Module - 2

String Handling

String Fundamentals

→ String is a class that represents sequence of characters

→ Strings are constants and their values cannot be changed after they are created. so it is called as immutable.

→ In java, when we create a string, it actually creates an object of type string.

→ String is the only class where operator overloading is supported in java.

→ we can concat two strings using "+" operator.

String constructors

We can construct the strings in 2 ways.

→ **String Literal**

You can create string objects with string literal.

```
String s1 = "Hello";
```

→ **Using new keyword**

This is common way to create a string object in java.

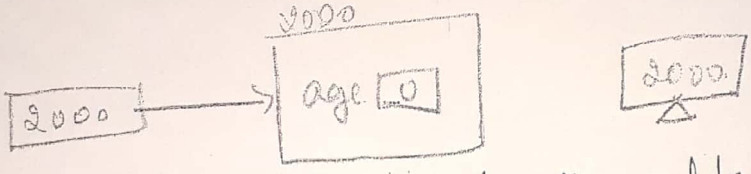
```
String s1 = new String("Sachin");
```

We have 10 Rules in string

Rule1

Eg:
```
class person
{
  int age;
}
Public class Firstpgm
{
  Public static void main (String[] args)
  {
    Person  p = new person();
    System.out.println(p);
  }
}
```

O/p: person@52e922



when ever we print the addressvariable of the user defined object we would not get the content of the objects rather we would get only the address of the object.

Eg: public class StringApp (string is inbuilt)
```
{
  Public static void main (String[] args)
  {
    String s = "Sachin"
    System.out.println(s);
  }
}
```

O/p: Sachin

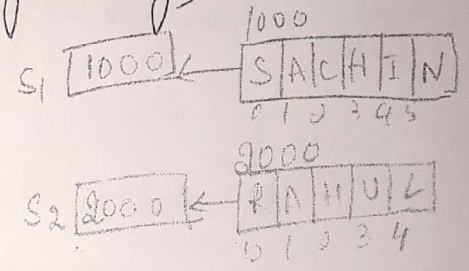
Rule1: On refering to string address variable we would get the data present in the string.

Rule2: Strings in java are refered as objects In order to compare 2 strings

1. With case sensitivity we use equals()
2. Without Case sensitivity we use equalsIgnoreCase()

Eg:

```
Public class StringRule2
{
  Public static void main(String[] args)
  {
    String S1 = "Sachin";
    String S2 = "Rahul";
    if (S1.equals(S2) == true)
    {
      System.out.println("Two strings are equal");
    }
    else
    {
      System.out.println("Two strings are not equal");
    }
  }
}
```



O/P: Two strings are not equal.

Eg 2:
```
public class StringRule22
{
Public static void main (String[] args)
{
String s1 = "sachin";
String s2 = "SACHIN";
if (s1.equalsIgnoreCase(s2) == true)
{
System.out.println("Two strings are equal");
}
else
{
System.out.println("Two strings are not equal");
}
}
}
```

O/P: Two strings are equal

Rule 3

In order to compare the address of the reference variable we use equal to operator (==).

Eg:
```
Public class StringRule3
{
Public static void main (String[] args)
{
String s1 = "sachin";
String s2 = "sachin";
if (s1 == s2)
{
System.out.println("Both s1 and s2 are pointing to the same object");
}
```

```
else
{
  System.out.println("s1 and s2 are not pointing
                      to the same object");
 }
}
```

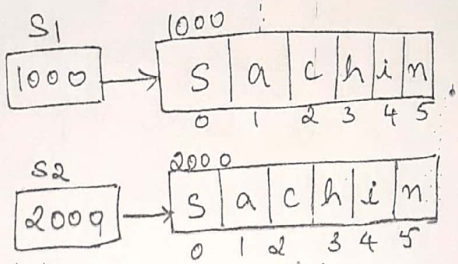
O/P: Both s_1 and s_2 are pointing to the same object.

