OOP using Java

Trainer: Mr. Rohan Paramane

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 without new operator.
  - String str = “Sunbeam”
  - String str = new String(“Rohan”);
- Two ways to create a String in Java which are String Literal and String Object.
- The main difference between String Literal and String Object is –
  - String Literal is String created using double quotes
  - String Object is a String created using the new() operator.
- 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
  - java.lang.String
  - java.lang.StringBuffer
  - java.lang.StringBuilder
  - java.util.StringTokenizer

Literal String

- **String s1 = “Hello World”;**
- Here, the s1 is referring to “Hello World” in the String pool.
- If there is another statement as follows.
- **String s2 = “Hello World”;**
- As “Hello World” already exists in the String pool, the s2 reference variable will also point to the already existing “Hello World” in the String pool. In other words, now both s1 and s2 refer to the same “Hello World” in the String pool.
- Therefore, if the programmer writes a statement as follows, it will display true.
- **System.out.println(s1==s2);**
- String Pool in Java is a special storage space in Java heap memory. It is also known as String Constant Pool or String Intern Pool.
- Whenever a new string is created, JVM first checks the string pool. If it encounters the same string, then instead of creating a new string, it returns the same instance of the found string to the variable.
- The String.intern() method puts the string in the String pool or refers to another String object from the string pool having the same value.

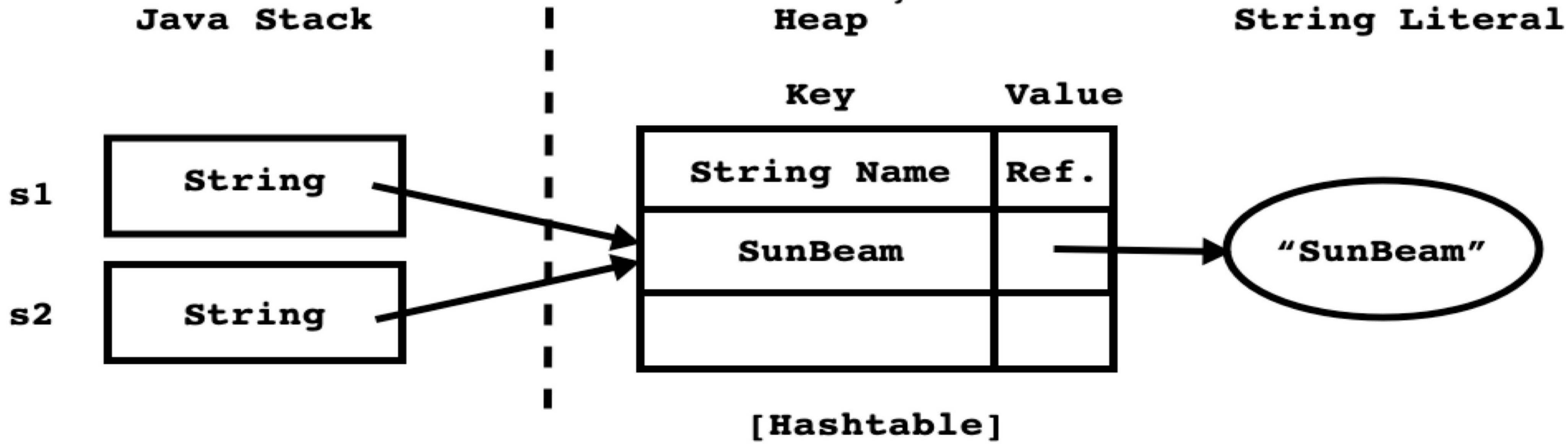
Non Literal String

- **String s1 = new (“Hello World”);**
- **String s2 = new (“Hello World”);**
- Unlike with String literals, in this case, there are two separate objects.
- In other words, s1 refers to one “Hello World” while s2 refers to another “Hello World”.
- Here, the s1 and s2 are reference variables that refer to separate String objects.
- Therefore, if the programmer writes a statement as follows, it will display false.
- **System.out.println(s1==s2);**

String Examples

```java
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
   }
}
```

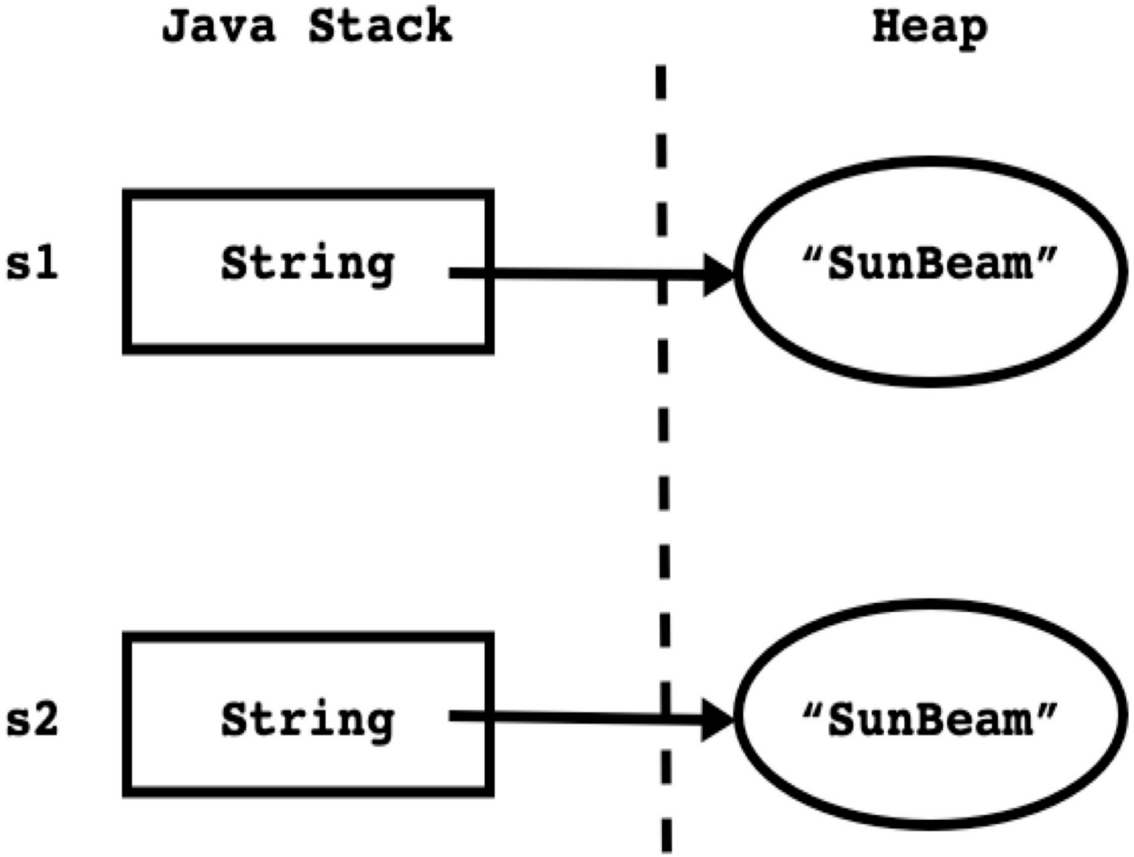
```java
public class Program {
   public static void main(String[] args) {
      String s1 = "SunBeam";
      String s2 = "SunBeam";
      if( s1.equals(s2) )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Equal
   }
}
```



