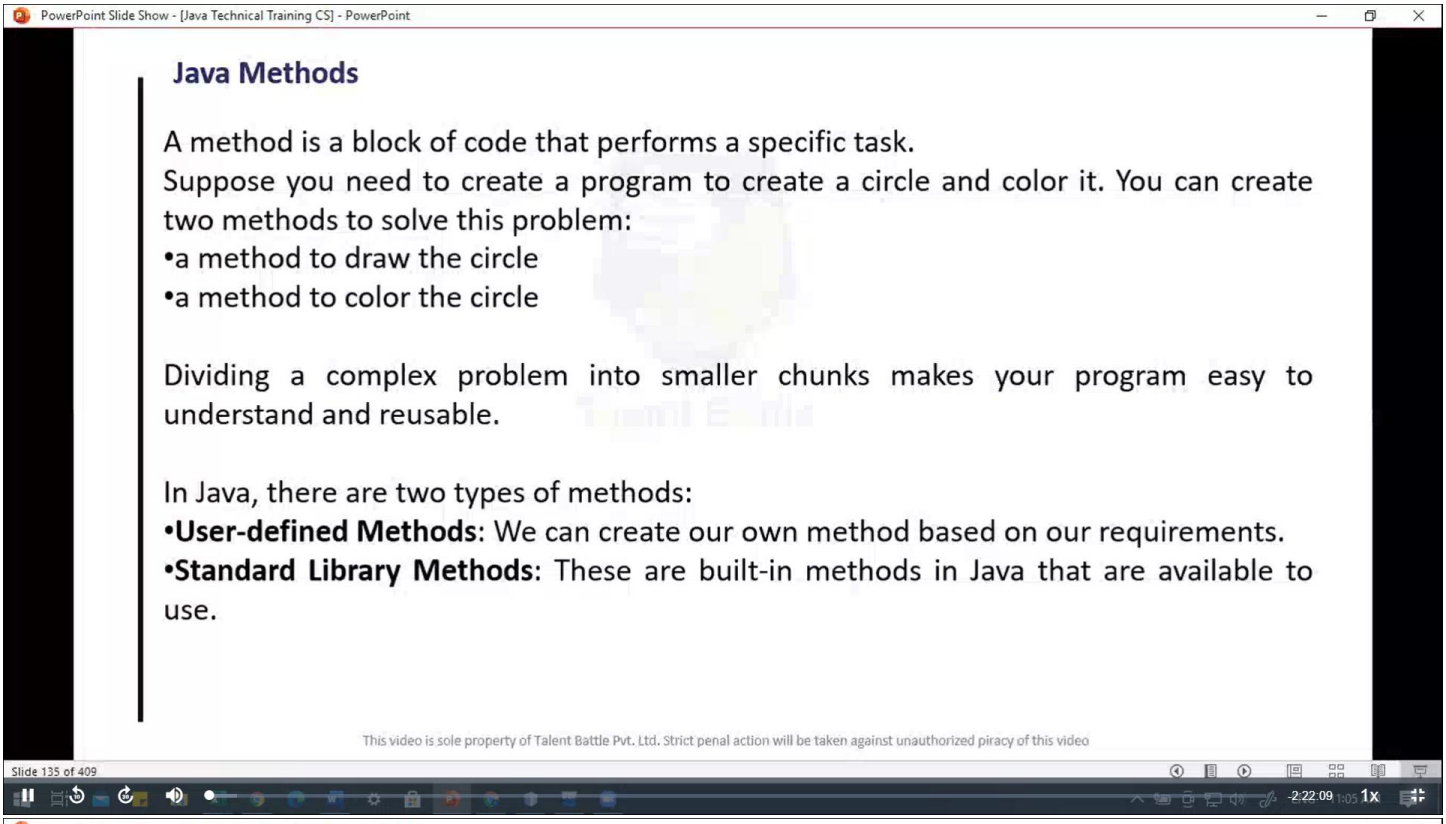
