Java Programming

Arrays, Classes and Objects

Module 2

Agenda

1 Arrays

2 Classes and Objects

3 Static Block

4 String and StringBuffer

Objectives

At the end of this module, you will be able to :

- Understand Array declaration, definition and access of array elements
- Create classes and Objects
- Understand the importance of static block
- Implement String and StringBuffer class methods

Arrays

Arrays

- An array is a container object that holds a fixed number of values of a single type
- When an array is created, the length of an array is fixed
- Array elements are automatically initialized with the default value of their type, When an array is created
- Array can be created using the **new** keyword

Ex:

`int[] x =  new int[5];` // defining an integer array for 5 blocks

Arrays (Contd.).

- Alternatively, we can create and initialize array as below format

```
int[] x = {10, 20, 30};
int[] x = new int[]{10, 20, 30};
```

- Here the length of an array is determined by the number of values provided between *{* and *}*

- The built-in length property determines the size of any array

Ex:

```
int[] x = new int[10];
int x_len = x.length;
```

Array - Example

```
public class ArrayDemo {
    public static void main(String[] args) {
        int[] x; // declares an array of integers
        x = new int[5]; // allocates memory for 5integers
        x[0] = 11;
        X[4] = 22;
        System.out.println("Element at index 0: " + x[0]);
        System.out.println("Element at index 1: " + x[1]);
        System.out.println("Element at index 4: " + x[4]);
    }
}
```

Output:

```
Element at index 0: 11
Element at index 1: 0
Element at index 4: 22
```

Array Bounds, Array Resizing

- Array subscripts begin with 0
- Can't access an array element beyond the range
- Can't resize an array. Can use the same reference variable to refer new array

```
int x[] = new int [5];
x= new int [10];
```

Array copy

- To copy array elements from one array to another array, we can use arraycopy static method from System class

- Syntax:

  public static void arraycopy(Object s,int sIndex,Object d,int dIndex,int lngth)

- Ex:

```
int source[] = {1, 2, 3, 4, 5, 6};

int  dest[]  = new int[10];

System.arraycopy(source,0, dest,0,source.length);
```

Array Copy - Example

```
public class ArrayLengthDemo {
    public static void main(String[] args) {
        // creates and initializes an array of integers
        int[] source = {100, 200, 300};
        // creates an integer array with 3 element
        int[]  dest = new int[3];
        // copying an elements from source to dest array
        System.arrayCopy(source, 0, dest, 0, source.length);
        for (int i =0;  i <  dest.length;  i++)
        System.out.println("Element at index " + i + ": " +
          dest[i]);
    }
}
```

```
Output:
Element at index 0: 100
Element at index 1: 200
Element at index 3: 300
```

Two-Dimensional Arrays

- Two-dimensional arrays are arrays of arrays
- Initializing two-dimensional arrays:

```
int[][] y = new int[3][3];
```

The 1st dimension represent rows or number of one dimension, the 2nd dimension represent columns or number of elements in the each one dimensions

- The curly braces { } may also be used to initialize two dimensional arrays
- Ex:

```
int[][] y = { {1,2,3}, {4,5,6}, {7,8,9} };

int[][] y = new int[3][] { {1,2,3}, {4,5,6}, {7,8,9} };
```

Two-Dimensional Arrays (Contd.).

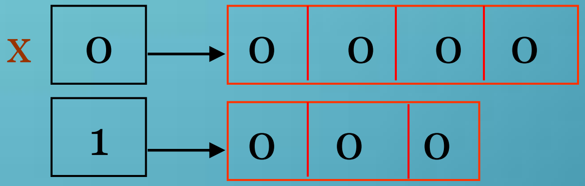
- You can initialize the row dimension without initializing the columns but not vice versa

```
int[][] x = new int[3][];

int[][] x = new int[][3];  //error
```

- The length of the columns can vary for each row and initialize number of columns in each row
- Ex1:

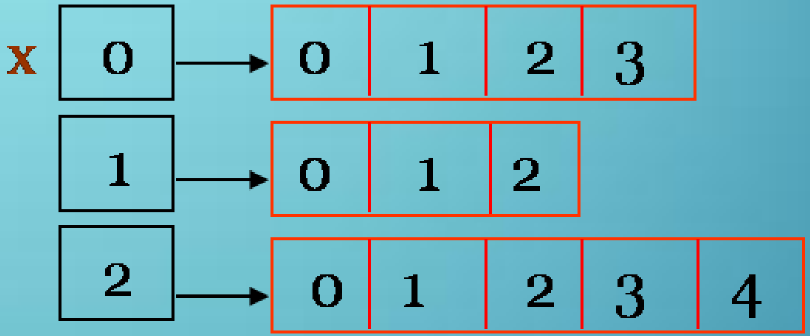
```
int [][]x = new int [2][];
x[0] = new int[5];
x[1] = new int [3];
```



Two-Dimensional Arrays (Contd.).

