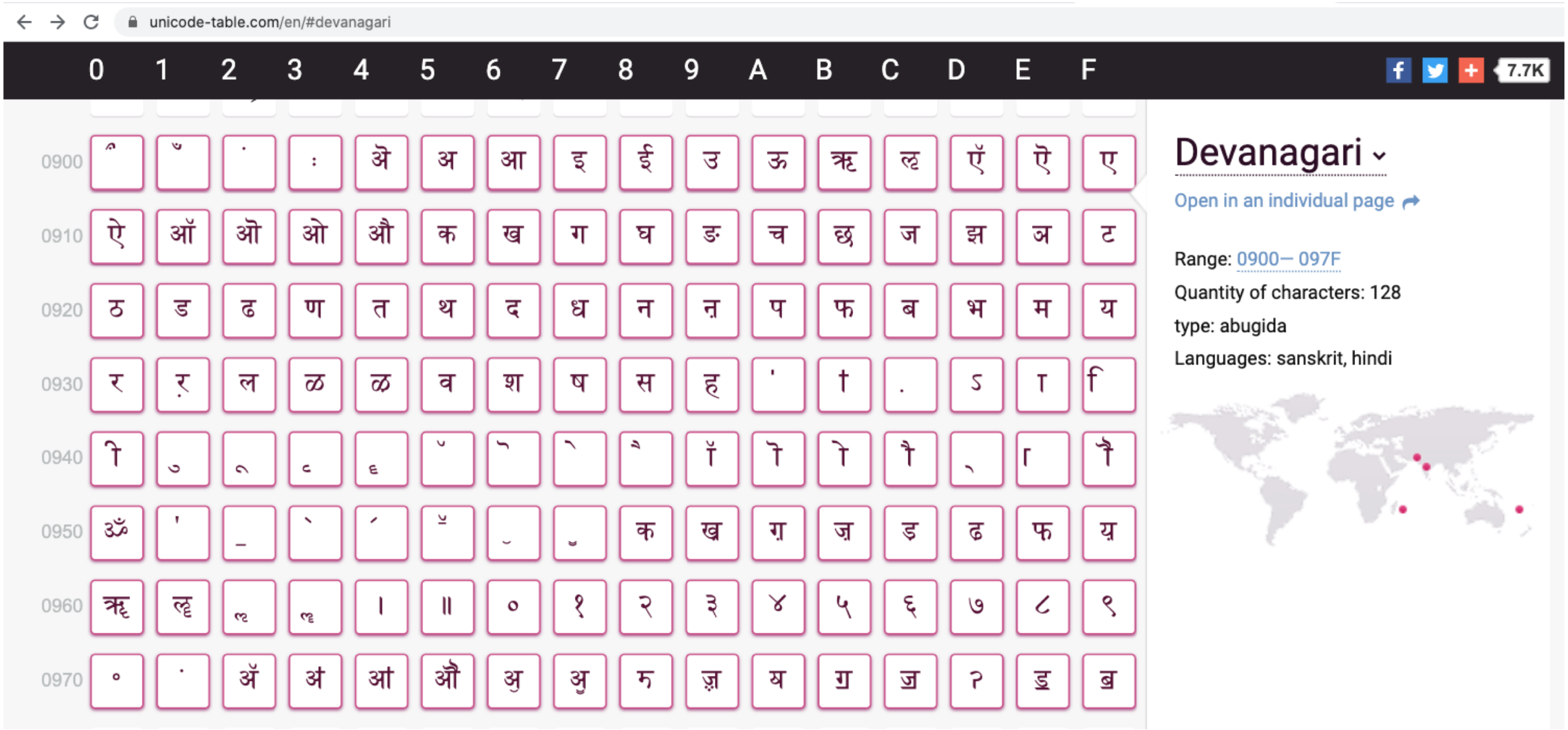
Object Oriented Programming with Java

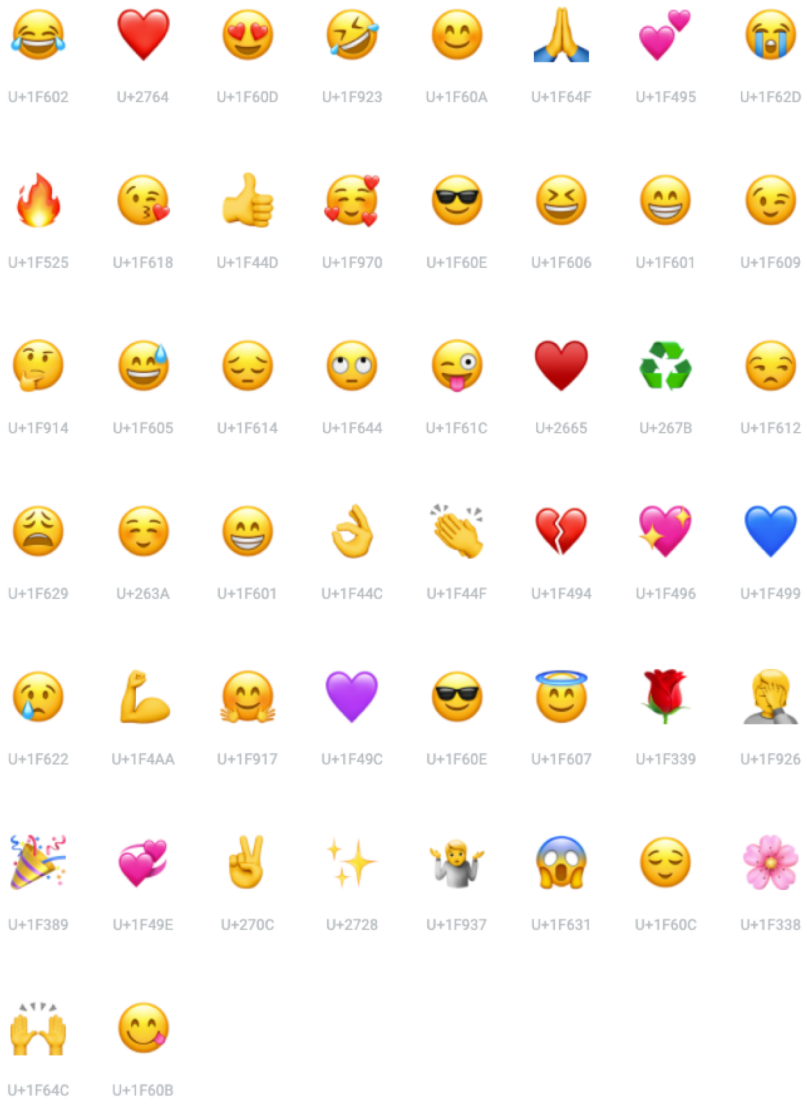
Sandeep Kulange

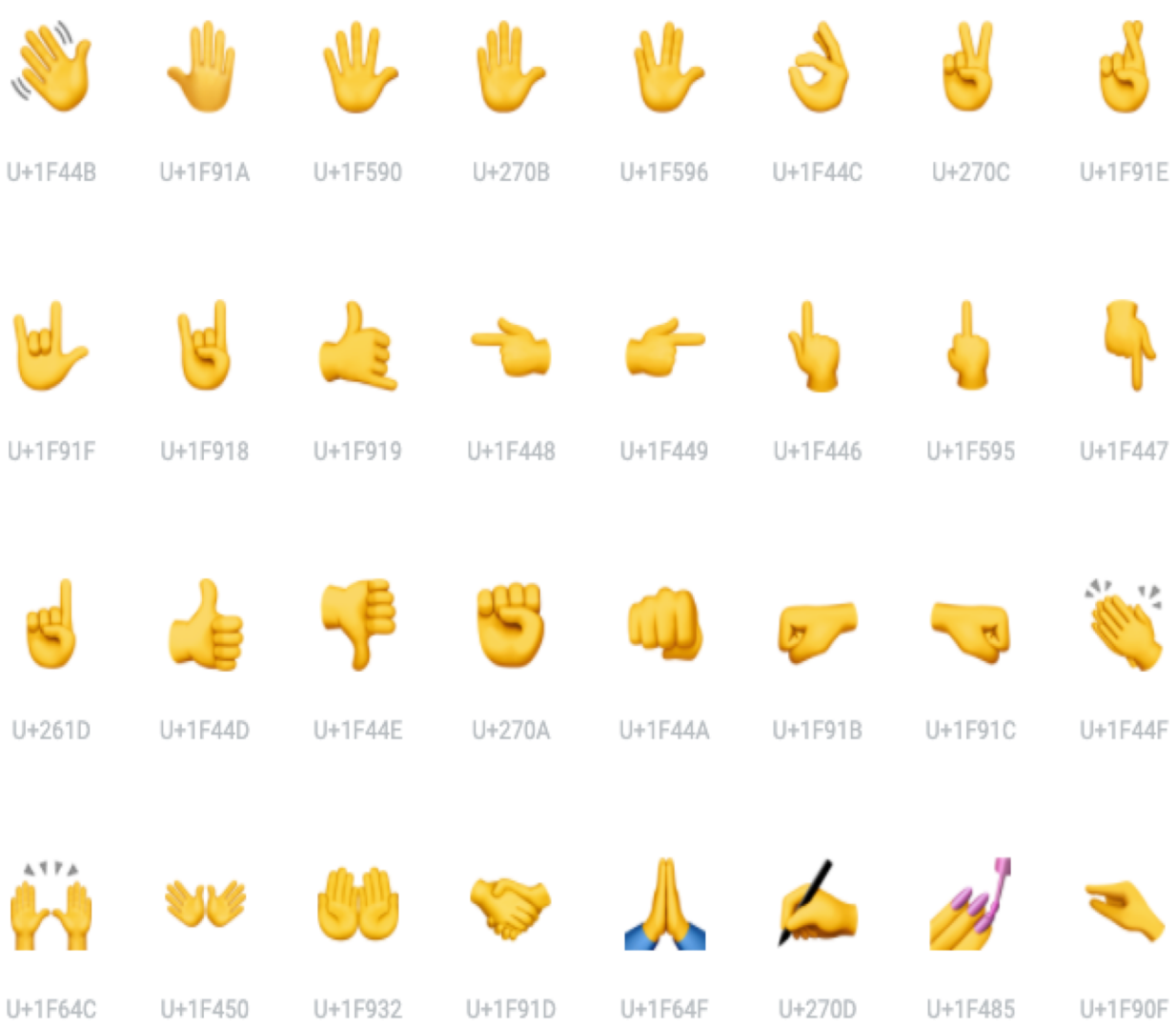
Unicode Character Table

Unicode Character Table

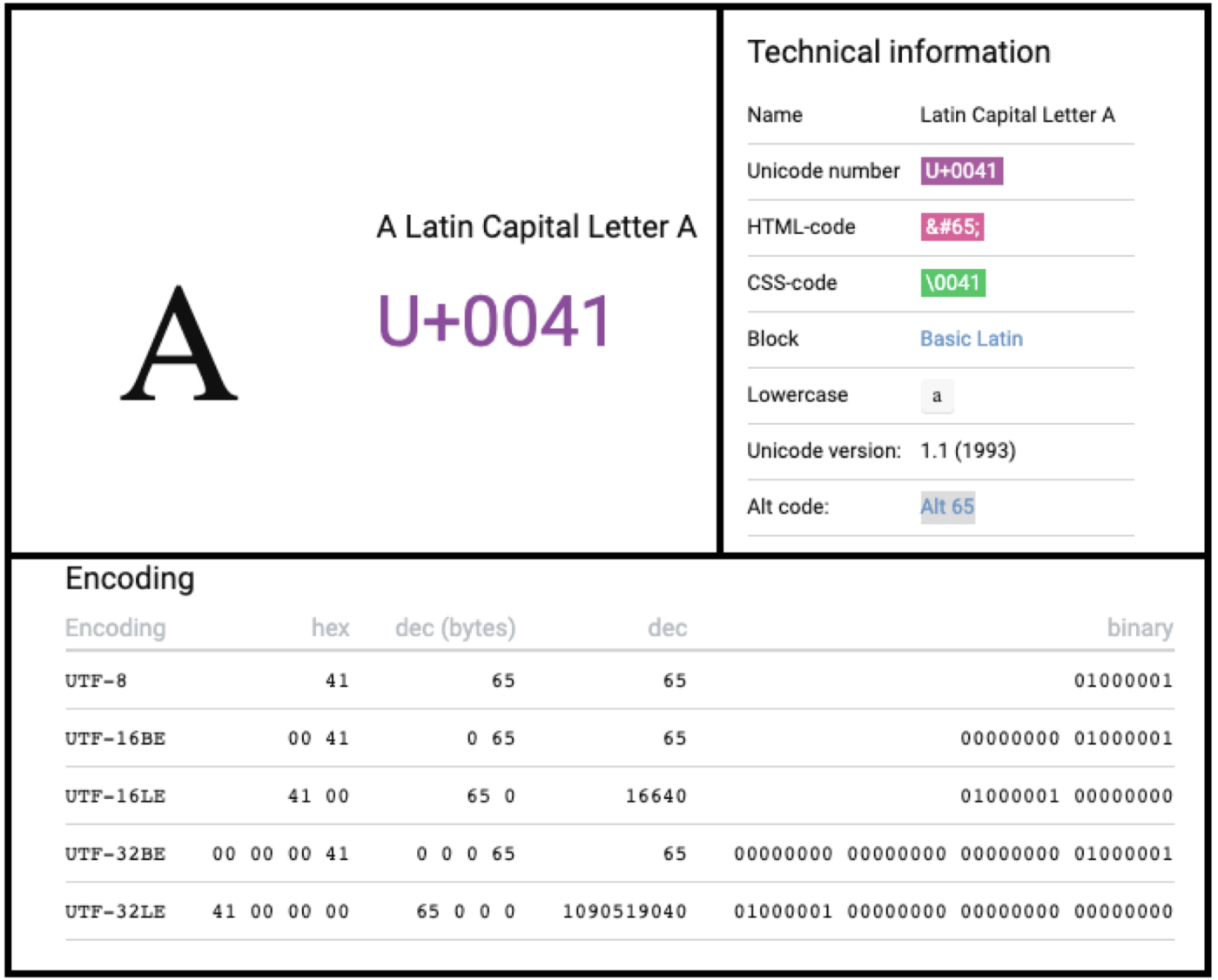


Unicode Character Table





Character Encoding



java.lang.Character

- It is a final class declared in java.lang package.
- The Character class wraps a value of the primitive type char in an object.
- This class provides a large number of static methods for determining a character's category (lowercase letter, digit, etc.) and for converting characters from uppercase to lowercase and vice versa.
- The fields and methods of class Character are defined in terms of character information from the Unicode Standard.