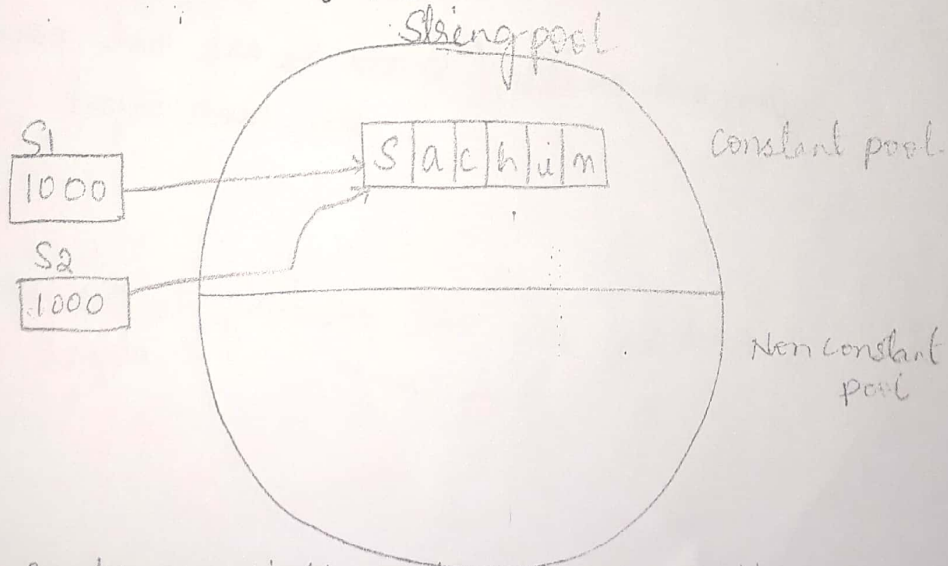
Rule 4:

if Strings are created without using new operator memory would come from constant pool.

Rule 5:

In constant pool duplicates are not allowed.





2 reference variable points to the same address

Rule 6:

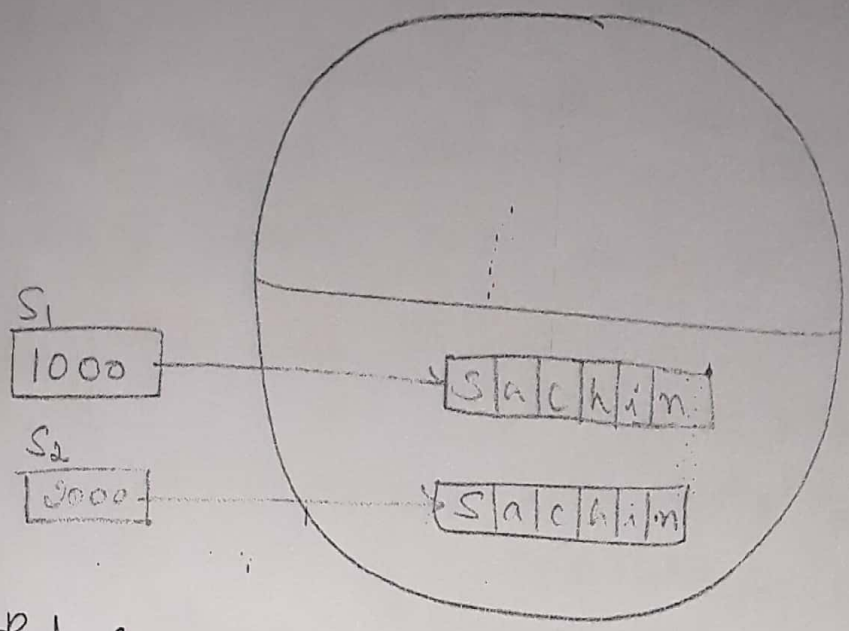
Whenever we create a string using new operator memory would come from non-constant pool.

Rule 7:

In non-constant pool duplicates are allowed.

Eg:
```
public class StringRule6
{
Public static void main(String[] args)
{
String s1 = new String("sachin");
String s2 = new String("sachin");
if (s1==s2)
{
System.out.println("s1 and s2 are pointing to
                    the same object");
}
else
{
System.out.println("s1 and s2 are not pointing
                    to the same object
}
}
```

O/P: s1 and s2 are not pointing to the same object.

