==========

Tutor: **Omprakash Chavan**

Reference: **UDEMY**

Course: **Selenium Java Test Framework & Best Practices - Masterclass**

Content: **Programming language - JAVA**

==========

1. Course URL: https://www.udemy.com/course/selenium-java-test-framework/
2. Document prepared by: **Rajat Verma**
    a. https://www.linkedin.com/in/rajat-v-3b0685128/
    b. https://github.com/rajatt95
    c. https://rajatt95.github.io/

Softwares:

1. Programming language - Java
2. IDE - IntelliJ/Eclipse

---

1. **Learnings from Course (UDEMY - OC)**
    a. **Links:**
        i. **Java**
        ii. **IDE-IntelliJ**
            1. https://www.jetbrains.com/idea/download/#section=mac

—----------

JAVA - START

—----------

b. **Java:**
    i. **Basics:**
        1. Package
        2. Class
            a. Components:
                i. Fields/Parameters/Variables
                ii. Constructors
                iii. Methods/Behavior
        3. Object:
        4. Method

5. Variables
    a. Initialize Variable using Method
    b. Initialize Variable using Default Constructor
    c. Initialize Variable using Parameterized Constructor
6. Constructor
    a. With Arguments
    b. Without Arguments
7. Data Types:
    a. Primitive
        i. int, float, double, boolean
    b. Reference
        i. String
8. Strings

```java
package com.learning.java;

public class _15_String_Operations {

    static String str = "Hello, Test Automation Engineer!";

    public static void main(String[] args) {
        System.out.println("str = " + str); // str = Hello, Test Automation Engineer!
        System.out.println("str.toLowerCase() = " + str.toLowerCase()); // hello, test automation engineer!
        System.out.println("str.toUpperCase() = " + str.toUpperCase()); // HELLO, TEST AUTOMATION ENGINEER!
        System.out.println("str.charAt(1) = " + str.charAt(1)); // e
        System.out.println("str.substring(5) = " + str.substring(5)); // , Test Automation Engineer!
        System.out.println("str.substring(5,11) = " + str.substring(5, 11)); // , Test
        System.out.println("str.contains(\"Test\") = " + str.contains("Test")); // true
        System.out.println("str.length() = " + str.length()); // 32
        System.out.println("str.indexOf('T') = " + str.indexOf('T')); // 7
        System.out.println("str.concat(\" -> Rajat\") = " + str.concat(" -> Rajat")); // Hello, Test Automation Engineer! -> R
        System.out.println("str.equals(\"Test\") = " + str.equals("Test")); // false
        System.out.println("str.equalsIgnoreCase(\"Hi\") = " + str.equalsIgnoreCase( anotherString: "Hi")); // false

    }// main
}// class
```

    a. **Coversion:**
        i. String to int -> **Integer.parseInt(str)**
        ii. int to String -> **String.valueOf(number1)**

9. Constants
    a. Access constants from other files
10. Enums
    a. The special class that represents a group of constants
    b. Access Enum constants from other files
11. Keyword:
    a. static:
        i. Shared between all objects
    b. final:
        i. Assign value only once
    c. return
    d. this
        i. Returns class object

e. super
    i. It is used to differentiate members of Superclass from members of Subclass if they have the same names
    ii. It is used to invoke the constructor of Superclass from Subclass

ii. Control Structures:

1. If …. Else
    a. Multiple Conditions
        i. **&** -> Both the conditions have to be true
        ii. **|** -> One condition has to be true
2. For
3. For Each
4. While
5. Switch-Case
    a. With String
    b. With Enum

iii. OOP - Inheritance:

1. Inheritance

a. It is the mechanism by which one class acquires properties (Fields and Methods) of another class
b. Why?
    i. Re-Usability
c. How?
    i. Using **extends** keyword
d. Sub/Derived/Child class inherits the properties of Super/Base/Parent class
e. Types of Inheritance:
    i. Single inheritance
    ii. Multilevel inheritance
    iii. Hierarchical inheritance
    iv. Multiple inheritance (using interface)

iv. Access Modifiers:

1. public, protected, private, default

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

2.

3. Applicability:

a. Class: public, default

b. Attributes and methods: public, private, protected, default

v. **OOP - Encapsulation**

1. **Encapsulation:**
    a. Wrapping up of **data under a single unit**! The data is protected!
    b. Use getter and setter methods to access private attributes

vi. **OOP - Polymorphism**

1. **Polymorphism:**
    a. The ability of an object to take many forms!
    b. **How?**
        i. By Overriding or Overloading methods.
    c. Polymorphism Types
        i. **Runtime polymorphism**
            1. E.g. Method Overriding
        ii. **Compile-time polymorphism**:
            1. E.g. Method Overloading
    d. Notes:
        i. final methods cannot be overridden
        ii. Static methods cannot be overridden (method hiding)

vii. **OOP - Abstraction using Abstract class**

1. **Abstraction:**
    a. Hide details and show only essential information!
    b. Partial abstraction (0 to 100%).

viii. **OOP - Abstraction using Interface**

1. **Abstraction:**
    a. Hide details and show only essential information!
    b. 100% abstraction.

    c. Interface
        i. Java8: Can have a default method
        ii. Java8: Can have a static method

    d. Advantages:
        i. Helps achieve multiple inheritance

ix. **Exception Handling**

1. **Exception types**
    a. Compile-time exception (checked exceptions)

i. FileNotFoundException

b. Runtime exception (unchecked exception)

i. ArrayIndexOutOfBoundsException

x. File Operations

1. Creation of a new file
2. Opening an existing file
3. Reading from file
4. Writing to a file
5. Closing a file
6. Deleting a file

7. **Java classes**

i. FileReader, BufferedReader, Files, Scanner, FileInputStream, FileWriter, BufferedWriter,

ii. FileOutputStream, etc.

