STRING Top Interview Questions – JAVA

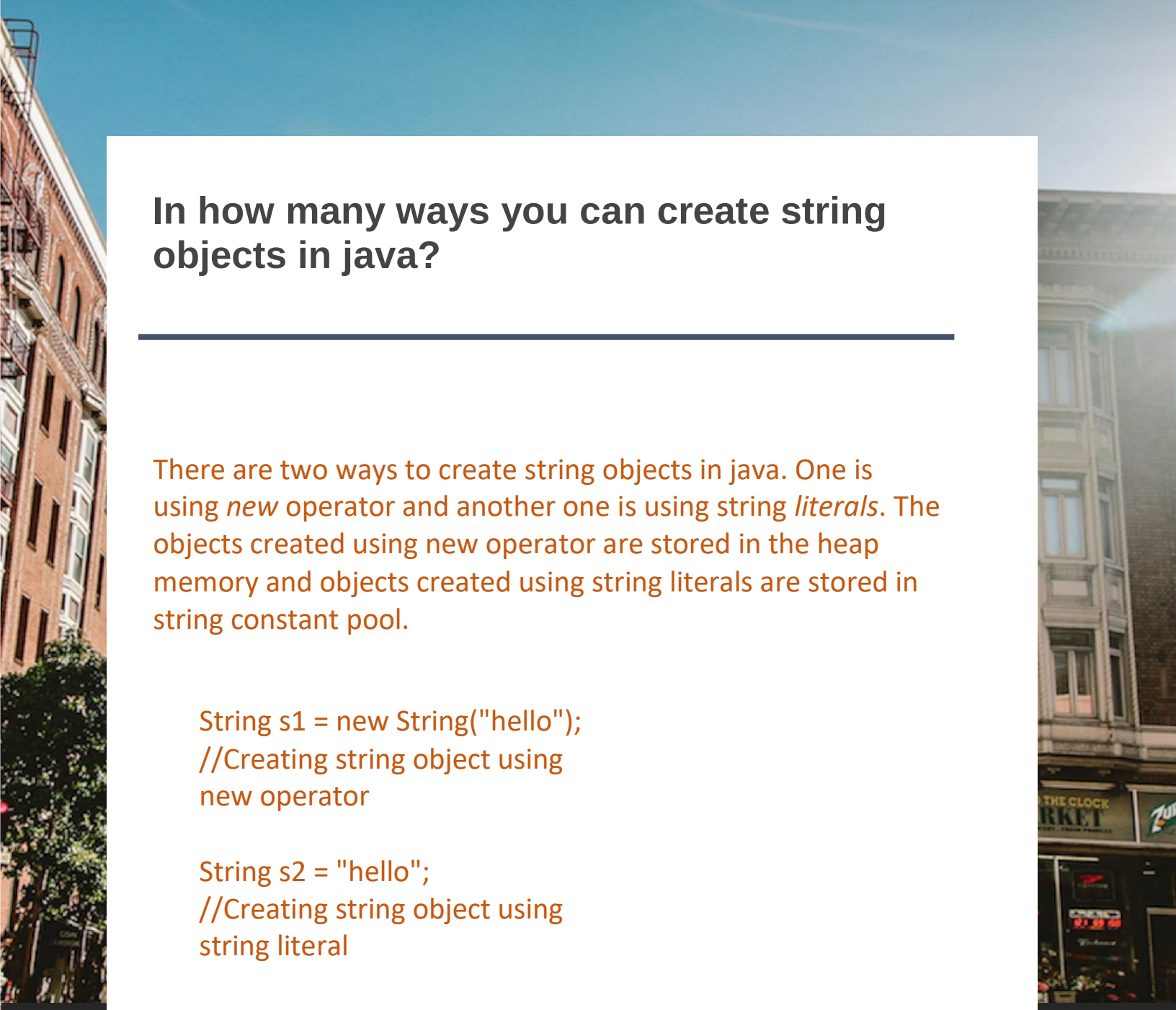
By

Prashant Kumar

https://www.linkedin.com/in/prashant-kumar-76b786168/

https://github.com/prashantt17

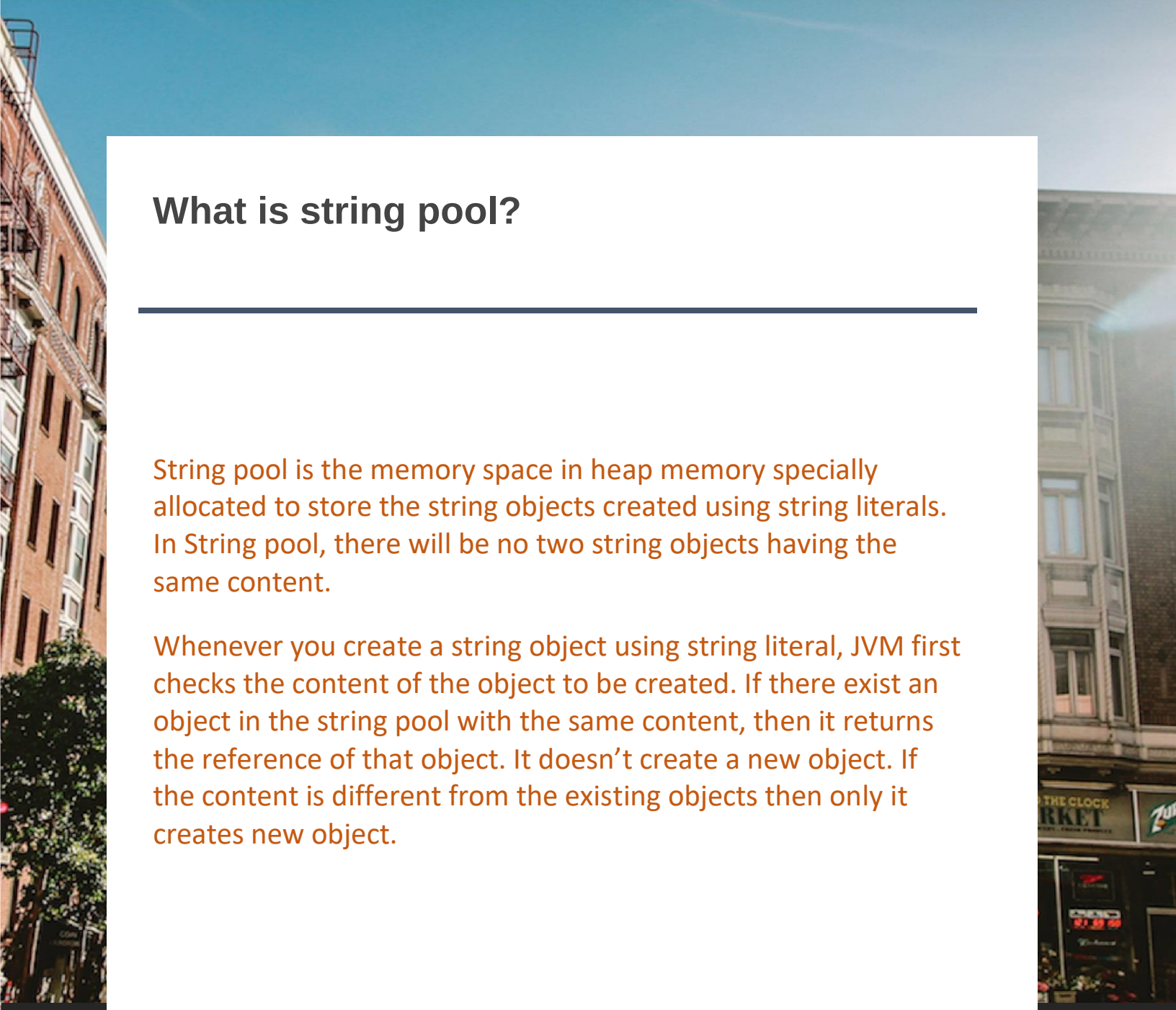
@prashantkumar



In how many ways you can create string objects in java?

There are two ways to create string objects in java. One is using *new* operator and another one is using string *literals*. The objects created using new operator are stored in the heap memory and objects created using string literals are stored in string constant pool.

