Module 2

Class, Object, Packages and Input/output

Java Classes/Objects

Java is an object-oriented programming language.

Everything in Java is associated with classes and objects, along with its attributes and methods.

For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword `class`:

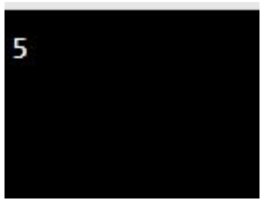
```
public class Main {
  int x = 5;
}
```

Create an Object

In Java, an object is created from a class. We have already created the class named `Main`, so now we can use this to create objects.

To create an object of `Main`, specify the class name, followed by the object name, and use the keyword `new`:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

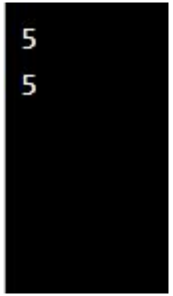


```
5
```

Multiple Objects

You can create multiple objects of one class:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();
    Main myObj2 = new Main();
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```



```
5
5
```

Java Class Attributes

In the previous slide we used the term "variable" for `x` in the example (as shown below). It is actually an attribute of the class. Or you could say that class attributes are variables within a class:

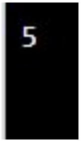
Create a class called "`Main`" with two attributes: `x` and `y`:

```
public class Main {
  int x = 5;
  int y = 3;
  }
```

Accessing Attributes

You can access attributes by creating an object of the class, and by using the dot syntax (`.`):The following example will create an object of the `Main` class, with the name `myObj`. We use the `x` attribute on the object to print its value:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```



```
5
```

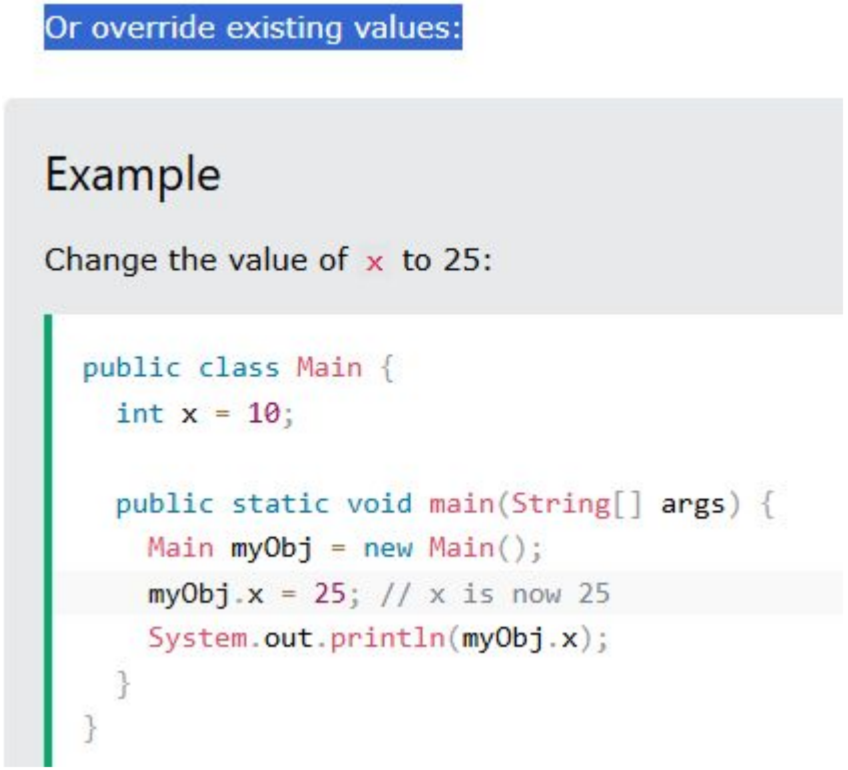
Modify Attributes

You can also modify attribute values:

Example

Set the value of x to 40:

```
public class Main {
  int x;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```



Or override existing values:

Example

Change the value of x to 25:

```
public class Main {
  int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // x is now 25
    System.out.println(myObj.x);
  }
}
```

If you don't want the ability to override existing values, declare the attribute as `final`:

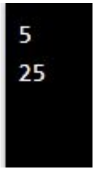
```
public class Main {
  final int x = 10;


  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // will generate an error
    System.out.println(myObj.x);
  }
}
```

IT will generate error

The `final` keyword is useful when you want a variable to always store the same value,

Multiple Objects

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
 }
```



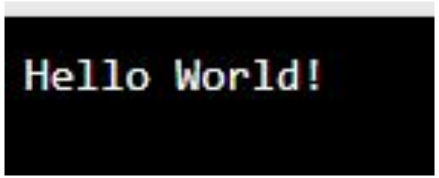
```
5
25
```

Java Class Methods

Create a method named `myMethod()` in Main:

`myMethod()` prints a text (the action), when it is called. To call a method, write the method's name followed by two parentheses () and a semicolon;