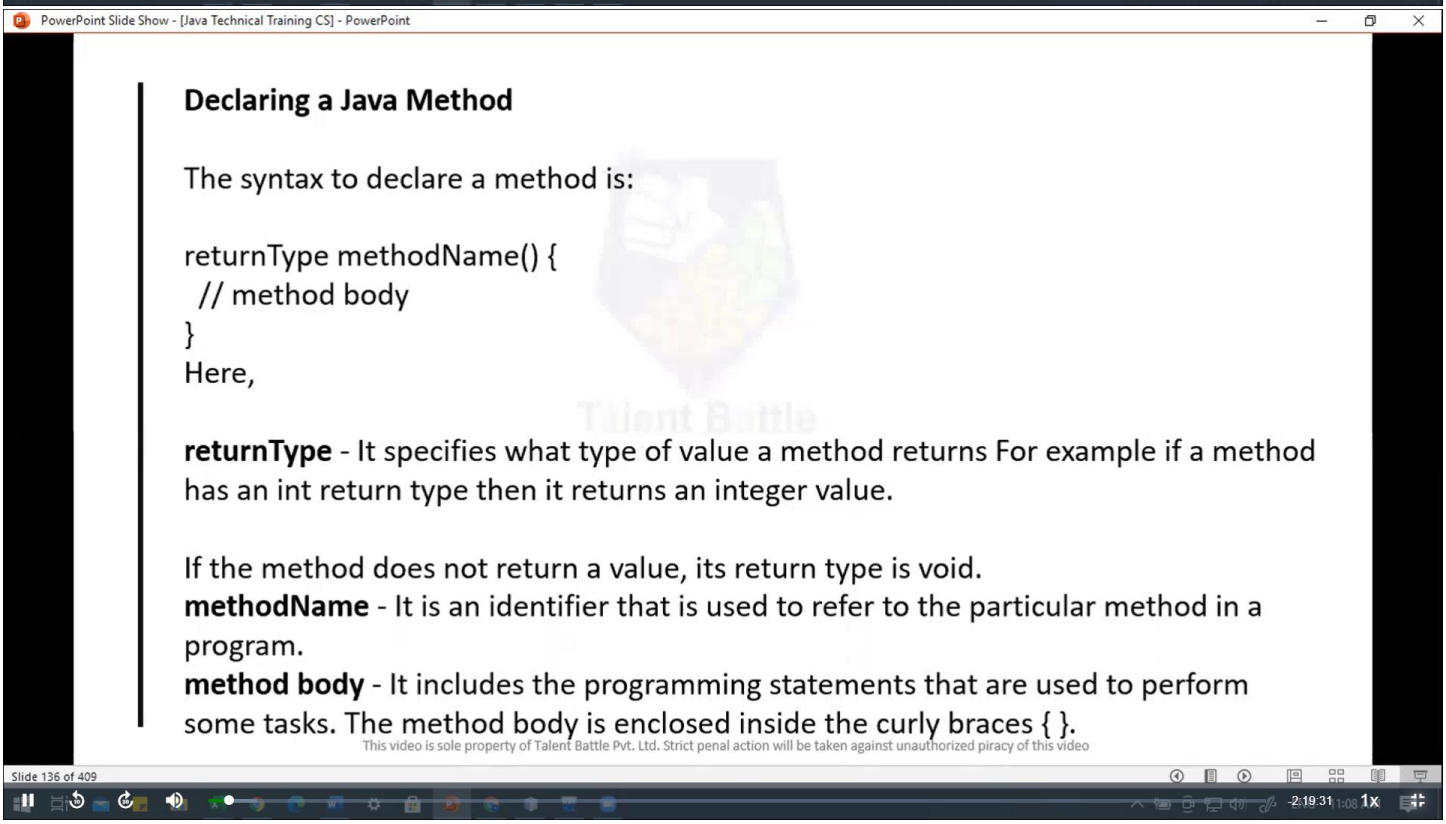
Day 5