Ex2:

```
int [][]x = new int [3][];
x[0] = new int[]{0,1,2,3};
x[1] = new int []{0,1,2};
x[2] = new
int[]{0,1,2,3,4};
```



Two-Dimensional Array - Example

/* Program to under stand two-dimensional arrays */

```
class TwoDimDemo {
    public static void main(String[] args) {
     int [][] x = new int[3][]; // initialize number of rows
     /* define number of columns in each row */
     x[0] = new int[3];
         x[1] = new int[2];
         x[2] = new int[5];

  /* print array elements */
         for(int i=0;  i < x.length; i++) {
              for (int j=0;  j < x[i].length;  j++) {
                   x[i][j] = i;
                System.out.print(x[i][j]);
              }
              System.out.println();
         }
    }
}
```

<u>Output:</u>

```
000
11
22222
```

Quiz

- Select which of the following are valid array definition

1. int[] a;
   a = new int[5];
2. int a[] = new int[5]
3. int a[5] = new int[5];
4. int a[] = {1,2,3};
5. int[] a = new int[]{1,2,3};
6. int[] a = new int[5]{1,2,3,4};

Quiz (Contd.).

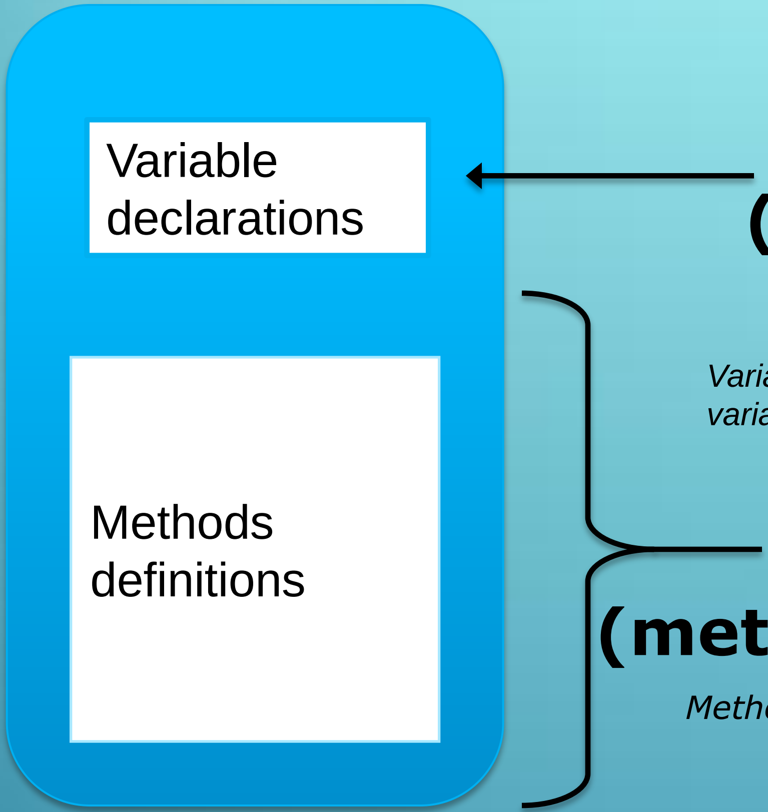
- Predict the output

```
class Sample{
   public static void main(String[] args){
   int[] a = new int[5]{1,2,3};
    for(int i : a)
      System.out.println(i);
    }
  }
```

Classes & Objects

Classes

A class contains variable declarations and method definitions



Variable declarations

(variable describes the attributes)

Variable may be: instance variables or static variables or final variables

Method definitions

(methods handle the behavior)

Methods may be: instance methods or static methods

Defining a Class in java

Define an Employee class with instance variables and instance methods

class Employee

Instance variables

- id
- name
- salary

Instance methods

- setId(…)
- setName(…)
- setSalary(…)
- getEmployeeDetails()

```
class Employee{
    int id;
    String name;
    int salary;

     void setId(int i) {
        id = i;
      }
    void setName(String n) {
        name = n;
      }
      void setSalary(int s) {
        salary = s;
      }
      void getEmployeeDetails( ) {
        System.out.println (name + " salary is
 " + salary);
      }
}
```

Basic information about a class

```
public class Account {
    double balance;
    public void deposit( double amount ){
        balance += amount;
    }
    public double withdraw( double amount ){
        int minimum_balance=5000;
        if (balance >= (amount+minimum_balance)){
                balance -= amount;
                return amount;
        }
        else {
                System.out.println(“Insufficient Balance”);
                return 0.0;
        }   }
   public double getbalance(){
                return balance;
   }  }
```

Instance Variable

Parameter or argument

local Variable

Basic information about a class (Contd.).

```
        else {
            System.out.println(“Insufficient Balance”);
            return 0.0;
        }
      }
      public double getbalance(){
              return balance;
      }
    }
```

Member variables

The previous slide contains definition of a class called Accounts.