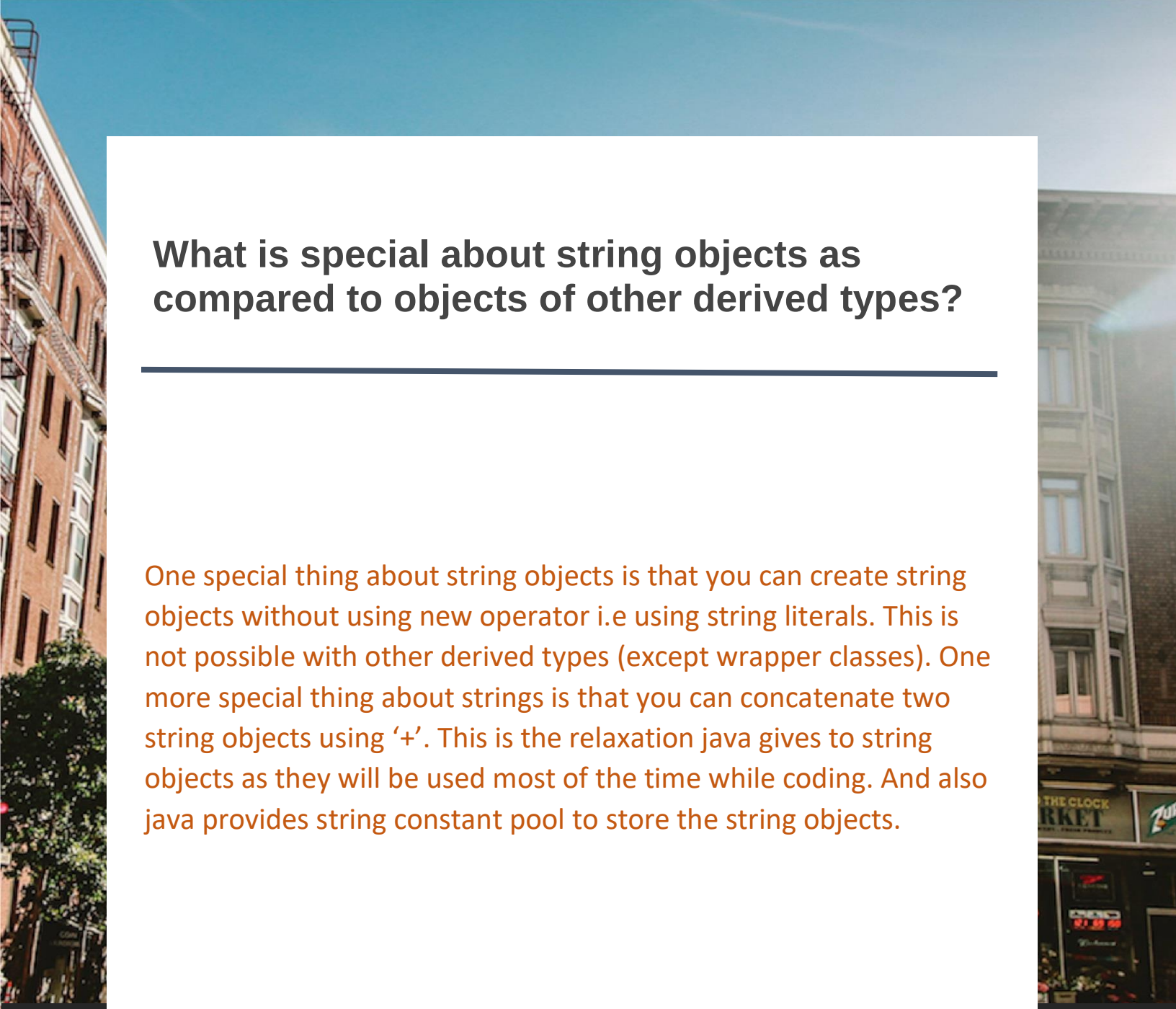
```
String s1 = new String("hello");
//Creating string object using
new operator

String s2 = "hello";
//Creating string object using
string literal
```



What is string pool?

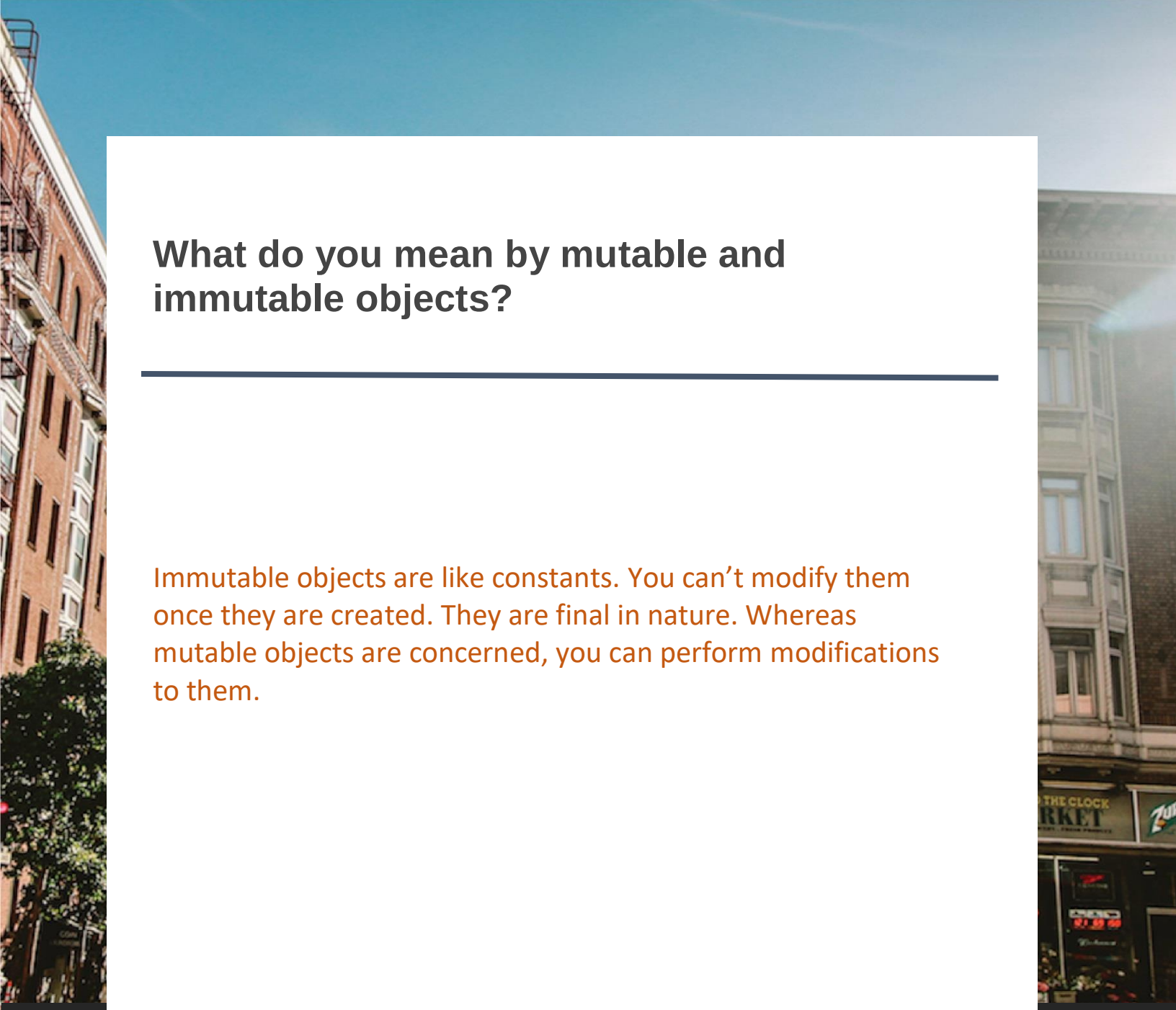
String pool is the memory space in heap memory specially allocated to store the string objects created using string literals. In String pool, there will be no two string objects having the same content.

Whenever you create a string object using string literal, JVM first checks the content of the object to be created. If there exist an object in the string pool with the same content, then it returns the reference of that object. It doesn't create a new object. If the content is different from the existing objects then only it creates new object.



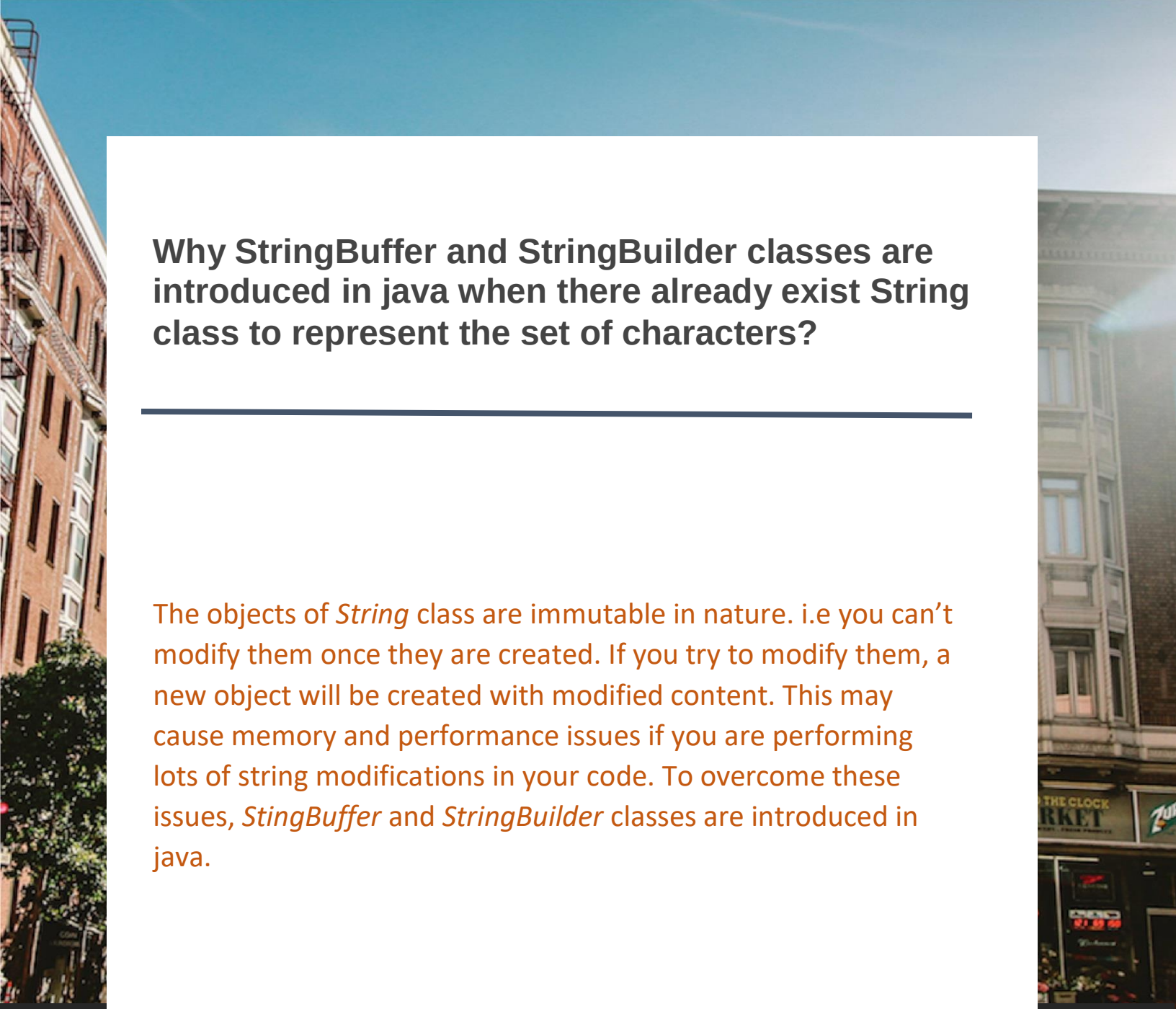
What is special about string objects as compared to objects of other derived types?

