ASSIGNMENT 2

Name: M.Manaswini

Reg. Number: 23BCE9853

AIM: To demonstrate String and String function

APPLICATION:

The applications of strings and string functions in programming are extensive and fundamental across various domains. Here are some common applications:

1. **Text Processing:** Strings are used extensively in text processing tasks such as parsing, searching, and manipulating textual data. String functions like `substring()`, `indexOf()`, and `replace()` are vital for performing these operations efficiently.

2. **User Input Handling:** When developing applications that interact with users, string functions are essential for handling user input validation, formatting, and sanitization. They help ensure that the input provided by users meets certain criteria and is processed correctly.

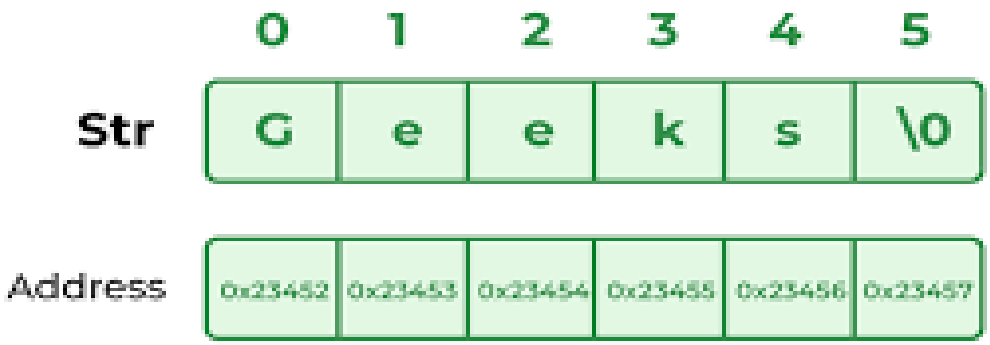
3. **Data Storage and Retrieval:** Strings are commonly used for storing and representing data in databases, files, and data structures. String functions facilitate tasks such as converting data types,

formatting data for storage, and extracting relevant information from stored data.

4. **Text-Based User Interfaces:** In applications with text-based user interfaces (e.g., command-line interfaces), strings are used to display information, prompts, and messages to users. String functions help with formatting text and aligning content for better readability.

Overall, strings and string functions are fundamental building blocks in programming, enabling developers to work with textual data effectively across a wide range of applications and use cases.

DESIGN:



PSEUDO CODE:

```
public class hel {
    public static void main(String[] args) {
        // Creating strings
        String str1 = "Hello";
        String str2 = "World";

        // Concatenating strings
        String concatenatedString = str1 + " " + str2;
        System.out.println("Concatenated String: " + concatenatedString);

        // Length of a string
        int length = concatenatedString.length();
        System.out.println("Length of the string: " + length);

        // Finding index of a character
        int index = concatenatedString.indexOf('W');
        System.out.println("Index of 'W': " + index);

        // Substring
        String substring = concatenatedString.substring(6);
        System.out.println("Substring from index 6: " + substring);
```

```
        // Converting case
        String uppercase = concatenatedString.toUpperCase();
        System.out.println("Uppercase: " + uppercase);

        String lowercase = concatenatedString.toLowerCase();
        System.out.println("Lowercase: " + lowercase);

        // Checking equality
        boolean isEqual = str1.equals(str2);
        System.out.println("Are str1 and str2 equal? " + isEqual);

        // Checking equality (case-insensitive)
        boolean isEqualIgnoreCase = str1.equalsIgnoreCase("hello");
        System.out.println("Are str1 and 'hello' equal ignoring case? " +
isEqualIgnoreCase);
    }
}
```

OUTPUT:

```
Concatenated String: Hello World
Length of the string: 11
Index of 'W': 6
```