A class contains members which can either be variables(fields) or methods(behaviors).

A variable declared within a class(outside any method) is known as an instance variable.

A variable declared within a method is known as local variable.

Variables with method declarations are known as parameters or arguments.

A class variable can also be declared as static where as a local variable cannot be static.

Objects and References

- Once a class is defined, you can declare a variable (object reference) of type class

```
Student stud1;
Employee emp1;
```

- The **new** operator is used to create an object of that reference type

```
Employee e = new Employee();
```

Object reference ↑ object ↑

Objects and References (Contd.).

- The **new** operator,

  Dynamically allocates memory for an object

  Creates the object on the heap

  Returns a reference to it

  The reference is then stored in the variable

Employee class - Example

```
class Employee{
int id;
String name;
int salary;
void setId(int no){
id = no;
}
void setName(String n){
name = n;
}
void setSalary(int s){
salary = s;
}
void getEmployeeDetails(){
System.out.println(name + " salary is "+ salary);
}
}
public class EmployeeDemo {
public static void main(String[] args) {
Employee emp1 = new Employee();
emp1.setId(101);
emp1.setName("John");
emp1.setSalary(12000);
emp1.getEmployeeDetails();
}
}
```

<u>Output:</u>

```
John salary is 12000
```

Constructors

- While designing a class, the class designer can define within the class, a special method called ‘constructor’
- Constructor is automatically invoked whenever an object of the class is created
- Rules to define a constructor
  - A constructor has the same name as the class name
  - A constructor should not have a return type
  - A constructor can be defined with any access specifier (like private, public)
  - A class can contain more than one constructor, So it can be overloaded

Constructor - Example

```java
class Sample{
private int id;
Sample(){
id = 101;
System.out.println("No Arg constructor, with ID: "+id);
}
Sample(int no){
id = no;
System.out.println("One argument constructor,with ID: "+ id);
}
}
public class ConstDemo {
public static void main(String[] args) {
Sample s1 = new Sample();
Sample s2 = new Sample(102);
}
}
```

Output:

```
No Arg constructor, with ID: 101
One argument constructor,with ID: 102
```

this reference keyword

- Each class member function contains an implicit reference of its class type, named **this**
- this reference is created automatically by the compiler
- It contains the address of the object through which the function is invoked
- Use of this keyword
  - this can be used to refer instance variables when there is a clash with local variables or method arguments
  - this can be used to call overloaded constructors from another constructor of the same class

this Reference (Contd.).

- Ex1:

```
void  setId (int id){
    this.id = id;
}
```

argument variable

instance variable

- Ex2:

```
class Sample{
Sample(){
this("Java"); // calls overloaded constructor
System.out.println("No Arg constructor ");
}
Sample(String str){
System.out.println("One argument constructor "+
  str);
}
}
```

Static Class Members

- Static class members are the members of a class that do not belong to an instance of a class
- We can access static members directly by prefixing the members with the class name

  ClassName.staticVariable

  ClassName.staticMethod(...)

Static variables:

- Shared among all objects of the class
- Only one copy exists for the entire class to use

Static Class Members (Contd.).

- Stored within the class code, separately from instance variables that describe an individual object
- Public static final variables are global constants

Static methods:

- Static methods can only access directly the static members and manipulate a class’s static variables
- Static methods cannot access non-static members(instance variables or instance methods) of the class
- Static method cant access this and super references

Static Class Members – Example

```
class StaticDemo
{
private static int a = 0;
private int b;
public void set ( int i, int j)
{
a = i; b = j;
}
public void show( )
{
System.out.println("This is static a: " + a );
System.out.println( "This is non-static b: " + b );
}
```

Static Class Members – Example (Contd.).

```
public static void main(String args[ ])
{
StaticDemo x = new StaticDemo( );
StaticDemo y = new StaticDemo( );
x.set(1, 1);
x.show( );
y.set(2, 2);
y.show( );
x.show( );
}
}
```

Output:

```
This is static a: 1
This is non-static b: 1

This is static a: 2
This is non-static b: 2

This is static a: 2
This is non-static b: 1
```

Discussion

Why is main() method static ?