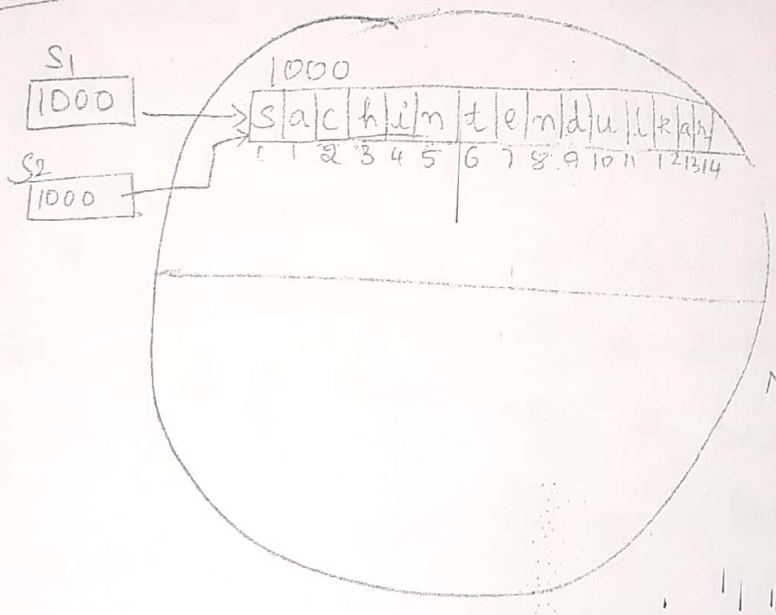


Rule 8:

The operator '+' is used for concatenation of 2 strings.

Rule 9:

During concatenation if values are used memory would come from constant pool.



Eg: public class StringRule
{
public static void main(String[] args)
{
String s_1;
String s_2;
s_1 = "sachin" + "tendulkar";
s_2 = "sachin" + "tendulkar";
System.out.println(s_1);
System.out.println(s_2);
if (s_1 == s_2)
{
System.out.println("s_1 and s_2 are pointing to the same object");
}
else
{
System.out.println("s_1 and s_2 are not pointing to the same object");
}
}
}

O/p: Sachintendulkar
Sachintendulkar
s_1 and s_1 are pointing to the same object.

Rule 10
Durin
menor
Eg:

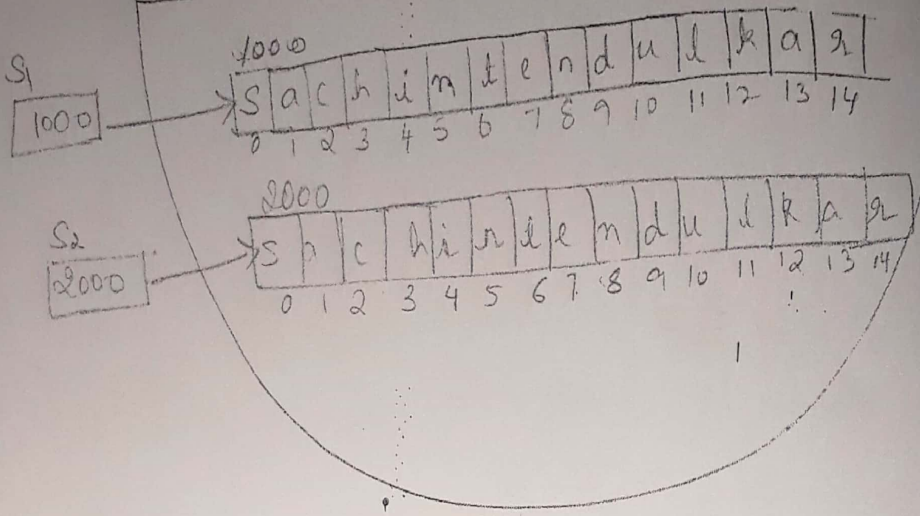
Rule 10

During concatenation if variables are used memory would come from non-constant pool.