Substring from index 6: World

Uppercase: HELLO WORLD

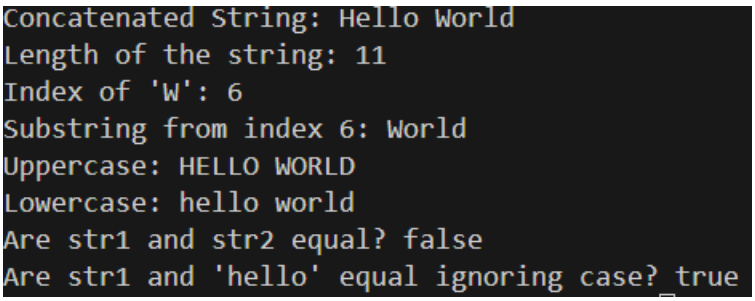
Lowercase: hello world

Are str1 and str2 equal? false

Are str1 and 'hello' equal ignoring case? True

RESULT:

```java
public class hel {
    public static void main(String[] args) {
        // Creating strings
        String str1 = "Hello";
        String str2 = "World";

        // Concatenating strings
        String concatenatedString = str1 + " " + str2;
        System.out.println("Concatenated String: " + concatenatedString);

        // Length of a string
        int length = concatenatedString.length();
        System.out.println("Length of the string: " + length);

        // Finding index of a character
        int index = concatenatedString.indexOf(ch:'W');
        System.out.println("Index of 'W': " + index);

        // Substring
        String substring = concatenatedString.substring(beginIndex:6);
        System.out.println("Substring from index 6: " + substring);

        // Converting case
        String uppercase = concatenatedString.toUpperCase();
        System.out.println("Uppercase: " + uppercase);

        String lowercase = concatenatedString.toLowerCase();
        System.out.println("Lowercase: " + lowercase);

        // Checking equality
        boolean isEqual = str1.equals(str2);
        System.out.println("Are str1 and str2 equal? " + isEqual);

        // Checking equality (case-insensitive)
        boolean isEqualIgnoreCase = str1.equalsIgnoreCase(anotherString:"hello");
        System.out.println("Are str1 and 'hello' equal ignoring case? " + isEqualIgnoreCa
    }
```



CONCLUSION:

The applications of strings and string functions in programming are diverse and essential across various domains. From text processing and user input handling to data storage, web development, and internationalization, strings play a fundamental role in modern software development.Through their versatility, strings enable developers to manipulate textual data efficiently, facilitating tasks such as parsing, searching, formatting, and validating user input. String functions provide a rich set of tools for performing these operations, empowering developers to work with strings effectively and expressively.In addition to their practical utility, strings and string functions contribute to the readability, maintainability, and flexibility of code. By leveraging string functions and best practices, developers can write cleaner, more concise code that is easier to understand and maintain over time. Furthermore, the widespread use of strings extends beyond traditional software development to fields such as data analysis, natural language processing, and web development. Whether it's analyzing text data, building web applications, or handling multilingual content, strings serve as a foundational element in modern computing. In conclusion, the importance of strings and string functions in programming cannot be overstated. Their versatility, efficiency, and ubiquity make them indispensable tools for developers across a wide range of applications and use cases, shaping the way we interact with and process textual data in the digital age.

AIM: To define a class, describe its constructor , overload the constructors and instantiate its object

APPLICATION:

The applications of classes, constructors, and objects in Java are numerous and fundamental to object-oriented programming (OOP). Here are some key applications:

1. **Modeling Real-World Entities:** Classes allow developers to model real-world entities or abstract concepts within a software system. For example, a `Person` class can represent individuals with attributes such as name, age, and address.

2. **Encapsulation:** Classes facilitate encapsulation by bundling data (attributes) and behavior (methods) together into a single unit. This helps in organizing code and preventing unauthorized access to internal data by using access modifiers such as private, protected, and public.

3. **Code Reusability:** Once a class is defined, it can be reused in multiple parts of a program or in different programs altogether. This promotes code reuse and reduces redundancy, leading to more maintainable and scalable software systems.

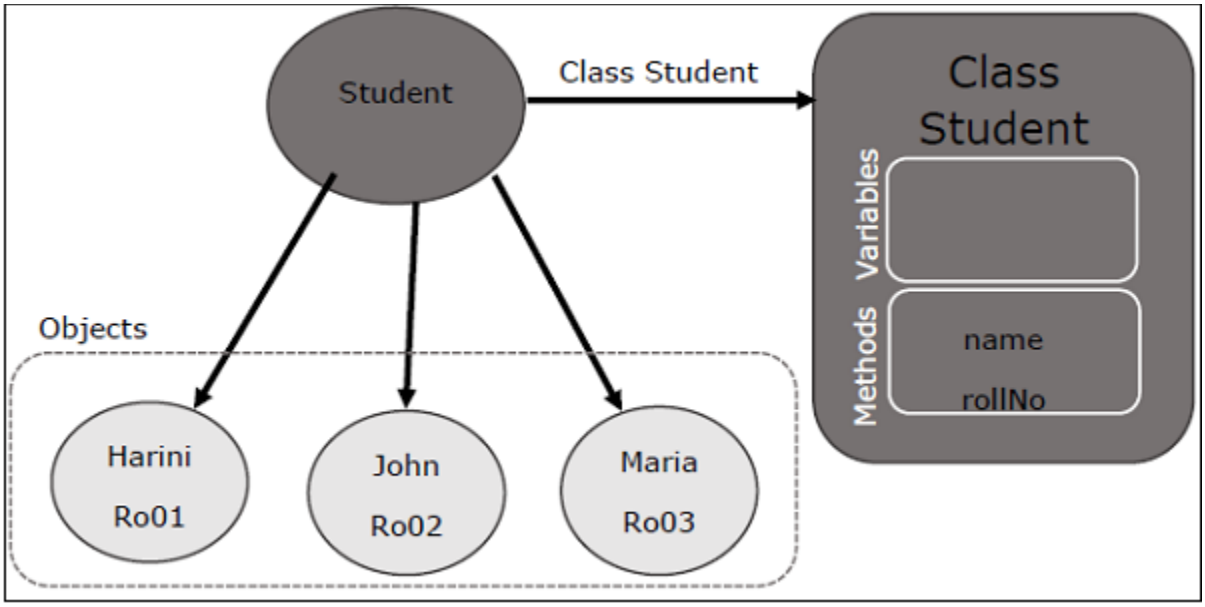
4. **Modularity and Maintainability:** Classes promote modularity by breaking down complex systems into smaller, manageable units. Each class encapsulates a specific set of functionalities, making it easier to understand, maintain, and update the codebase.

