```
Inbuilt functional interfaces used in realtime Coding
++++++++++++++++++++++++++++++++++++++++++++++++++

Predicate(logical condition) -> test(T) :: boolean

Fuction(Perform some operation and return result) -> apply(T) :: R

Consumer(Consume the input supplied by user) -> accept(T) :: void

Supplier(Supplies the data to the user)  -> get() :: T


Optional API
++++++++++++
public final class java.util.Optional<T> {

  public static <T> java.util.Optional<T> empty();


  public static <T> java.util.Optional<T> of(T);         :: returns the Optional
object filled with non-null value
  public static <T> java.util.Optional<T> ofNullable(T); :: returns non-empty
optional object if a value is present inside the object


  public T get();

  public boolean isPresent();
  public boolean isEmpty();

  public void ifPresent(java.util.function.Consumer<? super T>);


  public java.util.Optional<T>     filter(java.util.function.Predicate<? super T>);
  public <U> java.util.Optional<U> map(java.util.function.Function<? super T, ?
extends U>);


  public T orElse(T);
  public T orElseGet(java.util.function.Supplier<? extends T>);
  public T orElseThrow();
  public <X extends java.lang.Throwable> T
orElseThrow(java.util.function.Supplier<? extends X>) throws X;


}

eg#1.
import java.util.function.*;
import java.util.*;
public class Test{
      public static void main(String[] args){

            Optional<String> empty = Optional.empty();
            System.out.println(empty.isPresent());

            System.out.println();

            String name= "nitin";
```

```
            if(name!=null)
                  System.out.println(name.length());

            System.out.println();

            Optional<String> opt = Optional.of("nitin");
            opt.ifPresent(str->System.out.println(str.length()));

      }

}
Output
false

5

5


eg#2.

import java.util.function.*;
import java.util.*;
public class Test{
      public static void main(String[] args){

            String nullName = "[sachin,51,true]";

            String result = Optional.ofNullable(nullName).orElse("Record not
found");
            System.out.println(result);
      }
}
Output
[sachin,51,true]

eg#3.
import java.util.function.*;
import java.util.*;

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

            String nullName = null;

            String result = Optional.ofNullable(nullName).orElse("Record not
found");
            System.out.println(result);
      }
}
Output
Record not found


KeyPoints
 orElse()               -> If the value is present, it would return the value
otherwise use orElse() to return the value.
 orElseGet(Supplier)    -> If the value is present, it would return the value
otherwise get it from Supplier API call.
```

orElseThrow(Supplier) -> If the value is present, it would return the value otherwise it throws an Exception.

```
eg#1.
import java.util.function.*;
import java.util.*;

class StudentRecordNotFoundException extends RuntimeException
{
      StudentRecordNotFoundException(String msg)
      {
            super(msg);
      }
}

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

                  String nullValue = null;
                  String result = Optional.ofNullable(nullValue).orElseGet(()->"not
found");
                  System.out.println(result);

                  System.out.println();

                  System.out.println(Optional
                                    .ofNullable(nullValue)
                                    .orElseThrow(
                                        ()-> new
StudentRecordNotFoundException("Record not found"))
                                  );

      }
}
```

Supplier(I)
++++++++++
Sometime our requirment is we have to get some value based on some operation like supply student object, supply otp, supply random password,
supply randomName etc.... for this type of requirment we need to go for "Supplier".

-> Supplier won't take any input,but it will always supply objects.

```
public interface java.util.function.Supplier<T> {
      public abstract T get();
}
```

Note: Math.random() :: double -> generates a number greater than or equal to 0.0 and lessthan 1.0

```
eg#1.
import java.util.function.*;
import java.util.*;
public class Test{
      public static void main(String[] args){
```

```
            //public abstract T get();
            Supplier<String> s = () ->{

                  String names[] = {"sachin","saurav","rahul","dhoni","kohli"};
                  int index=(int)(Math.random()*5);
                  System.out.print(index + "::");
                  return names[index];
            };
            System.out.println(s.get());
            System.out.println(s.get());
            System.out.println(s.get());
      }
}
Output
0::sachin
4::kohli
1::saurav


eg#2.
import java.util.function.*;
import java.util.*;

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

            //public abstract T get();
            Supplier<String> s = () ->{

                  //Generating OTP(6 digits:: 0 to 9)
                  String otp = "";
                  for (int i =1;i<=6 ;i++ )
                  {
                        otp+=(int)(Math.random()*10);
                  }

                  return otp;
            };
            System.out.println(s.get());
            System.out.println(s.get());
            System.out.println(s.get());
      }
}

Output
517276
947456
419555

eg#3.
Rules
  1. length should be 8 characters
  2. 2,4,6,8 places only digits
  3. 1,3,5,7 only capital letters and special symobls like @,#,$


import java.util.function.*;
import java.util.*;
public class Test{
```