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}
```



```
Hello World!
```

Static members and functions

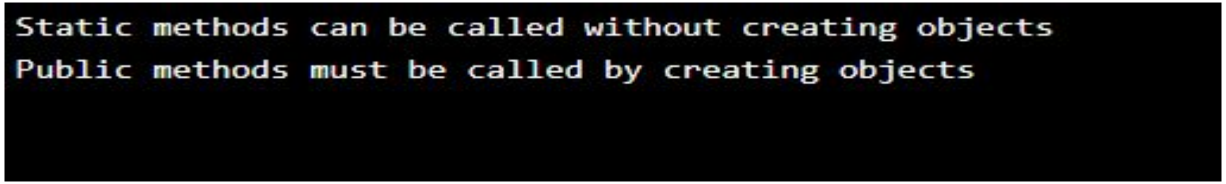
- Static members are those which belongs to the class and you can access these members without instantiating the class.

- The static keyword can be used with methods, fields, classes (inner/nested), blocks.

- **Static Methods** − You can create a static method by using the keyword static. Static methods can access only static fields, methods. To access static methods there is no need to instantiate the class, you can do it just using the class name.

You will often see Java programs that have either `static` or `public` attributes and methods. In the example above, we created a `static` method, which means that it can be accessed without creating an object of the class, unlike `public`, which can only be accessed by objects:

```
public class Main {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method

    Main myObj = new Main(); // Create an object of MyClass
    myObj.myPublicMethod(); // Call the public method
  }
}
```



```
Static methods can be called without creating objects
Public methods must be called by creating objects
```

Constructors

- Constructor is a block of codes similar to the method.
- It is called when an instance of the class is created.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.

Characteristics of Constructors:

- **Same Name as the Class:** A constructor has the same name as the class in which it is defined.
- **No Return Type:** Constructors do not have any return type, not even void. The main purpose of a constructor is to initialize the object, not to return a value.
- **Automatically Called on Object Creation:** When an object of a class is created, the constructor is called automatically to initialize the object's attributes.
- **Used to Set Initial Values for Object Attributes:** Constructors are primarily used to set the initial state or values of an object's attributes when it is created.

Create a constructor:

```
// Create a Main class
public class Main {
  int x;  // Create a class attribute

  // Create a class constructor for the Main class
  public Main() {
    x = 5;  // Set the initial value for the class attribute x
  }

  public static void main(String[] args) {
    Main myObj = new Main(); // Create an object of class Main
(This will call the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}

 // Outputs 5
```

The below table demonstrates the key difference between Java Constructor and Java Methods.

Features	Constructor	Method
Name	Constructors must have the same name as the class name	Methods can have any valid name
Return Type	Constructors do not return any type	Methods have the return type or void if does not return any value.
Invocation	Constructors are called automatically with new keyword	Methods are called explicitly
Purpose	Constructors are used to initialize objects	Methods are used to perform operations

Types of constructors

1. Default constructor (no-arg constructor)

2. Parameterized constructor

1. Default constructor

- **A constructor that has no parameters is known as default constructor.**

- **Implicit Default Constructor:** If no constructor is defined in a class, the Java compiler automatically provides a default constructor. This constructor doesn’t take any parameters and initializes the object with default values, such as `0` for numbers, `null` for objects.
- **Explicit Default Constructor:** If we define a constructor that takes no parameters, it's called an explicit default constructor. This constructor replaces the one the compiler would normally create automatically. Once you define any constructor (with or without parameters), the compiler no longer provides the default constructor for you.

```
// Java Program to demonstrate
// Default Constructor


// Driver class
class Cars{

    // Default Constructor
    Cars() {
        System.out.println("Default constructor");

    }

    // Driver function
    public static void main(String[] args)
    {
        Cars c1= new Cars();
    }
}
```

2. Parameterized Constructor in Java

A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

Copy Constructor in Java

Unlike other constructors copy constructor is passed with another object which copies the data available from the passed object to the newly created object.

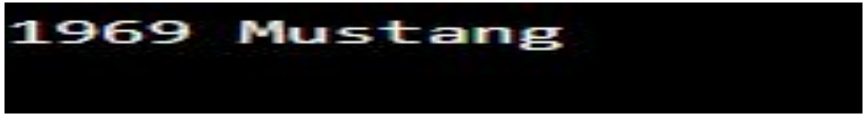
- There are many ways to copy the values of one object into another in Java.

- They are:
  - By constructor
  - By assigning the values of one object into another
  - By clone() method of Object class

Constructor Parameters

Constructors can also take parameters, which is used to initialize attributes.

```
public class Main {
  int modelYear;
  String modelName;

  public Main(int year, String name) {
    modelYear = year;
    modelName = name;
  }

  public static void main(String[] args) {
    Main myCar = new Main(1969, "Mustang");
    System.out.println(myCar.modelYear + " " + myCar.modelName);
  }
}
```



```
1969 Mustang
```

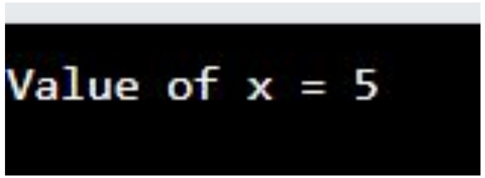
Java this Keyword

The `this` keyword in Java refers to the current object in a method or constructor.

The `this` keyword is often used to avoid confusion when class attributes have the same name as method or constructor parameters.

Accessing Class Attributes

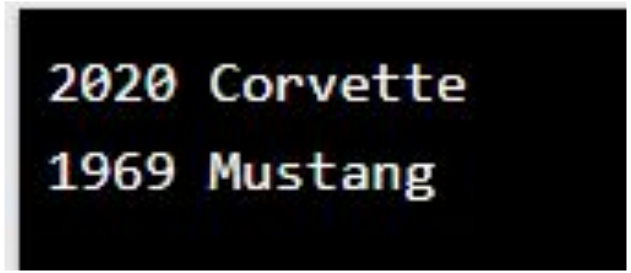
```
public class Main {
  int x;

  public Main(int x) {
    this.x = x; // refers to the class variable x
  }

  public static void main(String[] args) {
    Main myObj = new Main(5);
    System.out.println("Value of x = " + myObj.x);
  }
}
```