Eg:

```
Public class StringRule10
{
  Public class void main (String[] args)
  {
    String S1 = "sachin";
    String S2 = "tendulkar";
    String S3 = S1 + S2;
    String S4 = S1 + S2;
    System.out.println (S3);
    System.out.println (S4);
    if (S3==S4)
    {
     System.out.println ("S3 and S4 are pointing
                          to the same object");
    }
    else
    {
    System.out.println ("S3 and S4 are not pointing
                         to the same object");
    }
  }
}
```

O/p: S3 and S4 are not pointing to the same object



Obtaining the characters within a string

By default strings are "immutable" in java. it means strings cannot be accessed at character level.

```
Public class ObtainingCharacter
{
  Public static void main(String[] args)
  {
    String S1 = "Sachin"
    System.out.println (S1[2])
  }
}
```

→ error

data cannot be accessed at character level because it is a string (immutable)

String comparision

Different methods used for string comparision are

1. equals() : it compares the strings to the specified object.
   if the strings are equal then the result is true otherwise false.
   Syntax: boolean equals();
2. equalIgnoreCase(): Compares the string to the specified object without checking the case
   Syntax: boolean equalIgnoreCase();
3. StartsWith() EndsWith(): Determines whether a given string begin with & end with a specified string.
   Syntax: boolean startsWith(String Str);
   boolean EndsWith(String Str)
4. CompareTo(): Compares a string to another string object.
   The result of the comparision is one of the following
   → Less than Zero: Invoking string is less than Str.
   → Greater than zero: Invoking string is greater than Str
   → Zero: two strings are equal.

Syntax: int compareTo (String str)

5. CompareToIgnoreCase(): Compares the string to the specified string object without checking the case.

Syntax: int compareToIgnoreCase (String str)

Eg: 3 way comparision

* write a java pgm to compare the string in 3 ways.

1. if two strings are equal print as it is equal.
2. if first string is greater than second string print $S_1 > S_2$
3. if first string is less than second string print $S_1 < S_2$.

In java compareTo method is used compare the strings in 3 ways, it would return the result in the following manner.

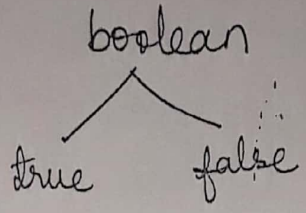
1. if two strings are equal it would return 0.
2. if first string is greater than second string it would return a positive number.
3. if first string is less than second string it would return a negative number.

Eg1 S_1 = rama
S_2 = rama

Eg2 S_1 = rama
S_2 = ramie

Eg3 S_1 = yuvi
S_2 = yuva

if we compare 2 strings then we would get the result in



In C we call it as ASCII values. But in java it is unicode.

a = 97	f = 102	k = 107	P = 112	u = 117	z = 122
b = 98	g = 103	l = 108	Q = 113	V = 118	
C = 99	h = 104	m = 109	R = 114	W = 119	
d = 100	i = 105	n = 110	S = 115	X = 120	
e = 101	j = 106	O = 111	t = 116	Y = 121	

Here in java to compare 2 strings we use a method called compareTo

Now, (S_1)compareTo(S_2)

↓ JVM

1) S_1 = rama 114 97 109 97

S_2 = rama (-) 114 97 109 97

0 0 0 0

if it returns zero (0) "2 strings are equal"

2) S_1 = rama 114 97 109 97

S_2 = rame 114 97 109 101

0 0 0 -4

if it returns a value < 0 "1st string is less than 2nd string

3) S_1 = yuvi 121 117 118 105
S_2 = yuva (-) 121 117 118 97
0 0 0 8

if it return a value >0 "1st string is greater than 2nd string"

Eg: Public class CompareToApp
{
Public static void main (String[] args)
{
String S_1 = "yuvi"
String S_2 = "yuva"
int m = S_1.compareTo(S_2);
System.out.println(m);
if (m!=0)
{
System.out.println("Two strings are equal");
}
else if (m>0)
{
System.out.println ("$S_1 > S_2$");
}
else
{
System.out.println("$S_1 < S_2$");
}
}
}

O/P: $S_1 > S_2$

Searching a substring using indexOf() and lastIndexOf()

Methods for searching string

indexOf()

→ Searches for the first occurance of the specified character or substring.

→ if the character or substring is not within the source string, then return -1.

→ if the character or substring is found, then return the position of search string in source string.