String Examples

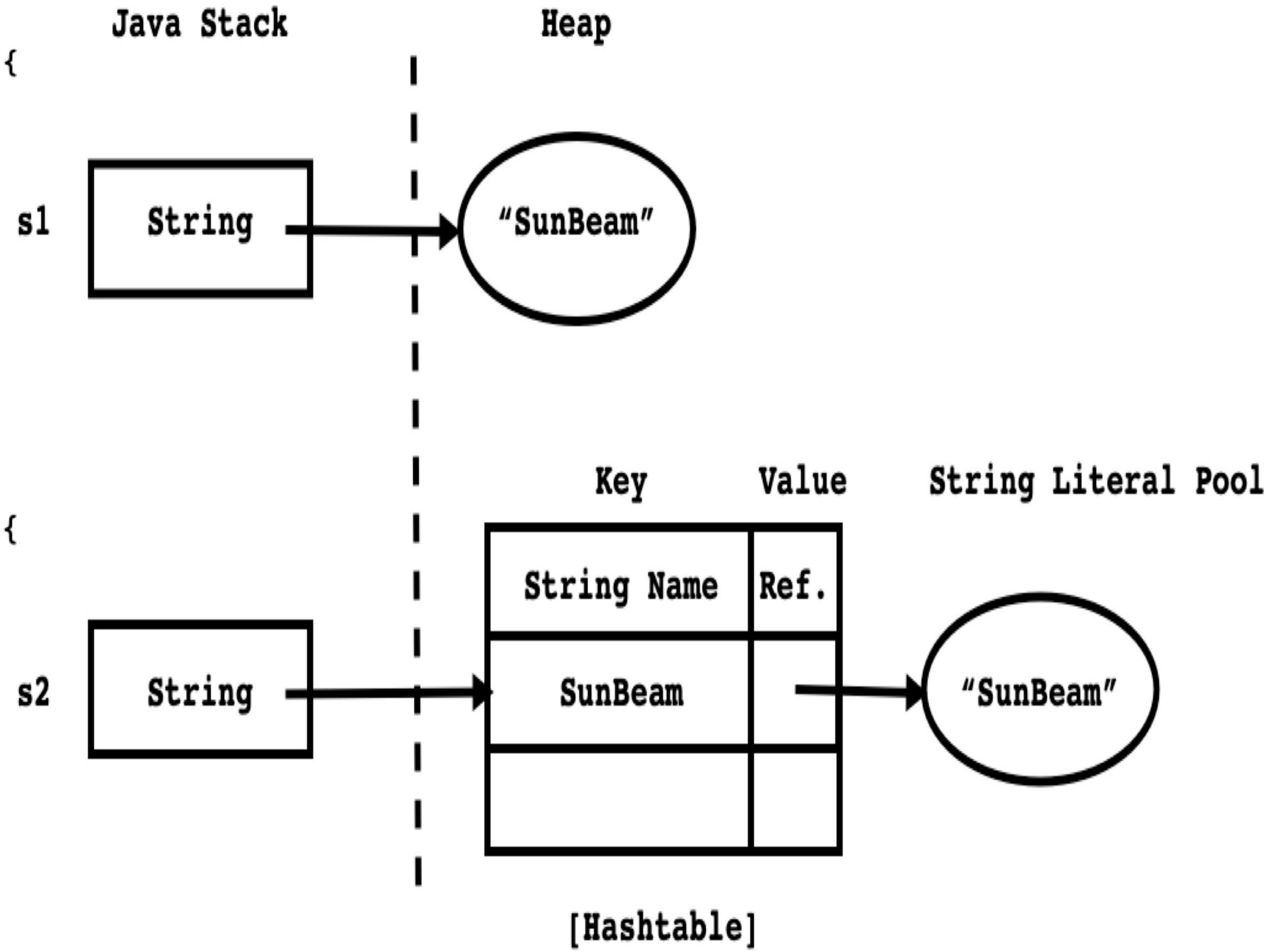
```java
public class Program {
   public static void main(String[] args) {
      String s1 = new String("SunBeam");
      String s2 = new String("SunBeam");
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not Equal
   }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

```java
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
   }
}
```



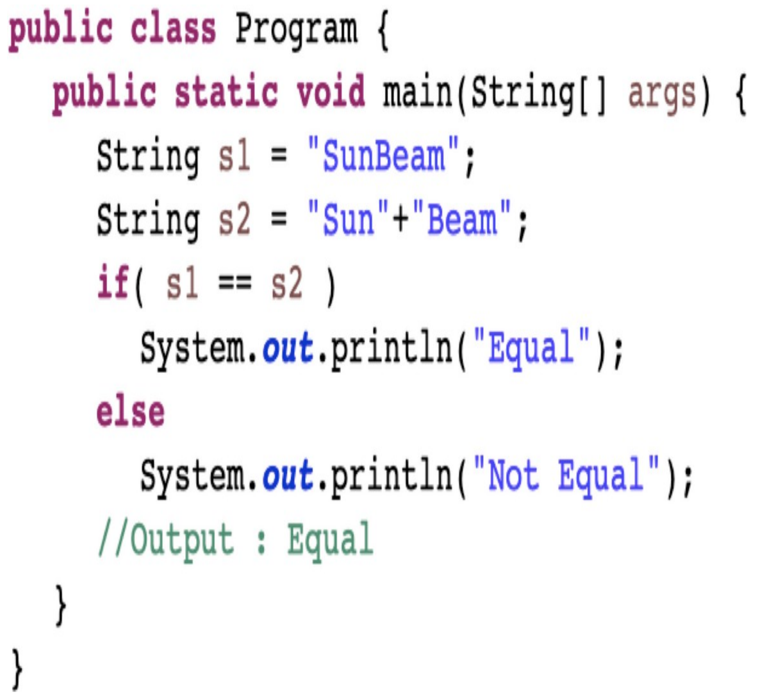
String Examples

```java
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
   }
}
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
   }
}
```

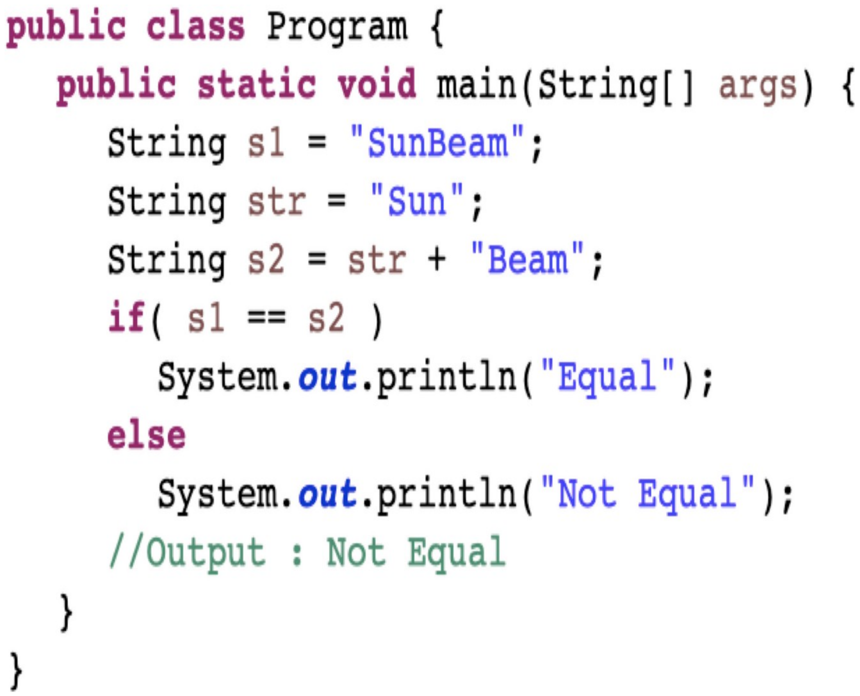


String Examples

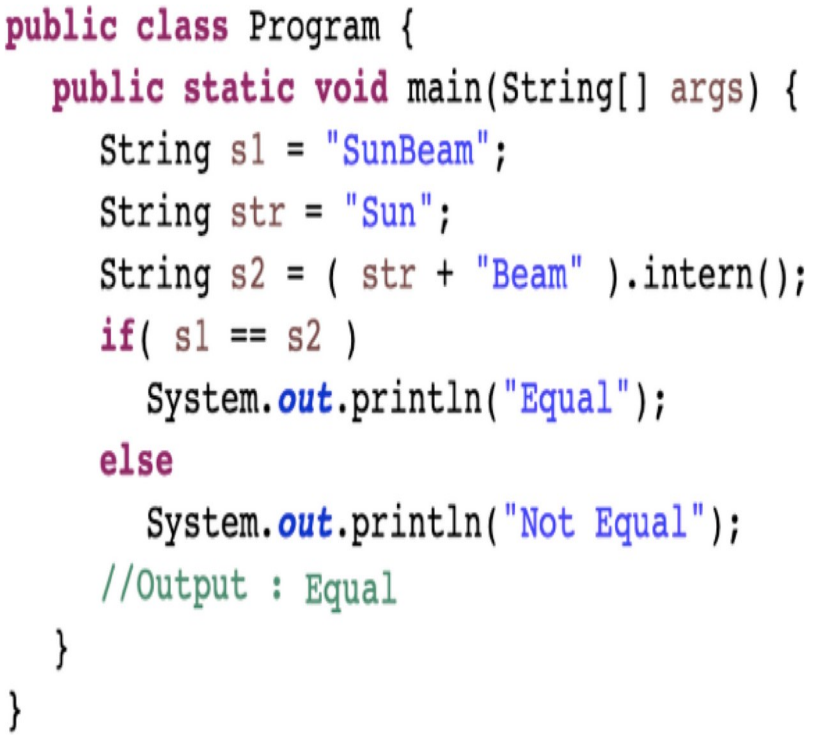
- Constant expression gets evaluated at compile time.
- “int result = 2 + 3;” becomes “int result = 5;” at compile time
- ”String s2 = “Sun”+”Beam”;” becomes “String s2=“SunBeam”;” at compile time.