Java Methods and Strings



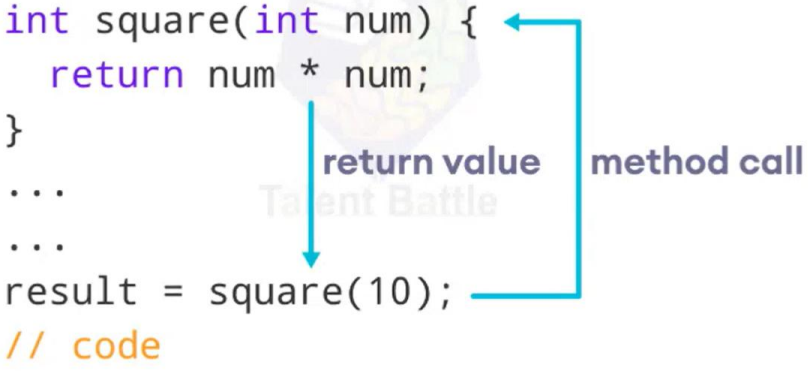


PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

```
class Main {

 // create a method
 public int addNumbers(int a, int b) {
  int sum = a + b;
  // return value
  return sum;
 }

 public static void main(String[] args) {

  int num1 = 25;
  int num2 = 15;

  // create an object of Main
  Main obj = new Main();
  // calling method
  int result = obj.addNumbers(num1, num2);
  System.out.println("Sum is: " + result);
 }
}
```


-2:16:42 1:11 1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint



-2:12:56 1:15 1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Method Parameters in Java

```
class Main {
  // method with no parameter
  public void display1() {
    System.out.println("Method without parameter");
  }
  // method with single parameter
  public void display2(int a) {
    System.out.println("Method with a single parameter: " + a);
  }
  public static void main(String[] args) {
    // create an object of Main
   Main obj = new Main();
    // calling method with no parameter
    obj.display1();
    // calling method with the single parameter
    obj.display2(24);
  }
}
```


-2:12:24 1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Standard Library Methods

The standard library methods are built-in methods in Java that are readily available for use. These standard libraries come along with the Java Class Library (JCL) in a Java archive (*.jar) file with JVM and JRE.

For example,

print() is a method of java.io.PrintSteam. The print("...") method prints the string inside quotation marks.

sqrt() is a method of Math class. It returns the square root of a number.

ENG 11:26 AM

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

To exit full screen, press Esc

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

    // using the sqrt() method
    System.out.print("Square root of 4 is: " + Math.sqrt(4));
  }
}
```


-1:56:10 1:31 1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

What are the advantages of using methods?

1. The main advantage is **code reusability**.
2. Methods make code more **readable and easier** to debug.

```
public class Main {
  // method defined
  private static int getSquare(int x){
    return x * x;
  }
  public static void main(String[] args) {
    for (int i = 1; i <= 5; i++) {
      // method call
      int result = getSquare(i);
      System.out.println("Square of " + i + " is: " + result);
    }
  }
}
```


-1:41:22 1.5x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Java Constructors

What is a Constructor?

A constructor in Java is similar to a method that is invoked when an object of the class is created.

Unlike Java methods, a constructor has the same name as that of the class and does not have any return type. For example,

```
class Test {
  Test() {
    // constructor body
  }
}
```


-1:35:57 1.5x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

```
class Main {
  private String name;

  // constructor
  Main() {
    System.out.println("Constructor Called:");
    name = "Programming";
  }

  public static void main(String[] args) {

    // constructor is invoked while
    // creating an object of the Main class
    Main obj = new Main();
    System.out.println("The name is " + obj.name);
  }
}
```


-1:34:28 1.5x

Types of Constructor

In Java, constructors can be divided into 3 types:

1.No-Arg Constructor

2.Parameterized Constructor

3.Default Constructor

1. Java No-Arg Constructors

Similar to methods, a Java constructor may or may not have any parameters (arguments).

If a constructor does not accept any parameters, it is known as a no-argument constructor. For example,

```
private Constructor() {
  // body of the constructor
}
```

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

```
class Main {

  int i;