One special thing about string objects is that you can create string objects without using new operator i.e using string literals. This is not possible with other derived types (except wrapper classes). One more special thing about strings is that you can concatenate two string objects using '+'. This is the relaxation java gives to string objects as they will be used most of the time while coding. And also java provides string constant pool to store the string objects.



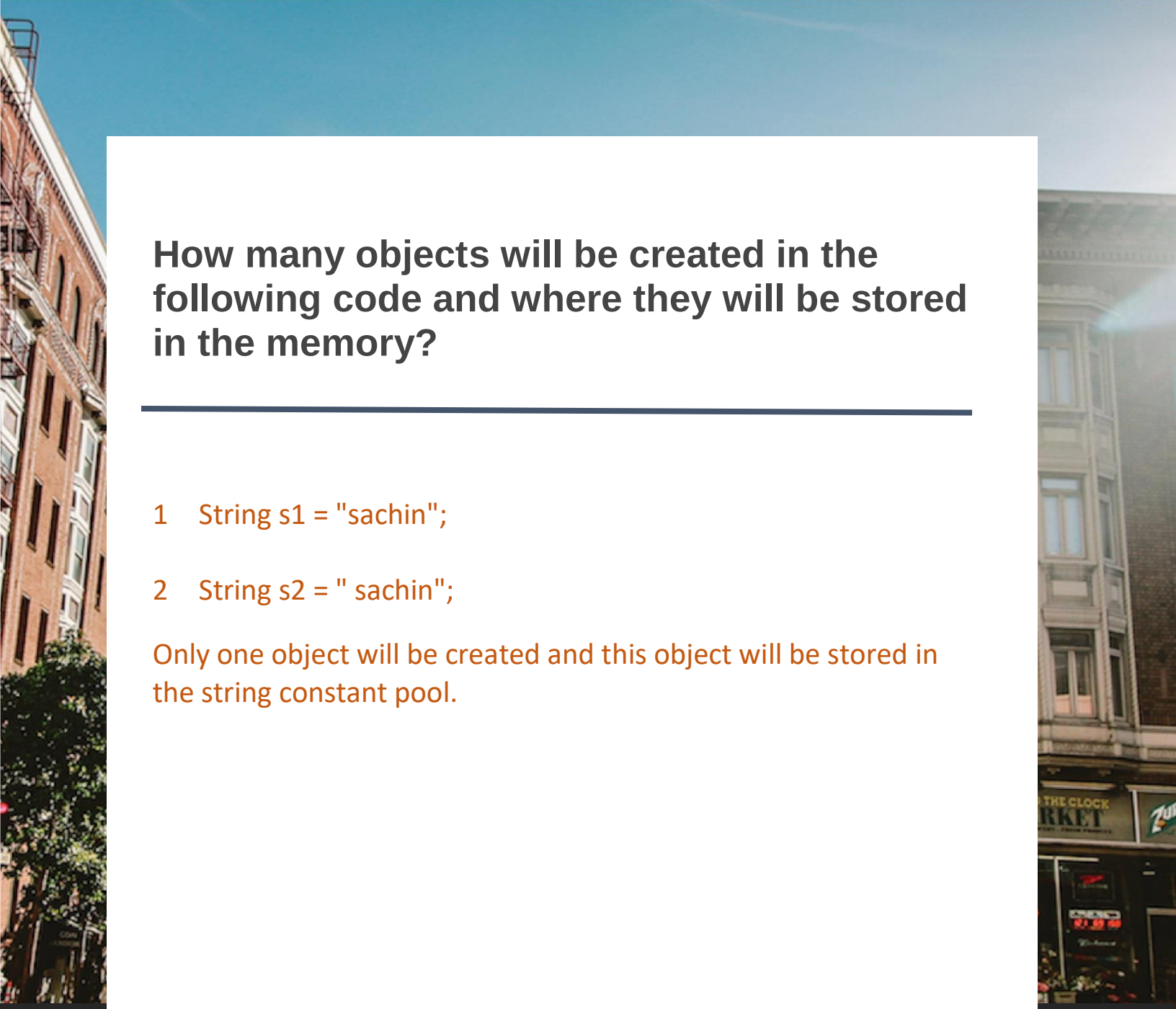
What do you mean by mutable and immutable objects?

Immutable objects are like constants. You can’t modify them once they are created. They are final in nature. Whereas mutable objects are concerned, you can perform modifications to them.



Why StringBuffer and StringBuilder classes are introduced in java when there already exist String class to represent the set of characters?

The objects of *String* class are immutable in nature. i.e you can't modify them once they are created. If you try to modify them, a new object will be created with modified content. This may cause memory and performance issues if you are performing lots of string modifications in your code. To overcome these issues, *StingBuffer* and *StringBuilder* classes are introduced in java.

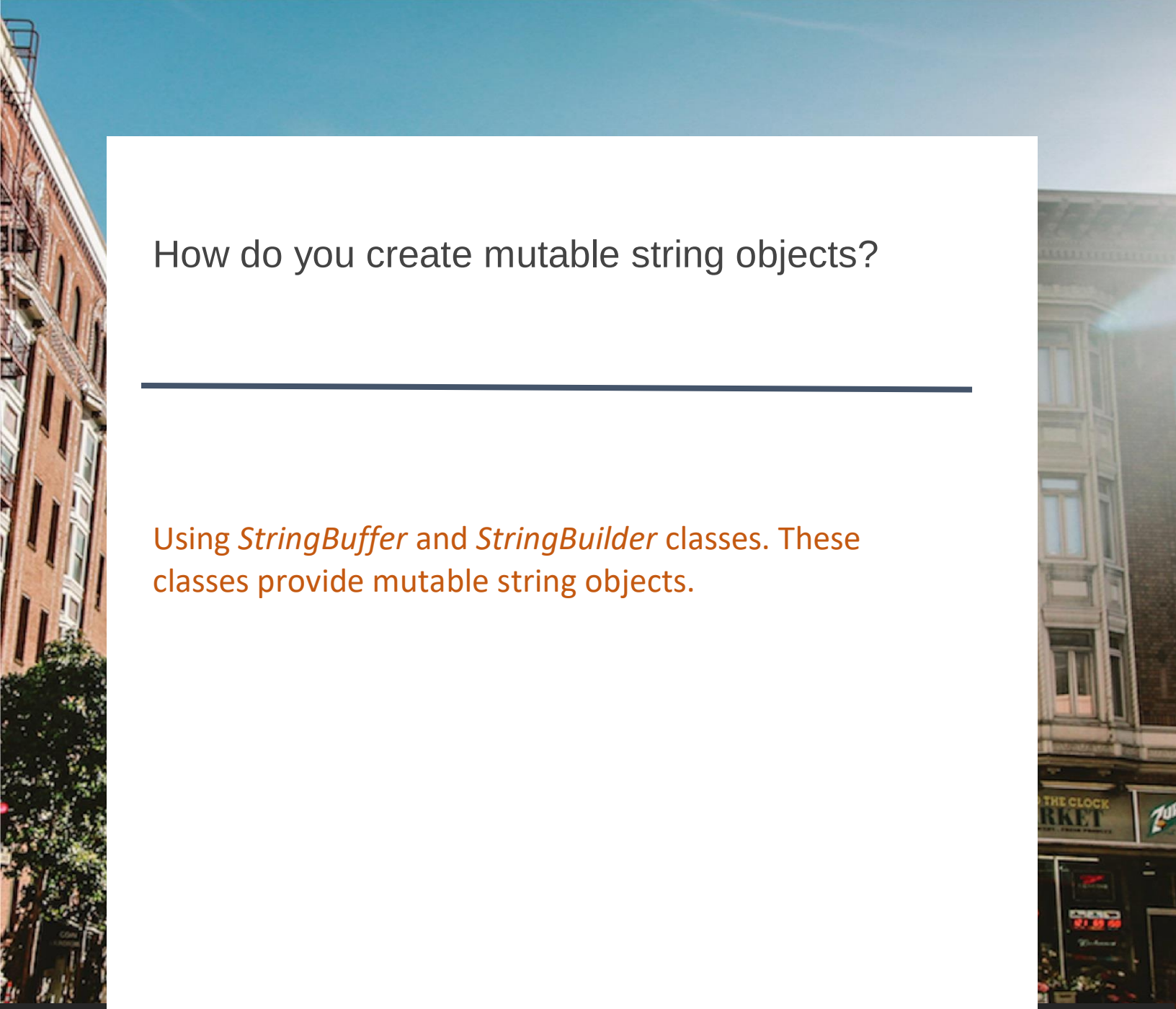


How many objects will be created in the following code and where they will be stored in the memory?