```java
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
  }
}
```

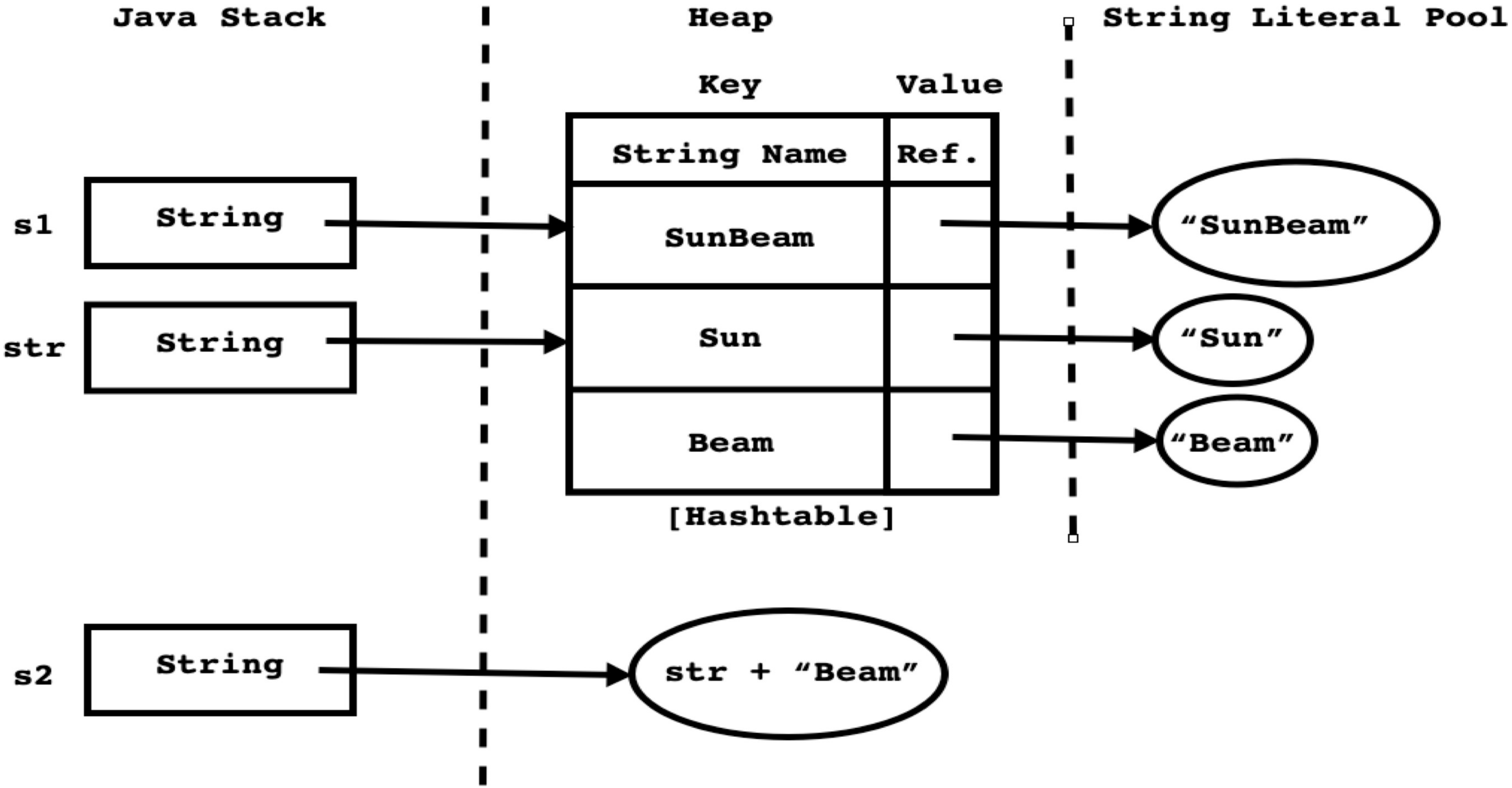


```java
public class Program {
  public static void main(String[] args) {
    String s1 = "SunBeam";
    String str = "Sun";
    String s2 = str + "Beam";
    if( s1 == s2 )
      System.out.println("Equal");
    else
      System.out.println("Not Equal");
    //Output : Not Equal
  }
}
```

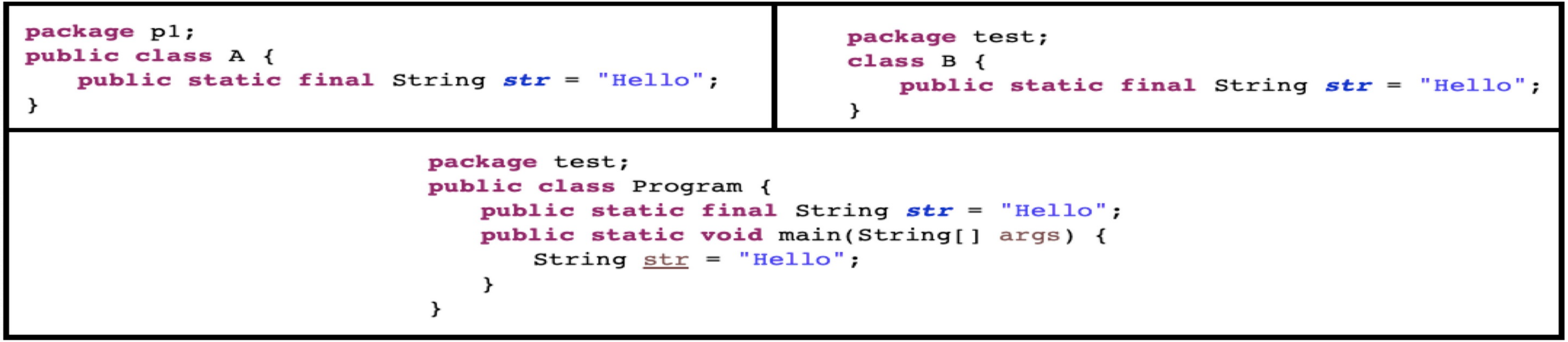


```java
public class Program {
  public static void main(String[] args) {
    String s1 = "SunBeam";
    String str = "Sun";
    String s2 = ( str + "Beam" ).intern();
    if( s1 == s2 )
      System.out.println("Equal");
    else
      System.out.println("Not Equal");
    //Output : Equal
  }
}
```

String Examples



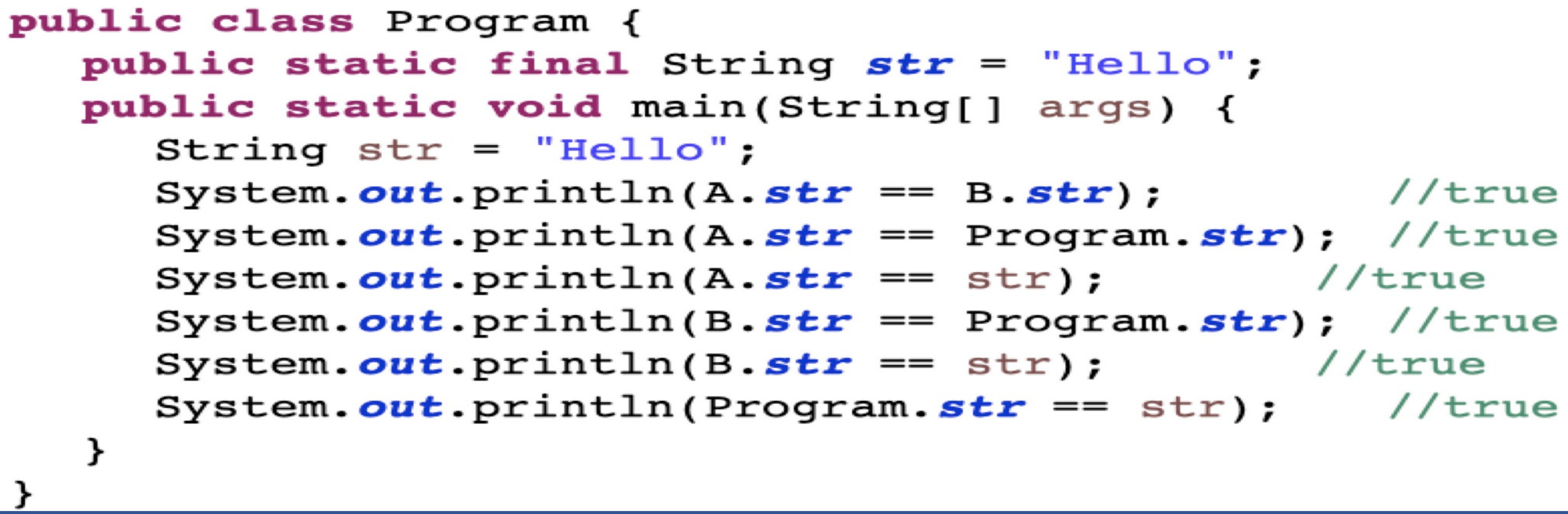
String Examples



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

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

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

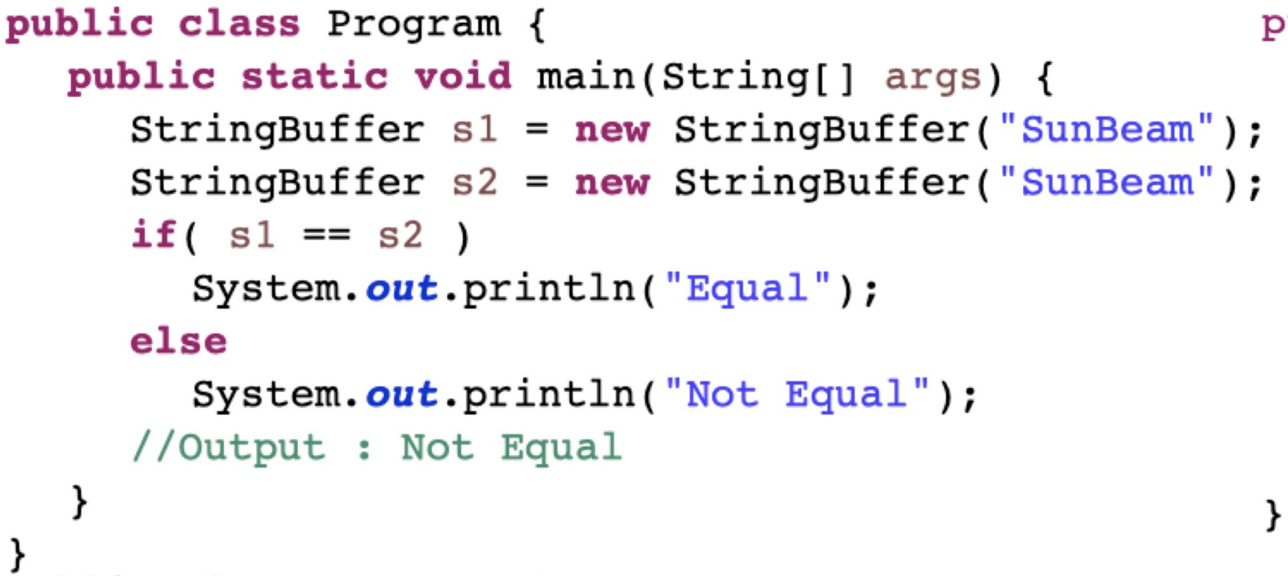