- The char data type are based on the original Unicode specification, which defined characters as fixed-width 16-bit entities.

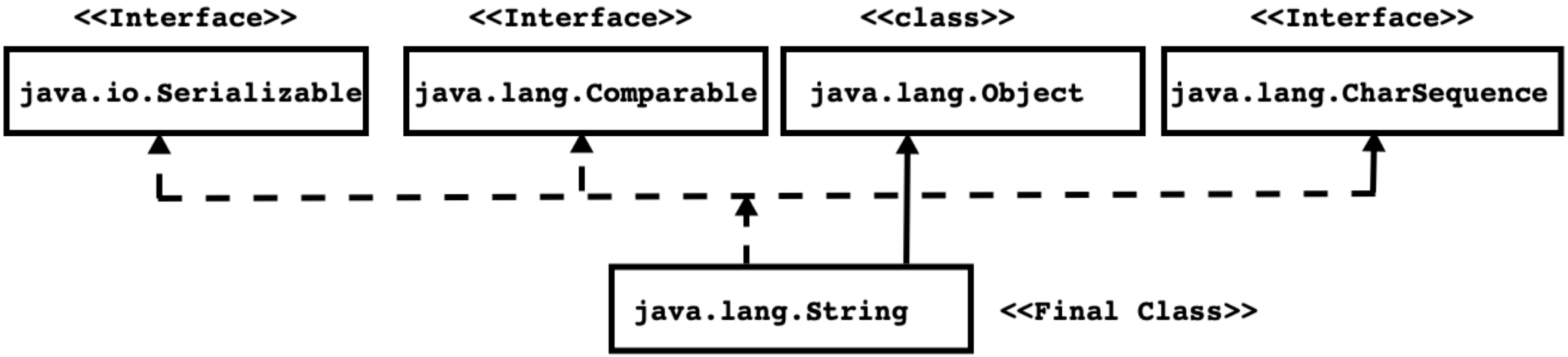
java.lang.Character

- The range of legal *code points* is now U+0000 to U+10FFFF, known as *Unicode scalar value*.
- The set of characters from U+0000 to U+FFFF is sometimes referred to as the *Basic Multilingual Plane (BMP)*.
- Characters whose code points are greater than U+FFFF are called *supplementary characters*.
- The Java platform uses the UTF-16 representation in char arrays and in the String and StringBuffer classes.
- A char value, therefore, represents Basic Multilingual Plane (BMP) code point
- https://medium.com/jspoint/introduction-to-character-encoding-3b9735f265a6

String Introduction

- Strings, which are widely used in Java programming, are a sequence of characters.
- In the Java programming language, strings are objects.
- We can use following classes to manipulate string:
    1. java.lang.String
    2. java.lang.StringBuffer
    3. java.lang.StringBuilder
    4. java.util.StringTokenizer
    5. java.util.regex.Pattern
    6. java.util.regex.Matcher

String Class Hierarchy



String Introduction

- Serializable is a Marker interface declared in java.io package.
- Comparable is Functional interface declared in java.lang package.
  1. int compareTo(T other)
- CharSequence is interface declared in java.lang package.
  1. char charAt(int index)
  2. int length()
  3. CharSequence subSequence(int start, int end)
  4. default IntStream chars()
  5. default IntStream codePoints( )
- Object is non final, concrete class declared in java.lang package.
  1. It is having 11 methods(5 Non final + 6 final)
- String is a final class declared in java.lang package.