```
Value of x = 5
```

Calling a Constructor from Another Constructor

```
public class Main {
  int modelYear;
  String modelName;

  // Constructor with one parameter
  public Main(String modelName) {
    // Call the two-parameter constructor to reuse code and set a default year
    this(2020, modelName);
  }

  // Constructor with two parameters
  public Main(int modelYear, String modelName) {
    // Use 'this' to assign values to the class variables
    this.modelYear = modelYear;
    this.modelName = modelName;
  }

  // Method to print car information
  public void printInfo() {
    System.out.println(modelYear + " " + modelName);
  }

  public static void main(String[] args) {
    // Create a car with only model name (uses default year)
    Main car1 = new Main("Corvette");

    // Create a car with both model year and name
    Main car2 = new Main(1969, "Mustang");

    car1.printInfo();
    car2.printInfo();
  }
}
```



```
2020 Corvette
1969 Mustang
```

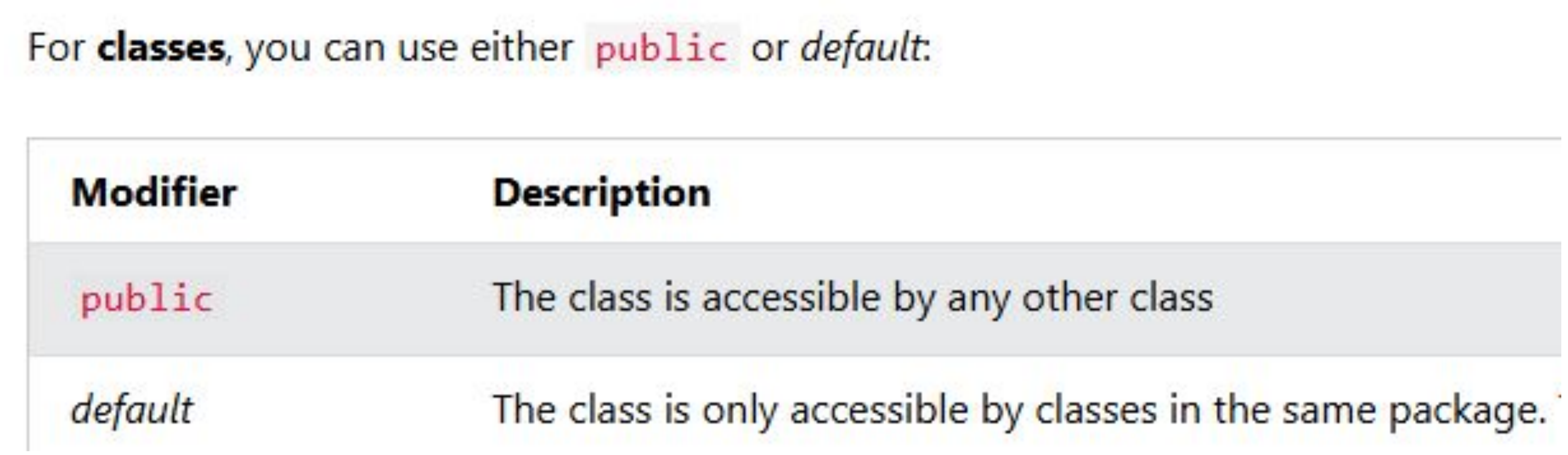
Java Modifiers

The `public` keyword is an access modifier, meaning that it is used to set the access level for classes, attributes, methods and constructors.

We divide modifiers into two groups:

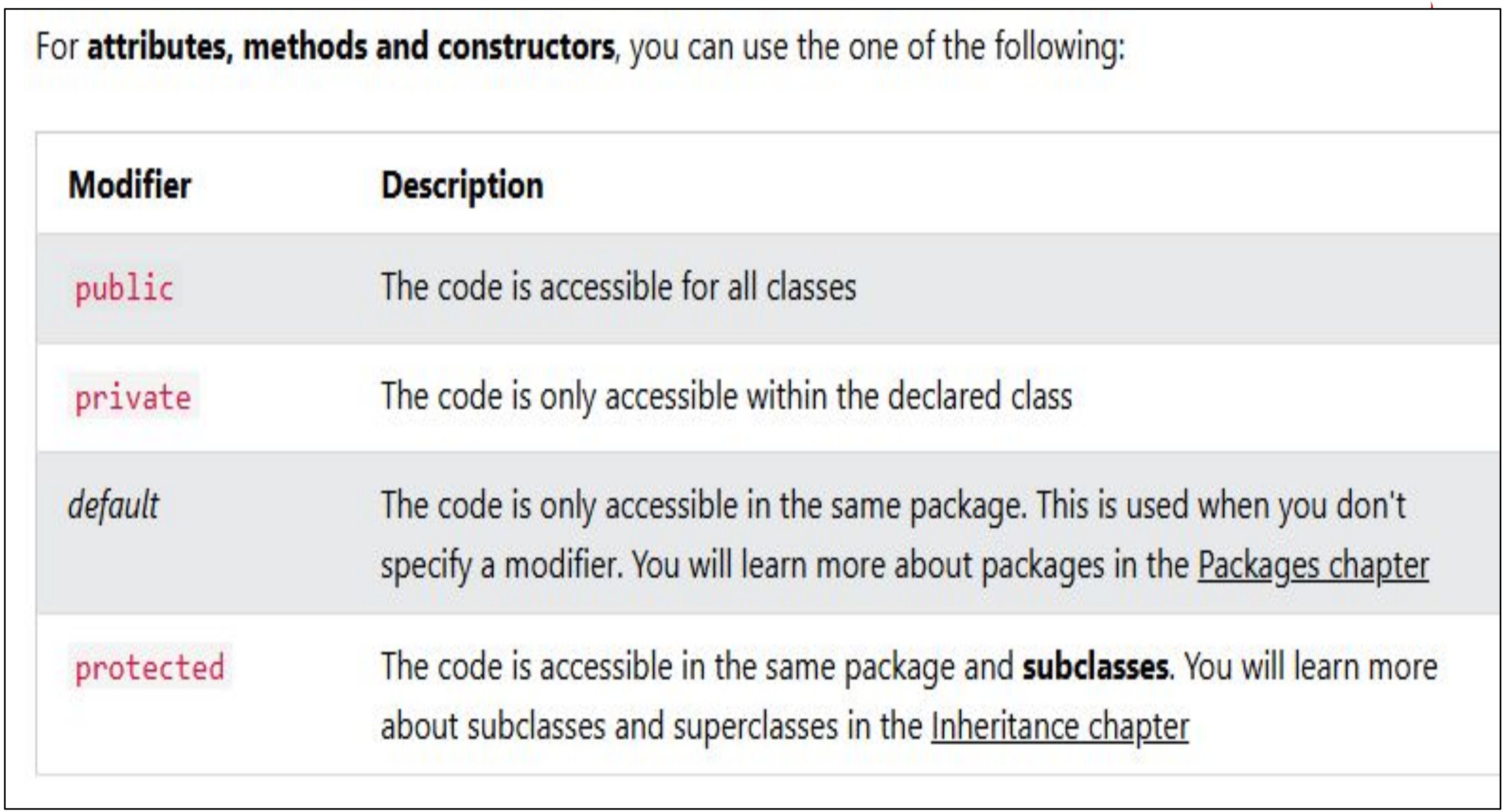
- Access Modifiers - controls the access level
- Non-Access Modifiers - do not control access level, but provides other functionality

Access Modifiers



For **classes**, you can use either `public` or *default*:

Modifier	Description
`public`	The class is accessible by any other class
default	The class is only accessible by classes in the same package.



For **attributes, methods and constructors**, you can use the one of the following:

Modifier	Description
`public`	The code is accessible for all classes
`private`	The code is only accessible within the declared class
default	The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter
`protected`	The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the Inheritance chapter