  // constructor with no parameter
  private Main() {
    i = 5;
    System.out.println("Constructor is called");
  }

  public static void main(String[] args) {

    // calling the constructor without any parameter
    Main obj = new Main();
    System.out.println("Value of i: " + obj.i);
  }
}
```


-1:31:02 1.5x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Example 1: Java program to create a private constructor

```
class Test {
  // create private constructor
  private Test () {
    System.out.println("This is a private constructor.");
  }
  // create a public static method
  public static void instanceMethod() {
    // create an instance of Test class
    Test obj = new Test();
  }
}
class Main {
 public static void main(String[] args) {
    // call the instanceMethod()
    Test.instanceMethod();
  }
}
```


-1:29:26 1.5x

2. Java Parameterized Constructor

A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructor with parameters).

```
class Main {
  String languages;
  // constructor accepting single value
  Main(String lang) {
    languages = lang;
    System.out.println(languages + " Programming Language");
  }
  public static void main(String[] args) {
    // call constructor by passing a single value
    Main obj1 = new Main("Java");
    Main obj2 = new Main("Python");
    Main obj3 = new Main("C");
  }
}
```


3. Java Default Constructor

If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.

```
class Main {
  int a;
  boolean b;
  public static void main(String[] args) {
   // A default constructor is called
    Main obj = new Main();
    System.out.println("Default Value:");
    System.out.println("a = " + obj.a);
    System.out.println("b = " + obj.b);
  }
}
```

Important Notes on Java Constructors

- Constructors are invoked implicitly when you instantiate objects.
- The two rules for creating a constructor are:

The name of the constructor should be the same as the class.
A Java constructor must not have a return type.

- If a class doesn't have a constructor, the Java compiler automatically creates a default constructor during run-time. The default constructor initializes instance variables with default values. For example, the int variable will be initialized to 0
- Constructor types:
- **No-Arg Constructor** - a constructor that does not accept any arguments
- **Parameterized constructor** - a constructor that accepts arguments
- **Default Constructor** - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
- A constructor cannot be abstract or static or final.
- A constructor can be overloaded but can not be overridden.

Constructors Overloading in Java

Similar to Java method overloading, we can also create two or more constructors with different parameters. This is called constructors overloading.

```
class Main {
  String language;
 Main() {
    this.language = "Java";
  }
  Main(String language) {
    this.language = language;
  }
  public void getName() {
    System.out.println("Programming Langauage: " + this.language);
  }
  public static void main(String[] args) {
    Main obj1 = new Main();
    Main obj2 = new Main("Python");
    obj1.getName();
    obj2.getName();
  }
}
```

```java
package Day5;
// Method
/*
public class Main {


  // create a method
  public int addNumbers(int a, int b){
    int sum = a + b;
    // return value
    return sum;
  }

  public static void main(String[] args){
    int num1 = 25;
    int num2 = 15;

    // create an object of Main
    Main obj = new Main();
    // calling method
    int result = obj.addNumbers(num1, num2);
    System.out.println("Sum is: " + result);

    // ============ output ================
    // PS C:\Users\hp\Desktop\TCS IT\Java> java Day5/Main.java
    // Sum is: 40

  }

}
 */

import java.lang.reflect.Method;

//=================================================================

 // Method Parameters in java
/*
 public class Main {

  // method with no parameter
  public void display1(){
    System.out.println("Method without parameter");
  }

  // Method with single parameter
  public void display2(int a){
    System.out.println("Method with a single parameter: " + a);
  }
```

```
  public static void main(String[] args) {
    // create an object of Main
    Main obj = new Main();

    // calling method with no parameter
    obj.display1();

    // calling method with the single parameter
    obj.display2(24);

    // ======== output =============
    // PS C:\Users\hp\Desktop\TCS IT\Java> java Day5/Main.java
    // Method without parameter
    // Method with a single parameter: 24
  }
 }
   */


//=====================================================================

// Standerd library method
/*
public class Main {

  public static void main(String[] args) {
    // using the sqrt() method
    System.out.println("Square root of 4 is: " + Math.sqrt(4));
  }
}
 */



 /*
// user input
import java.util.*;

public class Main {

  public static void main(String[] args) {
    // taking input from user
    System.out.println("Enter a number: ");

    double num;