—----------

JAVA - END

—----------

==========

========4_JAVA Essentials [Optional]==========

—--------19. Section Intro —----------

1. **Basics:**
    - **a.** Class
    - b. Method
    - c. Variable
    - d. Constructor
    - e. Strings
    - f. Constants
    - g. Enums

2. **Control Structures:**
    - a. Switch-Case
    - b. For
    - c. If …. else
    - d. For Each
3. **OOP concepts:**
    - **a.** Inheritance
    - b. Encapsulation
    - c. Abstraction
    - d. Polymorphism

4. **Access Modifiers:**
    - a. public
    - b. protected
    - c. private

5. **Exception Handling**

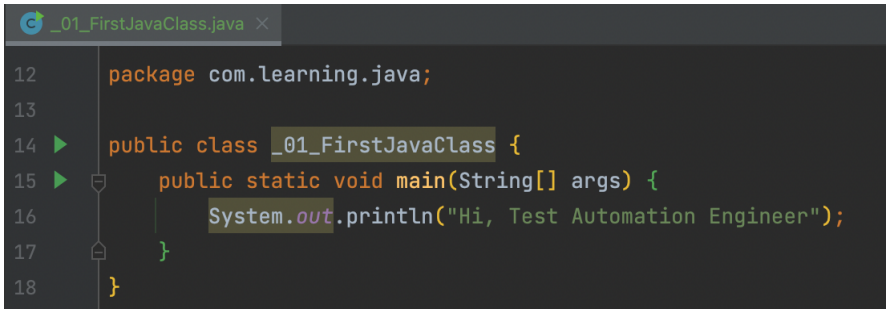
6. **File Operations**

—--------20. Java Basics - Part 1 —----------

1. Package
    a. Naming:
        i. lower case
        ii. Example:
            1. com.learning.java
2. Class
    a. Naming:
        i. Camel Case
        ii. Example:
            1. MyFirstJavaClass
3. Object:
    a. Naming:
        i. lower case
4. Method
5. Data Types:
    a. Primitive
        i. int, float, double, boolean
    b. Reference
        i. String
6. Variables
    a. Naming:
        i. lower case
7. Constructor
    a. With Arguments
    b. Without Arguments
8. Keyword:
    a. static:
        i. Shared between all objects
    b. final:
        i. Assign value only once
    c. return
    d. this
        i. Returns class object

9. Constants
10. Enums
    a. The special class that represents a group of constants
11. Strings
12. Data Type conversions

—--------21. Java Basics - Part 2 —----------

1. 

—----------

1. **Class has:**
    a. Fields/Parameters/Variables
    b. Constructors
    c. Methods/Behavior

2. **Data Types:**
    a. **Primitive:**
        i. int
        ii. float
        iii. double
        iv. boolean
    b. **Reference**
        i. Strings

—----------

1. _05_Constructor.java

```java
package com.learning.java;

public class _05_Constructor {

    // Constructor
    public _05_Constructor(){
        System.out.println("Default Constructor"); // Gets called when object is created
    }// Constructor

    // Class variable
    int number1 = 10;

    // Pre-Defined method
    public static void main(String[] args) {
        System.out.println("Hi, Test Automation Engineer");

        // Creating object of class
        // ClassName objectName = new ClassName();
        _05_Constructor obj1 = new _05_Constructor();

        System.out.println("obj1.number1 = " + obj1.number1);
    }// main
}// class
```

2. _06_CustomMethod.java

```java
package com.learning.java;

public class _06_CustomMethod {

    // Pre-Defined method
    public static void main(String[] args) {
        displayMessage();
    }// main

    public static void displayMessage() { System.out.println("Hi, Test Automation Engineer"); }// displayMessage
}// class
```

3.

_07_InitializeVariable_Method.java

```java
package com.learning.java;

public class _07_InitializeVariable_Method {

    // Variable/Field/Parameter
    int number1;

    // Pre-Defined method
    public static void main(String[] args) {

        System.out.println("Hi, Test Automation Engineer");

        _07_InitializeVariable_Method obj1 = new _07_InitializeVariable_Method();
        obj1.initialize();
        System.out.println("obj1.number1 = " + obj1.number1);

    }// main

    public void initialize() { number1 = 100; }// initialize
}// class
```

4.

_08_InitializeVariable_DefaultConstructor.java

```java
package com.learning.java;

public class _08_InitializeVariable_DefaultConstructor {

    // Variable/Field/Parameter
    int number1;

    // Constructor
    public _08_InitializeVariable_DefaultConstructor() { number1 = 100; }

    // Pre-Defined method
    public static void main(String[] args) {

        System.out.println("Hi, Test Automation Engineer");

        _08_InitializeVariable_DefaultConstructor obj1 = new _08_InitializeVariable_DefaultConstructor();
        System.out.println("obj1.number1 = " + obj1.number1);
    }
}
```

5.

_09_InitializeVariable_ParameterizedConstructor.java

```java
public class _09_InitializeVariable_ParameterizedConstructor {

    // Variable/Field/Parameter
    int number1;

    // Parameterized Constructor
    public _09_InitializeVariable_ParameterizedConstructor(int number){
        // this is keyword used to access class level variables
        this.number1 = number;
    }

    // Pre-Defined method
    public static void main(String[] args) {

        System.out.println("Hi, Test Automation Engineer");

        _09_InitializeVariable_ParameterizedConstructor obj1 = new _09_InitializeVariable_ParameterizedConstructor( number: 200);
        System.out.println("obj1.number1 = " + obj1.number1);
    }// main
}// class
```

—--------22. Java Basics - Part 3 —----------

1.

_10_Constants.java

```java
package com.learning.java;

public class _10_Constants {

    // Variables
    public static final int LONG_WAIT = 30;
    public static final int SHORT_WAIT = 5;


}// class
```

2.

_10_Constants.java _11_AccessConstants.java

```java
/.../

package com.learning.java;

public class _11_AccessConstants {

    public static void main(String[] args) {

        System.out.println("_10_Constants.LONG_WAIT = " + _10_Constants.LONG_WAIT);
        System.out.println("_10_Constants.SHORT_WAIT = " + _10_Constants.SHORT_WAIT);

    }// main

}// class
```

—----------

1. _10_Constants.java | _11_AccessConstants.java | _12_EnumConstants.java

```java
/.../

package com.learning.java;

public enum _12_EnumConstants {

    SUNDAY,
    MONDAY;

}// class
```

2. _12_EnumConstants.java | _13_AccessEnumConstants.java

```java
/.../

package com.learning.java;

public class _13_AccessEnumConstants {

    public static void main(String[] args) {
        System.out.println("_12_EnumConstants.SUNDAY = " + _12_EnumConstants.SUNDAY);
        System.out.println("_12_EnumConstants.MONDAY = " + _12_EnumConstants.MONDAY);
    }// main
}// class
```