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

		//public abstract T get();
		Supplier<String> s = () ->{

		String password = "";
		String symbols = "ABCDEFGHIJKLMNOPQRSTUVWXYZ@$#";
		Supplier<Integer> i1=()->(int)(Math.random()*10);
		Supplier<Character>c1=()->symbols.charAt((int)(Math.random()*29));

		for (int i =1;i<=8 ;i++)
		{
			if (i%2==0)
			{
				//even places
				password+=i1.get();
			}
			else
			{
				//odd places
				password+=c1.get();
			}
		}

		return password;

	};

		System.out.println(s.get());
		System.out.println(s.get());
		System.out.println(s.get());
	}
}
Output
V7R8Y3D3
L5D4C3K4
F4Z8@8Y5


StringJoiner class
++++++++++++++++++
 => It is used to join 2 String seperated with a delimitor
 => Delimitor can be ,|,......

public final class java.util.StringJoiner {
  public java.util.StringJoiner(java.lang.CharSequence);
  public java.util.StringJoiner(java.lang.CharSequence, java.lang.CharSequence,
java.lang.CharSequence);

  public java.util.StringJoiner setEmptyValue(java.lang.CharSequence);
  public java.lang.String toString();


  public java.util.StringJoiner add(java.lang.CharSequence);
  public java.util.StringJoiner merge(java.util.StringJoiner);

  public int length();

}
```

eg#1.

```
import java.util.*;

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

			delimitorDemonstration();
			addingPrefixAndSuffix();
			mergeTwoStringJoiners();


	}

	public static void addingPrefixAndSuffix()
	{
		StringJoiner sj =new StringJoiner(",","[","]");
			sj.add("sachin");
			sj.add("saurav");
			sj.add("dhoni");
			sj.add("kohli");
			sj.add("dravid");

			System.out.println(sj);
	}
	public static void delimitorDemonstration()
	{
		StringJoiner sj =new StringJoiner(",");
			sj.add("sachin");
			sj.add("saurav");
			sj.add("dhoni");
			sj.add("kohli");
			sj.add("dravid");

			System.out.println(sj);

			System.out.println();

			sj =new StringJoiner("|");
			sj.add("sachin");
			sj.add("saurav");
			sj.add("dhoni");
			sj.add("kohli");
			sj.add("dravid");

			System.out.println(sj);
	}

	public static void mergeTwoStringJoiners()
	{
			StringJoiner sj1 =new StringJoiner(",","[","]");
			sj1.add("sachin");
			sj1.add("saurav");
			sj1.add("dravid");

			StringJoiner sj2 =new StringJoiner(",","[","]");
			sj2.add("dhoni");
			sj2.add("kohli");
			sj2.add("rohith");
```

```
                StringJoiner sj3 = sj1.merge(sj2);
                System.out.println(sj3);

      }

}
Output
sachin,saurav,dhoni,kohli,dravid

sachin|saurav|dhoni|kohli|dravid

[sachin,saurav,dhoni,kohli,dravid]

[sachin,saurav,dravid,dhoni,kohli,rohith]



eg#2.
import java.util.*;

public class Test{
      public static void main(String[] args){
            stringJoinerMethods();
      }
      public static void stringJoinerMethods()
      {
            StringJoiner sj =new StringJoiner(",");
            System.out.println(sj);

            sj.setEmptyValue("It is empty");
            System.out.println(sj);

            sj.add("sachin");
            sj.add("dravid");
            System.out.println(sj);

            int length = sj.length();
            System.out.println(length);

            String data = sj.toString();
            System.out.println(data.charAt(3));
            System.out.println(data.toUpperCase());

      }

}
Output

It is empty
sachin,dravid
13
h
SACHIN,DRAVID

Date and Time API
+++++++++++++++++
  Untill JDK1.7Version to handle the data and time related information, we use
```

classes like
      a. Date
        b. Calendar
        c. TimeZone etc....
  But these classes are not up to the mark w.r.t performance and convinence.

=>To Overcome this problem in JDK1.8V Oracle team had introducted an API called "JODA-API".
=>This API is developed by an organisation called "joda.org' and it is present inside the package called "java.time" package.

JODA-API
  a. LocalDate
  b. LocalTime
  c. LocalDateTime

Methods of LocalDate
++++++++++++++++++++
public static java.time.LocalDate of(int, java.time.Month, int);
public static java.time.LocalDate of(int, int, int);

Methods of LocalTime
++++++++++++++++++++
public static java.time.LocalTime of(int, int);
public static java.time.LocalTime of(int, int, int);
public static java.time.LocalTime of(int, int, int, int);

Methods of LocalDateTime
++++++++++++++++++++++++
public static java.time.LocalDateTime of(int, java.time.Month, int, int, int);
public static java.time.LocalDateTime of(int, java.time.Month, int, int, int, int);
public static java.time.LocalDateTime of(int, java.time.Month, int, int, int, int, int);
public static java.time.LocalDateTime of(int, int, int, int, int);
public static java.time.LocalDateTime of(int, int, int, int, int, int);
public static java.time.LocalDateTime of(int, int, int, int, int, int, int);

eg#1.
```
import java.time.*;
public class Test{
      public static void main(String[] args){