    Scanner obj = new Scanner(System.in);
    num = obj.nextFloat();

    double result = Math.sqrt(num);
```

```
    System.out.println("Square Root of entered value is: " + result);

    // ============== output ===============
      // PS C:\Users\hp\Desktop\TCS IT\Java> java Day5/Main.java
      // Enter a number:
      // 25
      // Square Root of entered value is: 5.0
      // PS C:\Users\hp\Desktop\TCS IT\Java>
  }
}
 */


 //============================================================

 /*
 // what are advantaes of using methods?
 // 1. Reusability of code
 // 2. Code organization
 // 3. Code readability

 public class Main {

  // method defined
  private static int getSquare(int x){
    return x * x;
  }

  public static void main(String[] args) {
    for (int i = 1; i <= 5; i++){
      // method call
      int result = getSquare(i);
      System.out.println("Square of " + i + " is: " + result);
    }
  }
 }

 // ============ output ================
  // PS C:\Users\hp\Desktop\TCS IT\Java> java Day5/Main.java
  // Square of 1 is: 1
  // Square of 2 is: 4
  // Square of 3 is: 9
  // Square of 4 is: 16
  // Square of 5 is: 25

 */
```

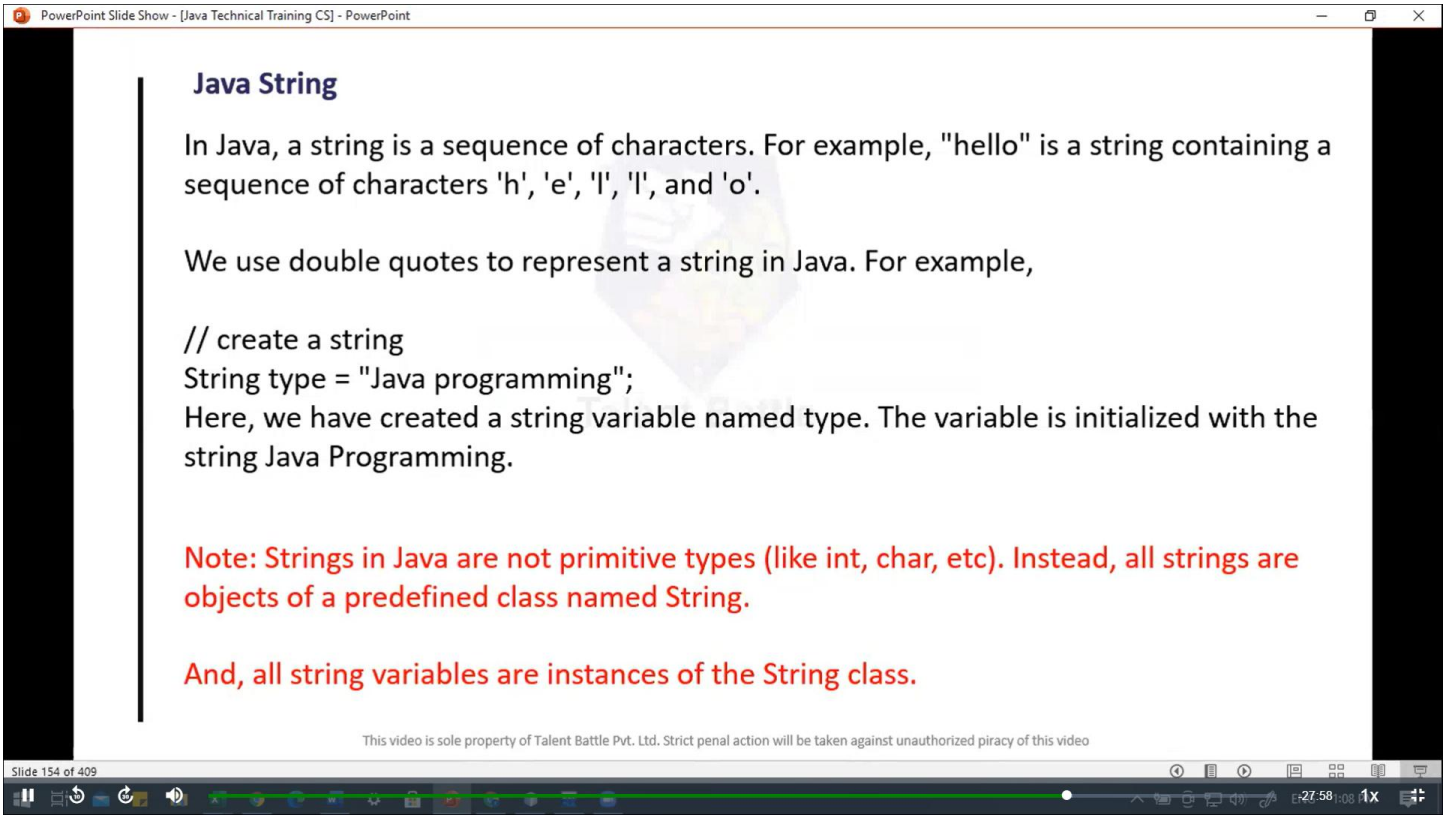
```java
//============================================================