Quiz

- What will be the result, if we try to compile and execute the following code as

  java Sample

```
class Sample{
  int i_val;
   public static void main(String[] xyz){
         System.out.println("i_val is
  :"+this.i_val);
   }
   }
```

Quiz

- What will be the result, if we try to compile and execute the following code as

  java Sample

```
class Sample{
 int i_val=10;
  Sample(int i_val){
        this.i_val=i_val;
        System.out.println("inside Sample 
 i_val: "+this.i_val);
   }
  public static void main(String[] xyz){
        Sample o = new Sample();}
 }
```

Static Block

The “static” block

- A static block is a block of code enclosed in braces, preceded by the keyword static

Ex :

```
static {
  System.out.println(“Within static
  block”);
}
```

- The statements within the static block are executed automatically when the class is loaded into JVM

The “static” block (Contd.).

- A class can have any number of static blocks and they can appear anywhere in the class
- They are executed in the order of their appearance in the class
- JVM combines all the static blocks in a class as single static block and executes them
- You can invoke static methods from the static block and they will be executed as and when the static block gets executed

Example on the “static” block

```
class StaticBlockExample {
  StaticBlockExample() {
    System.out.println("Within constructor");
  }
  static {
    System.out.println("Within 1st static block");
  }
  static void m1() {
    System.out.println("Within static m1 method");
  }
static {
    System.out.println("Within 2nd static block");
    m1();
  }
```

contd..

Example on the “static” block (Contd.).

```
  public static void main(String [] args) {
    System.out.println("Within main");
    StaticBlockExample x = new
  StaticBlockExample();
  }
  static {
    System.out.println("Within 3rd static
  block");
  }
}
```

Output:

```
Within 1st static block
Within 2nd static block
Within static m1 method
Within 3rd static block
Within main
Within constructor
```

Quiz

- What will be the result, if we try to compile and execute the following code as

  java Sample

```
class Sample{
  public static void main(String[] xyz){
       System.out.println("Inside main method line1");
  }
  static {
       System.out.println("Inside class line1");
  }
}
```

String and StringBuffer

String

- String is a group of characters. They are objects of type String.
- Once a String object is created it cannot be changed. Strings are Immutable.
- To get changeable strings use the class called StringBuffer.
- String and StringBuffer classes are declared as final, so there cannot be subclasses of these classes.
- The default constructor creates an empty string.

```
String s = new String();
```

Creating Strings

- To Create a String in JAVA is

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

If data array in the above example is modified after the string object str is created, then str remains unchanged.

Construct a string object by passing another string object.

```
String str2 = new String(str);
```

String class Methods

The length() method returns the length of the string.

Eg: System.out.println("Varun".length()); // prints 5

The + operator is used to concatenate two or more strings.

Eg: String myName = "Varun";

String s = "My name is" + myName+ ".";

For string concatenation the Java compiler converts an operand to a String whenever the other operand of the + is a String object.

String class Methods

- Characters in a string can be retrieved in a number of ways

public char **charAt**(int index)

- Method returns the character at the specified index. An index ranges from 0 to length() - 1

```
char c;
c = "abc".charAt(1); // c = “b”
```

String class Methods (Contd.).

equals() – Method This method is used to compare the invoking String to the object specified. It will return true, if the argument is not null and it is String object which contains the same sequence of characters as the invoking String.