```
class Person {
  public String name = "John";   // Public - accessible everywhere
  private int age = 30;          // Private - only accessible inside this class
}

public class Main {
  public static void main(String[] args) {
    Person p = new Person();
    System.out.println(p.name);   // Works fine
    System.out.println(p.age);    // Error: age has private access in Person
  }
}
```

Encapsulation

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables/attributes as `private`
- provide public get and set methods to access and update the value of a `private` variable

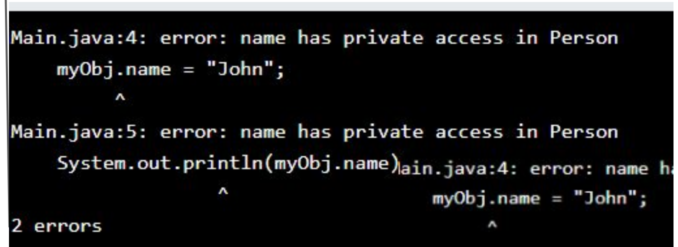
Get and Set

`private` variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public get and set methods.

The `get` method returns the variable value, and the `set` method sets the value.

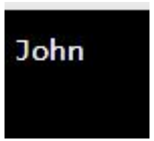
```java
public class Person {
  private String name; //
private = restricted
access

  // Getter
  public String getName()
{
    return name;
  }

  // Setter
  public void
setName(String newName) {
    this.name = newName;
  }

 }
```

```java
public class Main {
  public static void
main(String[] args) {
    Person myObj = new
Person();
    myObj.name = "John";  //
error

System.out.println(myObj.name)
; // error
   }

  }
```



```
Main.java:4: error: name has private access in Person
    myObj.name = "John";
         ^
Main.java:5: error: name has private access in Person
    System.out.println(myObj.name)ain.java:4: error: name h
              ^                  myObj.name = "John";
2 errors                              ^
```

```java
public class Main {
  public static void
main(String[] args) {
    Person myObj =
new Person();

myObj.setName("John")
; // Set the value of
the name variable to
"John"