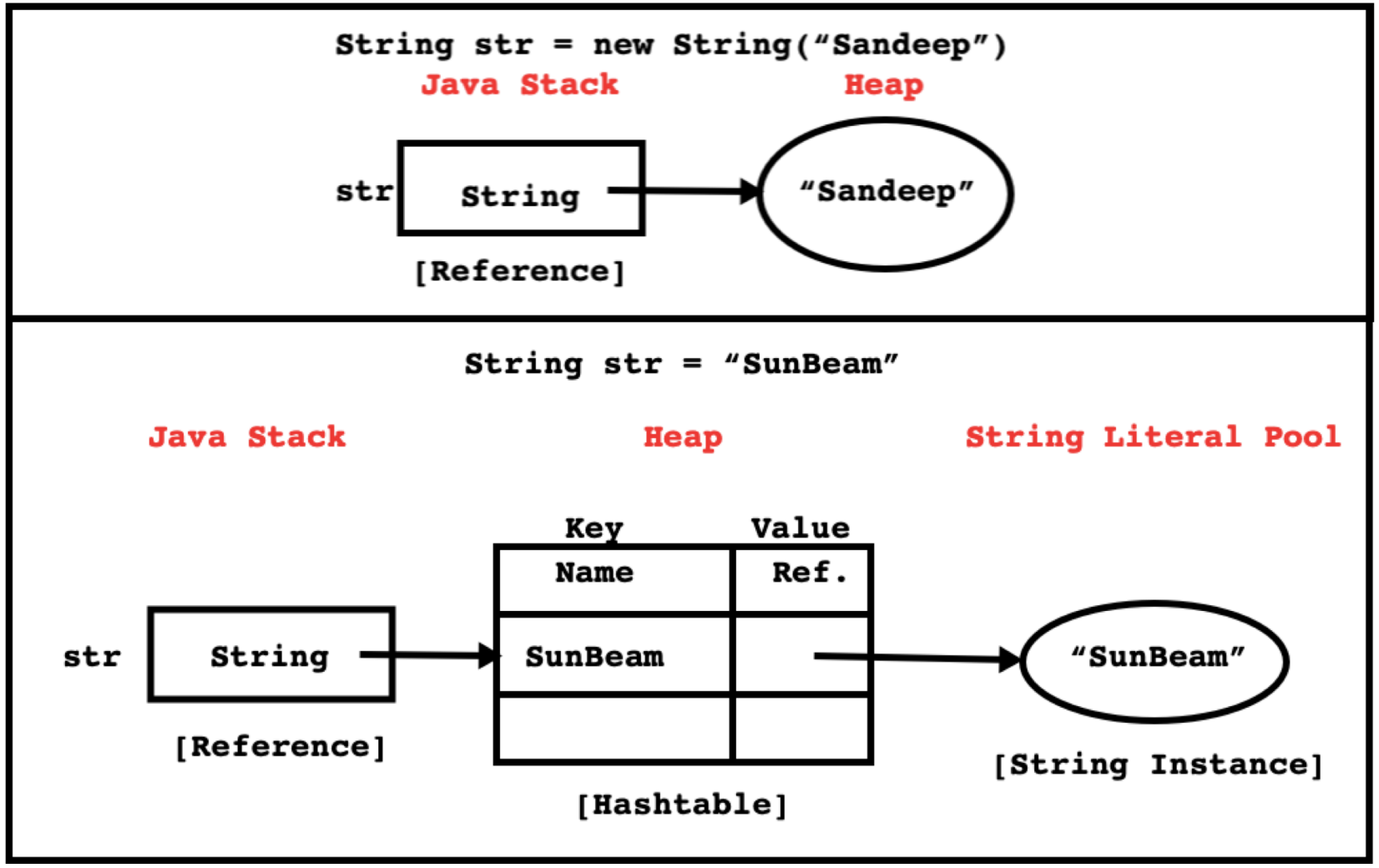
String Introduction

- String is not a built-in or primitive type. It is a class, hence considered as non primitive/reference type.

- We can create instance of String with and W/O new operator.
  - String str = new String("Sandeep"); //String Instance
  - String str = "SunBeam"; //String Literal

- String str = "SunBeam", is equivalent to:
- ```
  char[] data = new char[ ]{'S','u','n','B','e','a','m'};
  ```
- ```
  String str = new String( data );
  ```

String Memory Representation



String concatenation

- If we want to concatenate Strings then we should use concat() method:
  - "public String concat(String str)"

- Consider following Example:

```
String s1 = "SunBeam";
String s2 = "Pune/Karad";
String s3 = s1.concat( s2 );
```

- The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings.

```
String s1 = "SunBeam";
String s2 = s1 +" Pune/Karad";
```

-

```
int pinCode = 411057;
```

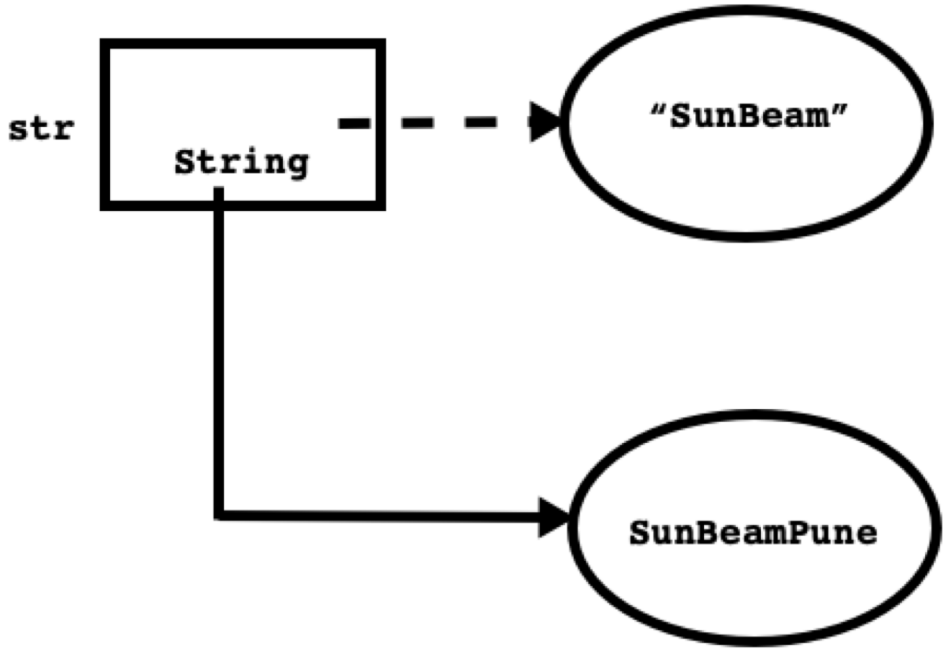
```
String str = "Pune,"+pinCode;
```

Immutable Strings

- Strings are constant; their values cannot be changed after they are created.
- Because String objects are immutable they can be shared.

```
String str = "SunBeam";

str = str + "Pune";
```



A Strategy for Defining Immutable Objects

1. Don't provide "setter" methods — methods that modify fields or objects referred to by fields.
2. Make all fields final and private.
3. 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.
4. 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. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

String Class Constructors

1. public String()

2. public String(byte[] bytes)

3. public String(char[] value)

4. public String(String original)

5. public String(StringBuffer buffer)

6. public String(StringBuilder builder)

String Class Methods

1. public char charAt(int index)
2. public int compareTo(String anotherString)
3. public String concat(String str)
4. public boolean equalsIgnoreCase(String anotherString)
5. public boolean startsWith(String prefix)
6. public boolean endsWith(String suffix)
7. public static String format(String format, Object... args)
8. public byte[] getBytes()
9. public int indexOf(int ch)
10. public int indexOf(String str)
11. public String intern( )
12. public boolean isEmpty()
13. public int length()
14. public boolean matches(String regex)

String Class Methods

15.public String[] split(String regex)

16.public String substring(int beginIndex)

17.public String substring(int beginIndex, int endIndex)

18.public char[] toCharArray()

19.public String toLowerCase()

20.public String toUpperCase()