1 String s1 = "sachin";

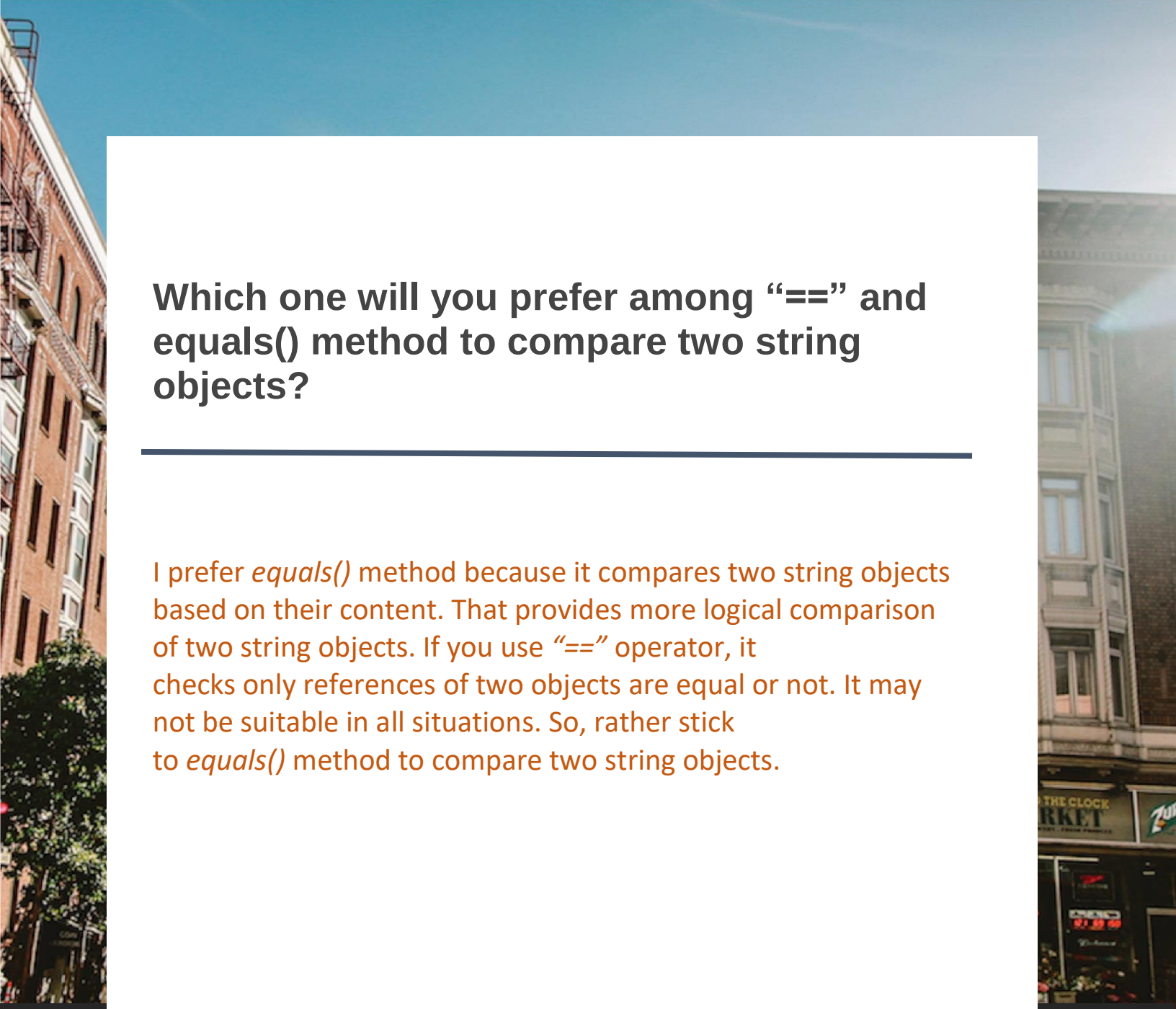
2 String s2 = " sachin";

Only one object will be created and this object will be stored in the string constant pool.



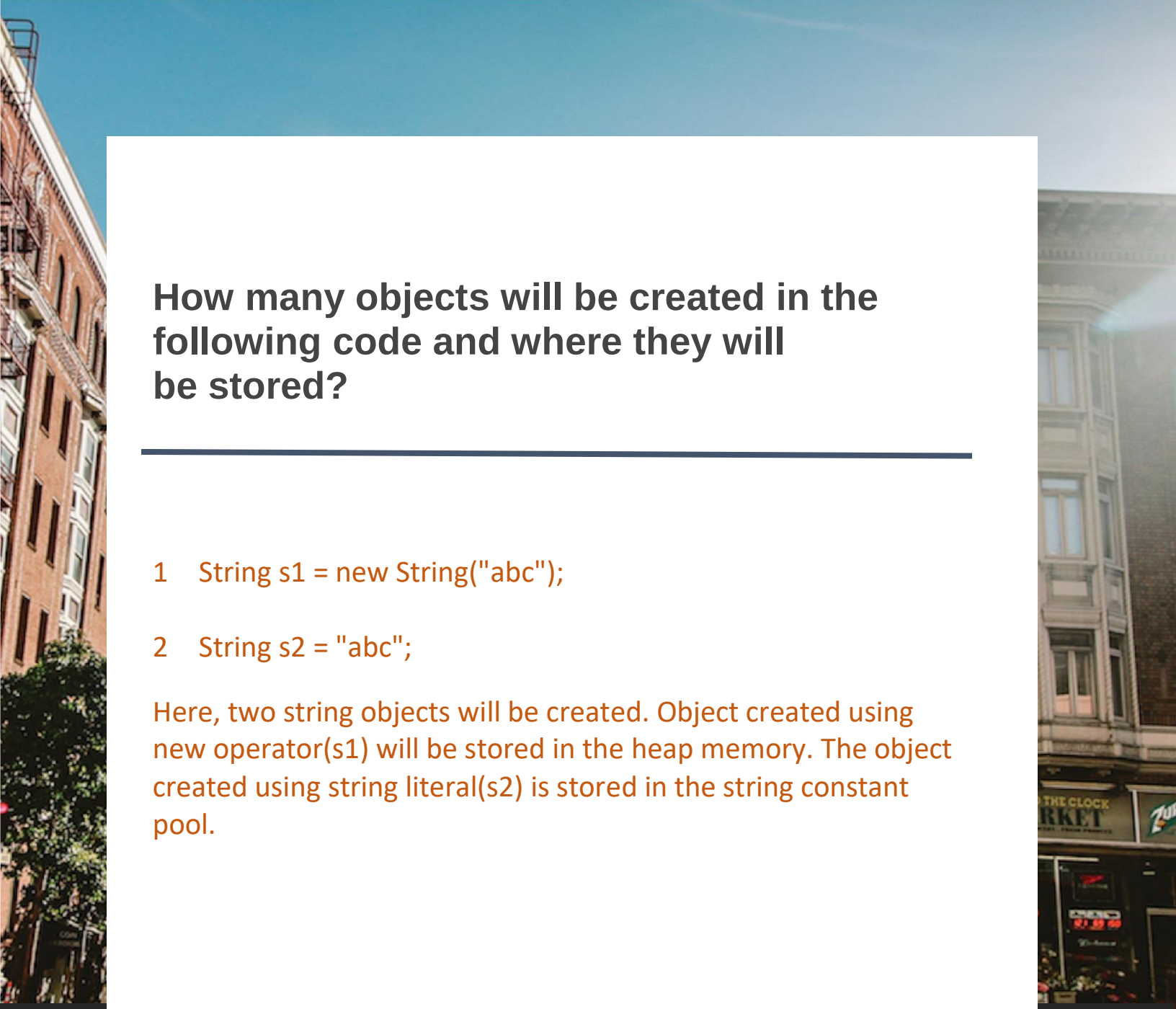
How do you create mutable string objects?

Using *StringBuffer* and *StringBuilder* classes. These classes provide mutable string objects.



Which one will you prefer among “==” and equals() method to compare two string objects?

I prefer *equals()* method because it compares two string objects based on their content. That provides more logical comparison of two string objects. If you use *“==”* operator, it checks only references of two objects are equal or not. It may not be suitable in all situations. So, rather stick to *equals()* method to compare two string objects.



