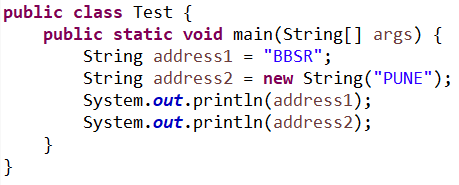
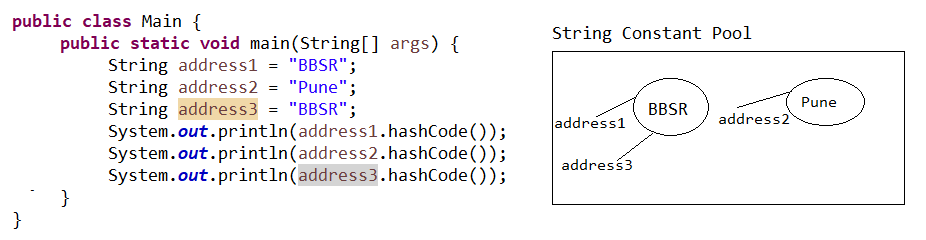
**String**

* String is used to represent group of characters enclosed with in the double quotes.



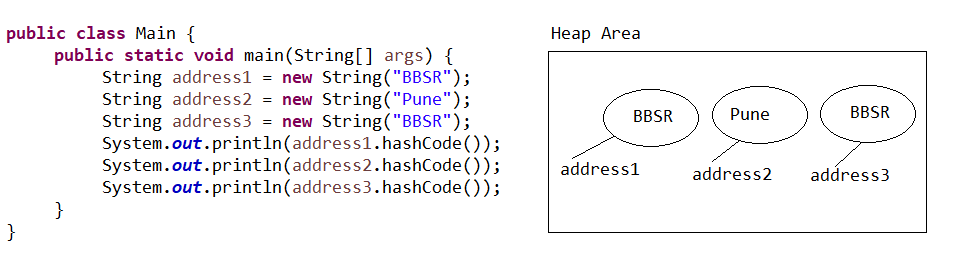
**Creating a string object without using new operator**

* When we create String object without using new operator the objects are created in SCP (String constant pool) area.
* When we create object in SCP area then just before object creation it is always checking previous objects.
  + If the previous object is available with the same content then it won’t create new object that reference variable pointing to existing object.
  + If the previous objects are not available then JVM will create new object.



**Creating a string object by using new operator**

* Whenever we are creating String object by using new operator the object created in heap area.
* When we create object in Heap area instead of checking previous objects it directly creates objects.

****

**== operator**

* It is comparing reference type. If two reference variables are pointing to same object then it returns true otherwise false.

**Example#1**

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String address1 = "BBSR";

String address2 = "Pune";

String address3 = "BBSR";

System.***out***.println(address1 == address2);

System.***out***.println(address1 == address3);

}

}

**Example#2**

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String address1 = **new** String("BBSR");

String address2 = **new** String("Pune");

String address3 = **new** String("BBSR");

System.***out***.println(address1 == address2);

System.***out***.println(address1 == address3);

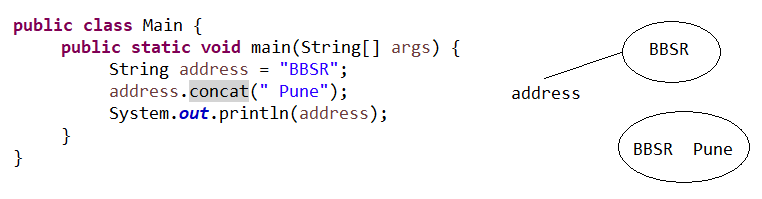
}

}

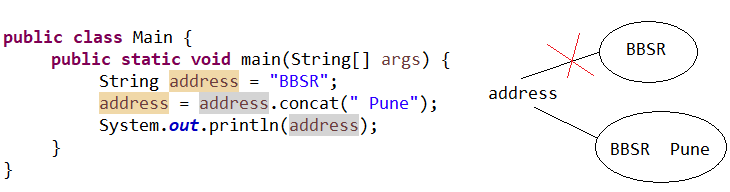
**String Immutability**

* String is immutable, it means once we are creating String objects it is not possible to perform modifications on existing object.