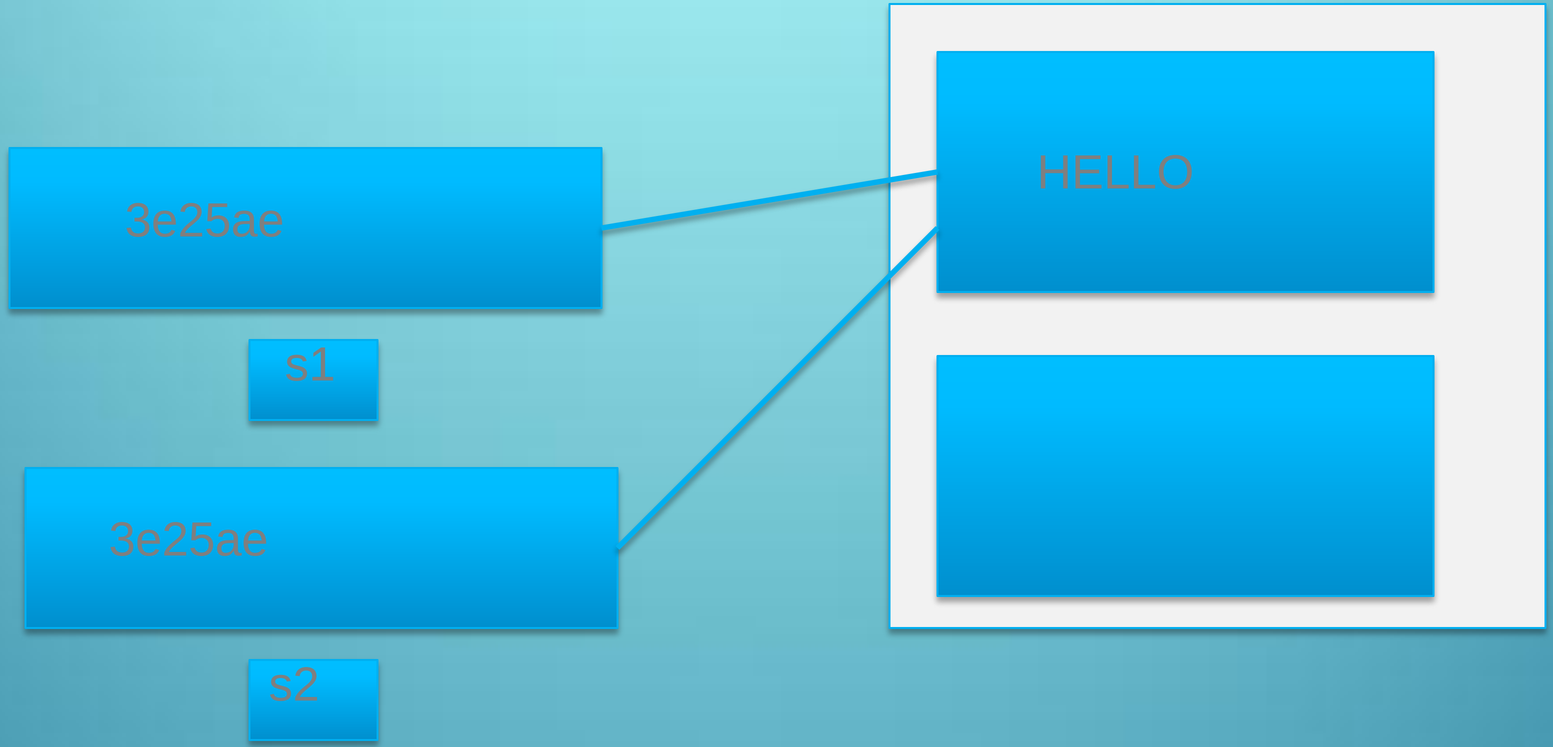
```
public boolean equals(Object anObject)
```

equalsIgnoreCase()- Method Compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

```
public boolean equalsIgnoreCase(String anotherString)
```

Strings created using assignment operator

- String s1 = “HELLO”;
- String s2 = “HELLO”;



Comparing Strings using == operator

What is the output ?

```
public class StringTest{
  public static void main(String[] args){
    String s1="Hello";
    String s2="Hello";
    if(s1==s2)
        System.out.println("String objects referenced
  are same");
    else
        System.out.println("String objects referenced
  are not same");
  }
}
```