```
public class Program {
    public static final String str = "Hello";
    public static void main(String[] args) {
        String str = "Hello";
        System.out.println(A.str == B.str);          //true
        System.out.println(A.str == Program.str);   //true
        System.out.println(A.str ==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

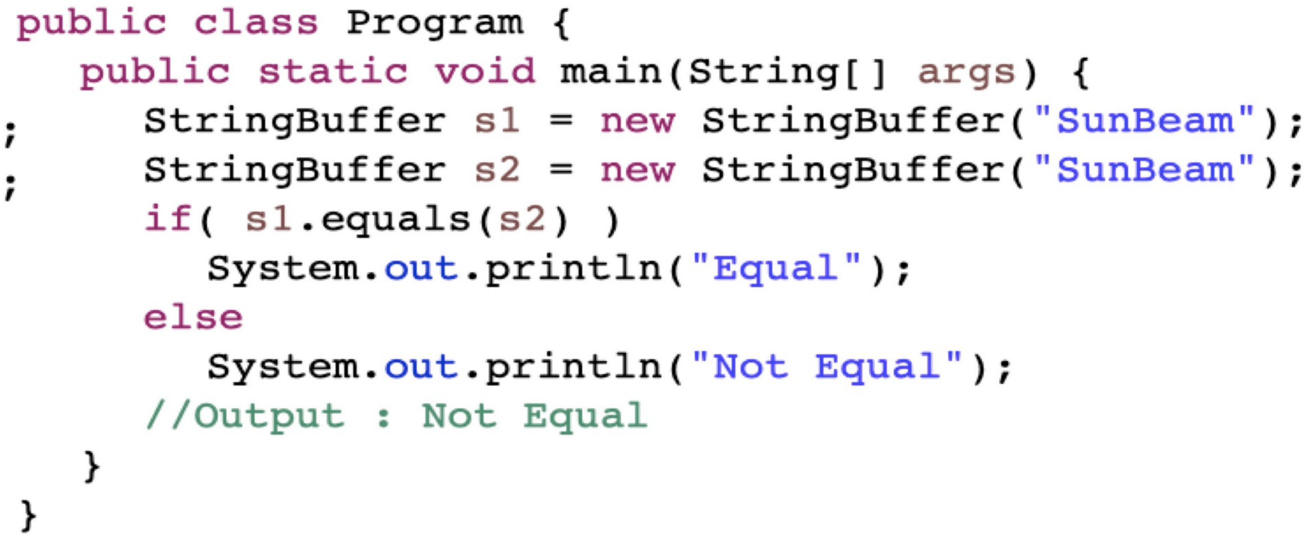
StringBuffer versus StringBuilder

- 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.

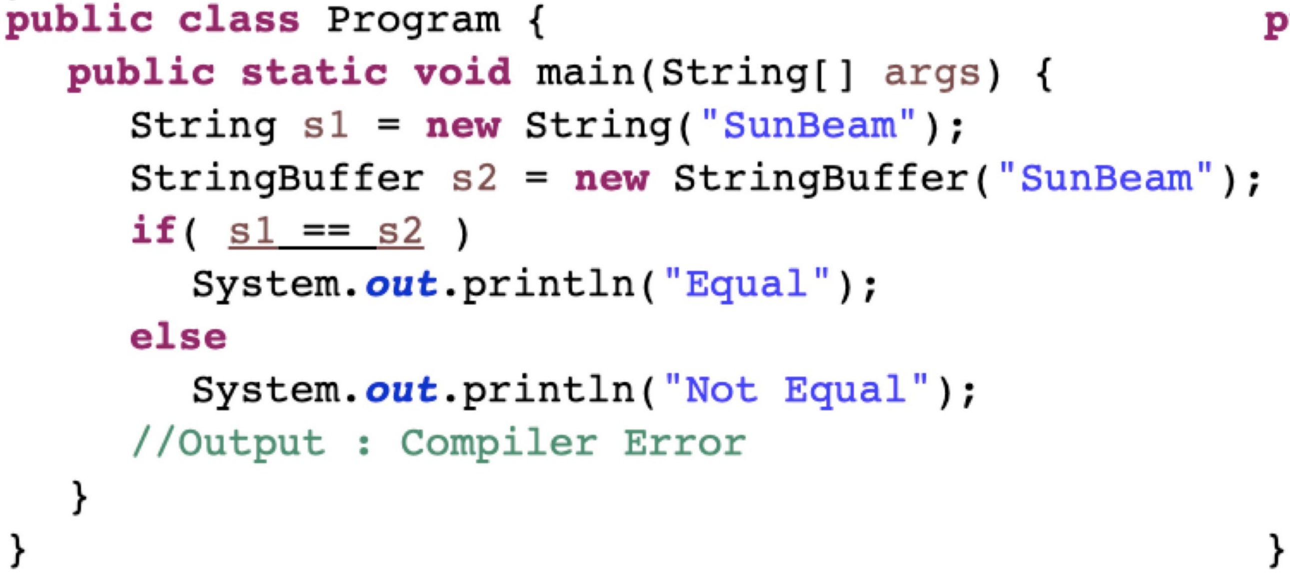
StringBuffer Example



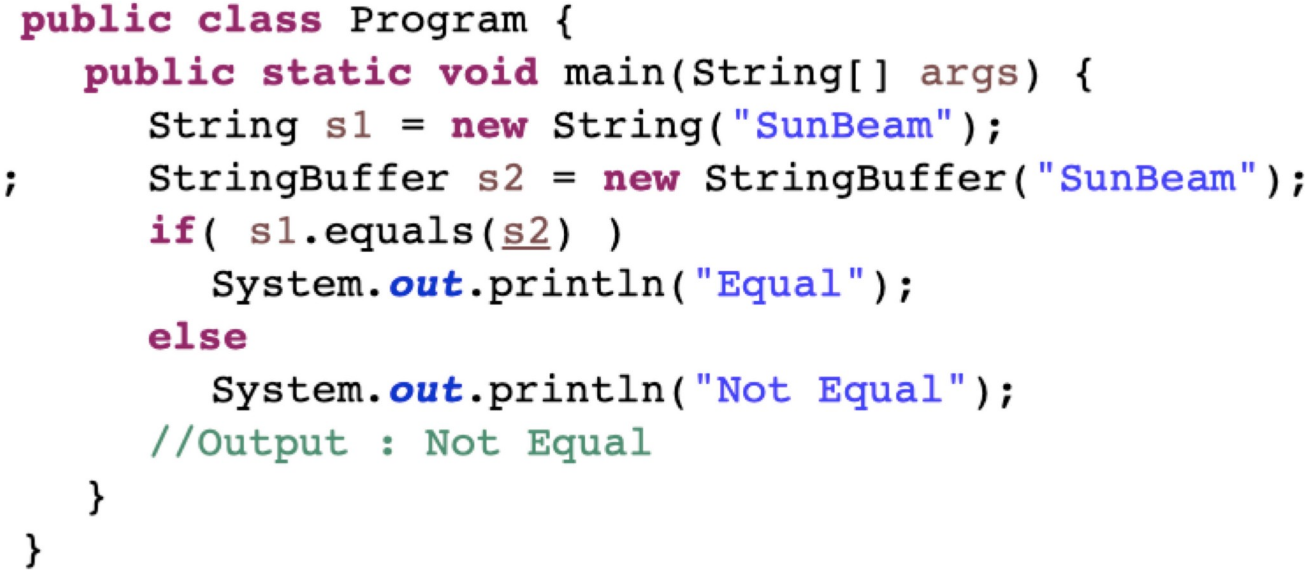
```
public class Program {
   public static void main(String[] args) {
      StringBuffer s1 = new StringBuffer("SunBeam");
      StringBuffer s2 = new StringBuffer("SunBeam");
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output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