**Example#1**

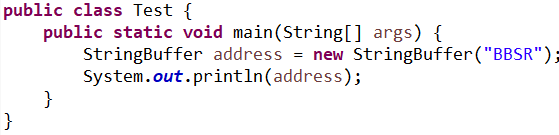


**Example#2**



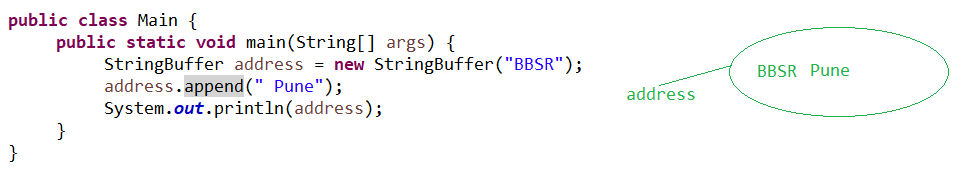
**StringBuffer**

* StringBuffer is used to represents group of character like String.
* We are able to create StringBuffer object by using new operator.



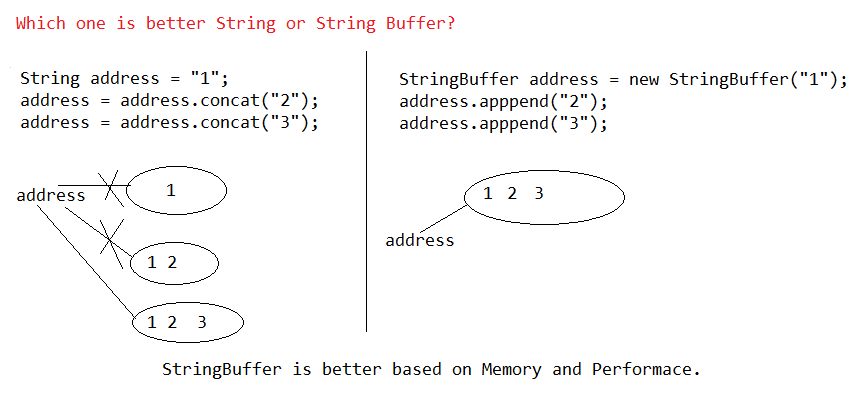
**StringBuffer Mutability**

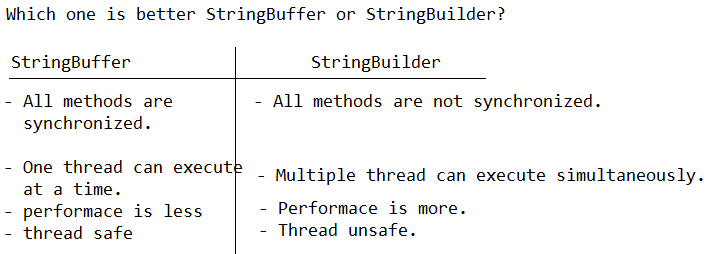
* StringBuffer is mutable, it means once we are creating StringBuffer objects on that existing object it is possible to perform modification.



**StringBuilder**

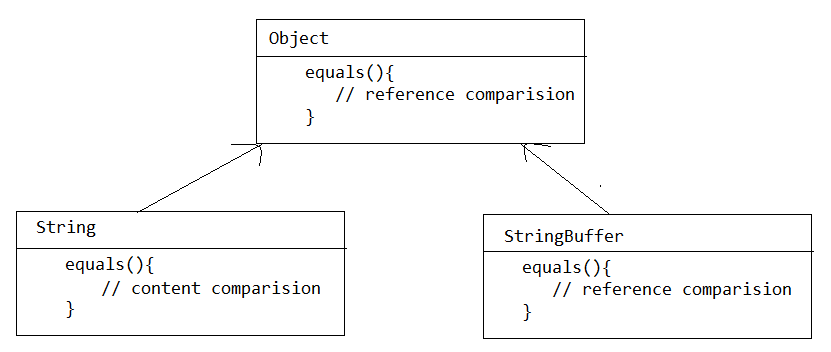
* StringBuilder is identical to StringBuffer except one important difference.
* *Every method present in the StringBuffer is Synchronized but method in StringBuilder is not Synchronized.*
* Multiple threads are allow to operate on StringBuilder methods hence the performance of the application is increased.
* Not Synchronized means not thread safe.