How many objects will be created in the following code and where they will be stored?

```
1    String s1 = new String("abc");

2    String s2 = "abc";
```

Here, two string objects will be created. Object created using new operator(s1) will be stored in the heap memory. The object created using string literal(s2) is stored in the string constant pool.

Where exactly string constant pool is located in the memory?

Inside the heap memory. JVM reserves some part of the heap memory to store string objects created using string literals.

Whenever you create a string object using string literal, that object is stored in the **string constant pool** and whenever you create a string object using new keyword, such object is stored in the heap memory.

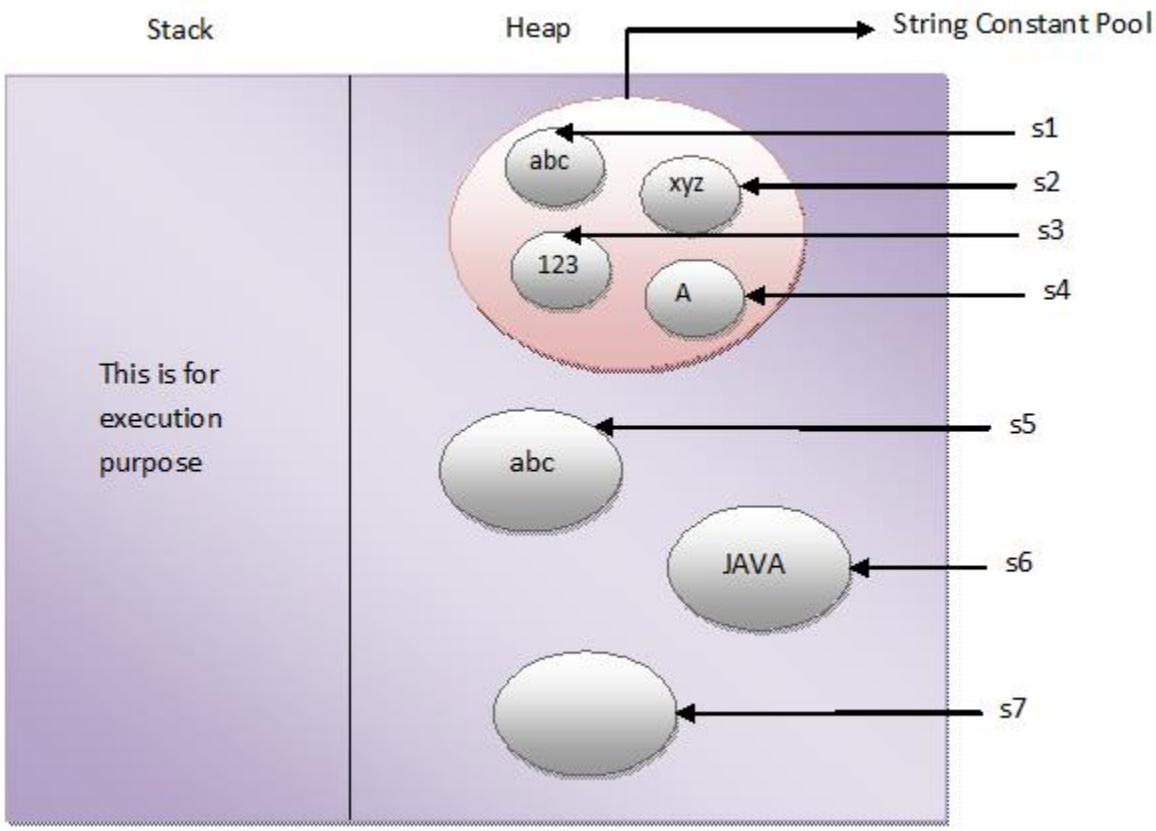
For example, when you create string objects like below, they will be stored in the String Constant Pool.

```
1    String s1 = "abc";
2
3    String s2 = "xyz";
4
5    String s3 = "123";
6
7    String s4 = "A";
```

And when you create string objects using new keyword like below, they will be stored in the heap memory.

```
1    String s5 = new String("abc");
2
3    char[] c = {'J', 'A', 'V', 'A'};
4
5    String s6 = new String(c);
6
7    String s7 = new String(new StringBuffer());
```

This is how String Constant Pool looks like in the memory.



One more interesting thing about String Constant Pool is that, **pool space is allocated to an object depending upon it's content**. There will be no two objects in the pool having the same content.

This is what happens when you create string objects using string literal,

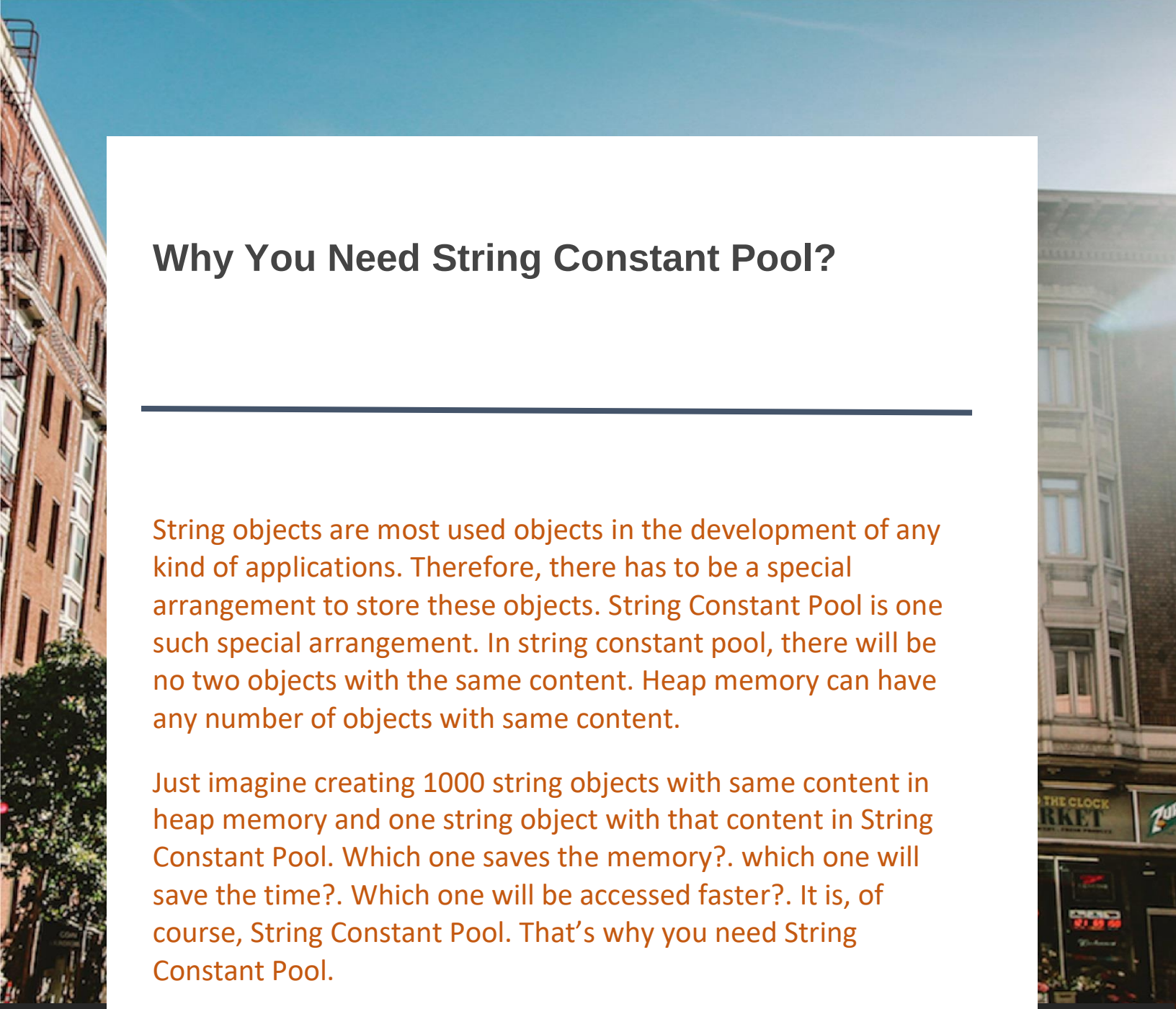
"When you create a string object using string literal, JVM first checks the content of to be created object. If there exist an object in the pool with the same content, then it returns the reference of that object. It doesn't create new object. If the content is different from the existing objects then only it creates new object."

But, when you create string objects using new keyword, a new object is created whether the content is same or not.