—----------

—--------23. Java Basics - Part 4 —----------

1. _15_String_Operations.java

```java
package com.learning.java;

public class _15_String_Operations {

    static String str = "Hello, Test Automation Engineer!";

    public static void main(String[] args) {
        System.out.println("str = " + str); // str = Hello, Test Automation Engineer!
        System.out.println("str.toLowerCase() = " + str.toLowerCase()); // hello, test automation engineer!
        System.out.println("str.toUpperCase() = " + str.toUpperCase()); // HELLO, TEST AUTOMATION ENGINEER!
        System.out.println("str.charAt(1) = " + str.charAt(1)); // e
        System.out.println("str.substring(5) = " + str.substring(5)); // , Test Automation Engineer!
        System.out.println("str.substring(5,11) = " + str.substring(5, 11)); // , Test
        System.out.println("str.contains(\"Test\") = " + str.contains("Test")); // true
        System.out.println("str.length() = " + str.length()); // 32
        System.out.println("str.indexOf('T') = " + str.indexOf('T')); // 7
        System.out.println("str.concat(\" -> Rajat\") = " + str.concat(" -> Rajat")); // Hello, Test Automation Engineer! -> Ra
        System.out.println("str.equals(\"Test\") = " + str.equals("Test")); // false
        System.out.println("str.equalsIgnoreCase(\"Hi\") = " + str.equalsIgnoreCase( anotherString: "Hi")); // false

    }// main
}// class
```

2. _16_Convert_String_To_Int.java

```java
/.../

package com.learning.java;

public class _16_Convert_String_To_Int {

    public static void main(String[] args) {
        String str = "5";
        System.out.println("Integer.parseInt(str) = " + Integer.parseInt(str));
    }// main
}// class
```

3. _17_Convert_Int_To_String.java

```java
/.../

package com.learning.java;

public class _17_Convert_Int_To_String {

    public static void main(String[] args) {
        int number1 = 10;
        System.out.println("String.valueOf(number1) = " + String.valueOf(number1));

    }// main
}// class
```

—----------

—--------24. Control Structures —----------

1. **Control Structures:**
    a. If …. Else

i.

_18_ControlStructure_IfElse.java

```java
package com.learning.java;

public class _18_ControlStructure_IfElse {

    public static void main(String[] args) {
        int number1 = 10;

        if (number1>5){
            System.out.println("number1: "+number1+" is greater than 5");
        }else if (number1 ==5 ){
            System.out.println("number1: "+number1+" is equal to 5");
        }else {
            System.out.println("number1: "+number1+" is lesser than 5");
        }

    }// main
}// class
```

ii.

_19_CS_IfElse_Condition_AND.java

```java
package com.learning.java;

public class _19_CS_IfElse_Condition_AND {

    public static void main(String[] args) {

        // & -> Both the conditions has to be true
        if(true & true){
            System.out.println("true & true");
            System.out.println("Inside if block"); // Inside if block
        }else{
            System.out.println("Inside else block");
        }

        if(true & false){
            System.out.println("Inside if block");
        }else{
            System.out.println("true & false");
            System.out.println("Inside else block"); // Inside else block
        }

    }// main
}// class
```

iii.

_19_CS_IfElse_Condition_OR.java

```java
public class _19_CS_IfElse_Condition_OR {

    public static void main(String[] args) {

        // | -> One condition has to be true

        if(true | true){
            System.out.println("true | true");
            System.out.println("Inside if block"); // Inside if block
        }else{
            System.out.println("Inside else block");
        }

        if(true | false){
            System.out.println("true | false");
            System.out.println("Inside if block"); // Inside if block
        }else{
            System.out.println("Inside else block");
        }

    }// main
}// class
```

b. For

i.

_21_CS_Loop_FOR_Break.java

```java
package com.learning.java;

public class _21_CS_Loop_FOR_Break {

    public static void main(String[] args) {

        for (int i=1;i<=5;i++){
            System.out.print(i+", "); // 1, 2, 3,
            if(i == 3){
                break; // Come out of the FOR loop when value of variable 'i' is 3
            }
        }

    }// main
}// class
```

c. For Each

i.

_21_CS_Loop_FOR_Break.java | _23_CS_Loop_FOR_EACH_Break.java

```java
/.../

package com.learning.java;

public class _23_CS_Loop_FOR_EACH_Break {

    public static void main(String[] args) {

        String[] strArray ={"Shreya","Shweta","Shruti","Preeti","Tanya"};

        for (String str: strArray) {
            System.out.println(str);
            if(str.equalsIgnoreCase( anotherString: "Shruti")){
                break; // Come out of the FOR EACH loop when value of variable 'str' is Shruti
            }
        }

    }// main
}// class
```

d. While

i.

_25_CS_Loop_WHILE_Break.java

```java
package com.learning.java;

public class _25_CS_Loop_WHILE_Break {

    public static void main(String[] args) {

        int i = 5;
        while (i>=0){
            System.out.println("i = " + i);
            i--;
            if(i == 3){
                break; // Come out of the WHILE loop when value of variable 'i' is 3
            }
        }

    }// main
}// class
```

e. Switch-Case

i.

_26_CS_SwitchCase.java

```java
package com.learning.java;

public class _26_CS_SwitchCase {

    public static void main(String[] args) {

      String str = "a";

      switch (str){
          case "a":
              System.out.println("Value is a"); // Value is a
              break;
          case "b":
              System.out.println("Value is b");
              break;
          default:
              System.out.println("Option not-in-scope");
      }
    }// main
}// class
```

ii.

_27_CS_SwitchCase_WithEnum.java

```java
public class _27_CS_SwitchCase_WithEnum {

    enum custom_Char{
        a,b;
    }
    public static void main(String[] args) {
      custom_Char required_Char =  custom_Char.a;

      switch (required_Char){
          case a:
              System.out.println("Value is a"); // Value is a
              break;
          case b:
              System.out.println("Value is b");
              break;
          default:
              System.out.println("Option not-in-scope");
      }
    }// main
}// class
```

—--------25. OOP - Inheritance and super Keyword —----------

1. **Inheritance:**
    a. It is the mechanism by which one class acquires properties (Fields and Methods) of another class
    b. Why?
        i. Re-Usability
    c. How?
        i. Using **extends** keyword
    d. Classes:
        i. SubClass | DerivedClass | ChildClass
        ii. SuperClass | BaseClass | ParentClass
    e. Points:
        i. A subclass can have its own methods and fields in addition to Superclass's methods and fields
        ii. A subclass can have only one Superclass. In other words, multiple inheritance is not supported by using classes
        iii. A subclass cannot inherit Superclass's constructor, but it can invoke the constructor
        iv. Sub/Derived/Child class inherits the properties of Super/Base/Parent class

2. Types of Inheritance:
    a. Single inheritance
    b. Multilevel inheritance
    c. Hierarchical inheritance
    d. Multiple inheritance (using interface)

3. **super** keyword:
    a. It is used to differentiate members of Superclass from members of Subclass if they have the same names
    b. It is used to invoke the constructor of Superclass from Subclass

—----------

1.