****

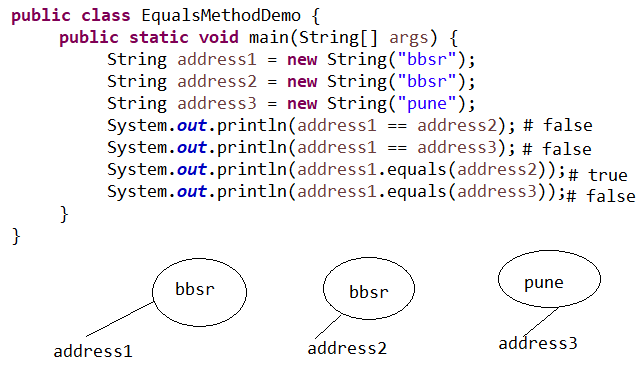
****

**Internal Implementation of equals method**

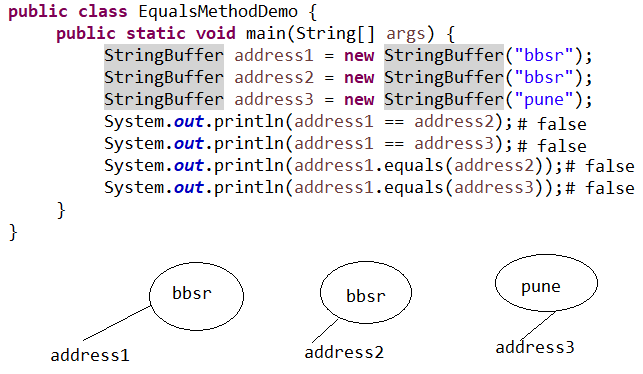
* equals() method present in object used for reference comparison.
* String is child class of object and it is overriding equals( ) methods used for content comparison.
* StringBuffer class is child class of object and it is not overriding equals() method hence it is using parent class equals() method.

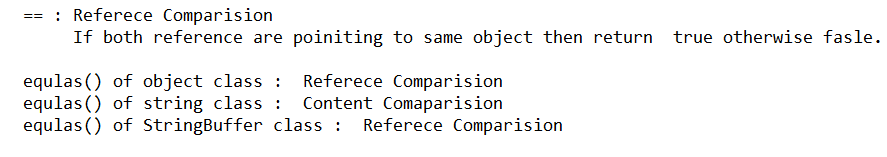


**Example#1**



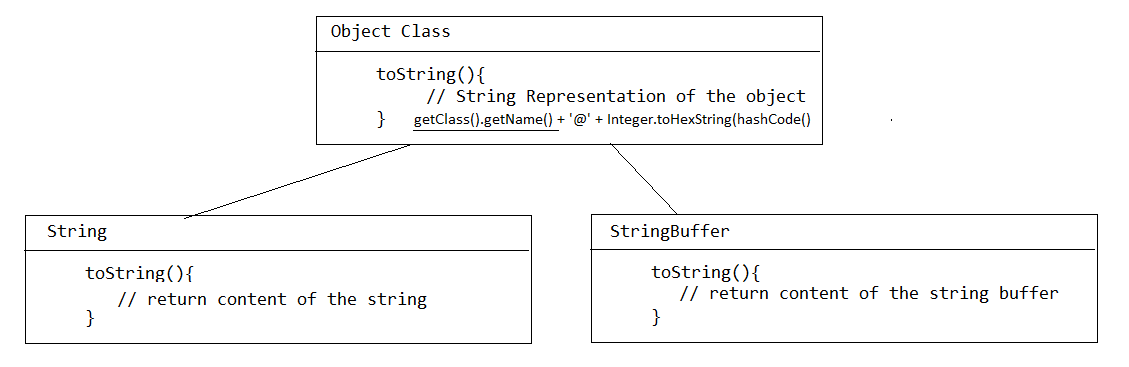
**Example#2**





**Internal implementation toString () method**

* toString( ) method present in object class returns a string representation of the object.
* String is overriding toString() used to return content of the String.
* StringBuffer is overriding toString() used to return content of the StringBuffer.