Output: String objects referenced are same

Comparing Strings using equals method

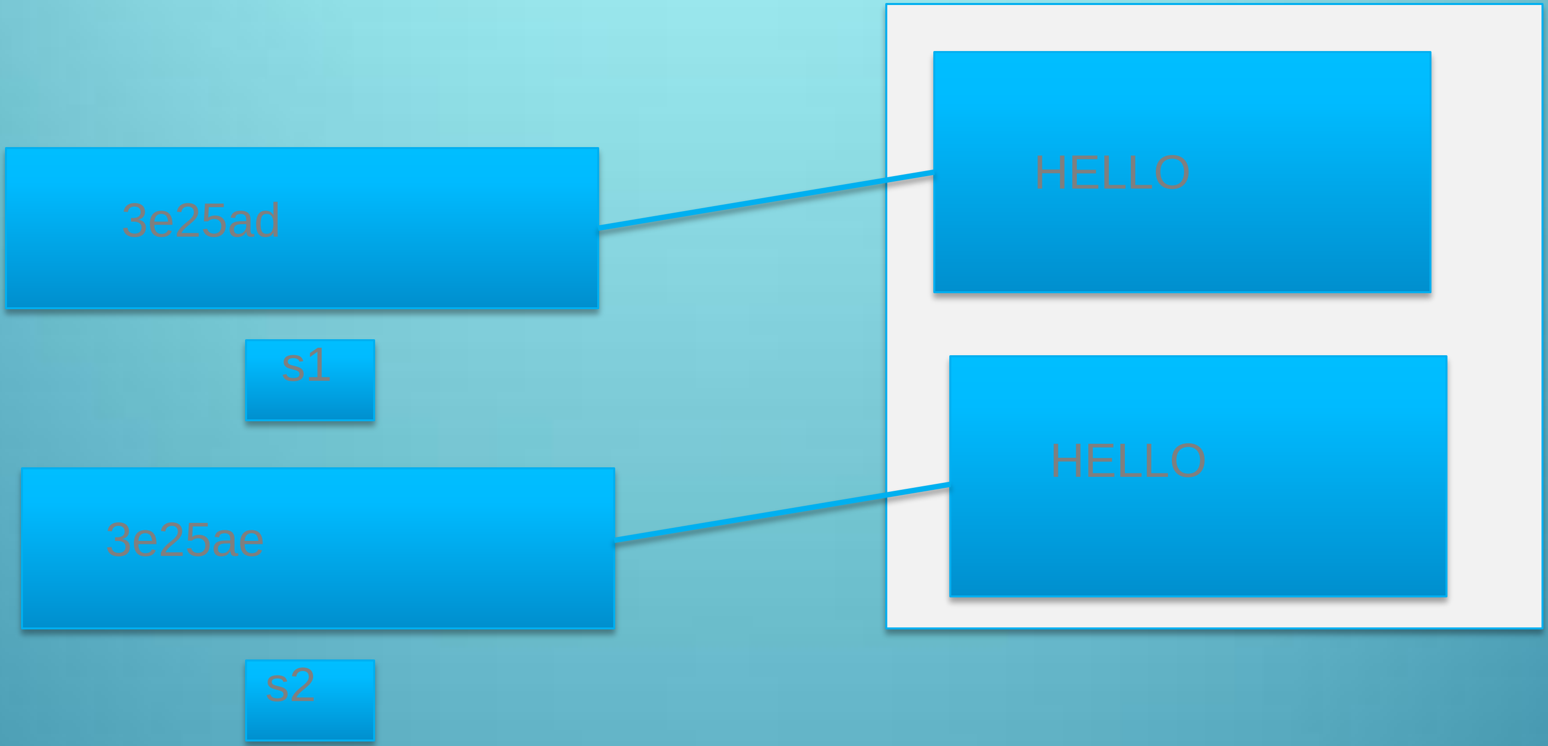
What is the output ?

```
public class StringTest{
  public static void main(String[] args){
    String s1="Hello";
    String s2="Hello";
    if(s1.equals(s2))
        System.out.println("Strings are equal");
    else
        System.out.println("Strings are not equal");
  }
}
```

Output: Strings are equal

Strings created using new keyword

- String s1 = new String(“HELLO”);
- String s2 = new String(“HELLO”);



Comparing Strings using == operator

What is the output ?

```
public class StringTest{
  public static void main(String[] args){
    String s1= new String("Hello");
    String s2= new String("Hello");
    if(s1==s2)
        System.out.println("String objects referenced
  are same ");
    else
        System.out.println("String objects referenced
  are not same");
  }
}
```

Output: String objects referenced are not same

Comparing Strings using equals method

What is the output ?

```
public class StringTest{
  public static void main(String[] args){
    String s1= new  String("Hello");
    String s2= new String("Hello");
    if(s1.equals(s2))
        System.out.println("Strings are equal");
    else
        System.out.println("Strings are not equal");
  }
}
```

Output: Strings are equal

String class Methods

startsWith() – Tests if this string starts with the specified prefix.

```
public boolean startsWith(String prefix)
"January".startsWith("Jan"); //
  true
```

endsWith() - Tests if this string ends with the specified suffix.

```
public boolean endsWith(String suffix)
"January".endsWith("ry"); // true
```

String class Methods (Contd.).

- **compareTo()** - Compares two strings and to know which string is bigger or smaller
  - We will get a negative integer, if this String object is less than the argument string
  - We will get a positive integer if this String object is greater than the argument string.
  - We will get a return value 0(zero), if these strings are equal.

public int **compareTo**(String anotherString)

public int **compareToIgnoreCase**(String str)

This method is similar to compareTo() method but this does not take the case of strings into consideration.

String class Methods (Contd.).

indexOf – Searches for the first occurrence of a character or substring. Returns -1 if the character does not occur

public int **indexOf**(int ch)- It searches for the character represented by ch within this string and returns the index of first occurrence of this character

- public int **indexOf**(String str) - It searches for the substring specified by str within this string and returns the index of first occurrence of this substring

String str = “How was your day today?”;
str.indexof(‘t’);
str(“was”);

String class Methods (Contd.).