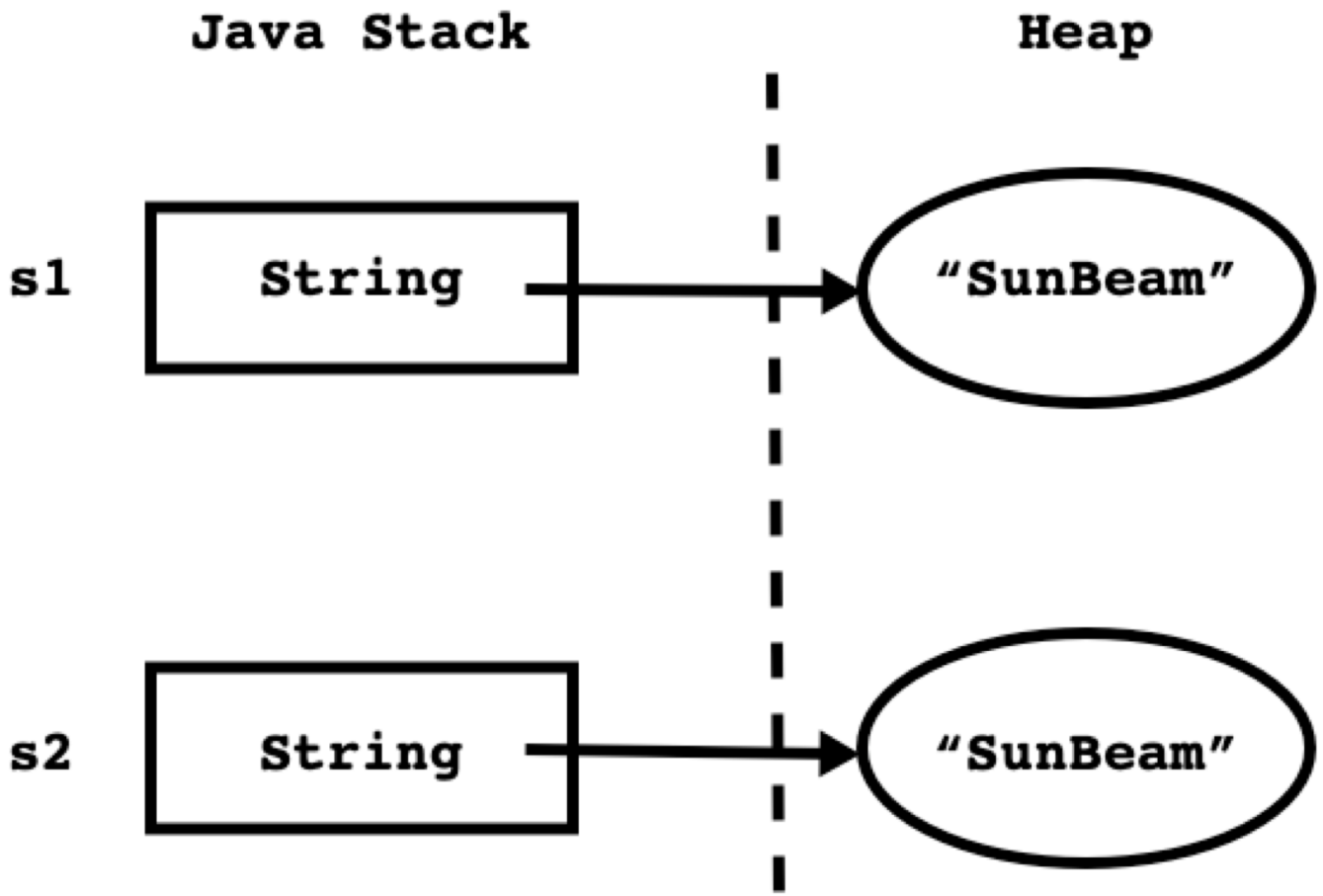
21.public String trim()

22.public static String valueOf(Object obj)

String Twister

```
public class Program {
   public static void main(String[] args) {
      String s1 = new String("SunBeam");
      String s2 = new String("SunBeam");
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not Equal
   }
}
```

String Twister



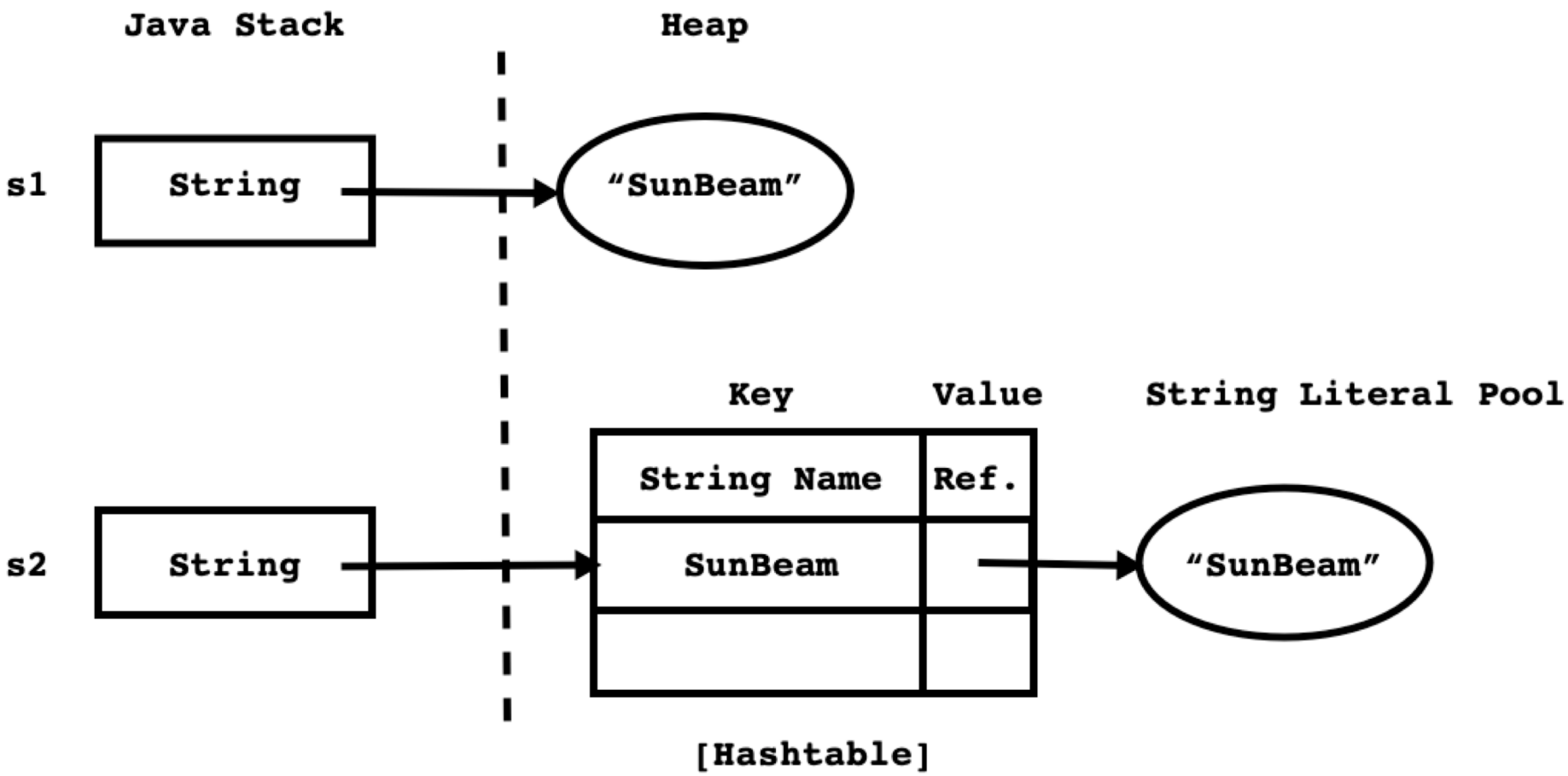
String Twister

```
public class Program {
   public static void main(String[] args) {
      String s1 = new String("SunBeam");
      String s2 = new String("SunBeam");
      if( s1.equals(s2) )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Equal
   }
}
```