_28_OOPS_Inheritance_SuperClass.java

```java
/.../

package com.learning.java;

public class _28_OOPS_Inheritance_SuperClass {

    // Field - Variable - Parameter
    String superStr = "Super class Field";
    String commonStr = "Super common String";


    // Constructor
    public _28_OOPS_Inheritance_SuperClass() { System.out.println("Super class Constructor"); }

    // Method
    public void superClassMethod() { System.out.println("Super class Method"); }
}// class
```

2.

_29_OOPS_Inheritance_SubClass.java

```java
package com.learning.java;

public class _29_OOPS_Inheritance_SubClass extends _28_OOPS_Inheritance_SuperClass{

    // Fields
    String commonStr = "Sub common String";

    // Constructor
    public _29_OOPS_Inheritance_SubClass(){
        System.out.println("Sub class Constructor");
    }

    // Method
    public void subClassMethod() { System.out.println("Sub class Method"); }

    public void printCommonString(){
        System.out.println("--------------------");
        System.out.println("commonStr = " + commonStr); // Sub common String
        System.out.println("super.commonStr = " + super.commonStr); // Super common String
    }
}// class
```

3.

```java
package com.learning.java;

public class _30_OOPS_Inheritance_App {

    public static void main(String[] args) {
        _29_OOPS_Inheritance_SubClass obj = new _29_OOPS_Inheritance_SubClass(); // Super class Constructor
                                                                                 // Sub class Constructor

        // Super class properties
        System.out.println("obj.superStr = " + obj.superStr); // obj.superStr = Super class Field
        obj.superClassMethod(); // Super class Method

        // Sub class properties
        obj.subClassMethod(); // Sub class Method

        obj.printCommonString();
    }
}// class
```

—--------26. Access Modifiers —----------

1. **Access Modifiers:**
    a. public, protected, private, default
    b.

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

2. **Definition**
    a. public: Everywhere
    b. protected: by class in the same package or by subclass if the outside package
    c. private: Within the class
    d. default: by class in the same package

3. **Applicability:**
    a. Class: public, default
    b. Attributes and methods: public, private, protected, default

4. **NOTES:**
    a. Use private for attributes unless there is a good reason not to do so
    b. Use public for constants
    c. Use protected if you want class members to be accessed by subclasses or by classes in the same package
    d. Use private if you think the method will be accessed only within the class

—--------27. OOP - Encapsulation —----------

1. **Encapsulation:**
    a. Wrapping up of **data under a single unit**! The data is protected!
    b. **How**?
        i. By making class attributes (or variables) private
        ii. By making methods private
    c. Use getter and setter methods to access private attributes
    d. Advantages
        i. Data hiding
        ii. Flexibility to use variables as read-only or write-only
        iii. Reusability

—----------

1. _31_OOPS_Inheritance_Encapsulation.java

```java
package com.learning.java;

public class _31_OOPS_Inheritance_Encapsulation {

    // Encapsulation:
        // Wrapping up of data under a single unit! The data is protected!

    // Field - Private
    private String str ="Private String";

    // Method - Public
    public String getPrivateString(){
        return str;
    }
}// class
```

2. _32_OOPS_Inheritance_Encapsulation_App.java

```java
/.../

package com.learning.java;

public class _32_OOPS_Inheritance_Encapsulation_App {

    public static void main(String[] args) {
        _31_OOPS_Inheritance_Encapsulation obj = new _31_OOPS_Inheritance_Encapsulation();
        System.out.println("obj.getPrivateString() = " + obj.getPrivateString()); // Private String
    }// main
}// class
```

—--------28. OOP - Polymorphism —----------

1. **Polymorphism:**
    a. The ability of an object to take many forms!
    b. **How?**
        i. By Overriding or Overloading methods.

    c. Method Overriding
        i. Allows a subclass to provide a specific implementation of a method that is already provided by its superclass
        ii. The method in the subclass should have the same name, same signature, and same return type(or sub-type) as the method in its superclass

    d. Method Overloading
        i. Allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both

    e. Polymorphism Types
        i. **Runtime polymorphism** or Dynamic method dispatch:
            1. Call resolved at runtime based on the type of the object being referred to at the time the call occurs
            2. E.g. Method Overriding
        ii. **Compile-time polymorphism**: Call resolved at compile time
            1. E.g. Method Overloading

    f. Notes:
        i. final methods cannot be overridden
        ii. Static methods cannot be overridden (method hiding)

—----------

—----------

1. **Method Overriding:**

a.

_33_OOPS_Polymorphism_Runtime_Employee.java

```java
package com.learning.java;

public class _33_OOPS_Polymorphism_Runtime_Employee {

    int base = 20000;

    void salary(){
        System.out.println("Employee Salary: " + base);
    }

    static void designation(){
        System.out.println("Tester");
    }
}// class
```

b.

_34_OOPS_Polymorphism_Runtime_FullTime.java

```java
package com.learning.java;

// OOPS - Inheritance
    // _34_OOPS_Polymorphism_Runtime_FullTime - Sub/Derived/Child Class
    // _33_OOPS_Polymorphism_Runtime_Employee - Super/Base/Parent Class
public class _34_OOPS_Polymorphism_Runtime_FullTime extends  _33_OOPS_Polymorphism_Runtime_Employee{

    // OOPS Polymorphism
        // Runtime
            // Method Overriding
    @Override
    void salary(){
        System.out.println("Employee (Full Time) Salary: " + (base+20000));
    }

    void methodFullTime() { System.out.println("Method in Full Time class"); }

    // We can not override static and final methods
    //@Override
    static void designation() { System.out.println("Tester: Full Time"); }

}// class
```

c.