/*
// Constructor
public class Main {

 private String name;

 // constructor
 Main(){
   System.out.println("Constructor Called: ");
   name = "Programming";
 }

 public static void main(String[] args) {
   // constructor is invoked while
   // creating an object of the Main class
   Main obj = new Main();
   System.out.println("The name is " + obj.name);
 }
}

// ========== output ===============
 // PS C:\Users\hp\Desktop\TCS IT\Java> java Day5/Main.java
 // Constructor Called:
 // The name is Programming

*/


//============================================================

/*
// No argument constructor
public class Main {
 int i;
 // constructor with no parameter
 private Main(){
   i = 5;
   System.out.println("Constructor is called ");
 }

 public static void main(String[] args) {
   // calling the constructor without any parameter
   Main obj = new Main();
   System.out.println("Value of i: " + obj.i);
 }
}
```

```
 // =============== output ======================
  // PS C:\Users\hp\Desktop\TCS IT\Java> java Day5/Main.java
  // Constructor is called
  // Value of i: 5

 */



//=========================================================

/*
// Example 1: Java program to create a private constructor
class Test{
  // create private constructor
  private Test(){
    System.out.println("Private constructor called");
  }

  // create a public static method
  public static void instanceMethod(){
    // create an instance of Test class
    Test obj = new Test();
  }
}
public class Main {

  public static void main(String[] args) {
    // call the instanceMethod()
    Test.instanceMethod();
  }
}

// ============== output ================
// PS C:\Users\hp\Desktop\TCS IT\Java> javac .\Day5\Main.java
// PS C:\Users\hp\Desktop\TCS IT\Java> java Day5.Main
// Private constructor called
// PS C:\Users\hp\Desktop\TCS IT\Java>

*/



//=====================================================


/*
// 2. Java parameterized constructor
// A java constructor can also accept one or more parameters. Such constructors are known as
parameterized constructors (constructor with parameters)
```

```
public class Main {

  String languages;
  // constructor accepting single value
  Main(String lang){
    languages = lang;
    System.out.println(languages + " Programming Languages");
  }

  public static void main(String[] args) {
    // call constructor by passing a single value
    Main obj1 = new Main("Java");
    Main obj2 = new Main("Python");
    Main obj3 = new Main("C++");
  }
}

// ========== output ================
// PS C:\Users\hp\Desktop\TCS IT\Java> javac .\Day5\Main.java
// PS C:\Users\hp\Desktop\TCS IT\Java> java Day5/Main.java
// Java Programming Languages
// Python Programming Languages
// C++ Programming Languages
// PS C:\Users\hp\Desktop\TCS IT\Java>
 */


 //=====================================================

/*
 // 3. Java Default constructor
 // If we do not create any constructor, the java compiler automatically create a no-argument
constructor during the execution of the program. This constructor is called default
constructor.