String Twister

```
public class Program {
   public static void main(String[] args) {
      String s1 = new String("SunBeam");
      String s2 = "SunBeam";
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not Equal
   }
}
```

String Twister



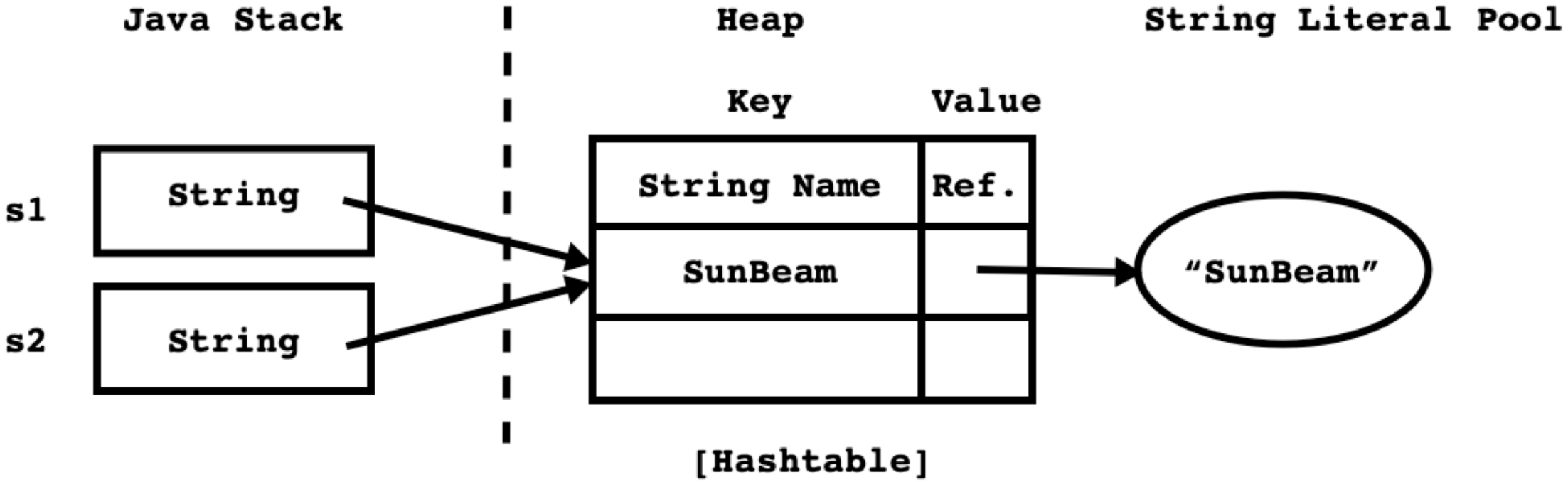
String Twister

```
public class Program {
   public static void main(String[] args) {
      String s1 = new String("SunBeam");
      String s2 = "SunBeam";
      if( s1.equals(s2) )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Equal
   }
}
```

String Twister

```
public class Program {
   public static void main(String[] args) {
      String s1 = "SunBeam";
      String s2 = "SunBeam";
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Equal
   }
}
```

String Twister



String Twister

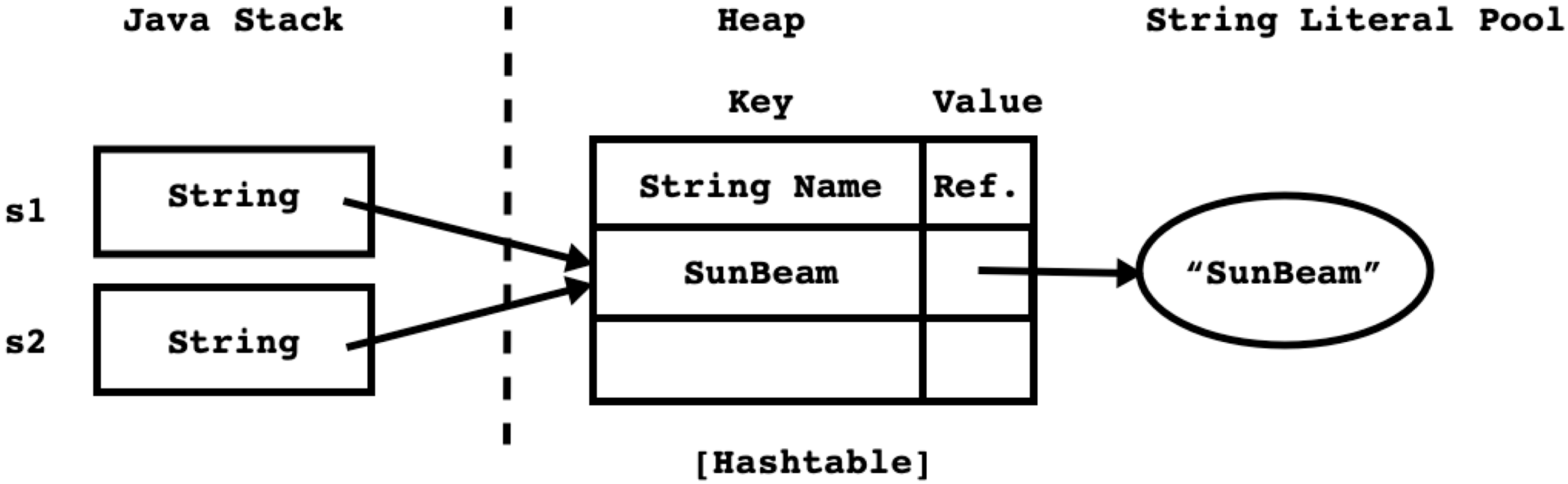
```
public class Program {
   public static void main(String[] args) {
      String s1 = "SunBeam";
      String s2 = "SunBeam";
      if( s1.equals(s2) )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Equal
   }
}
```

String Twister

```
public class Program {
   public static void main(String[] args) {
      String s1 = "SunBeam";
      String s2 = "Sun"+"Beam";
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Equal
   }
}
```