Syntax: int indexOf (char ch)
int indexOf (String str)

int indexOf (char c, int index)
int indexOf (String str, int index)

Eg:
```
String str = "Java is powerful";
str.indexOf('a');      // → 1
str.indexOf("is");     // → 5
str.indexOf('a', 7);   → -1
```

lastIndexOf()

→ Searches for the last occurence of the specified character or substring

→ if the character or substring is not within the string, then return -1

→ if the character or substring is found, then return the position of search string in source string.

Syntax: int lastIndexOf(char ch, int index)
int lastIndexOf(String str, int index)

int lastIndexOf(char ch)
int lastIndexOf(String str)

~~Eg: int lastIndexOf(ch~~

Eg: String str = "Java is powerful";
str.lastIndexOf('a', 7); // (3), -1

Changing the case of characters within a string

The string case can change using following methods.

1. toLowerCase():

Converts all of the character in string to lowercase

Syntax: String toLowerCase();

Eg: String str = "JAVA IS POWERFUL"
str.toLowerCase(); // java is powerful

2. toUpperCase():

Converts all of the characters in string to uppercase.

Syntax: String toUpperCase();

Eg: String str = "java is powerful";
str.toUpperCase(); // JAVA IS POWERFUL

Obtaining the characters

String class provides three ways in which character can be obtained from a string

methods for obtaining the characters

1. charAt()

returns a single character from a string at specified value.

Syntax: char charAt() (int index)

Eg: String str = "java is powerful";
char ch = str.charAt(3); // v

2. getChars()

Obtain more than one character

Syntax: void getchars(int srcstart, int endstart, char[] target, int targetstart)

where, reads the set of characters from index src.start to the index srcend and store in the target character array from a given index target start.

Eg: String str = "java is powerful";
str.getchars(5, 10, b, 0)
System.out.println(b); // is pow

toCharArray()

Converts this string to a new character array.

Syntax: char[] toCharArray()

Eg: String str = "java is powerful";
str.toCharArray();
// System.out.println(str[8]); // p

The length() method

→ The length() method is applicable for string objects but not for arrays

→ length() invokes a method to access a field member.

→ Strings does have a method called length() method, that returns the length of a string in integer type.

Syntax: int length()

Eg: String s1 = "Good morning";
System.out.println(s1.length()); // 12

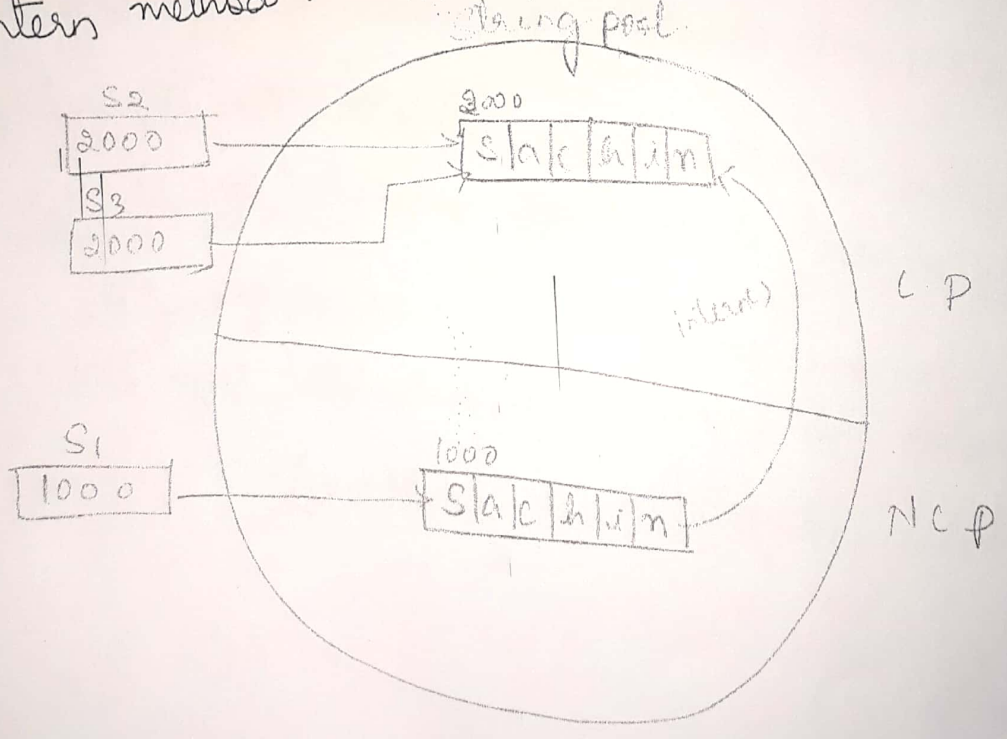
write a pgm on inbuilt method

```
Public class InbuiltMethod
{
  Public static void main (String[] args)
  {
    Staing S1 = "RajaRamMohanRoy";
    System.out.println (S1.length());
    System.out.println (S1.indexOf ('R'));
    System.out.println (S1.lastIndexOf ('R'));
    System.out.println (S1.contains ("Ram"));
    System.out.println (S1.contains ("Sachin"));
    System.out.println (S1.toUpperCase());
    System.out.println (S1.toLowerCase());
    System.out.println (S1.substring (7));
    System.out.println (S1.substring (4, 11));
    System.out.println (S1.endsWith ("Roy"));
    System.out.println (S1.endsWith ("Joy"));
    System.out.println (S1.startsWith ("Roy"));
    System.out.println (S1.startsWith ("Raja"));
    System.out.println (S1.charAt (10));
  }
}
```