_35_OOPS_Polymorphism_Runtime_Contract.java

```java
package com.learning.java;

// OOPS - Inheritance
    // _35_OOPS_Polymorphism_Runtime_Contractor - Sub/Derived/Child Class
    // _33_OOPS_Polymorphism_Runtime_Employee - Super/Base/Parent Class
public class _35_OOPS_Polymorphism_Runtime_Contract extends  _33_OOPS_Polymorphism_Runtime_Employee{

    // OOPS Polymorphism
        // Runtime
            // Method Overriding
    @Override
    void salary() { System.out.println("Employee (Contract) Salary: " + (base+10000)); }

    void methodContract(){
        System.out.println("Method in Contract class");
    }

    // We can not override static and final methods
    //@Override
    static void designation() { System.out.println("Tester: Contract"); }

}// class
```

d.

_36_OOPS_Polymorphism_Runtime_App.java

```java
package com.learning.java;

public class _36_OOPS_Polymorphism_Runtime_App{

    public static void main(String[] args) {

        // _33_OOPS_Polymorphism_Runtime_Employee -> Reference
        // _34_OOPS_Polymorphism_Runtime_FullTime -> Type of the Object
        // At Runtime, method will be called/executed on the basis of type of Object
        _33_OOPS_Polymorphism_Runtime_Employee employee_FT = new _34_OOPS_Polymorphism_Runtime_FullTime();
        employee_FT.salary(); // Employee (Full Time) Salary: 40000

        _33_OOPS_Polymorphism_Runtime_Employee employee_C = new _35_OOPS_Polymorphism_Runtime_Contract();
        employee_C.salary(); // Employee (Contract) Salary: 30000

        // Method specific to child class
        _34_OOPS_Polymorphism_Runtime_FullTime employeeFullTime = new _34_OOPS_Polymorphism_Runtime_FullTime();
        employeeFullTime.methodFullTime(); // Method in Full Time class


        /********** STATIC METHODS *************/
        // We can not override static and final methods
        _33_OOPS_Polymorphism_Runtime_Employee.designation(); // Tester
        _34_OOPS_Polymorphism_Runtime_FullTime.designation(); // Tester: Full Time
        _35_OOPS_Polymorphism_Runtime_Contract.designation(); // Tester: Contract
    }// main
}// class
```

—----------

—----------

2. Method Overloading:

a.

_37_OOPS_Polymorphism_CompileTime.java

```java
package com.learning.java;

public class _37_OOPS_Polymorphism_CompileTime {

    // Method Overloading
    public void display(String name){
        System.out.println("-----------------");
        System.out.println("name = " + name);
    }

    public void display(String name, int age){
        System.out.println("-----------------");
        System.out.println("name = " + name);
        System.out.println("age = " + age);
    }

    public void display(String name, int age, String country){
        System.out.println("-----------------");
        System.out.println("name = " + name);
        System.out.println("age = " + age);
        System.out.println("country = " + country);
    }

}// class
```

b.

_38_OOPS_Polymorphism_CompileTime_App.java

```java
/.../

package com.learning.java;

public class _38_OOPS_Polymorphism_CompileTime_App {

    public static void main(String[] args) {
        _37_OOPS_Polymorphism_CompileTime obj = new _37_OOPS_Polymorphism_CompileTime();

        obj.display( name: "Rajat Verma"); // name = Rajat Verma

        obj.display( name: "Rajat Verma",  age: 27); //name = Rajat Verma
                                                    // age = 27

        obj.display( name: "Rajat Verma", age: 27,  country: "India"); //name = Rajat Verma
                                                                       // age = 27
                                                                       // country = India

    }

}// class
```

—----------

—--------29. OOP - Abstraction using Abstract Class —----------

1. **Abstraction:**
    a. Hide details and show only essential information!
    b. Partial abstraction (0 to 100%).

    c. Abstract class
        i. Provides partial abstraction
        ii. An abstract method is declared without an implementation
        iii. An abstract class cannot be directly instantiated
        iv. A subclass can access the Abstract class using extends keyword
        v. A subclass must implement all abstract methods i.e. Overriding is compulsory
        vi. An abstract class can have default and parameterized constructor
            1. But, we can not create the object of an Abstract class

    d. Advantages:
        i. Reduces complexity by hiding implementation
        ii. Better viewing
        iii. Avoids code duplication and promotes reusability
        iv. Increases security by providing only important details to the user

    e. Example
        i. Shapes: Abstract
        ii. Triangle: SubClass1
        iii. Square: SubClass2

—----------

1.

_39_OOPS_Abstraction_AbstractClass_Shape.java

```java
package com.learning.java;


public abstract class _39_OOPS_Abstraction_AbstractClass_Shape {

    String color;

    // Default Constructor in Abstract class
    //public _39_OOPS_Abstraction_AbstractClass_Shape(){}

    // Parameterized Constructor in Abstract class
    public _39_OOPS_Abstraction_AbstractClass_Shape(String color) { this.color = color; }

    // Abstract method
    abstract double area();
    abstract String info();

    // Concrete method
    public String getColor() { return color; }
}// class
```

2.

_40_OOPS_Abstraction_AbstractClass_Circle.java

```java
/.../

package com.learning.java;


public class _40_OOPS_Abstraction_AbstractClass_Circle extends  _39_OOPS_Abstraction_AbstractClass_Shape{

    double radius;

    public _40_OOPS_Abstraction_AbstractClass_Circle(String color, double radius) {
        super(color);
        this.radius = radius;
    }

    // area() and colorInfo() -> Abstract methods coming from _39_OOPS_Abstraction_AbstractClass_Shape
    @Override
    double area() { return Math.PI * Math.pow(radius, 2); }

    @Override
    String info() { return "I'm a "+super.color+" Circle with Area: "+area(); }
}// class
```

3.