String Twister

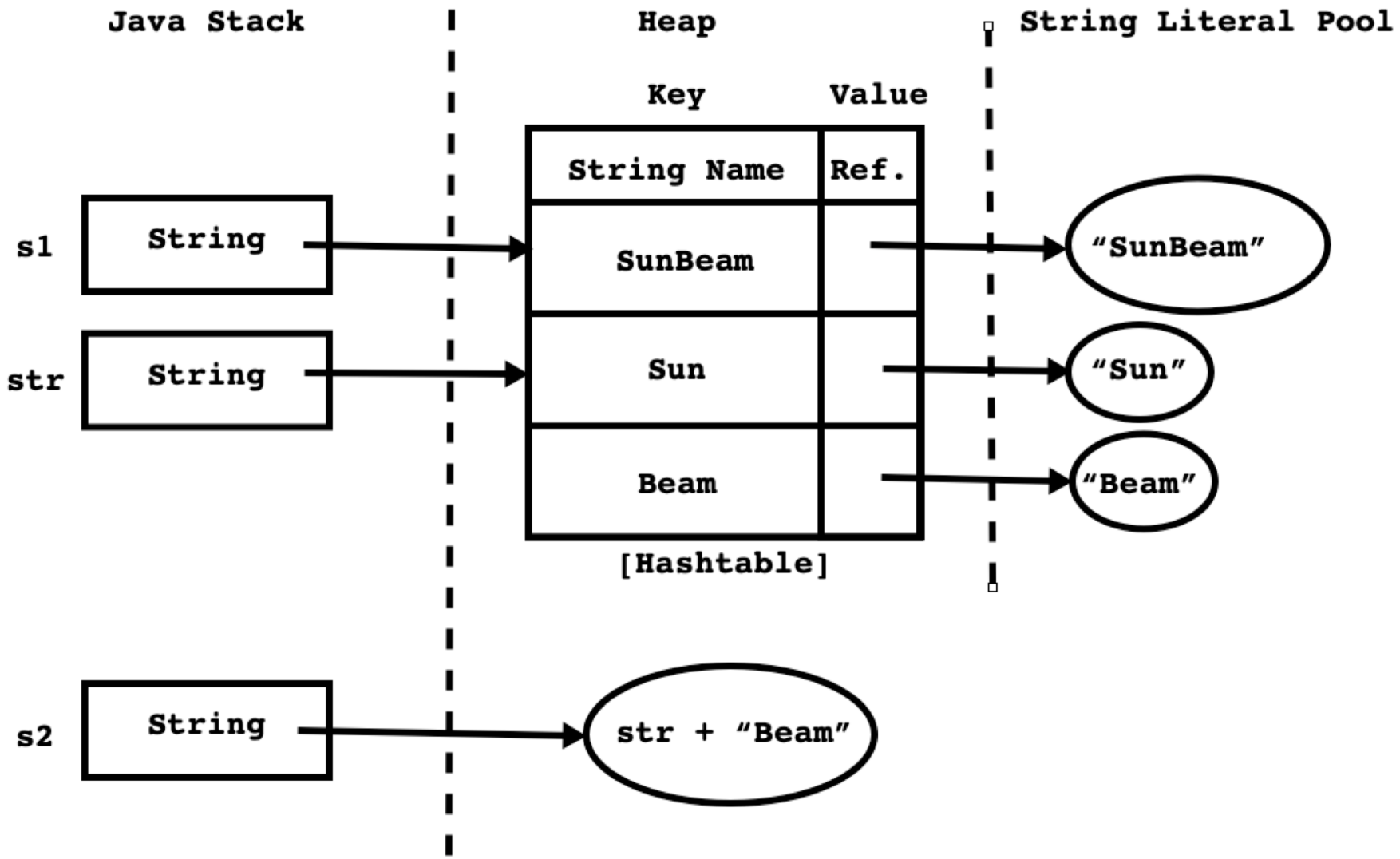
- Constant expression gets evaluated at compile time.
  - "int result = 2 + 3;" becomes "int result = 5;" at compile time
  - "String s2 = "Sun"+"Beam";" becomes "String s2="SunBeam";" at compile time.



String Twister

```
public class Program {
   public static void main(String[] args) {
      String s1 = "SunBeam";
      String str = "Sun";
      String s2 = str + "Beam";
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not Equal
   }
}
```

String Twister



String Twister

```
public class Program {
   public static void main(String[] args) {
      String s1 = "SunBeam";
      String str = "Sun";
      String s2 = ( str + "Beam" ).intern();
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Equal
   }
}
```

String Twister

```java
package p1;
public class A {
    public static final String str = "Hello";
}
```

```java
package test;
class B {
    public static final String str = "Hello";
}
```

```java
package test;
public class Program {
    public static final String str = "Hello";
    public static void main(String[] args) {
        String str = "Hello";
    }
}
```

String Twister

```
public class Program {
   public static final String str = "Hello";
   public static void main(String[] args) {
      String str = "Hello";
      System.out.println(A.str == B.str);          //true
      System.out.println(A.str == Program.str);  //true
      System.out.println(A.str == str);          //true
      System.out.println(B.str == Program.str);  //true
      System.out.println(B.str == str);          //true
      System.out.println(Program.str == str);    //true
   }
}
```

String Twister

```
public class Program {
   public static void main(String[] args) {
      String str = "SunBeam";
      //char ch = str.charAt( 0 );   //S
      //char ch = str.charAt( 6 );   //m
      //char ch = str.charAt(-1); //StringIndexOutOfBoundsException
      char ch = str.charAt( str.length() ); //StringIndexOutOfBoundsException
      System.out.println(ch);
   }
}
```

StringBuffer versus StringBuilder

- StringBuffer and StringBuilder are final classes.
- It is declared in java.lang package.
- It is used create to mutable string instance.
- equals() and hashCode() method is not overridden inside it.
- We can create instances of these classes using new operator only.
- Instances get space on Heap.
- **StringBuffer implementation is thread safe whereas StringBuilder is not.**
- **StringBuffer is introduced in JDK1.0and StringBuilder is introduced in JDK 1.5.**

StringBuffer Twister

```
public class Program {
   public static void main(String[] args) {
      StringBuffer s1 = new StringBuffer("SunBeam");
      StringBuffer s2 = new StringBuffer("SunBeam");
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not Equal
   }
}
```

StringBuffer Twister

```
public class Program {
   public static void main(String[] args) {
      StringBuffer s1 = new StringBuffer("SunBeam");
      StringBuffer s2 = new StringBuffer("SunBeam");
      if( s1.equals(s2) )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not Equal
   }
}
```

StringBuffer Twister

```
public class Program {
   public static void main(String[] args) {
      String s1 = new String("SunBeam");
      StringBuffer s2 = new StringBuffer("SunBeam");
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Compiler Error
   }
}
```