output:

15
0
12
true
false
RAJARAMMOHANROY
rajarammohanroy
MohanRoy
RamMoha
true
false
false
true
a

*Write a java pgm to show the working of intern method.



Intern method() is used to copy the string from non-constant pool to constant pool.

```
Public class InternApp
{
  Public static void main(String[] args)
  {
    String s1 = new String("sachin");
    String s2 = s1.intern();
    String s3 = "sachin";
    if (s2 == s3)
    {
    System.out.println("s2 and s3 are pointing
                        to the same object");

    }
    else
     {
      System.out.println("s2 and s3 are not
                          pointing to the same object");
     }
  }
}
```

O/P: s2 and s3 are pointing the same object.

Creation of Mutable strings

In java by default strings are immutable To create mutable strings we use 2 approches

1. String Builder()
2. String Buffer()

String Buffer()

→ The stringBuffer class is used to represent characters that can be modified. we can use stringBuffer to append, reverse, replace, concatenate and manipulate strings or sequence of characters. it is used to create mutable strings.

Eg:

```
Public class StringBufferApp
{
  Public static void main(String[] args)
  {
    StringBuffer sb = new stringBuffer("sachin");
    System.out.println(sb)
    Sb.append("Tendulkar");
    System.out.println("After appending the
                        string is :"+sb);
  }
}
```

O/P Sachin
After appending the string is : SachinTendulkar

StringBuilder()

→ StringBuilder objects are like String objects, except that they can be modified.

→ Internally, these objects are treated like variable-length arrays that contain a sequence of characters.

→ At any point, the length and the content of the sequence can be changed through method invocations.

Length and capacity

The StringBuilder class, like the String class, has a length() method that returns the length of the character sequence in the builder.

Unlike strings, every StringBuilder also has a capacity, the number of character spaces that have been allocated.

The capacity, which is returned by the capacity() method, is always greater than or equal to length and will automatically expand as necessary to accomodate additions to the StringBuilder.

ite a java pgm on StringBuilder (Inbuilt methods)

```
(ublic class StringBuilderMethodApp
{
Public static void main(String[] args)
{
  StringBuilder sb = new StringBuilder();
  System.out.println(sb.capacity());
  System.out.println(sb.length());
  System.out.println();
  sb.append("Sachin");
  System.out.println(sb.capacity());
  System.out.println(sb.length());
  System.out.println();
  sb.append("is a great batsman");
  System.out.println(sb.capacity());
  System.out.println(sb.length());
  System.out.println();
  sb.append("He is from India");
  System.out.println(sb.capacity());
  System.out.println(sb.length());
}
}
```

o/p: 16 → 34
0 → 25
16 → 70
6 → 42