`public int` **`indexOf`**`(int ch, int fromIndex)`- It searches for the character represented by ch within this string and returns the index of first occurrence of this character starting from the position specified by fromIndex

`public int` **`indexOf`**`(String str, int fromIndex)` - Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

String str = “How was your day today?”;
str.indexof(‘a’, 6);
str(“was”, 2);

String class Methods (Contd.).

lastIndexOf() –It searches for the last occurrence of a particular character or substring

substring() - This method returns a new string which is actually a substring of this string. It extracts characters starting from the specified index all the way till the end of the string

`public String` **`substring`**`(int beginIndex)`

Eg: `"unhappy".substring(2) returns "happy"`

String class Methods (Contd.).

- public String **substring**(int beginIndex, int endIndex)

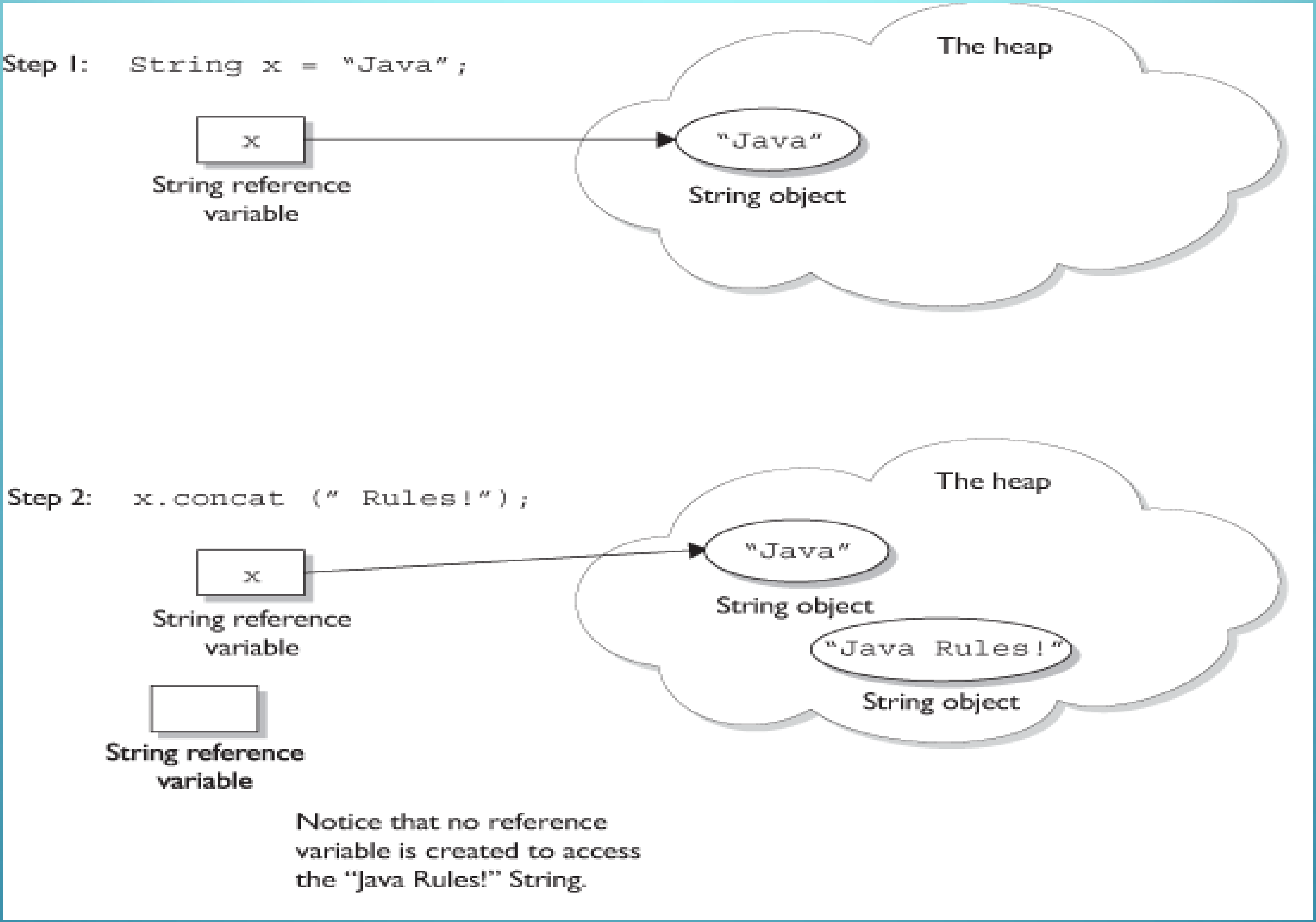
Eg: "smiles".substring(1, 5) returns "mile“

String class Methods (Contd.).

concat() - Concatenates the specified string to the end of this string

```
public String concat(String str)
"to".concat("get").concat("her") returns "together"
```

Example


Step 1: String x = "Java";
x
String reference variable
The heap
"Java"
String object
Step 2: x.concat (" Rules!");
x
String reference variable
The heap
"Java"
String object
"Java Rules!"
String object
String reference variable
Notice that no reference variable is created to access the "Java Rules!" String.

