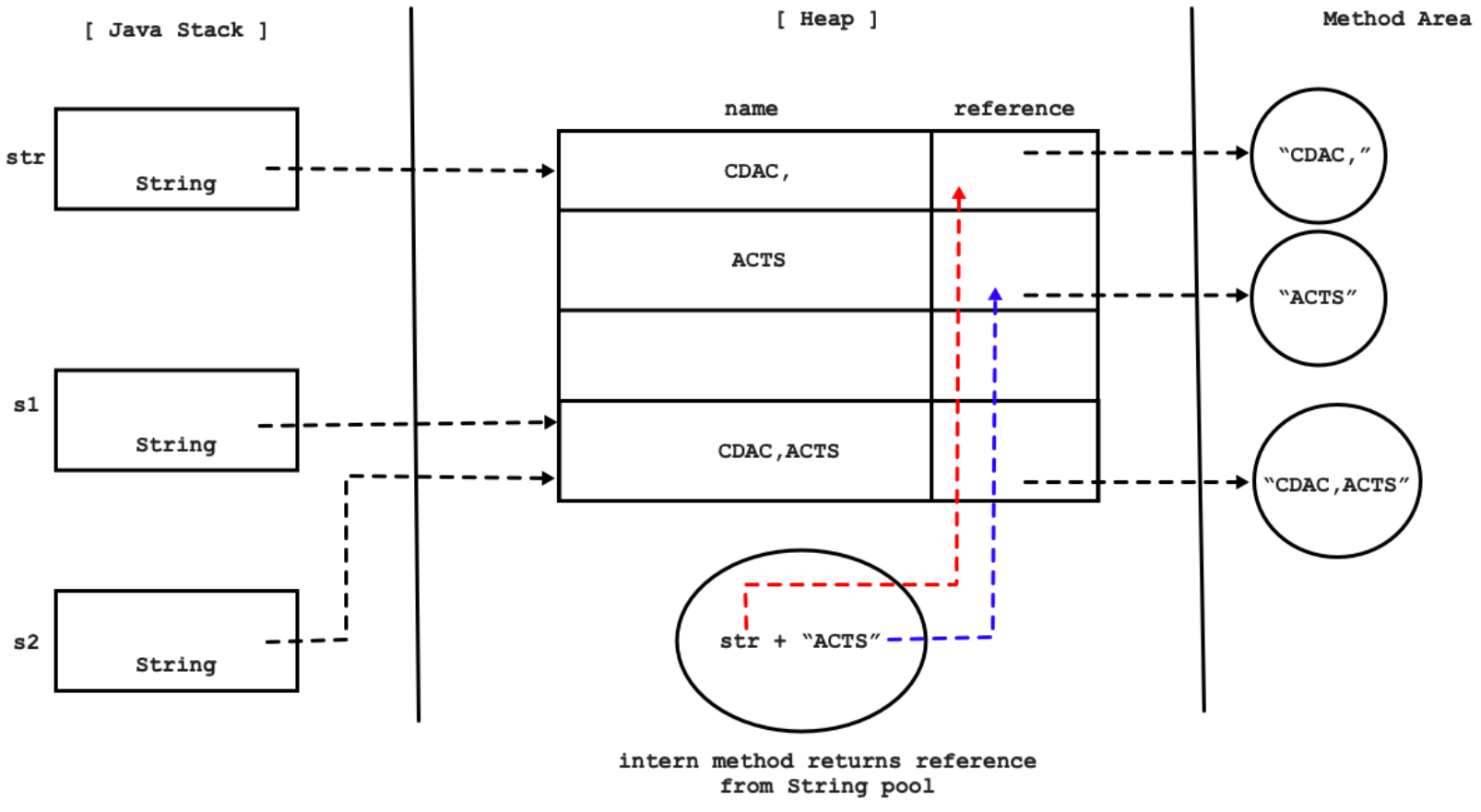


- Code Snippet 10

```
String str = "CDAC,";
String s1 = "CDAC,ACTS";
String s2 = str + "ACTS";
if( s1.equals( s2 ) )
  System.out.println("Equal");
else
  System.out.println("Not Equal");
//Output: Equal
```

- Code Snippet 11

```
String str = "CDAC,";
String s1 = "CDAC,ACTS";
String s2 = ( str + "ACTS" ).intern( );
if( s1 == s2 )
  System.out.println("Equal");
else
  System.out.println("Not Equal");
//Output: Equal
```



- Code Snippet 12

```
String str = "CDAC,";
String s1 = "CDAC,ACTS";
String s2 = ( str + "ACTS" ).intern( );
if( s1.equals( s2 ) )
  System.out.println("Equal");
else
  System.out.println("Not Equal");
//Output: Equal
```

ArrayIndexOutOfBoundsException versus StringIndexOutOfBoundsException

- ArrayIndexOutOfBoundsException indicate that an array has been accessed with an illegal index. Consider below code:

```
int[] arr = new int[ ]{ 10, 20, 30 };
int element = arr[ arr.length ];  //ArrayIndexOutOfBoundsException
```

- StringIndexOutOfBoundsException is thrown by String methods to indicate that an index is either negative or greater than the size of the string. Consider below code:

```
String str = "CDAC";
char ch = str.charAt( str.length( ) );  //StringIndexOutOfBoundsException
```

A Strategy for Defining Immutable Objects

- Don't provide setter methods.
- Make all fields final and private.
- Don't allow subclasses to override methods.
  - The simplest way to do this is to declare the class as final.
  - A more sophisticated approach is to make the constructor private and construct instances in factory methods.
- If the instance fields include references to mutable objects, don't allow those objects to be changed:
  - Don't provide methods that modify the mutable objects.
  - Don't share references to the mutable objects.
- Reference: https://docs.oracle.com/javase/tutorial/essential/concurrency/imstrat.html