_41_OOPS_Abstraction_AbstractClass_Square.java

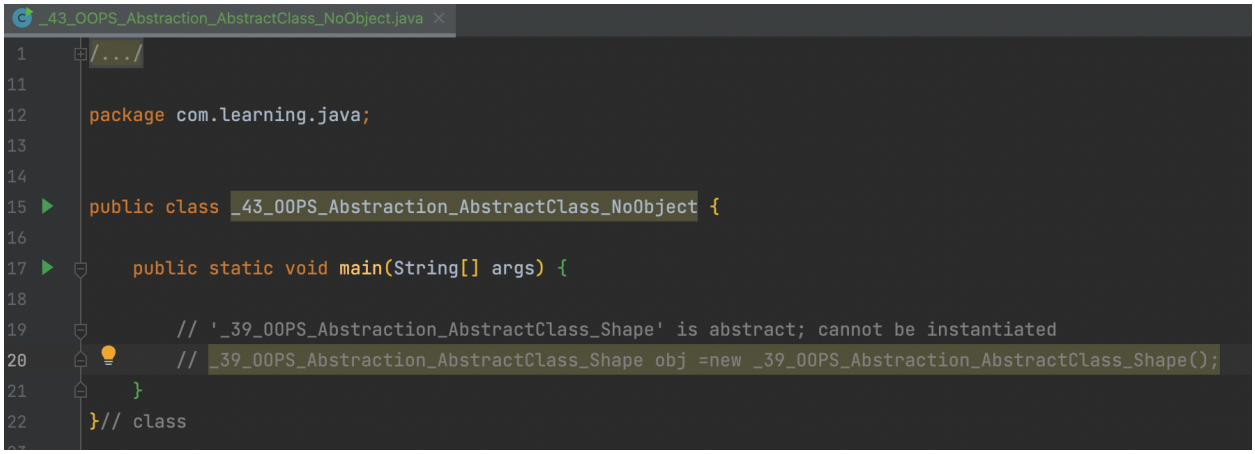
```java

package com.learning.java;


public class _41_OOPS_Abstraction_AbstractClass_Square extends  _39_OOPS_Abstraction_AbstractClass_Shape{

    double side;

    public _41_OOPS_Abstraction_AbstractClass_Square(String color, double side) {
        super(color);
        this.side = side;
    }

    // area() and colorInfo() -> Abstract methods coming from _39_OOPS_Abstraction_AbstractClass_Shape
    @Override
    double area() {
         return Math.pow(side, 2);
    }

    @Override
    String info() { return "I'm a "+super.color+" Square with Area: "+area(); }
}// class
```

4.

_42_OOPS_Abstraction_AbstractClass_App.java

```java

package com.learning.java;


public class _42_OOPS_Abstraction_AbstractClass_App {

    public static void main(String[] args) {
        _40_OOPS_Abstraction_AbstractClass_Circle circle =
                new _40_OOPS_Abstraction_AbstractClass_Circle( color: "Red",  radius: 2);

        System.out.println("circle.info() = " + circle.info()); // I'm a Red Circle with Area: 12.566370614359172

        _41_OOPS_Abstraction_AbstractClass_Square square =
                new _41_OOPS_Abstraction_AbstractClass_Square( color: "Yellow",  side: 4);

        System.out.println("square.info() = " + square.info()); // I'm a Yellow Square with Area: 16.0

    }
}// class
```

—----------

5. 

—--------30. OOP - Abstraction using Interface —----------

1. **Abstraction:**
    a. Hide details and show only essential information!
    b. 100% abstraction.

    c. Interface
        i. Provides complete abstraction (blueprint! - what to do and not how to do!)
        ii. Methods are by default abstract and public
        iii. Attributes are by default public, static and final. In other words, attributes are constants
        iv. Cannot contain a constructor
        v. Must be implemented by other class using the **implements** keyword
        vi. A subclass must implement all abstract methods
        vii. A subclass can implement multiple interfaces
        viii. Java8: Can have a default method
        ix. Java8: Can have a static method

    d. Advantages:
        i. Reduces complexity by hiding implementation
        ii. Increases security by providing only important details to the user
        iii. Helps achieve multiple inheritance

    e. Example
        i. Shapes: Abstract
        ii. Triangle: SubClass1
        iii. Square: SubClass2

—----------

1. _44_OOPS_Abstraction_Interface_Shape.java

```java
public interface  _44_OOPS_Abstraction_Interface_Shape {

    // In Interface,
        // Default ->
            // Variables are public, static and final
            // Methods are Abstract and public

    // Variable
    int number = 100;

    // Method
    String color();
    double area();
    String info();

    // From Java 8,
        // Interface can have a default method, static method
    default void defaultMethod(){
        System.out.println("Default method");
    }
    static void staticMethod(){
        System.out.println("Static method");
    }

}// interface
```

2.

_45_OOPS_Abstraction_Interface_Class_Circle.java

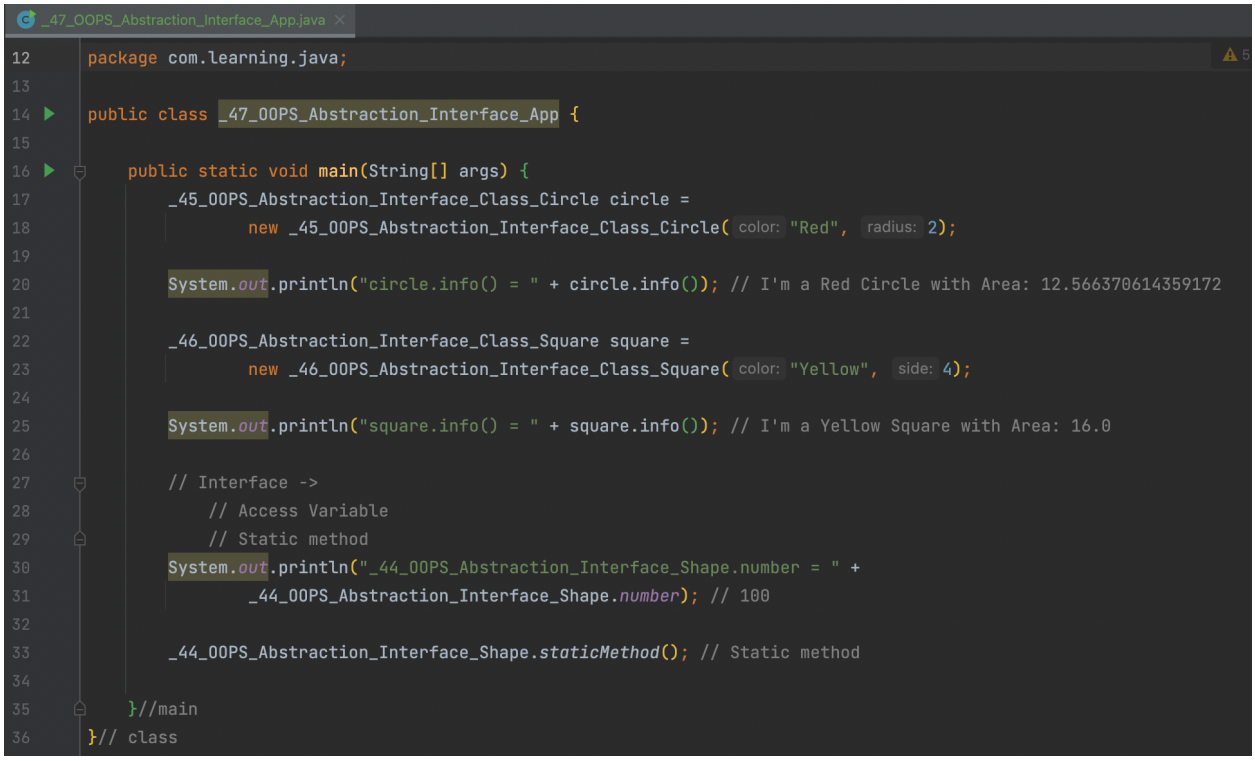
```java
package com.learning.java;

public class _45_OOPS_Abstraction_Interface_Class_Circle implements  _44_OOPS_Abstraction_Interface_Shape{

    String color;
    double radius;

    public _45_OOPS_Abstraction_Interface_Class_Circle(String color, double radius){
        this.color = color;
        this.radius = radius;
    }

    @Override
    public String color() {
        return color;
    }

    @Override
    public double area() { return Math.PI * Math.pow(radius,2); }

    @Override
    public String info() {
        return "I'm a "+color()+" Circle with Area: "+area();
    }
}// class
```