System.out.println(my
Obj.getName());
  }

 }
```



```
John
```

Java Packages

Packages in Java are a mechanism that encapsulates a group of classes, sub-packages and interfaces. Packages are used for:

- Prevent naming conflicts by allowing classes with the same name to exist in different packages, like college.staff.cse.Employee and college.staff.ee.Employee.
- Make it easier to organize, locate and use classes, interfaces and other components.
- Provide controlled access for Protected members that are accessible within the same package and by subclasses. Default members (no access specifier) are accessible only within the same package.

By grouping related classes into packages, Java promotes data encapsulation, making code reusable and easier to manage. Simply import the desired class from a package to use it in your program.

Types of Java Packages

A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)

Built-in Packages (packages from the Java API)

The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.

The library contains components for managing input, database programming, and much much more. The complete list can be found at Oracles website: https://docs.oracle.com/javase/8/docs/api/.

The library is divided into packages and classes. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.

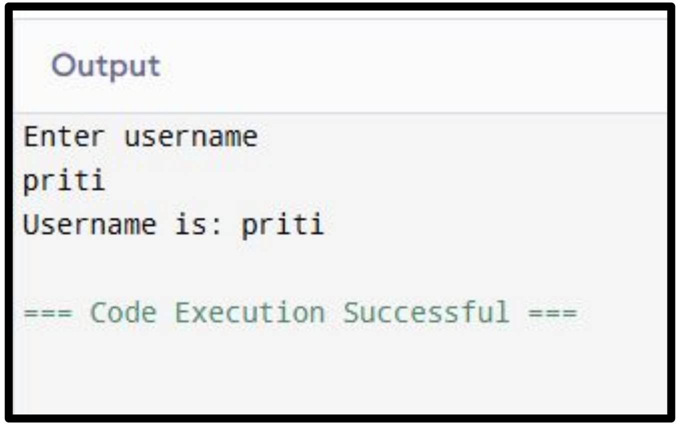
To use a class or a package from the library, you need to use the `import` keyword:

The `nextLine()` method returns a string containing all of the characters up to the next new line character in the scanner, or up to the end of the scanner if there are no more new line characters

- **Standard Input Stream:** `System.in` is one of three standard I/O streams provided by the `System` class in Java (the others being `System.out` for standard output and `System.err` for error output).

Example of scanner class

```
import java.util.Scanner; // import the Scanner class

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    String userName;

    // Enter username and press Enter
    System.out.println("Enter username");
    userName = myObj.nextLine();

    System.out.println("Username is: " + userName);
  }
}
```



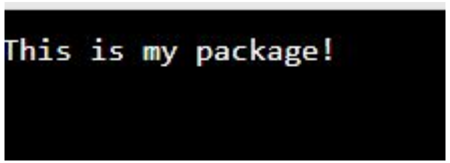
```
Enter username
priti
Username is: priti

=== Code Execution Successful ===
```

User-defined Packages

To create your own package, you need to understand that Java uses a file system directory to store them. Just like folders on your computer:

Example

```
package mypack;
class MyPackageClass {
  public static void main(String[] args) {
    System.out.println("This is my package!");
  }
 }
```



```
This is my package!
```

Strings

The `String` data type is used to store a sequence of characters (text). String values must be surrounded by double quotes:

Example:

```
String greeting = "Hello World";
 System.out.println(greeting);
```

String Length

A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the `length()` method:

```
String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

System.out.println("The length of the txt string is: " +

txt.length());
```

More String Methods

There are many string methods available, for example `toUpperCase()` and `toLowerCase()`:

```
String txt = "Hello World";

System.out.println(txt.toUpperCase());   // Outputs "HELLO WORLD"

System.out.println(txt.toLowerCase());   // Outputs "hello world"
```

Finding a Character in a String

The `indexOf()` method returns the index (the position) of the first occurrence of a specified text in a string (including whitespace):

```
String txt = "Please locate where 'locate' occurs!";

  System.out.println(txt.indexOf("locate")); // Outputs 7
```

String Concatenation

The `+` operator can be used between strings to combine them. This is called concatenation:

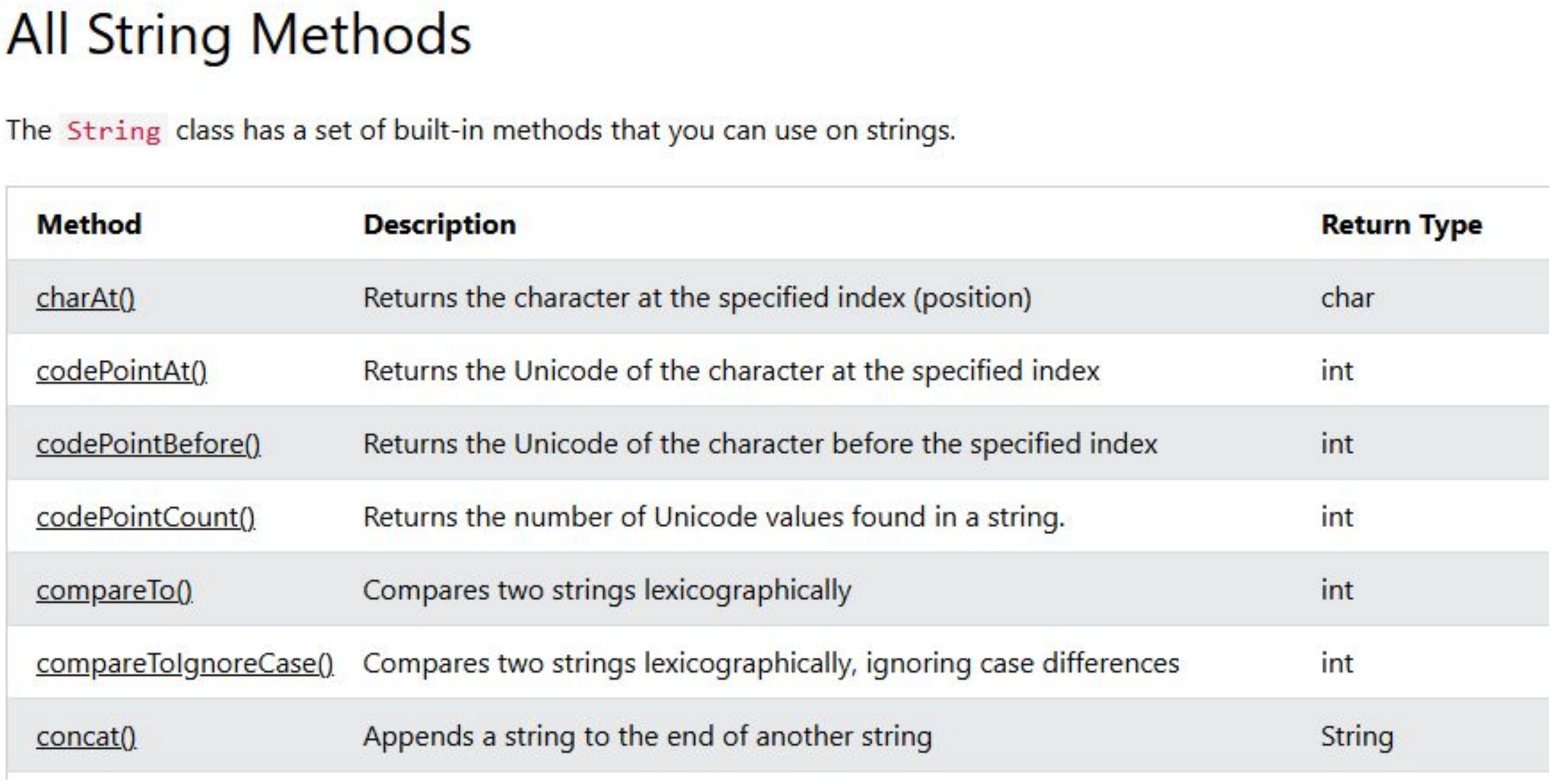
```
String firstName = "John";