```
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
   }
}
```

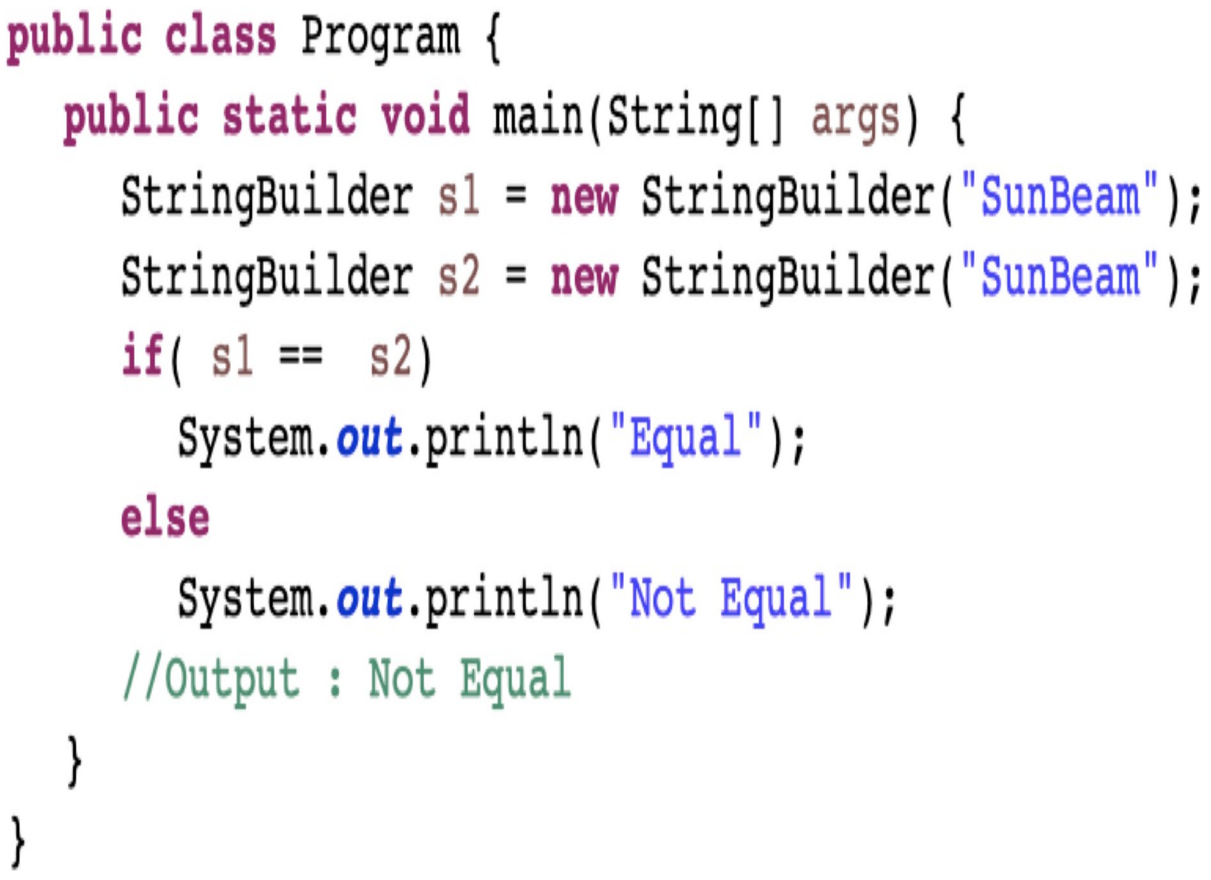


```
public class Program {
   public static void main(String[] args) {
      String s1 = new String("SunBeam");
      StringBuffer s2 = new StringBuffer("SunBeam");
      if( s1 == s2 )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Compiler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

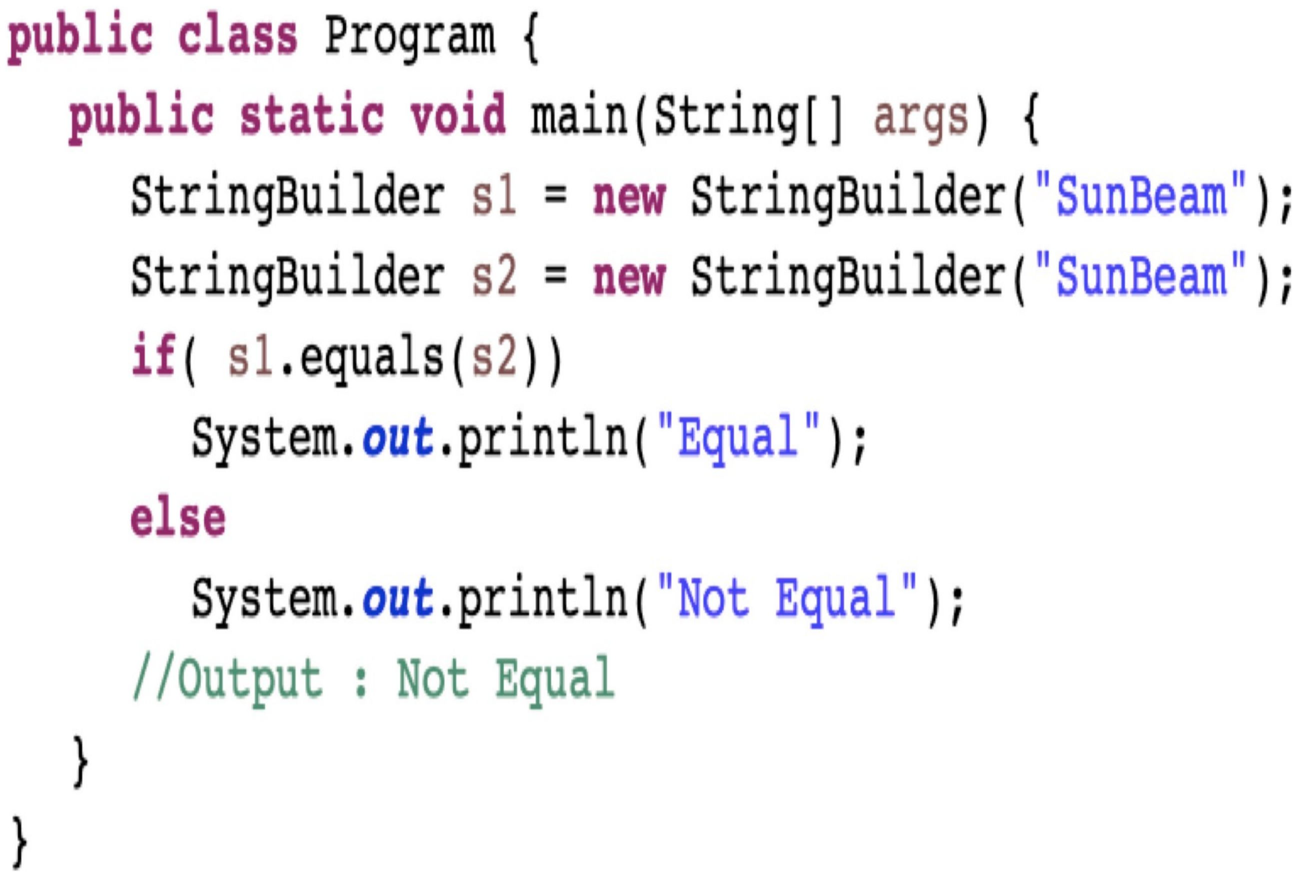


```
public class Program {
   public static void main(String[] args) {
      String s1 = new String("SunBeam");
      StringBuffer s2 = new StringBuffer("SunBeam");
      if( s1.equals(s2) )
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
      //Output : Not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

StringBuilder Example

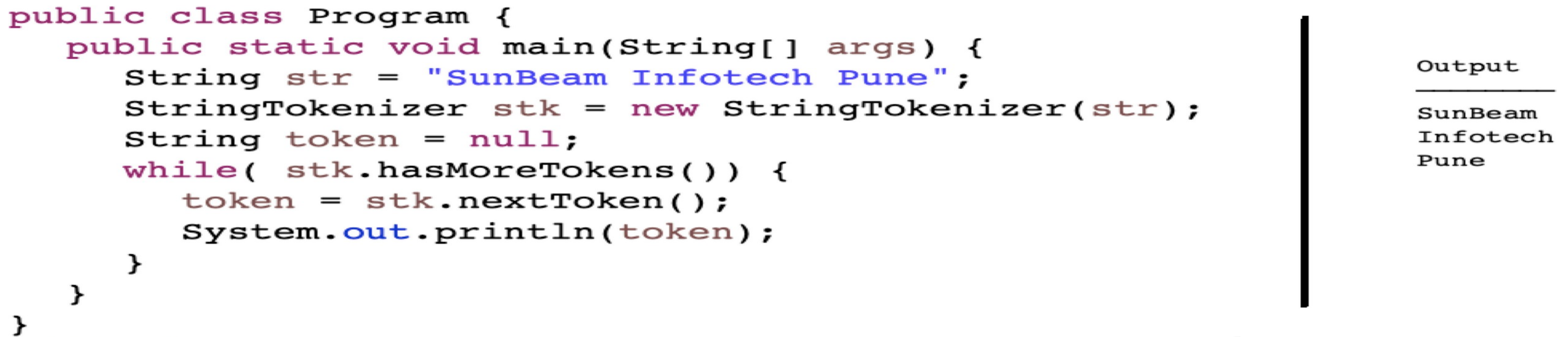