This can be proved by using “==” operator. As “==” operator returns true if two objects have same physical address in the memory otherwise it will return false. In the below example, s1 and s2 are created using string literal “abc”. So, s1 == s2 returns true. Where as s3 and s4 are created using new operator having the same content. But, s3 == s4 returns false.

```
?
1       public class StringExamples
2       {
3          public static void main(String[] args)
4          {
5             //Creating string objects using literals
6
7             String s1 = "abc";
8
9             String s2 = "abc";
10
11            System.out.println(s1 == s2);       //Output : true
12
13            //Creating string objects using new operator
14
15            String s3 = new String("abc");
16
17            String s4 = new String("abc");
18
19            System.out.println(s3 == s4);       //Output : false
20         }
21      }
```

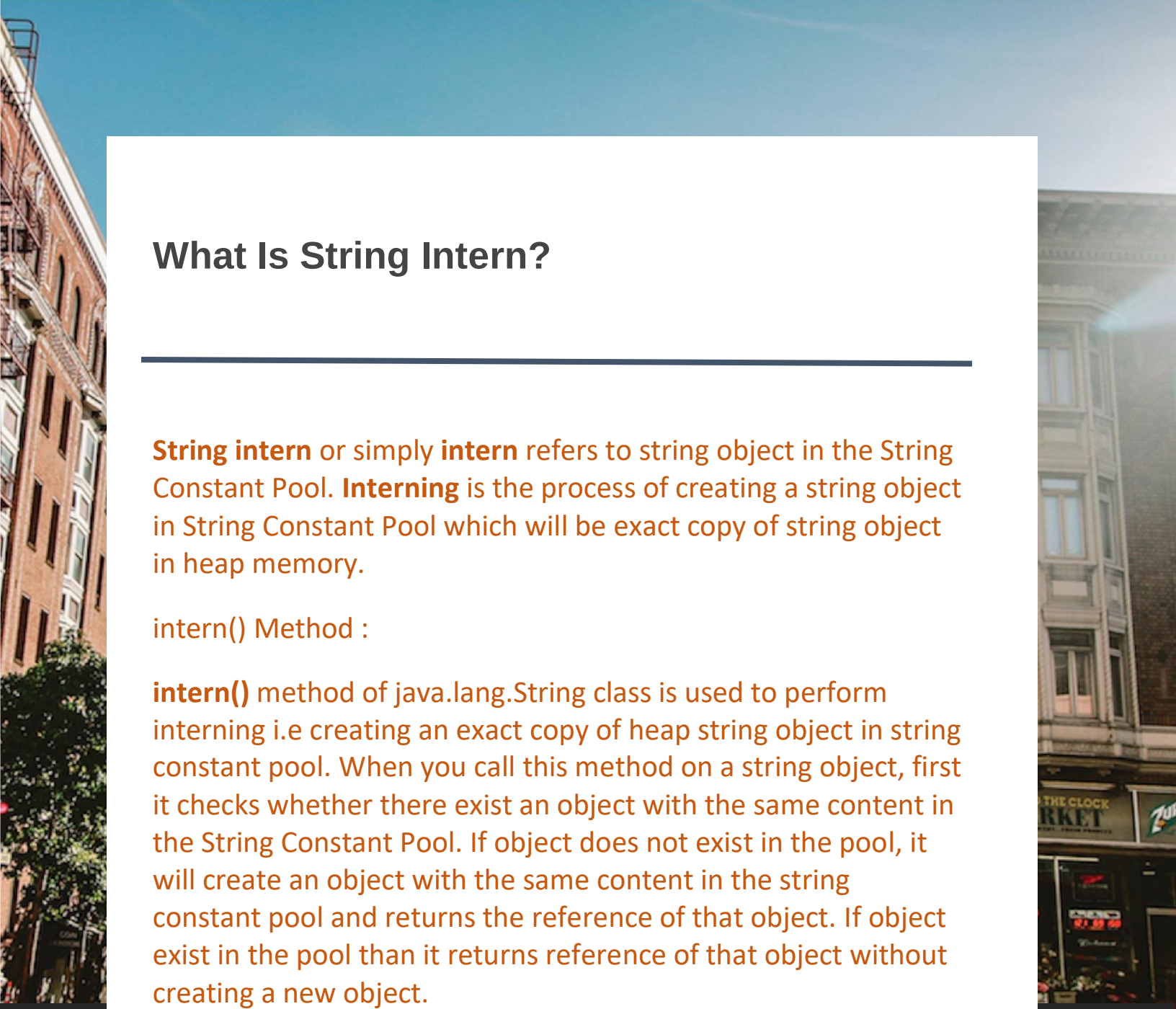
In simple words, there can not be two string objects with same content in the string constant pool. But, there can be two string objects with the same content in the heap memory.



Why You Need String Constant Pool?

String objects are most used objects in the development of any kind of applications. Therefore, there has to be a special arrangement to store these objects. String Constant Pool is one such special arrangement. In string constant pool, there will be no two objects with the same content. Heap memory can have any number of objects with same content.

Just imagine creating 1000 string objects with same content in heap memory and one string object with that content in String Constant Pool. Which one saves the memory?. which one will save the time?. Which one will be accessed faster?. It is, of course, String Constant Pool. That’s why you need String Constant Pool.



What Is String Intern?

String intern or simply **intern** refers to string object in the String Constant Pool. **Interning** is the process of creating a string object in String Constant Pool which will be exact copy of string object in heap memory.

intern() Method :

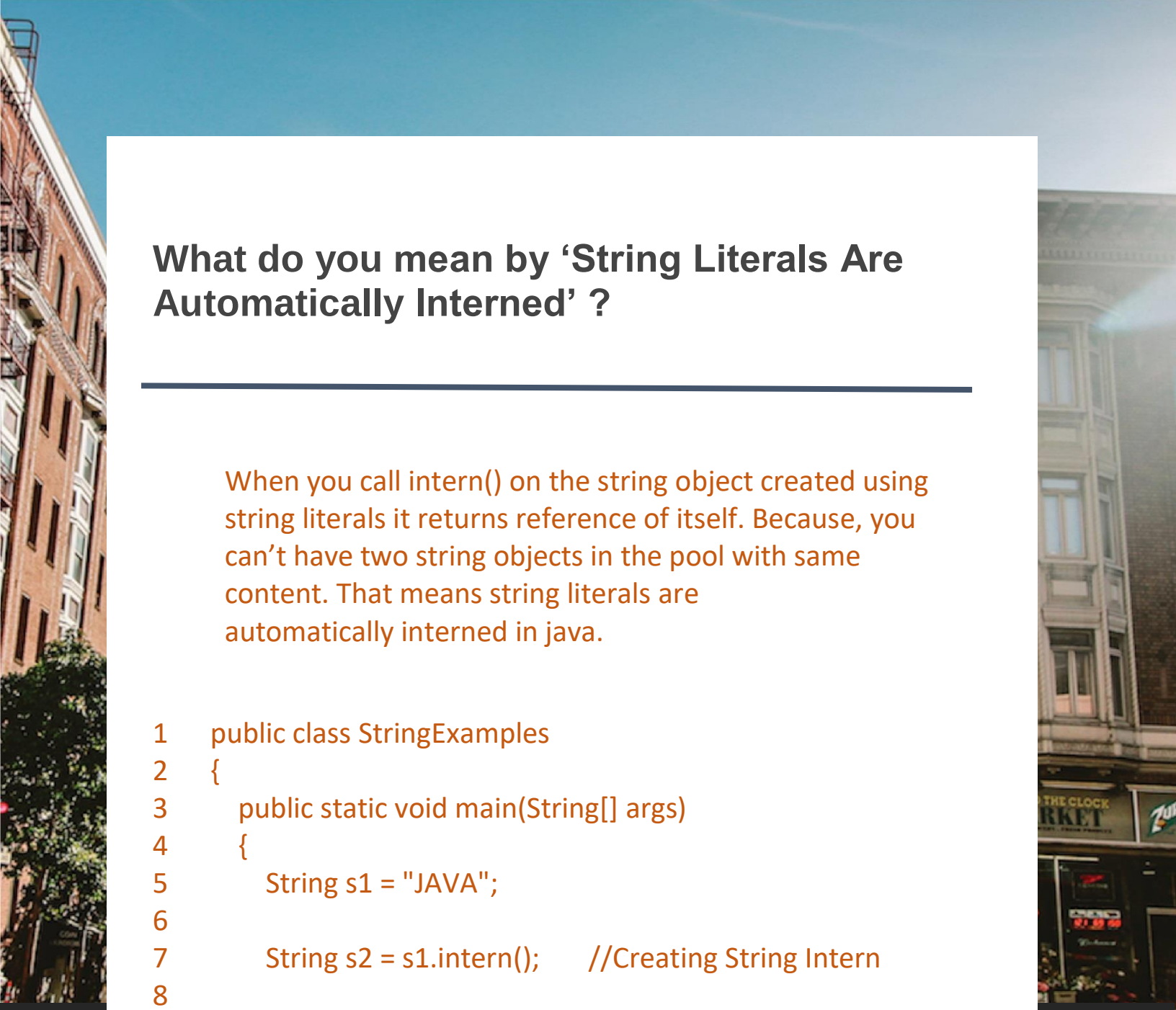
intern() method of java.lang.String class is used to perform interning i.e creating an exact copy of heap string object in string constant pool. When you call this method on a string object, first it checks whether there exist an object with the same content in the String Constant Pool. If object does not exist in the pool, it will create an object with the same content in the string constant pool and returns the reference of that object. If object exist in the pool than it returns reference of that object without creating a new object.