            LocalDate ld = LocalDate.now();
            System.out.println(ld);
            System.out.println("Year  is :: "+ld.getYear());
            System.out.println("Month is :: "+ld.getMonthValue());
            System.out.println("Date  is :: "+ld.getDayOfMonth());

            System.out.println();

            LocalTime lt = LocalTime.now();
            System.out.println(lt);
            System.out.println("HOUR    is  :: "+lt.getHour());
            System.out.println("MIN     is  :: "+lt.getMinute());
            System.out.println("SECONDS is  :: "+lt.getSecond());
            System.out.println("NANO    is  :: "+lt.getNano());
```

```
            System.out.println();


            LocalDateTime ldt = LocalDateTime.now();
            System.out.println(ldt);

      }
}
```

```
Output
2024-01-21
Year  is :: 2024
Month is :: 1
Date  is :: 21

12:57:24.688
HOUR    is  :: 12
MIN     is  :: 57
SECONDS is  :: 24
NANO    is  :: 688000000

2024-01-21T12:57:24.688
```

eg#2.
To create the date and time object as per our needs we use the following methods

```
import java.time.*;

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

            LocalDate ld = LocalDate.of(1993,Month.JANUARY,03);
            System.out.println(ld);

            LocalTime lt = LocalTime.of(19,45,35);
            System.out.println(lt);


            LocalDateTime ldt = LocalDateTime.of(1996,Month.FEBRUARY,24,9,30,45);
            System.out.println(ldt);
      }
}
```

```
Output
1993-01-03
19:45:35
1996-02-24T09:30:45
```

eg#3.
 To represent Zone we use ZoneId Object

```
Few ZoneId
America/Toronto
Asia/Singapore
Australia/Lindeman
America/Los_Angeles
++++++++++++++++++++
```

```
import java.time.*;
import java.util.*;
public class Test{
      public static void main(String[] args){

            ZoneId zone = ZoneId.systemDefault();
            System.out.println(zone);//Asia/Calcutta

            System.out.println();


            ZoneId zoneId= ZoneId.of("Australia/Melbourne");
            ZonedDateTime zdt = ZonedDateTime.now(zoneId);
            System.out.println(zdt);
      }
}
Output
Asia/Calcutta
2024-01-21T17:54:14.654+10:00[Australia/Lindeman]



Period Object
++++++++++++
 => It is used to represent quantity of time

eg#1.
import java.time.*;
import java.util.*;

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

            LocalDate todayDate = LocalDate.now();
            LocalDate birthDate = LocalDate.of(1993,1,03);

            Period period=Period.between(birthDate,todayDate);
            System.out.println(period);


            System.out.println("Your age is :: "+period.getYears() +
                                    " Months is :: "+period.getMonths() +
                                    " Days is :: "+period.getDays());

            System.out.println();

            System.out.printf("\nAge is %d years %d months %d days\n",
                        period.getYears(),period.getMonths(),period.getDays());

      }
}

Output
P31Y18D
Your age is :: 31 Months is :: 0 Days is :: 18
Age is 31 years 0 months 18 days

Write a program to check wheter the given year is leap year or not?
```

Logic::
 Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100,
 but these centurial years are leap years if they are exactly divisible by 400.

eg#1.

```
import java.time.*;
public class Test{
      public static void main(String[] args){

            //Converting String to Integer type
            Integer year = Integer.parseInt(args[0]);

            Year checkLeapear = Year.of(year);

            String output = checkLeapYear.isLeap() ?
                        year + " is a LeapYear " :
                        year + " is not a LeapYear ";

            System.out.println(output);
      }
}
```

Output

```
D:\OctBatchMicroservices>javac Test.java

D:\OctBatchMicroservices>java Test 2023
2023 is not a LeapYear

D:\OctBatchMicroservices>java Test 2024
2024 is a LeapYear

D:\OctBatchMicroservices>java Test 2020
2020 is a LeapYear
```