```java
public class Program {
   public static void main(String[] args) {
      StringBuilder s1 = new StringBuilder("SunBeam");
      StringBuilder s2 = new StringBuilder("SunBeam");
      if( s1 ==  s2)
         System.out.println("Equal");
      else
         System.out.println("Not Equal");
   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



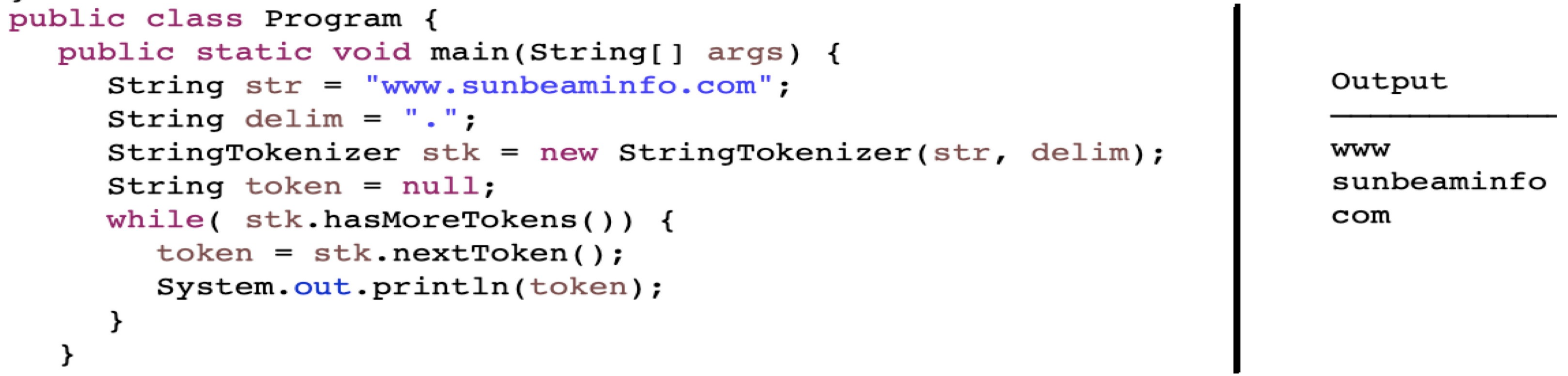
```java
public class Program {
   public static void main(String[] args) {
      StringBuilder s1 = new StringBuilder("SunBeam");
      StringBuilder s2 = new StringBuilder("SunBeam");
      if( s1.equals(s2))
         System.out.println("Equal");
      else
         System.out.println("Not Equal");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

StringTokenizer



```
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);
      }
   }
}
```

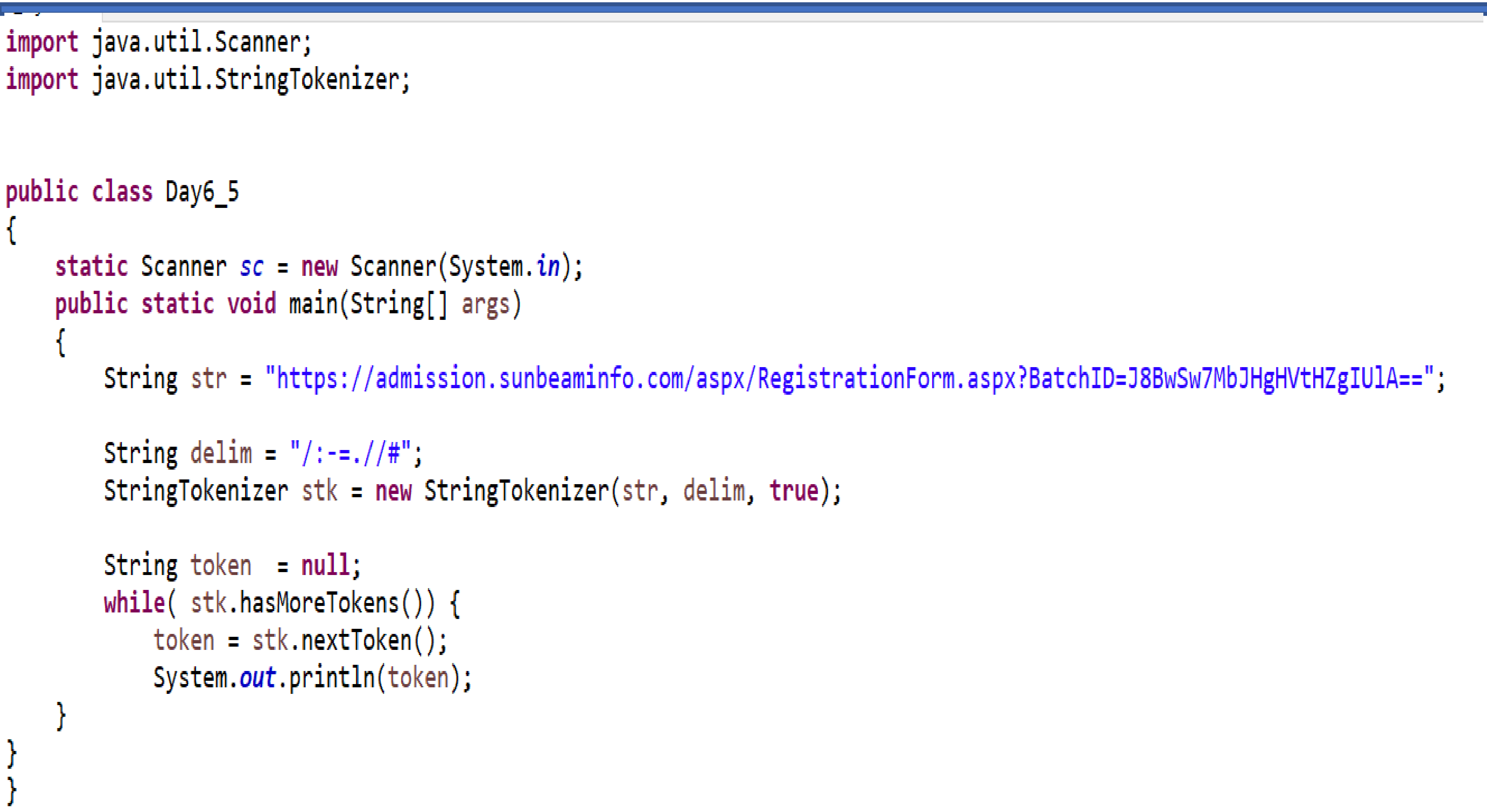
```
Output
______
SunBeam
Infotech
Pune
```