StringBuffer Twister

```
public class Program {
   public static void main(String[] args) {
      String s1 = new String("SunBeam");
      StringBuffer s2 = new StringBuffer("SunBeam");
      if( s1.equals(s2) )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not Equal
   }
}
```

StringBuffer Twister

```
public class Program {
   public static void main(String[] args) {
      StringBuffer s1 = new StringBuffer("SunBeam");
      String s2 = new String("SunBeam");
      if( s1.equals(s2) )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not Equal
   }
}
```

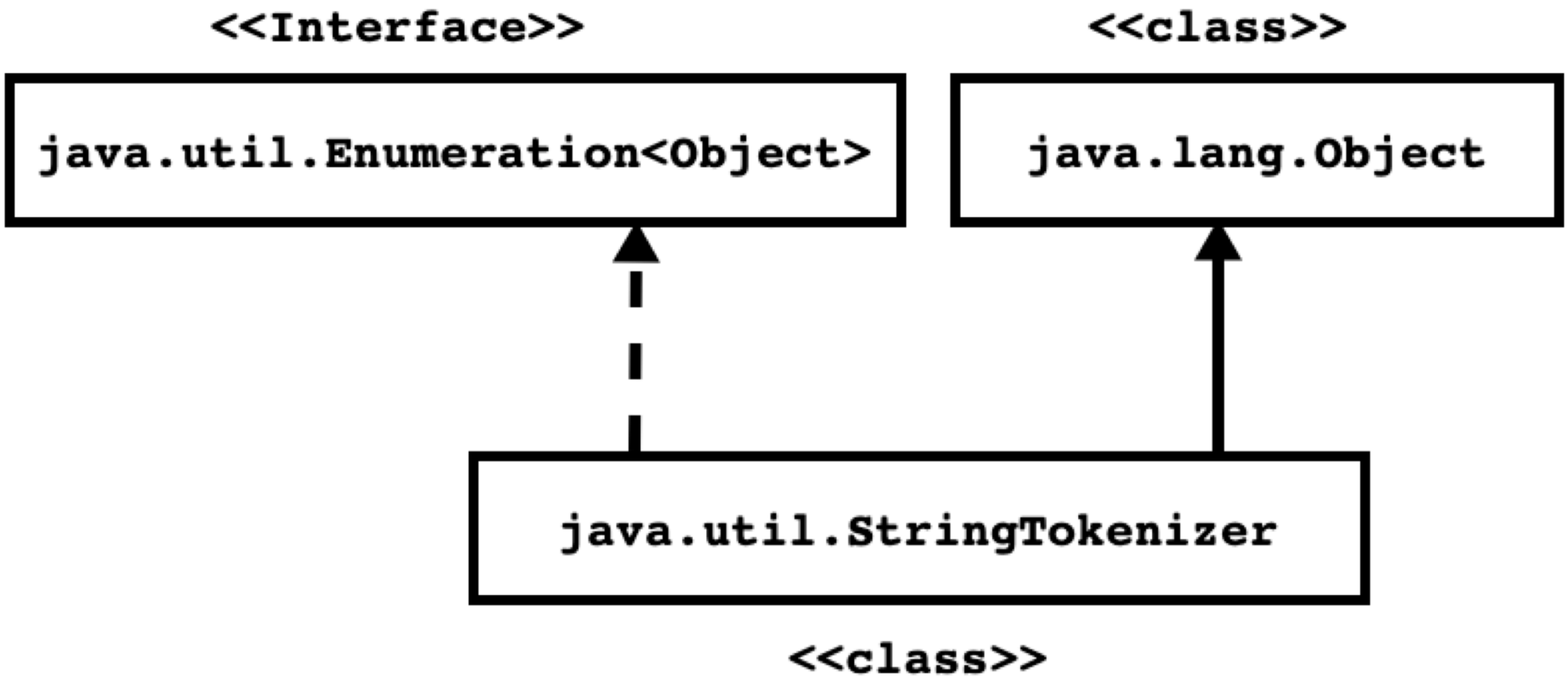
StringBuilder Twister

```
public class Program {
   public static void main(String[] args) {
      StringBuilder s1 = new StringBuilder("SunBeam");
      StringBuilder s2 = new StringBuilder("SunBeam");
      if( s1 ==  s2)
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not Equal
   }
}
```

StringBuilder Twister

```
public class Program {
   public static void main(String[] args) {
      StringBuilder s1 = new StringBuilder("SunBeam");
      StringBuilder s2 = new StringBuilder("SunBeam");
      if( s1.equals(s2))
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not Equal
   }
}
```

StringTokenizer



StringTokenizer

- The string tokenizer class allows an application to break a string into tokens.
- Methods of java.util.Enumeration<E> interface
  1. boolean hasMoreElements( )
  2. E nextElement()
- Methods of java.util.StringTokenizer class
  1. public int countTokens()
  2. public boolean hasMoreTokens()
  3. public String nextToken()
  4. public String nextToken(String delim)

StringTokenizer

```
public class Program {
   public static void main(String[] args) {
      String str = "SunBeam Infotech Pune";
      StringTokenizer stk = new StringTokenizer(str);
      String token = null;
      while( stk.hasMoreTokens()) {
         token = stk.nextToken();
         System.out.println(token);
      }
   }
}
```

Output

```
SunBeam
Infotech
Pune
```

StringTokenizer

```
public class Program {
   public static void main(String[] args) {
      String str = "www.sunbeaminfo.com";
      String delim = ".";
      StringTokenizer stk = new StringTokenizer(str, delim);
      String token = null;
      while( stk.hasMoreTokens()) {
         token = stk.nextToken();
         System.out.println(token);
      }
   }
```

Output

```
www
sunbeaminfo
com
```

StringTokenizer

```
public class Program {
    public static void main(String[] args) {
        String str = "https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions";
        String delim = "/.://-";
        StringTokenizer stk = new StringTokenizer(str, delim);
        String token = null;
        while( stk.hasMoreTokens()) {
            token = stk.nextToken();
            System.out.println(token);
        }
    }
}
```

```
Output
------
https
developer
mozilla
org
en
US
docs
Web
JavaScript
Guide
Regular_Expressions
```

Pattern and Matcher

- Java.util.regex.Pattern and Matcher Classes are used for matching character sequences against patterns specified by regular expressions.
- An instance of the Pattern class represents a regular expression that is specified in string form in a syntax similar to that used by Perl.
- Instances of the Matcher class are used to match character sequences against a given pattern.

```
Pattern p = Pattern.compile( regex );
Matcher m = p.matcher( input );
boolean b = m.matches();

//or

boolean b = Pattern.matches(regex, input);
```

Thank You.

[sandeepkulange@sunbeaminfo.com]