**Note:**

* Whenever we are printing reference variable internally it is calling toString() method.

class Object

{

public String toString()

{

return getClass().getName() + '@' + Integer.toHexString(hashCode());

}

}

class String extends Object

{

public String toString()

{

return "content of String";

}

}

class StringBuffer extends Object

{

public String toString()

{

return "content of String";

}

}

**Example**

**class** Test{

**int** i = 10;

}

**public** **class** EqualsMethodDemo {

**public** **static** **void** main(String[] args) {

Test t = **new** Test();

String address1 = **new** String("bbsr");

String address2 = **new** String("pune");

System.***out***.println(t);

System.***out***.println(address1);

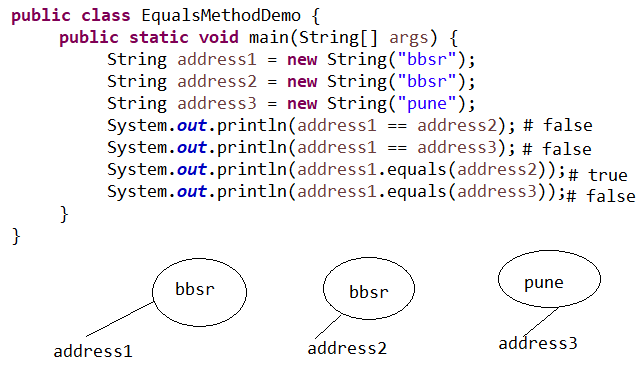
System.***out***.println(address2);

}

}

**== operator and equals() method**

* == operators for reference comparison and .equals() method for content comparison (e.g. for string).



**length variable vs. length() method**

* length variable used to find length of the Array.
* length() is method used to find length of the String.

**Example**

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**int** arr [] = {10,20,30,40,50};

String address = "bbsr";

System.***out***.println(arr.length);

System.***out***.println(address.length());

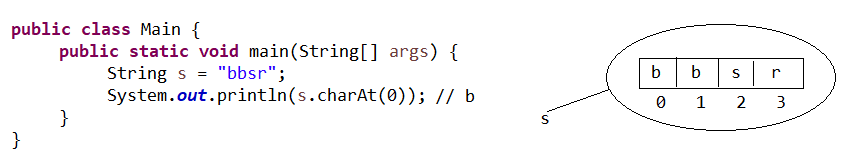
}

}

**charAt**

* It is used to extract the character from particular index position.

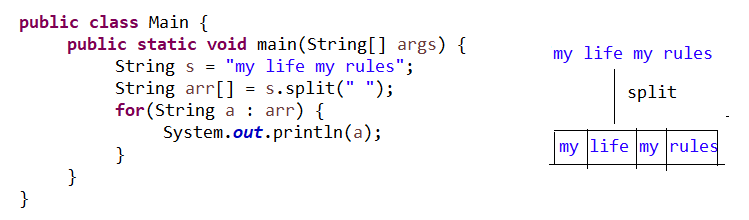
*char charAt(int)*



**split()**

* It is used to divide string into number of tokens.

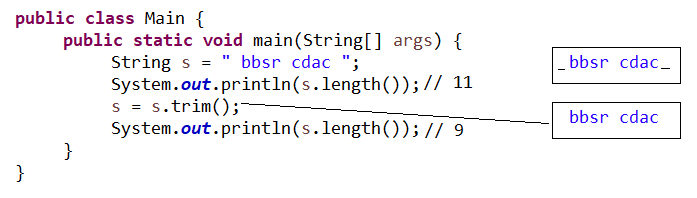
*String[] split(String)*



**trim()**

* It is used to remove the trail and leading spaces.

*String trim()*



**toUpperCase() and toLowerCase()**

**public** **class** Test {

**public** **static** **void** main(String[] args) {

String s1 = "BBSR";

String s2 = "bbsr";

System.***out***.println(s1.toLowerCase());

System.***out***.println(s2.toUpperCase());

}

}

**substring()**

* substring() used to find substring in main String.

String substring(int);

String substring(int, int)

* while printing substring() it includes starting index & it excludes ending index.