```
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
```

String Tokenizer



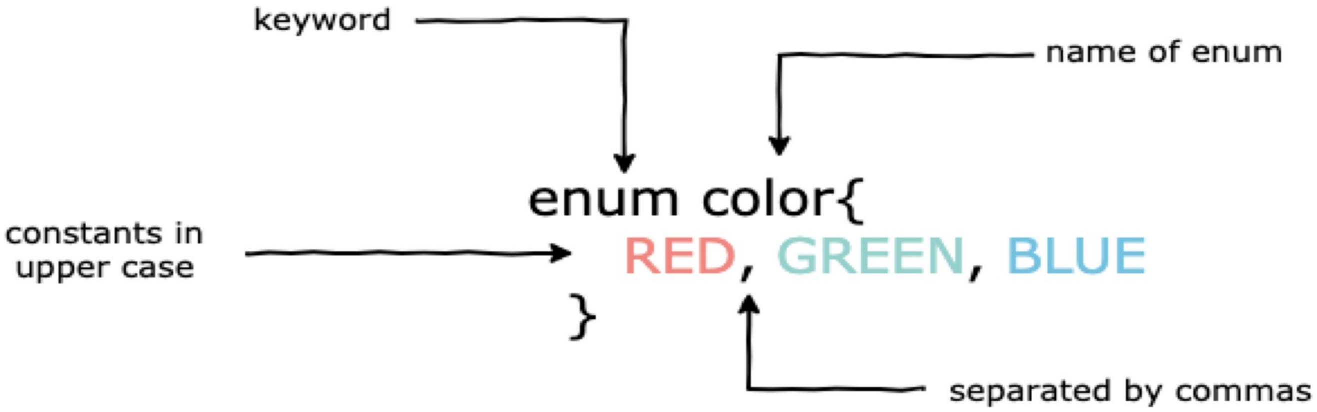
```java
import java.util.Scanner;
import java.util.StringTokenizer;


public class Day6_5
{
    static Scanner sc = new Scanner(System.in);
    public static void main(String[] args)
    {
        String str = "https://admission.sunbeaminfo.com/aspx/RegistrationForm.aspx?BatchID=J8BwSw7MbJHgHVtHZgIUlA==";

        String delim = "/:-=.//#";
        StringTokenizer stk = new StringTokenizer(str, delim, true);

        String token  = null;
        while( stk.hasMoreTokens()) {
            token = stk.nextToken();
            System.out.println(token);
    }
}
}
```

OUTPUT

```
https
:
/
/
admission
.
sunbeaminfo
.
com
/
aspx
/
RegistrationForm
.
aspx?BatchID
=
J8BwSw7MbJHgHVtHZ
gIUlA
=
=
```

Enum

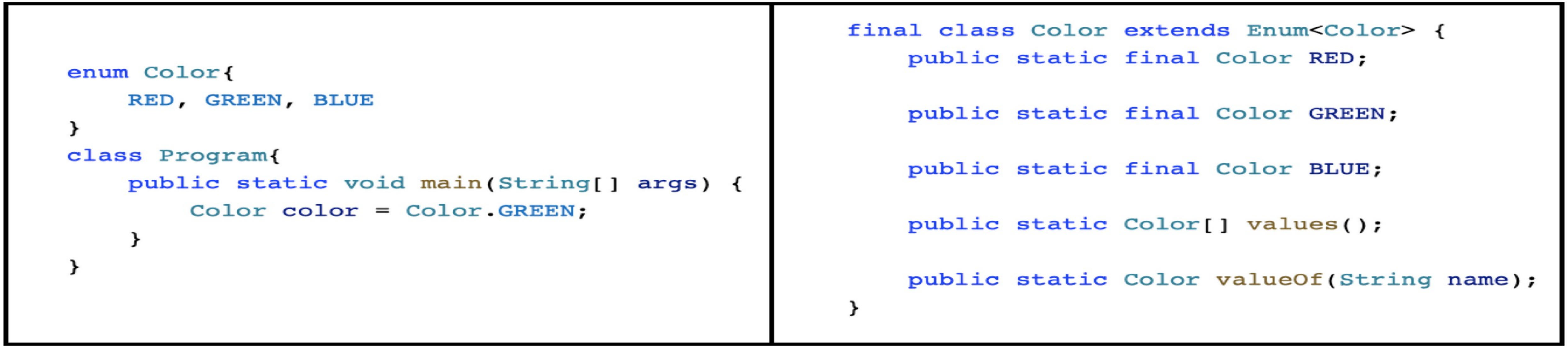
- According ANSI C standard, if we want to assign name to the integer constant then we should use enum.
- Enum helps developer to improve readability of source code.
- An enum is a class that represents a group of constants
- Enum keyword is used to create an enum. The constants declared inside are separated by a comma and should be in upper case.
- enum is used for values that are not going to change e.g. names of days, colors in a rainbow, number of cards in a deck etc.
- enum is commonly used in switch statements.



```
enum color{
    RED, GREEN, BLUE
}
```

Enum

- Similar to a class, an enum can have objects and methods. The only difference is that enum constants are public, static and final by default. Since it is final, we can’t extend enums
- It cannot extend other classes since it already extends the java.lang.Enum class.
- It can implement interfaces.
- The enum objects cannot be created explicitly and hence the enum constructor cannot be invoked directly.
- It can only contain concrete methods and no abstract methods.



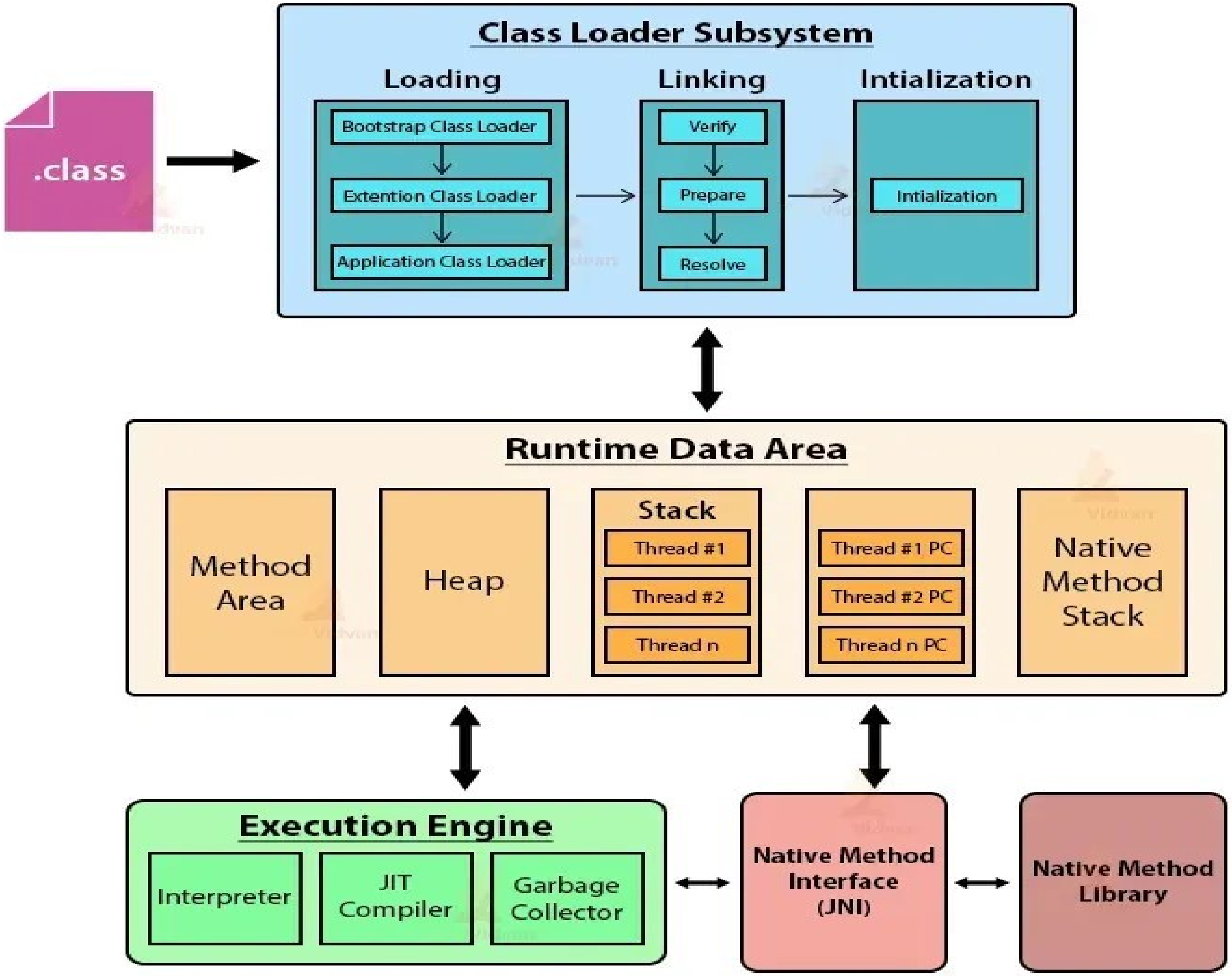
Java Source Code

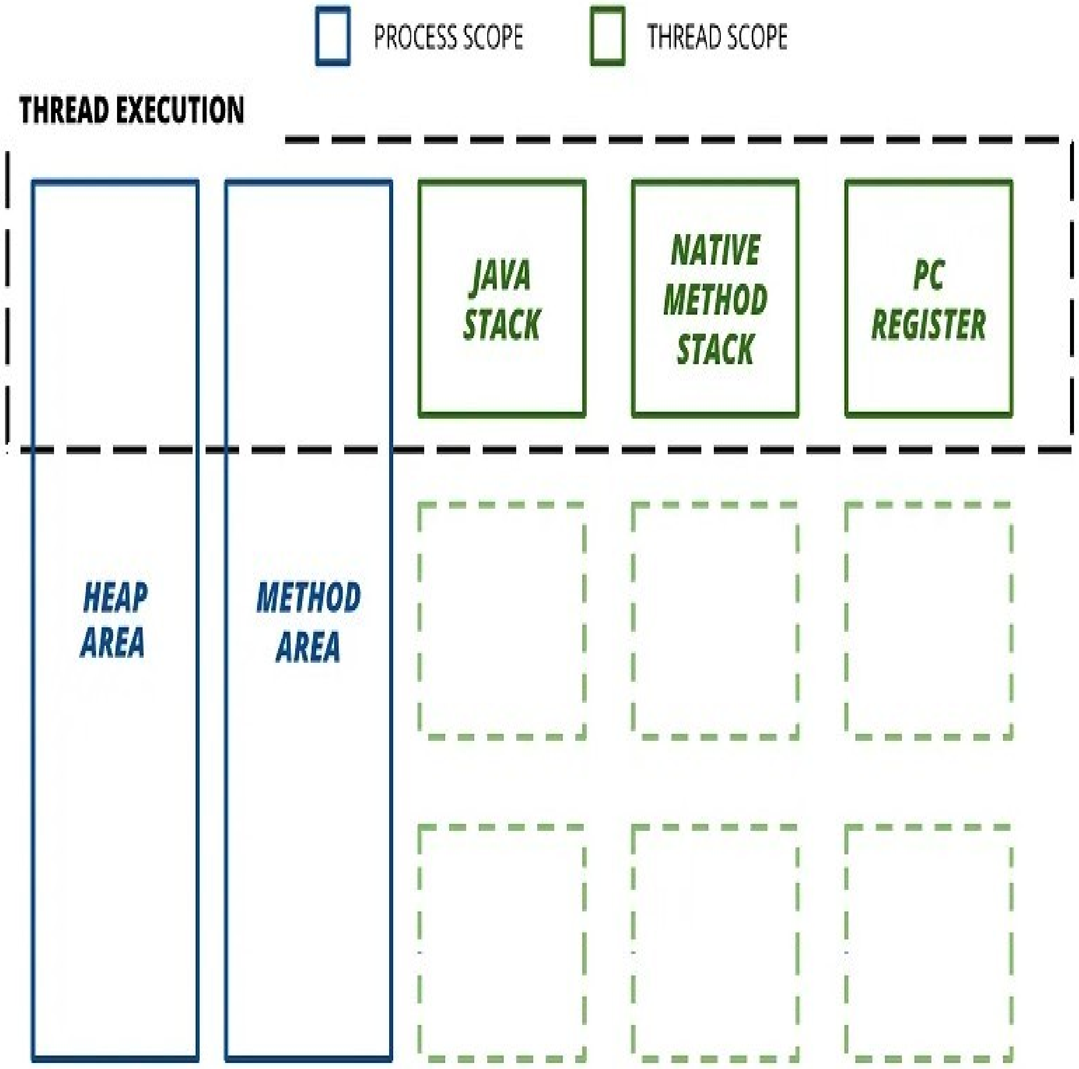
```java
enum Color{
    RED, GREEN, BLUE
}
class Program{
    public static void main(String[] args) {
        Color color = Color.GREEN;
    }
}
```

Compiled Code