String class Methods (Contd.).

- replace()- Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar

```
public String replace(char oldChar,
 char newChar)

"wipra technalagies".replace('a',
 'o') returns "wipro technologies"
```

String class Methods (Contd.).

- **trim() -** Returns a copy of the string, with leading and trailing whitespace omitted

```
public String trim()
```

```
String s = “  Hi Mom!  “.trim();
S = “Hi Mom!”
```

- **valueOf() –** This method is used to convert a character array into String. The result is a String representation of argument passed as character array

```
public static String valueOf(char[] data)
```

String class Methods (Contd.).

Other forms are:

```
public static String valueOf(char c)

public static String valueOf(boolean b)

public static String valueOf(int i)

public static String valueOf(long l)

public static String valueOf(float f)

public static String valueOf(double d)
```

String class Methods (Contd.).

- **toLowerCase():** Method converts all of the characters in a String to lower case
- **toUpperCase():** Method converts all of the characters in a String to upper case

```
public String toLowerCase()
public String toUpperCase()

Eg: "HELLO WORLD".toLowerCase();
   "hello world".toUpperCase();
```

StringBuffer

- StringBuffer class objects are mutable, so they can be modified
- The length and content of the StringBuffer sequence can be changed through certain method calls
- StringBuffer class defines three constructors:
  - StringBuffer()//empty object
  - StringBuffer(int capacity)//creates an empty object with a capacity for storing a string
  - StringBuffer(String str)//create StringBuffer object by using a string

String Buffer Operations

- StringBuffer has two main operations methods – append and insert

- Both these methods are overloaded so that they can accept any type of data

Here are few append methods:

```
StringBuffer append(String str)
StringBuffer append(int num)
```

- As the name suggests, the append method adds the specified characters at the end of the StringBuffer object

StringBuffer Operations (Contd.).

- The insert methods are used to insert characters at the specified index location

Here are few insert methods:

```
StringBuffer insert(int index, String str)
StringBuffer append(int index, char ch)
```

- Index specifies at which point the string will be inserted into the invoking StringBuffer object

StringBuffer Operations (Contd.).

- **delete() -** This method is used to delete specified substring within the StringBuffer object

```
public StringBuffer
  delete(int start, int end)
```

StringBuffer Operations (Contd.).

replace() - This method is used to replace part of this StringBuffer(substring) with another substring

```
public StringBuffer replace(int start, int end,
  String str)
```

substring() - This method returns a new string which is actually a substring of this StringBuffer. It extracts characters starting from the specified index all the way till the end of the StringBuffer

```
public String substring(int start)
```

StringBuffer Operations (Contd.).

- **reverse() -** As the name suggests, the character sequence is reversed with this method

```
public StringBuffer reverse()
```

- **length() –** Used to find the length of the StringBuffer

```
public int length()
```

StringBuffer Operations (Contd.).

capacity() – We can find the capacity of the StringBuffer using this method

What is capacity ?

The capacity is the amount of storage available for the characters that have just been inserted

```
public int capacity()
```

charAt() - Used to find the character at a particular index position

```
public char charAt(int index)
```

Quiz

What is the output ?

```
class StringExample {
  public static void main(String[] args) {
    String st = new String("Wipro Technologies");
    StringBuffer sb = new StringBuffer("Wipro Technologies");
    String result1 = st.substring(6,12);
    String result2 = sb.substring(6);
    String result3 = sb.substring(6,12);
    System.out.println("Substring of String st : "+result1);
    System.out.println("Substring of StringBuffer sb (with
  single argument): "+result2);
    System.out.println("Substring of StringBuffer sb (with two
  arguments) : "+result3); }
}
```

```
Substring of String st : Techno
Substring of StringBuffer sb (with single argument): Technologies
Substring of StringBuffer sb (with two arguments) : Techno
```

Summary

In this module, we were able to:

- Understand Array declaration, definition and access of array elements
- Create classes and Objects
- Understand the importance of static block
- Implement String and StringBuffer class methods

References

1. Oracle (2012). Java Tutorials: Arrays. Retrieved on May 12, 2012, from, http://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

2. Oracle (2012). Java Tutorials: Declaring Classes. Retrieved on May 12, 2012, from, http://docs.oracle.com/javase/tutorial/java/javaOO/classdecl.html

3. Oracle (2012). Java Tutorials: The Numbers Classes. Retrieved on May 12, 2012, from, http://docs.oracle.com/javase/tutorial/java/data/numberclasses.html

Thank You