OPTIONAL CLASS

```java
package com.java8Features.Optional;

import java.util.Date;
import java.util.Optional;

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

        /**
         * The main objective of Optional is to avoid null pointer exception
         * Three ways to create Optional Object
         */
         // 1. empty() method
        /**
         * Optional.empty():- This method returns Optional.empty
         * empty is a reference variable in Optional class which holds an empty object
         * The output of printing optional1 is Optional.empty because of the overriding of toString method in Optional class
         */
        Optional<String> optional1 =Optional.empty();

        //2.of() method
        /**
         * This method returns Optional object if there is any value
         * but this method throws NullPointerException if there is no value
         * when we are 100% sure the value is not null then we should go for of() method
         */
        Optional<String> optional2=Optional.of("Rajnish");

        //3.ofNullable method
        /**
```

```
 * This method returns Optional object if there is value
 * if there is no value there it return Empty Optional Object--Optional.empty
 */
Optional<String> optional3=Optional.ofNullable("Rajnish");


/**
 * Optional is a container which holds empty as well as non-empty Object
 * Try to print and track the behaviour of each method
 */
System.out.println(optional1);
System.out.println(optional2);
System.out.println(optional3);

/**
 * Because we consider Optional as a container so we are just trying to get the value from that container
 */
//1.get() --> get() method returns the Optional object
/**
 * When we tring to get the empty object then we will get NoSuchElementException
 */
System.out.println(optional1.get());                    //NoSuchElementException---->always when trying to get
System.out.println(optional2.get());                    //NullPointerException----->always when object is empty
System.out.println(optional3.get());                    //NosuchElementException--->always when trying to get when the value is null


//2.ifPresent() ---> this method returns boolean if the value is present return true else return false
if(optional3.isPresent()) {
    System.out.println(optional3.get());
}else {
    System.out.println("No value present");
}
```

```java
//3.ifPresent(Consumer)
/**
 * consumer is a functional interface  which contains andThen(consumer) method
 * if value is present, it calls the consumer with the value, otherwise does nothing or ifPresent() won't invoke
 */
optional3.ifPresent(string->System.out.println(string.toUpperCase()));

//4. orElse()
/**
 * if the value is present then perform the logic
 * if the value is not present then it return default value
 */
System.out.println(optional1.orElse("Default value or value not present"));

//5.orElseGet(Supplier)
/**
 * returns the value if present  otherwise invokes the  supplier --> it means we can write our own logic
 * supplier is a functional interface which contains get() method,
 * its meaning is to only give something don't take anything as argument
 */
Optional<Date> date = Optional.empty();
System.out.println(date.orElseGet( ()->new Date()  ));

//6.orElseThrow(supplier)
/**
 * returns the value if present otherwise invokes the supplier --> throw any exception (Based on requirement)
 *  supplier is a functional interface which contains get() method,
 * its meaning is to only give something don't take anything as argument
 */
Optional<Integer> item = Optional.empty();
System.out.println(item.orElseThrow(IllegalArgumentException::new));
```

```
        //7.filter(Predicate)
        /**
         * if the value present then the value matches the given predicate otherwise return Optional.empty
         * predicate is an functional interface which contains test() method
         */
        Optional<Integer> age = Optional.of(25);
        System.out.println(age.filter(i->i>18).get());

        //8.map(function)
        /**
         * if a value is present applies the provided logic and mapping or transform the value
         * function is a functional interface which contains apply function
         */

        Optional<Integer> marks=Optional.of(30);
        System.out.println(marks.map(i->i+10).get());
    }
}
```