5. **Inheritance:** Classes can inherit attributes and methods from other classes through inheritance, facilitating code reuse and promoting the "is-a" relationship between objects. This allows for hierarchical organization of classes and promotes code extensibility.

6. **Polymorphism:** Objects of different classes can be treated interchangeably through polymorphism. This enables developers to write code that operates on objects of a superclass but can be applied to objects of any subclass, enhancing flexibility and code reuse.

In summary, the applications of classes, constructors, and objects in Java are diverse and essential for building modular, maintainable, and extensible software systems following the principles of object-oriented programming.

DESIGN:



PSEUDO CODE:

```
class Car {
    private String make;
    private String model;
    private int year;

    // Default constructor
    public Car() {
        this.make = "Unknown";
        this.model = "Unknown";
        this.year = 0;
    }

    // Parameterized constructor
    public Car(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }

    // Overloaded constructor with only make and model
    public Car(String make, String model) {
```

```
        this.make = make;
        this.model = model;
        this.year = 0; // Assuming year as 0 for unspecified year
    }

    // Method to display car details
    public void display() {
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
    }
}

public class hel {
    public static void main(String[] args) {
        // Instantiating objects of the Car class using different constructors
        Car car1 = new Car(); // Default constructor
        Car car2 = new Car("Toyota", "Camry", 2020); // Parameterized
constructor
        Car car3 = new Car("Honda", "Civic"); // Overloaded constructor

        // Displaying details of each car
```

```
        System.out.println("Car 1:");
        car1.display();
        System.out.println();

        System.out.println("Car 2:");
        car2.display();
        System.out.println();

        System.out.println("Car 3:");
        car3.display();
    }
}
```

OUTPUT:

```
Car 1:
Make: Unknown
Model: Unknown
Year: 0

Car 2:
Make: Toyota
Model: Camry
Year: 2020
```

Car 3:

Make: Honda

Model: Civic

Year: 0

RESULT:

```java
class Car {
    private String make;
    private String model;
    private int year;

    // Default constructor
    public Car() {
        this.make = "Unknown";
        this.model = "Unknown";
        this.year = 0;
    }

    // Parameterized constructor
    public Car(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }

    // Overloaded constructor with only make and model
    public Car(String make, String model) {
        this.make = make;
        this.model = model;
        this.year = 0; // Assuming year as 0 for unspecified year
    }

    // Method to display car details
    public void display() {
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
    }
}
```

```java
public class hel {
    Run | Debug
    public static void main(String[] args) {
        // Instantiating objects of the Car class using different constructors
        Car car1 = new Car(); // Default constructor
        Car car2 = new Car(make:"Toyota", model:"Camry", year:2020); // Parameterized constructor
        Car car3 = new Car(make:"Honda", model:"Civic"); // Overloaded constructor

        // Displaying details of each car
        System.out.println(x:"Car 1:");
        car1.display();
        System.out.println();

        System.out.println(x:"Car 2:");
        car2.display();
        System.out.println();

        System.out.println(x:"Car 3:");
        car3.display();
    }
}
```

```
Car 1:
Make: Unknown
Model: Unknown
Year: 0

Car 2:
Make: Toyota
Model: Camry
Year: 2020

Car 3:
Make: Honda
Model: Civic
Year: 0
```

CONCLUSION:

In conclusion, the concepts of classes, constructors, and objects form the foundation of object-oriented programming (OOP) in Java, offering developers a powerful paradigm for building modular, maintainable, and scalable software solutions. Through the application of classes, developers can model real-world entities or abstract concepts, encapsulating data and behavior into cohesive units. Constructors play a crucial role in initializing objects of a class, ensuring that they are properly configured and ready for use. By defining constructors with various parameters and overloading them as needed, developers provide flexibility and customization options for object instantiation. Objects, as instances of classes, represent specific occurrences of the entities modeled by those classes. They encapsulate state and behavior, enabling interaction with other objects and facilitating dynamic behavior based on their internal state. Together, classes, constructors, and objects promote code reuse, modularity, and maintainability, enabling developers to create complex systems with ease. They support key principles of software engineering such as encapsulation, inheritance, and polymorphism, fostering a modular and extensible codebase.

In summary, the applications of classes, constructors, and objects in Java are vast and fundamental to modern software development practices. By understanding and leveraging these concepts effectively, developers can create robust, flexible, and scalable software solutions to address a wide range of requirements and challenges.