3.

_46_OOPS_Abstraction_Interface_Class_Square.java

```java
package com.learning.java;


public class _46_OOPS_Abstraction_Interface_Class_Square implements _44_OOPS_Abstraction_Interface_Shape{

    String color;
    double side;

    public _46_OOPS_Abstraction_Interface_Class_Square(String color, double side){
        this.color = color;
        this.side = side;
    }

    @Override
    public String color() { return color; }

    @Override
    public double area() { return Math.pow(side,2); }

    @Override
    public String info() { return "I'm a "+color()+" Square with Area: "+area(); }
}// class
```

4. _47_OOPS_Abstraction_Interface_App.java



```java
package com.learning.java;

public class _47_OOPS_Abstraction_Interface_App {

    public static void main(String[] args) {
        _45_OOPS_Abstraction_Interface_Class_Circle circle =
                new _45_OOPS_Abstraction_Interface_Class_Circle( color: "Red",  radius: 2);

        System.out.println("circle.info() = " + circle.info()); // I'm a Red Circle with Area: 12.566370614359172

        _46_OOPS_Abstraction_Interface_Class_Square square =
                new _46_OOPS_Abstraction_Interface_Class_Square( color: "Yellow",  side: 4);

        System.out.println("square.info() = " + square.info()); // I'm a Yellow Square with Area: 16.0

        // Interface ->
            // Access Variable
            // Static method
        System.out.println("_44_OOPS_Abstraction_Interface_Shape.number = " +
                _44_OOPS_Abstraction_Interface_Shape.number); // 100

        _44_OOPS_Abstraction_Interface_Shape.staticMethod(); // Static method

    }//main
}// class
```

—--------31. Exception Handling —----------

1. **Exception types**
    a. **Compile-time exception (checked exceptions)**
        i. FileNotFoundException
    b. **Runtime exception (unchecked exception)**
        i. ArrayIndexOutOfBoundsException

2. **Ways to handle exceptions**
    a. Handling exceptions using **try-catch-finally**
    b. throws keyword: postpone exception
    c. try with resources
    d. user-defined exception
    e. throw new Exception

—----------

1. _48_ExceptionHandling_Exception_Runtime.java

```java
package com.learning.java;

public class _48_ExceptionHandling_Exception_Runtime {

    public static void main(String[] args) {

        int[] intArray = {1,2,3};

        // java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 3
        // ArrayIndexOutOfBoundsException -> Runtime Exception
        System.out.println("intArray[4] = " + intArray[4]);

    }//main
}// class
```

2. _49_ExceptionHandling_Exception_Compiletime.java

```java

import java.io.File;
import java.io.FileReader;

public class _49_ExceptionHandling_Exception_Compiletime {

    public static void main(String[] args) {

        File file = new File( pathname: "./SomePath");

        // java.io.FileNotFoundException -> Compile Time Exception
        // FileReader fileReader = new FileReader(file);

    }//main
}// class
```

3.

_50_EH_AddToMethodSignature.java

```java
package com.learning.java;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;

public class _50_EH_AddToMethodSignature {

    public static void main(String[] args) throws FileNotFoundException {

        File file = new File( pathname: "./SomePath");

        // java.io.FileNotFoundException -> Compile Time Exception
        FileReader fileReader = new FileReader(file);

    }//main
}// class
```

4.

_50_EH_TryCatch_Block.java

```java
package com.learning.java;

import ...

public class _50_EH_TryCatch_Block {

    public static void main(String[] args) {

        try {
            File file = new File( pathname: "./SomePath");

            // java.io.FileNotFoundException -> Compile Time Exception
            FileReader fileReader = new FileReader(file);
        }catch (FileNotFoundException fileNotFoundException){
            fileNotFoundException.printStackTrace();
        }

        System.out.println("Exception handled using try-catch block.");
    }//main
}// class
```

5.

_51_EH_TryCatchFinally_Block_Keyword_throw.java

```java
package com.learning.java;

import ...

public class _51_EH_TryCatchFinally_Block_Keyword_throw {

    public static void main(String[] args) throws FileNotFoundException {

        try {
            File file = new File( pathname: "./SomePath");

            // java.io.FileNotFoundException -> Compile Time Exception
            FileReader fileReader = new FileReader(file);
        }catch (FileNotFoundException fileNotFoundException){
            fileNotFoundException.printStackTrace();
            throw fileNotFoundException;
        }finally {
            System.out.println("Inside finally block");
        } // finally block will execute (Does not matter, Exception comes or not)

        // This below code will not execute
            // because we are throwing the Exception explicitly on line # 29
                // and, added this exception in Method Signature
        System.out.println("Exception handled using try-catch block.");
    }//main
}// class
```

6.