Look at the below example. Object ‘s1’ will be created in heap memory as we are using new operator to create it. When we call intern() method on s1, it creates a new string object in the string constant pool with “JAVA” as it’s content and assigns it’s reference to s2. So, **s1 == s2** will return false because they are two different objects in the memory and s1.equals(s2) will return true because they have same content.

```
public class StringExamples
{
   public static void main(String[] args)
   {
      String s1 = new String("JAVA");

      String s2 = s1.intern();      //Creating String Intern

      System.out.println(s1 == s2);      //Output : false

      System.out.println(s1.equals(s2));    //Output : true
   }
}
```

Look at this example. Object s1 will be created in string constant pool as we are using string literal to create it and object s2 will be created in heap memory as we are using new operator to create it. When you call intern() method on s2, it returns reference of object to which s1 is pointing as it's content is same as s2. It does not create a new object in the pool. So, **S1 == s3** will return true as both are pointing to same object in the pool.

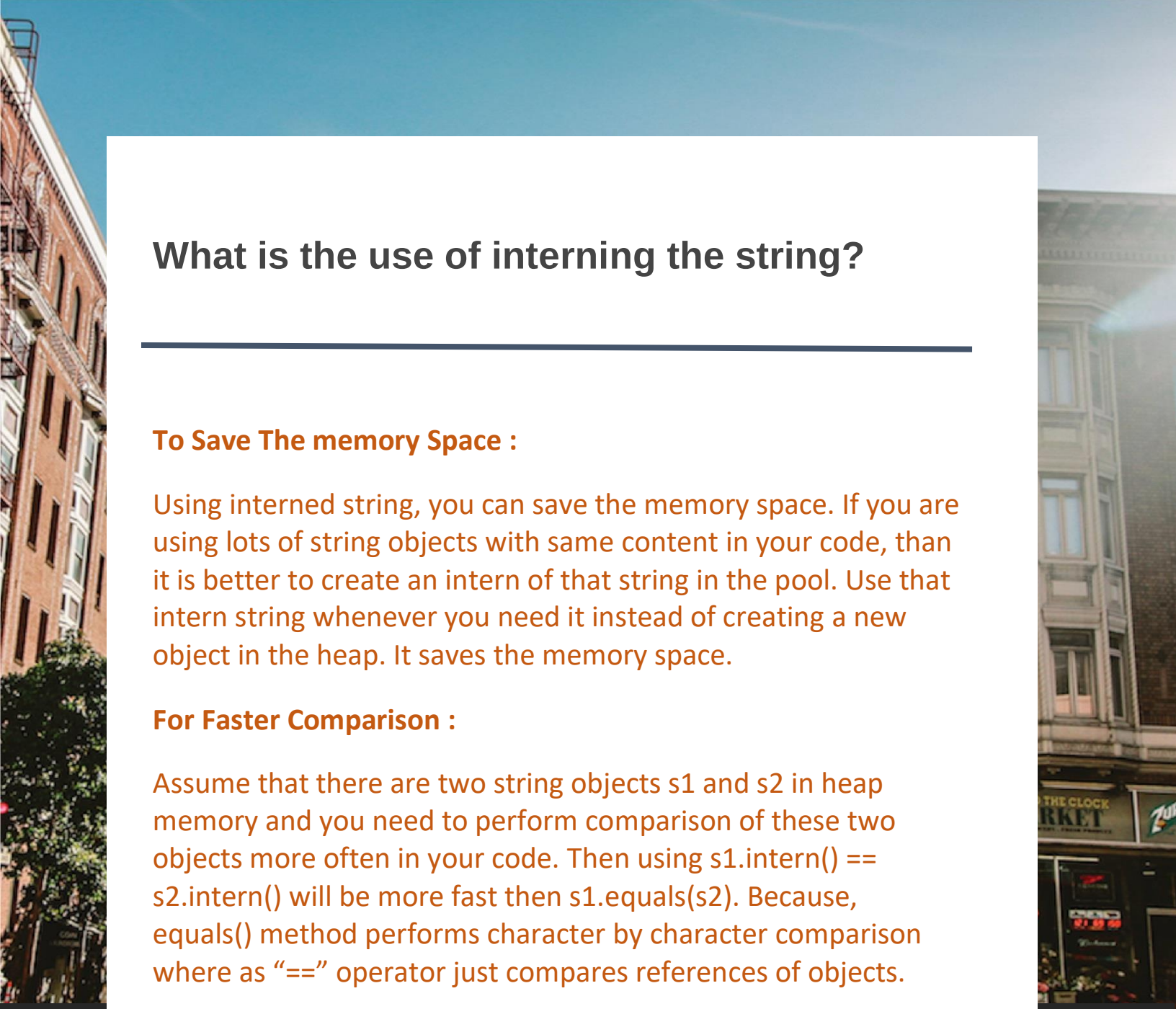
```
?
1       public class StringExamples
2       {
3          public static void main(String[] args)
4          {
5             String s1 = "JAVA";
6
7             String s2 = new String("JAVA");
8
9             String s3 = s2.intern();      //Creating String Intern
10
11            System.out.println(s1 == s3);      //Output : true
12         }
13      }
```



What do you mean by ‘String Literals Are Automatically Interned’ ?

When you call intern() on the string object created using string literals it returns reference of itself. Because, you can’t have two string objects in the pool with same content. That means string literals are automatically interned in java.

```
1   public class StringExamples
2   {
3      public static void main(String[] args)
4      {
5         String s1 = "JAVA";
6
7         String s2 = s1.intern();      //Creating String Intern
8
9         System.out.println(s1 == s2);      //Output : true
10     }
11  }
```



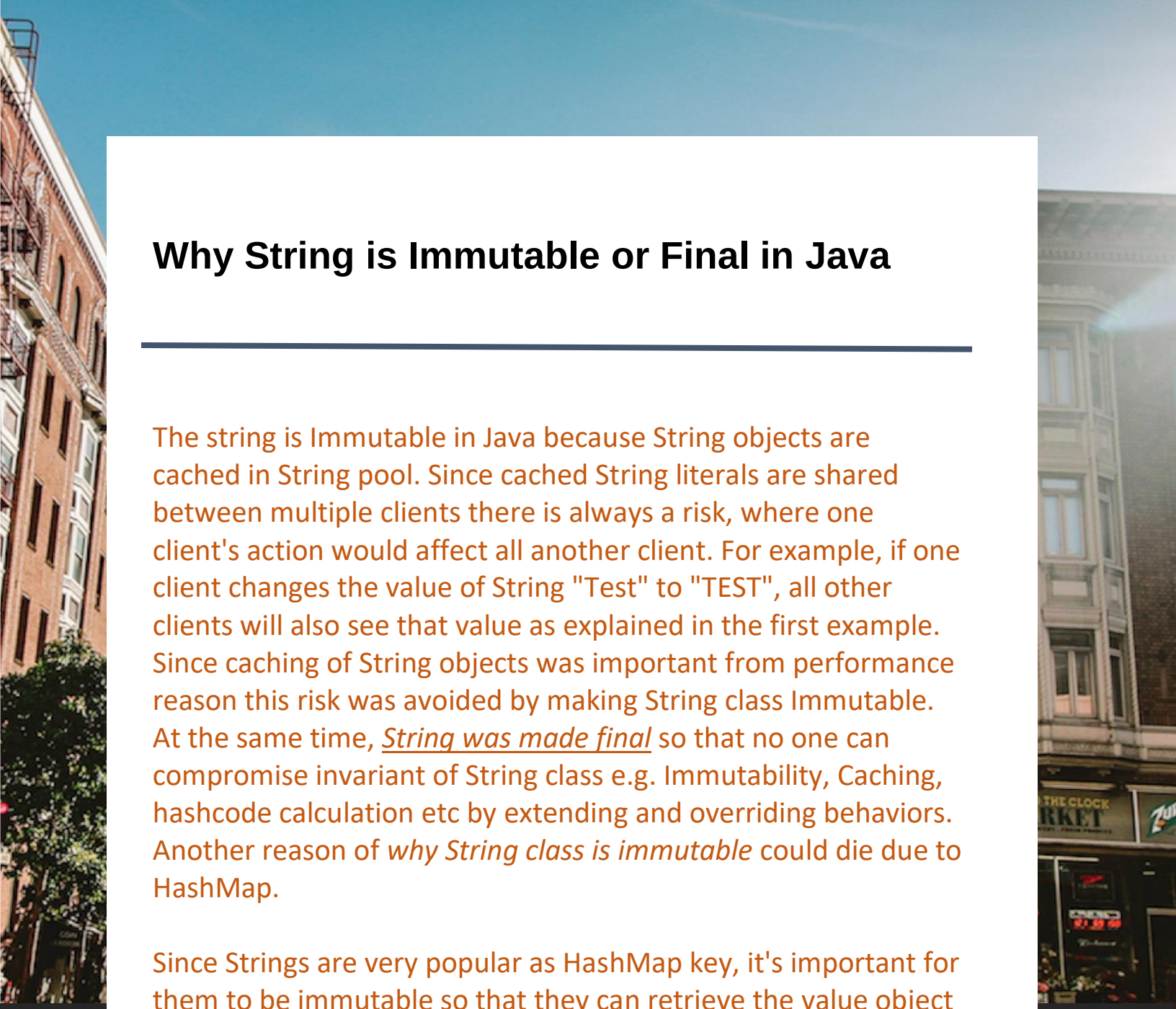
What is the use of interning the string?

To Save The memory Space :

Using interned string, you can save the memory space. If you are using lots of string objects with same content in your code, than it is better to create an intern of that string in the pool. Use that intern string whenever you need it instead of creating a new object in the heap. It saves the memory space.