String lastName = "Doe";

  System.out.println(firstName + " " + lastName);
```

Comparing Strings

```
String txt1 = "Hello";
String txt2 = "Hello";

String txt3 = "Greetings";
String txt4 = "Great things";

System.out.println(txt1.equals(txt2));  // true
 System.out.println(txt3.equals(txt4));  // false
```

Removing Whitespace

```
String txt = "   Hello World   ";

System.out.println("Before: [" + txt + "]");

 System.out.println("After:  [" + txt.trim() + "]");
```

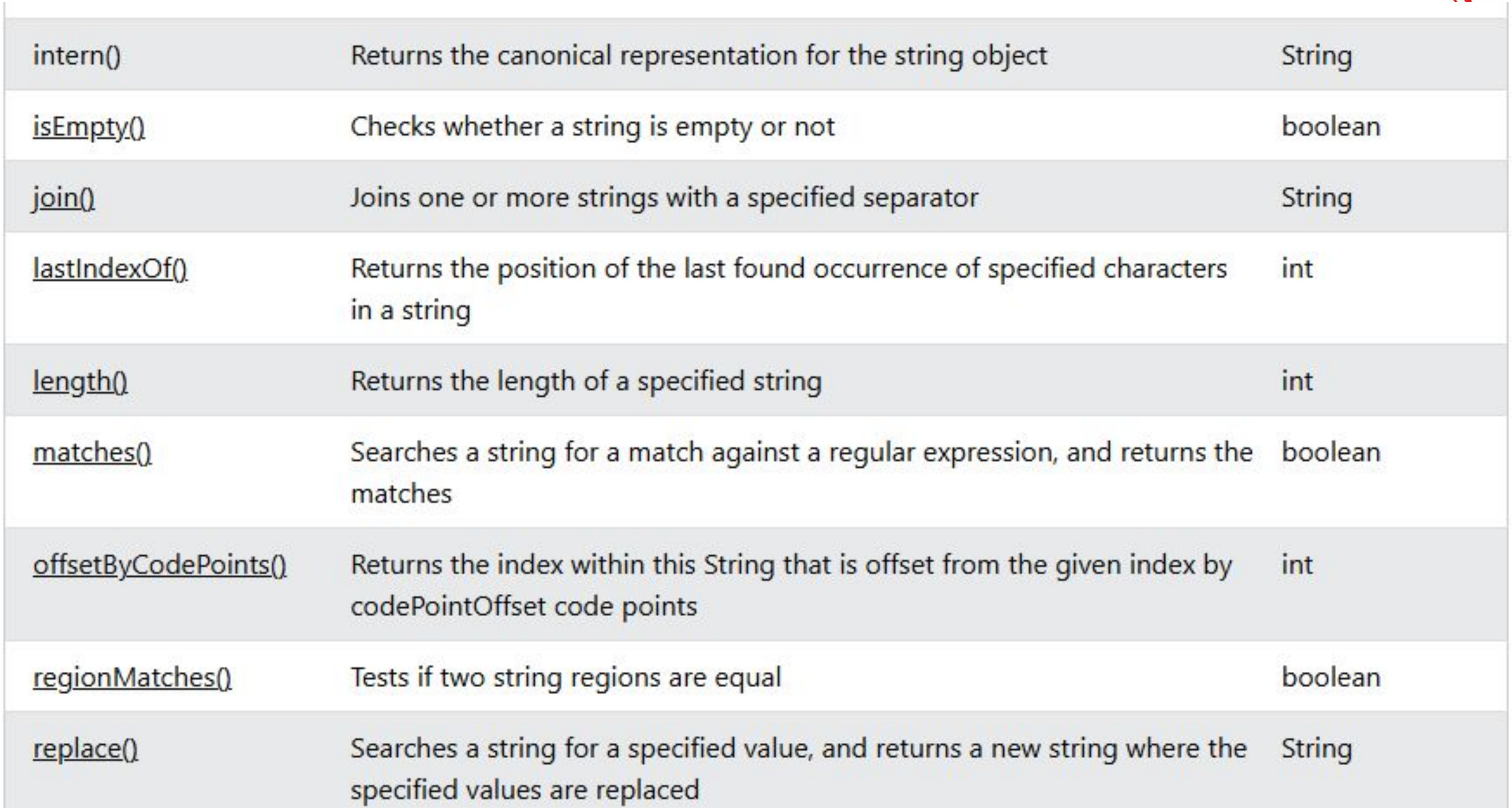


All String Methods

The `String` class has a set of built-in methods that you can use on strings.

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	char
codePointAt()	Returns the Unicode of the character at the specified index	int
codePointBefore()	Returns the Unicode of the character before the specified index	int
codePointCount()	Returns the number of Unicode values found in a string.	int
compareTo()	Compares two strings lexicographically	int
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences	int
concat()	Appends a string to the end of another string	String

contains()	Checks whether a string contains a sequence of characters	boolean
contentEquals()	Checks whether a string contains the exact same sequence of characters of the specified CharSequence or StringBuffer	boolean
copyValueOf()	Returns a String that represents the characters of the character array	String
endsWith()	Checks whether a string ends with the specified character(s)	boolean
equals()	Compares two strings. Returns true if the strings are equal, and false if not	boolean
equalsIgnoreCase()	Compares two strings, ignoring case considerations	boolean
format()	Returns a formatted string using the specified locale, format string, and arguments	String
getBytes()	Converts a string into an array of bytes	byte[]
getChars()	Copies characters from a string to an array of chars	void
hashCode()	Returns the hash code of a string	int
indexOf()	Returns the position of the first found occurrence of specified characters in a string	int



intern()	Returns the canonical representation for the string object	String
isEmpty()	Checks whether a string is empty or not	boolean
join()	Joins one or more strings with a specified separator	String
lastIndexOf()	Returns the position of the last found occurrence of specified characters in a string	int
length()	Returns the length of a specified string	int
matches()	Searches a string for a match against a regular expression, and returns the matches	boolean
offsetByCodePoints()	Returns the index within this String that is offset from the given index by codePointOffset code points	int
regionMatches()	Tests if two string regions are equal	boolean
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String

Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
To declare an array, define the variable type with square brackets:

```
  String[] cars;
```

We have now declared a variable that holds an array of strings. To insert values to it, you can place the values in a comma-separated list, inside curly braces:

```
  String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
  int[] myNum = {10, 20, 30, 40};
```

Access the Elements of an Array

You can access an array element by referring to the index number.
This statement accesses the value of the first element in cars:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
  // Outputs Volvo
```

Change an Array Element

To change the value of a specific element, refer to the index number:

```
public class Main {
  public static void main(String[] args) {
    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    cars[0] = "Opel";
    System.out.println(cars[0]);
  }
}
```

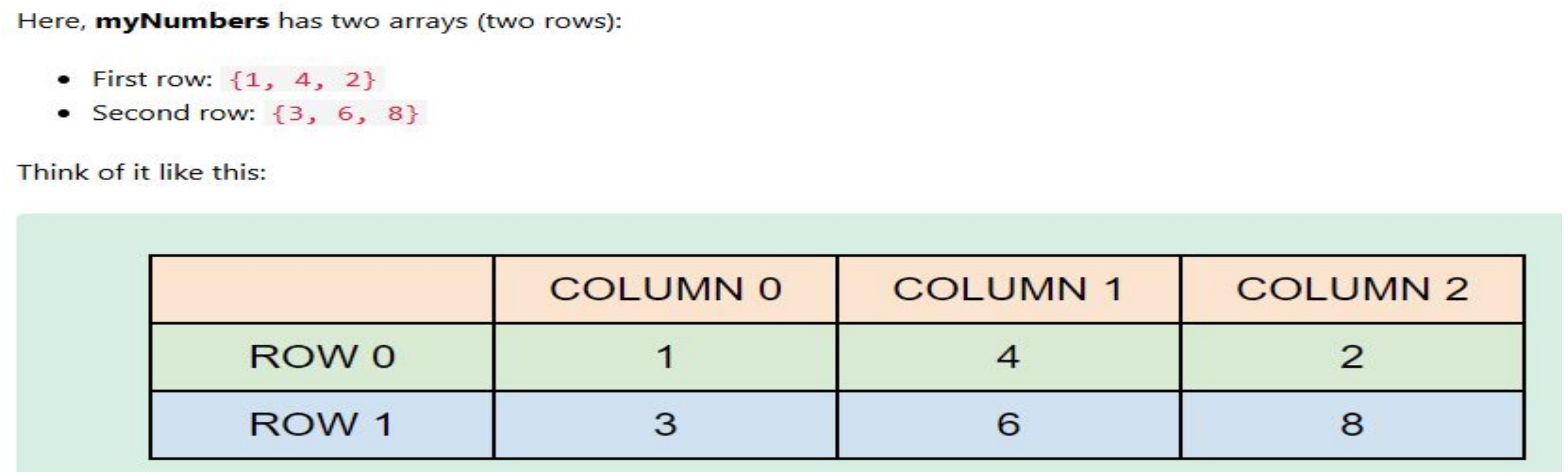
Multi-Dimensional Arrays

A multidimensional array is an array that contains other arrays.

You can use it to store data in a table with rows and columns.