```java
final class Color extends Enum<Color> {
    public static final Color RED;

    public static final Color GREEN;

    public static final Color BLUE;

    public static Color[] values();

    public static Color valueOf(String name);
}
```

JVM Architecture





JVM Architecture

- There are three main mechanisms inside the JVM as shown in the above diagram.
  1. Class Loader
  2. Memory Area
  3. Execution Engine

- **ClassLoader :**
- Classloader consists of 3 class loaders as below -
  1. Bootstrap ClassLoader — Load classes from JRE/lib/rt.jar
  2. Extension ClassLoader — Load classes from JRE/lib/ext
  3. Application/System ClassLoader — Load classes from classpath, -cp

JVM Architecture (ClassLoader)

- ClassLoader is responsible for loading class file into the memory area.
- The classloader is an abstract class, generates the data which constitutes a definition for the class using a class binary name which is constituted of the package name, and a class name.
- The ClassLoading mechanism consists of three main steps as follows.
  1. Loading
  2. Linking
  3. Initialization

- **Loading :**
- Whenever JVM loads a file, it will load and read,
  - Fully qualified class name
  - Variable information (instance variables)
  - Immediate parent information
  - Whether class or interface or enum

JVM Architecture (ClassLoader)

- **Linking :**
- This is the process of linking the data in the class file into the memory area. It begins with verification to ensure this class file and the compiler.
- Make sure this compiler is valid
- The class file has the correct formatting and the correct structure
- When the verification process is done, the next step is preparation.
- In this stage, all the variables are initialized with the default value.
- As an example, it will assign 0 for int variables, null for all objects, false for all boolean variables, etc.

- **Initialization :**
- This is the final stage of class loading. In this stage, the actual values are assigned to all static and instance variables.

JVM Architecture (Memory Area)

- This is the place data will store until the program is executed. It consists of 5 major components as follows.

1. Method Area :

- This is a shared resource (only 1 method area per JVM). Method area stores class-level information such as
  - ClassLoader reference
  - Run time constant pool
  - Constructor data — Per constructor: parameter types (in order)
  - Method data — Per method: name, return type, parameter types (in order), modifiers, attributes
  - Field data — Per field: name, type, modifiers, attributes

2. Heap Area

- All the objects and their corresponding instance variables are stored in the heap area. As an example, if your program consists of a class called “Employee” and you are declaring an instance as follows,
- Employee employee = new Employee();
- When the class is loaded, there is an instance of employee is created and it will be loaded into the Heap Area.

JVM Architecture (Memory Area)

3. Stack Area

- There are separate stack areas for each thread.
- The stack is responsible for hold the methods (method local variables, etc) and whenever we invoke a method, a new frame creates in the stack.
- These frames use Last-In-First-Out structure.
- They will be destroyed when the method execution is completed.

4. Program Counter (PC) Register

- PC Register keeps a record of the current instruction executing at any moment.
- That is like a pointer to the current instruction in a sequence of instructions in a program.
- Once the instruction is executed, the PC register is updated with the next instruction.
- If the currently executing method is ‘native’, then the value of the program counter register will be undefined.

5. Native Method Area

- A native method is a Java method that is implemented in another programming language, such as C or C++. This memory area is responsible to hold the information about these native methods.

JVM Architecture (Execution Engine)

- At the end of the loading and storing mechanisms, the final stage of JVM is executing the class file. Execution Engine has three main components.
  1. Interpreter
  2. JIT Compiler
  3. Garbage Collector

1. Interpreter :

- When the program is started to execute, the interpreter reads byte code line by line.
- This process will use some sort of dictionary that implies this kind of byte code should be converted into this kind of machine instructions.
- The main advantage of this process is the interpreter is very quick to load and fast execution.
- But whenever it executes the same code blocks (methods, loops, etc) again and again, the interpreter executes repeatedly.
- It means the interpreter cannot be optimized the run time by reducing the same code repeated execution.

JVM Architecture (Execution Engine)

2. JIT Compiler :

- Just In Time Compiler (JIT Compiler) is introduced to overcome the main disadvantage of the interpreter.
- That means the JIT compiler can remember code blocks that execute again and again.
- As an example, there is a class called “Employee” and getEmployeeID() method is declared in this class. If a program uses getEmployeeID() method 1000 times, each time it is executed by the interpreter.
- But the JIT compiler can identify those repeated code segments and they will be stored as native codes in the cache.
- Since it is stored, next time JIT compiler uses the native code which stored in the cache.

3. Garbage Collector (GC) :

- All the objects are stored in the heap area before JVM starts the execution.
- Since this area is limited, it is required to manage this area efficiently by removing the objects that are no longer in use.
- The process of removing unused objects from heap memory is known as Garbage collection and this is a part of memory management in Java.

Thank you!

Rohan Paramane

rohan.paramane@sunbeaminfo.com