 public class Main {
  int a;
  boolean b;

  public static void main(String[] args) {
    // a default constructor is called
    Main obj = new Main();
    System.out.println("Default value: ");
    System.out.println("a = " + obj.a);
    System.out.println("b = " + obj.b);
  }
```

```
}


// ============================ output ============================
 // PS C:\Users\hp\Desktop\TCS IT\Java> java Day5/Main.java
 // Default value:
 // a = 0
 // b = false
 // PS C:\Users\hp\Desktop\TCS IT\Java>

*/




//================================================================


// Constructors Overloading in java
// Similar to java method overloading, we can also create two or more constructors with different paramenters. This is called constructors overloading.


public class Main {
 String language;
 Main(){
   this.language = "Java";
 }

 Main(String language){
   this.language = language;
 }

 public void getName(){
   System.out.println("Programming Language: " + this.language);
 }

 public static void main(String[] args) {
   Main obj1 = new Main();
   Main obj2 = new Main("Python");
   obj1.getName();
   obj2.getName();
 }
}

// ============== output ======================
//  PS C:\Users\hp\Desktop\TCS IT\Java> java Day5/Main.java
//  Programming Language: Java
//  Programming Language: Python
//  PS C:\Users\hp\Desktop\TCS IT\Java>
```

Java Strings



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

    // create strings
    String first = "Java";
    String second = "Python";
    String third = "JavaScript";

    // print strings
    System.out.println(first);   // print Java
    System.out.println(second);  // print Python
    System.out.println(third);   // print JavaScript
  }
}
```


Java String Operations

1. Get Length of a String

To find the length of a string, we use the length() method of the String. For example,

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

    // create a string
    String greet = "Hello! World";
    System.out.println("String: " + greet);

    // get the length of greet
    int length = greet.length();
    System.out.println("Length: " + length);
  }
}
```

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

2. Join two Strings

We can join two strings in Java using the concat() method. For example,

```
class Main {
  public static void main(String[] args) {
    // create first string
    String first = "Java ";
    System.out.println("First String: " + first);
    // create second
    String second = "Programming";
    System.out.println("Second String: " + second);
    // join two strings
    String joinedString = first.concat(second);
    System.out.println("Joined String: " + joinedString);
  }
}
```


22:10 | 14 1x

3. Compare two Strings

In Java, we can make comparisons between two strings using the equals() method. For example,

```
class Main {
  public static void main(String[] args) {
    // create 3 strings
    String first = "java programming";
    String second = "java programming";
    String third = "python programming";
    // compare first and second strings
    boolean result1 = first.equals(second);
    System.out.println("Strings first and second are equal: " + result1);
    // compare first and third strings
    boolean result2 = first.equals(third);
    System.out.println("Strings first and third are equal: " + result2);
  }
}
```

Methods of Java String

Methods	Description
substring()	returns the substring of the string
replace()	replaces the specified old character with the specified new character
charAt()	returns the character present in the specified location
getBytes()	converts the string to an array of bytes
indexOf()	returns the position of the specified character in the string
compareTo()	compares two strings in the dictionary order
trim()	removes any leading and trailing whitespaces
format()	returns a formatted string
split()	breaks the string into an array of strings
toLowerCase()	converts the string to lowercase
toUpperCase()	converts the string to uppercase
valueOf()	returns the string representation of the specified argument
toCharArray()	converts the string to a char array

Creating strings using the new keyword

Since strings in Java are objects, we can create strings using the new keyword as well. For example,

```
// create a string using the new keyword
String name = new String("Java String");
```

In the above example, we have created a string name using the new keyword.

Here, when we create a string object, the String() constructor is invoked.

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Example: Create Java Strings using the new keyword

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

    // create a string using new
    String name = new String("Java String");

    System.out.println(name);  // print Java String
  }
}
```


7:11 1.5x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Create String using literals vs new keyword

In Java, the JVM maintains a string pool to store all of its strings inside the memory. The string pool helps in reusing the strings.

1. While creating strings using string literals,
String example = "Java";
Here, we are directly providing the value of the string (Java). Hence, the compiler first checks the string pool to see if the string already exists.