_52_EH_TryWithResources.java

```java
package com.learning.java;

import ...

public class _52_EH_TryWithResources {

    public static void main(String[] args) {

        File file = new File( pathname: "./SomePath");

        // java.io.FileNotFoundException -> Compile Time Exception
        try (FileReader fileReader = new FileReader(file)) {
        } catch (Exception exception) {
            exception.printStackTrace();
        }

        System.out.println("Exception handled with try with resources");

    }//main
}// class
```

7.

_53_EH_CustomException.java

```java
/.../

package com.learning.java;

import ...

public class _53_EH_CustomException extends Exception{

    public void customException() { System.out.println("This is User-Defined Exception"); }

}// class
```

8.

_54_EH_CustomException_Impl.java

```java
package com.learning.java;

import ...

public class _54_EH_CustomException_Impl extends Exception{

    public static void main(String[] args) {

        File file = new File( pathname: "./SomePath");

        // java.io.FileNotFoundException -> Compile Time Exception
        try (
        FileReader fileReader = new FileReader(file)) {
            throw new _53_EH_CustomException();
        } catch (_53_EH_CustomException exception) {
            exception.customException();
            // exception.printStackTrace();
        } catch (IOException e) {
            // e.printStackTrace();
        }

        System.out.println("Exception handled with Custom (User-Defined) Exception");

    }// main
}// class
```

—----------

—--------32. File Operations —----------

1. **File operations**
    a. Creation of a new file
    b. Opening an existing file
    c. Reading from file
    d. Writing to a file
    e. Closing a file
    f. Deleting a file

    g. Java classes
        i. FileReader, BufferedReader, Files, Scanner, FileInputStream, FileWriter, BufferedWriter,
        ii. FileOutputStream, etc.

    h. Operations
        i. Create file
        ii. Read file properties
        iii. Read and Write file using FileReader and FileWriter
        iv. Read and Write file using FileInputStream and FileOutputStream
        v. Use relative file path (File separator)

—----------

1.

_55_FileOperations.java

```
12  package com.learning.java;
13
14  import ...
17
18  public class _55_FileOperations extends Exception{
19
20      public static void main(String[] args) {
21
22          File file = new File( pathname: "./src/resources/DataFile.txt");
23
24          System.out.println("file.exists() = " + file.exists()); // true
25          System.out.println("file.canRead() = " + file.canRead()); // true
26          System.out.println("file.canExecute() = " + file.canExecute()); // false
27          System.out.println("file.canWrite() = " + file.canWrite()); // true
28
29      }// main
30  }// class
```

2.

_56_FileOperations_CreateNewFile.java

```java
public class _56_FileOperations_CreateNewFile {

    public static void main(String[] args) {

        File file = new File( pathname: "./src/resources/DataFile_2.txt");

        // If file - DataFile_2.txt is not present, Then, create new file
        if(!file.exists()){
            try {
                file.createNewFile();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }// main
}// class
```

3.

_57_FileOperations_Write_FileWriter.java

```java
public class _57_FileOperations_Write_FileWriter {

    public static void main(String[] args) {

        File file = new File( pathname: "./src/resources/DataFile_2.txt");

        // If file - DataFile_2.txt is not present, Then, create new file
        if(!file.exists()){
            try {
                file.createNewFile();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        // Write content into file using FileWriter
        try {
            FileWriter fileWriter = new FileWriter(file);
            fileWriter.write( str: "DATA-2.txt -> Hi, Test Automation Engineer!");
            fileWriter.flush();
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }// main
}// class
```

4.

_58_FileOperations_ReadFile_FileReader.java

```java
public class _58_FileOperations_ReadFile_FileReader {

    public static void main(String[] args) {

        File file = new File( pathname: "./src/resources/DataFile.txt");

        // Read content from file using FileReader
        try {
            FileReader fileReader = new FileReader(file);
            int i;
            while ((i = fileReader.read())!= -1){
                System.out.print((char)i);
            }
            fileReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }


    }// main
}// class
```

5.

_59_FileOperations_Write_FileOutputStream.java

```java
public class _59_FileOperations_Write_FileOutputStream {

    public static void main(String[] args) {

        File file = new File( pathname: "./src/resources/DataFile_3.txt");

        // If file - DataFile_2.txt is not present, Then, create new file
        if(!file.exists()){...}

        // Write content into file using FileOutputStream
        try {
            FileOutputStream fileOutputStream = new FileOutputStream(file);
            String content = "DATA-3.txt -> Hi, Test Automation Engineer!";
            fileOutputStream.write(content.getBytes());
            fileOutputStream.flush();
            fileOutputStream.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }


    }// main
}// class
```

6.

_60_FileOperations_ReadFile_FileInputStream.java

```java
public class _60_FileOperations_ReadFile_FileInputStream {

    public static void main(String[] args) {

        File file = new File( pathname: "./src/resources/DataFile.txt");

        // Read content from file using FileInputStream

        try {
            FileInputStream fileInputStream = new FileInputStream(file);
            int i;
            while ((i = fileInputStream.read())!= -1){
                System.out.print((char)i);
            }
            fileInputStream.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }// main
}// class
```

7.

_61_FileOperations_RetrieveFileProperties.java

```java
package com.learning.java;

import java.io.File;

public class _61_FileOperations_RetrieveFileProperties {

    public static void main(String[] args) {

        File file = new File( pathname: "./src/resources/DataFile.txt");

        // Retrieve File Properties

        System.out.println("file.getName() = " + file.getName()); // DataFile.txt
        System.out.println("file.getPath() = " + file.getPath()); //./src/resources/DataFile.txt
        System.out.println("file.getAbsolutePath() = " + file.getAbsolutePath()); //  /Users/rajatverma/Desktop/Work/

        // To Delete the file
        // file.delete();

    }// main
}// class
```

8.

_62_FileOperations_MakeDirectories.java

```java
package com.learning.java;

import java.io.File;
import java.io.IOException;

public class _62_FileOperations_MakeDirectories {

    public static void main(String[] args) {

        String directoryPath = "./src/resources/directories" +File.separator +
                "directory_1" + File.separator + "directory_2";

        File directory = new File(directoryPath);

        // Create Directories
        if(!directory.exists()){
            directory.mkdirs();
        }

    }// main
}// class
```

—-------- 22. Code Download—----------

1. https://github.com/rajatt95/Java_OC

==========

==========

1. **To connect:**
    a. https://www.linkedin.com/in/rajat-v-3b0685128/
    b. https://github.com/rajatt95
    c. https://rajatt95.github.io/

==========