For Faster Comparison :

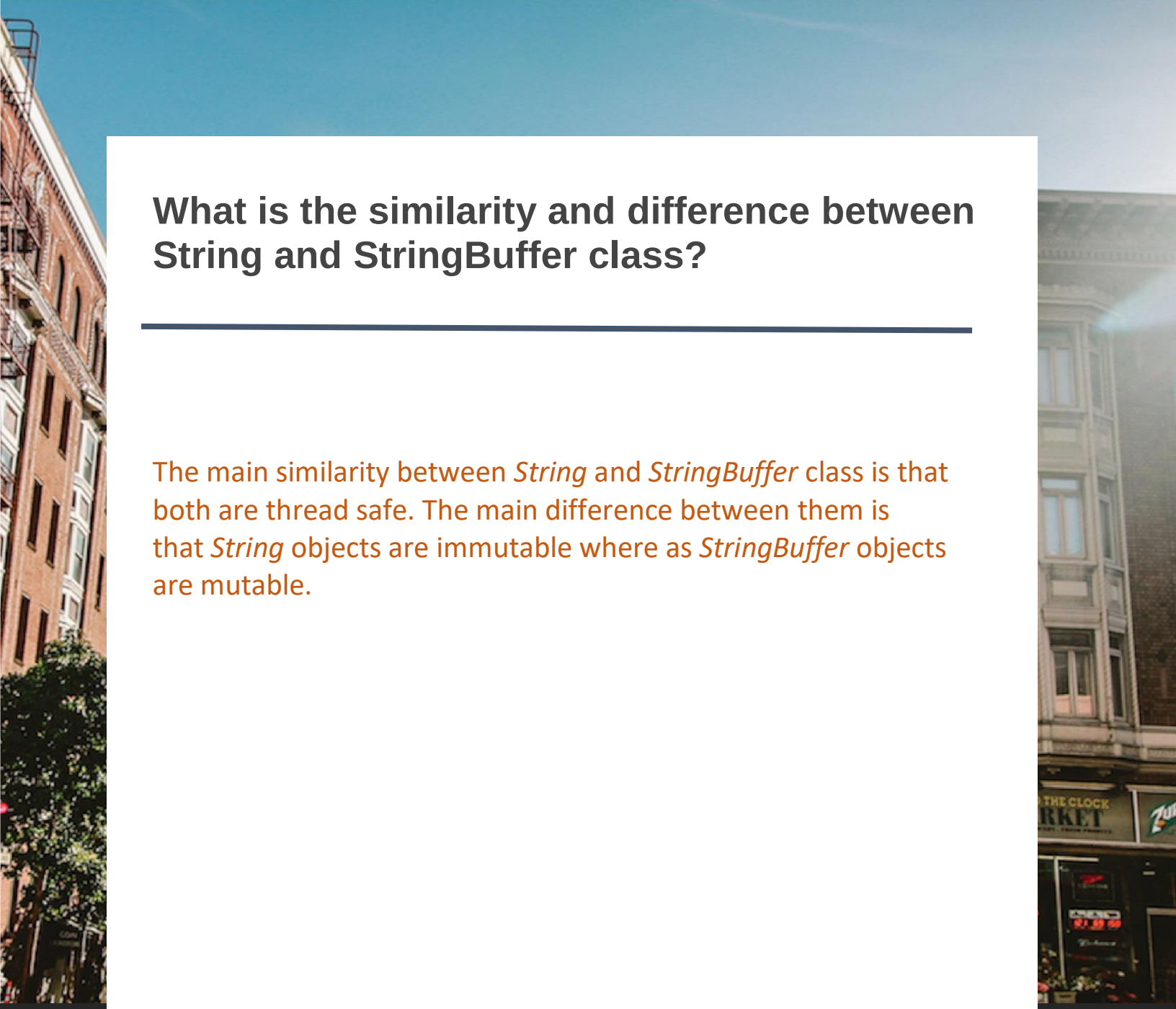
Assume that there are two string objects s1 and s2 in heap memory and you need to perform comparison of these two objects more often in your code. Then using s1.intern() == s2.intern() will be more fast then s1.equals(s2). Because, equals() method performs character by character comparison where as “==” operator just compares references of objects.



Why String is Immutable or Final in Java

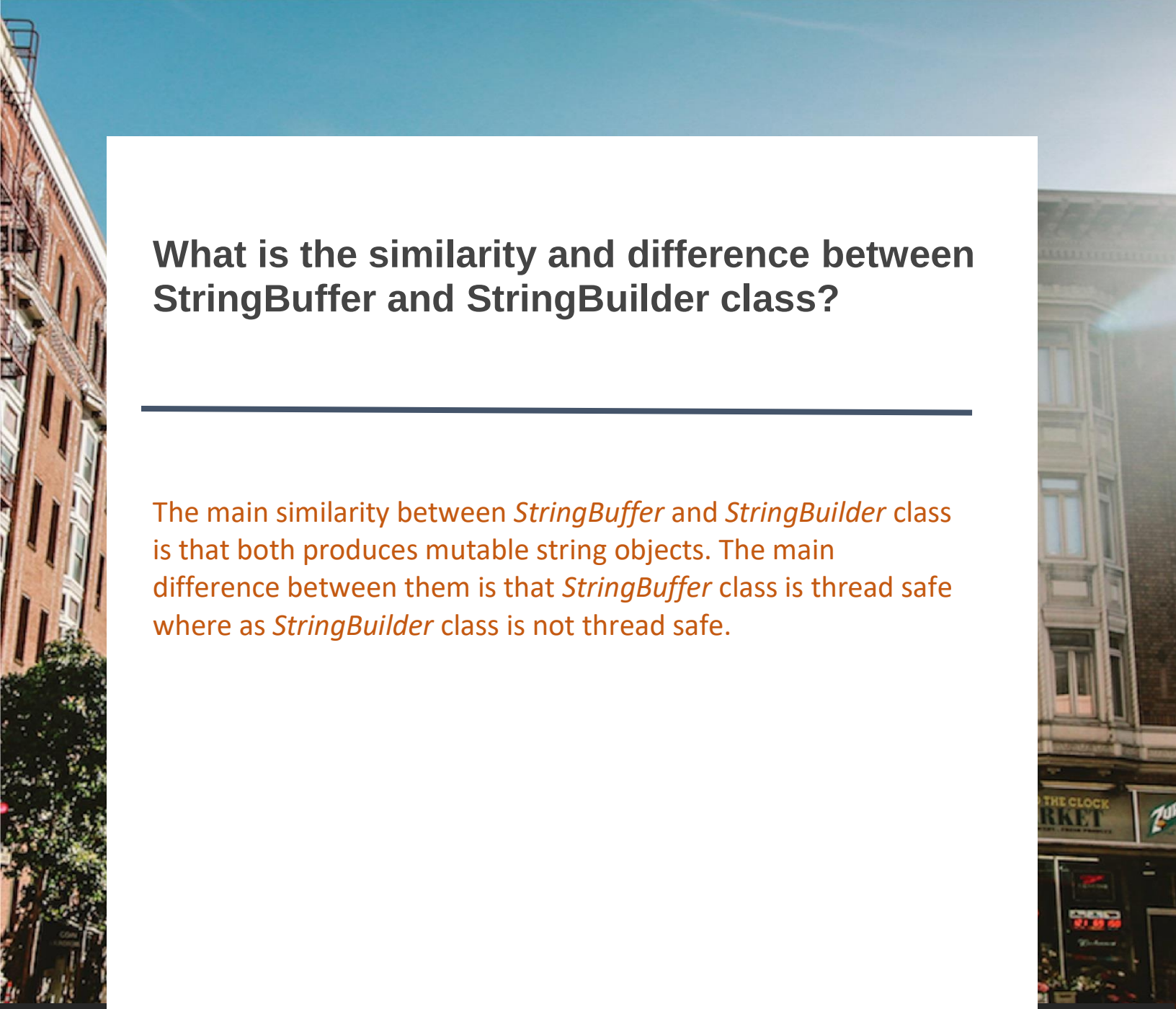
The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk, where one client's action would affect all another client. For example, if one client changes the value of String "Test" to "TEST", all other clients will also see that value as explained in the first example. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable. At the same time, *String was made final* so that no one can compromise invariant of String class e.g. Immutability, Caching, hashcode calculation etc by extending and overriding behaviors. Another reason of *why String class is immutable* could die due to HashMap.

Since Strings are very popular as HashMap key, it's important for them to be immutable so that they can retrieve the value object which was stored in HashMap. Since HashMap works in the principle of hashing, which requires same has value to function properly. Mutable String would produce two different hashcodes at the time of insertion and retrieval if contents of String was modified after insertion, potentially losing the value object in the map.



What is the similarity and difference between String and StringBuffer class?

The main similarity between *String* and *StringBuffer* class is that both are thread safe. The main difference between them is that *String* objects are immutable where as *StringBuffer* objects are mutable.



What is the similarity and difference between StringBuffer and StringBuilder class?

The main similarity between *StringBuffer* and *StringBuilder* class is that both produces mutable string objects. The main difference between them is that *StringBuffer* class is thread safe where as *StringBuilder* class is not thread safe.

https://www.linkedin.com/in/prashant-kumar-76b786168/

https://github.com/prashantt17

@prashantkumar