If the string already exists, the new string is not created. Instead, the new reference, example points to the already existed string (Java).
If the string doesn't exist, the new string (Java is created.

2. While creating strings using the new keyword,

String example = new String("Java");
Here, the value of the string is not directly provided. Hence, the new string is created all the time.

6:38 1.5x

Java Access Modifiers

What are Access Modifiers?

In Java, access modifiers are used to set the accessibility (visibility) of classes, interfaces, variables, methods, constructors, data members, and the setter methods. For example,

```
class Animal {
   public void method1() {...}

  private void method2() {...}
}
```

In the above example, we have declared 2 methods: method1() and method2(). Here,

method1 is public - This means it can be accessed by other classes.
method2 is private - This means it can not be accessed by other classes.
Note the keyword public and private. These are access modifiers in Java. They are also known as visibility modifiers.

Note: You cannot set the access modifier of getters methods.

```java
package Day5;
// java String

/*public class java_String {
  public static void main(String[] args) {
    // create strings
    String first = "java";
    String second = "Python";
    String third = "JavaScript";

    // print strings
    System.out.println(first); // print java
    System.out.println(second); // print Python
    System.out.println(third); // print JavaScript
  }
}


// ============= output ========================
// PS C:\Users\hp\Desktop\TCS IT\Java> javac .\Day5\java_String.java
// PS C:\Users\hp\Desktop\TCS IT\Java> java .\Day5\java_String.java
// java
// Python
// JavaScript
// PS C:\Users\hp\Desktop\TCS IT\Java>


 */




 //-------------------------------------------------------------

 // java string operation


 /*
 // 1. Get Length of a string
 // To find the length of a string, we use the length() method of the String. For example:

 public class java_String {

  public static void main(String[] args) {
    // create strings
    String greet = "Hello! World";
    System.out.println("Length of the string is: " + greet.length());

    // get the length of greet
    int length = greet.length();
    System.out.println("Length: " + length);
```

```
  }
 }

 // ============= output ==============
//  PS C:\Users\hp\Desktop\TCS IT\Java> java .\Day5\java_String.java
// Length of the string is: 12
// Length: 12
// PS C:\Users\hp\Desktop\TCS IT\Java>

 */


//-----------------------------------------------------------------

/*
// 2. Join two Strings
// We can join two strings in java using the concat() method. For example:

public class java_String {

  public static void main(String[] args) {
    // create strings
    String first = "Java";
    System.out.println("First String: " + first);

    // create second
    String second = "Programming";
    System.out.println("Second String: " + second);

    // join two strings
    String joinedString = first.concat(second);
    System.out.println("Joined String: " + joinedString);
  }
}

// ========== output ==============
// PS C:\Users\hp\Desktop\TCS IT\Java> java .\Day5\java_String.java
// First String: Java
// Second String: Programming
// Joined String: JavaProgramming
 */


 // ---------------------------------------------------------

/*
 // 3. Compare two Strings
 // In java, we can make comparisons between two strings using the equals() method. for
example:
```

```
public class java_String {

  public static void main(String[] args) {
    // create strings
    String first = "Java Programming";
    String second = "Java Programming";
    String third = "C++ Programming";

    // compare first and second strings
    boolean result1 = first.equals(second);
    System.out.println("First and Second Strings are equal: " + result1);

    // compare first and third strings
    boolean result2 = first.equals(third);
    System.out.println("First and Third Strings are equal: " + result2);
  }
 }

 //============== output ==================
//  PS C:\Users\hp\Desktop\TCS IT\Java> java .\Day5\java_String.java
// First and Second Strings are equal: true
// First and Third Strings are equal: false

 */



//-----------------------------------------------------------------

// Example: Create java Strings using the new keyword

public class java_String {

  public static void main(String[] args) {
    // create strings using new keyword
    String name = new String("Java String");

    System.out.println(name); // print Java String
  }
}

// ================ output ====================
// PS C:\Users\hp\Desktop\TCS IT\Java> java .\Day5\java_String.java
// Java String
```