To create a two-dimensional array, write each row inside its own curly braces:

```
Ex: int[][] myNumbers = { {1, 4, 2}, {3, 6, 8} };
```



Here, **myNumbers** has two arrays (two rows):

- First row: `{1, 4, 2}`
- Second row: `{3, 6, 8}`

Think of it like this:

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

Access Elements

To access an element of a two-dimensional array, you need two indexes: the first for the row, and the second for the column.

Remember: Array indexes start at 0. That means row `0` is the first row, and column `0` is the first column. (So row index `1` is the *second* row, and column index `2` is the *third* column.)

This statement accesses the element in the second row (index `1`) and third column (index `2`) of the myNumbers array:

```
int[][] myNumbers = { {1, 4, 2}, {3, 6, 8} };

  System.out.println(myNumbers[1][2]); // Outputs 8
```

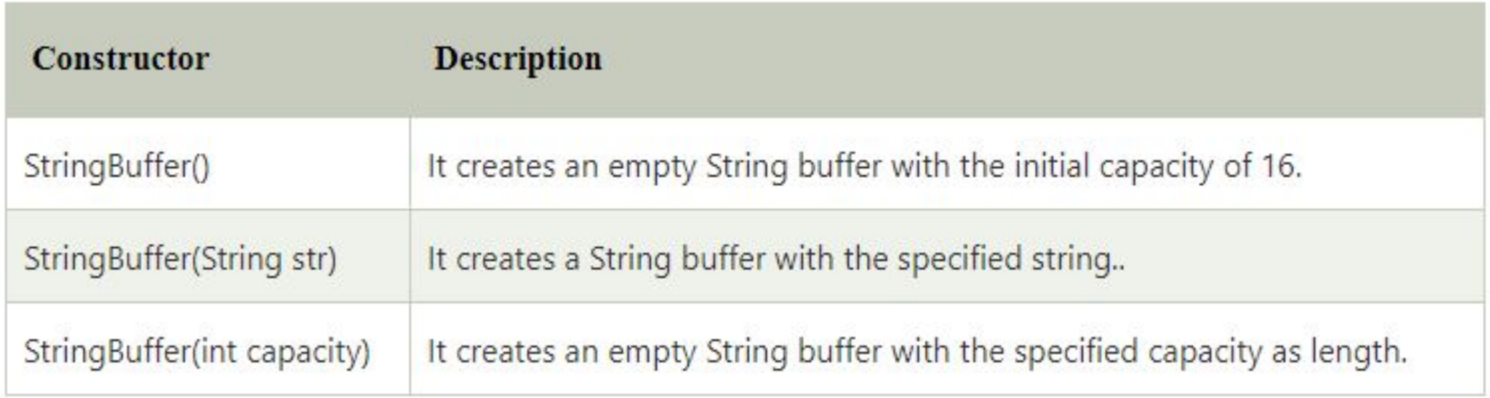
example

```
public class Main {
  public static void main(String[] args) {
    Int[ ][ ] myNumbers = { {1, 4, 2}, {3, 6, 8} };
    myNumbers[1][2] = 9;
    System.out.println(myNumbers[1][2]); // Outputs 9 instead of 8
  }
}
```

Output :2

String Buffer classes

- Java StringBuffer class is used to create mutable (modifiable) String objects.
- The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.
- Constructors of StringBuffer Class



Constructor	Description
StringBuffer()	It creates an empty String buffer with the initial capacity of 16.
StringBuffer(String str)	It creates a String buffer with the specified string..
StringBuffer(int capacity)	It creates an empty String buffer with the specified capacity as length.

1) StringBuffer Class append() Method

- The append() method concatenates the given argument with this String.

```
class StringBufferExample{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}  }
```

2) StringBuffer insert() Method

The insert() method inserts the given String with this string at the given position.

```
class StringBufferExample2{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);
} }
```

3. StringBuffer replace() Method

replace() method replaces the given string from the specified beginIndex and endIndex-1.

```
import java.io.*;

class Geeks {
    public static void main(String args[]) {

        StringBuffer sb = new StringBuffer("Hello");
        sb.replace(1, 3, "Java");
        System.out.println(sb);
    }
}
```

4.. StringBuffer delete() Method

delete() method is used to delete the string from the specified beginIndex to endIndex-1.

```
import java.io.*;

class Geeks {
    public static void main(String args[]) {

        StringBuffer sb = new StringBuffer("Hello");
        sb.delete(1, 3);
        System.out.println(sb);
    }
}
```

Vector Class in Java

Vector is a resizable array in Java, found in the java.util package. It is part of the Collection Framework and works like an ArrayList, but it is synchronized, meaning it is safe to use in multi-threaded programs. However, this makes it a bit slower than ArrayList.

Key Features of Vector

- It expands as elements are added.
- The Vector class is synchronized in nature means it is thread-safe by default.
- Like an ArrayList, it maintains insertion order.
- It allows duplicates and nulls.
- It implements List, RandomAccess, Cloneable and Serializable.

Java Program Implementing Vector

```
import java.util.Vector;

public class Geeks
{
    public static void main(String[] args)
    {
        // Create a new vector
        Vector<Integer> v = new Vector<>(3, 2);

        // Add elements to the vector
        v.addElement(1);
        v.addElement(2);
        v.addElement(3);

        // Insert an element at index 1
        v.insertElementAt(0, 1);

        // Remove the element at index 2
        v.removeElementAt(2);

        // Print the elements of the vector
        for (int i : v) {
            System.out.println(i);
        }
    }
}
```

Output

```
1
0
3
```