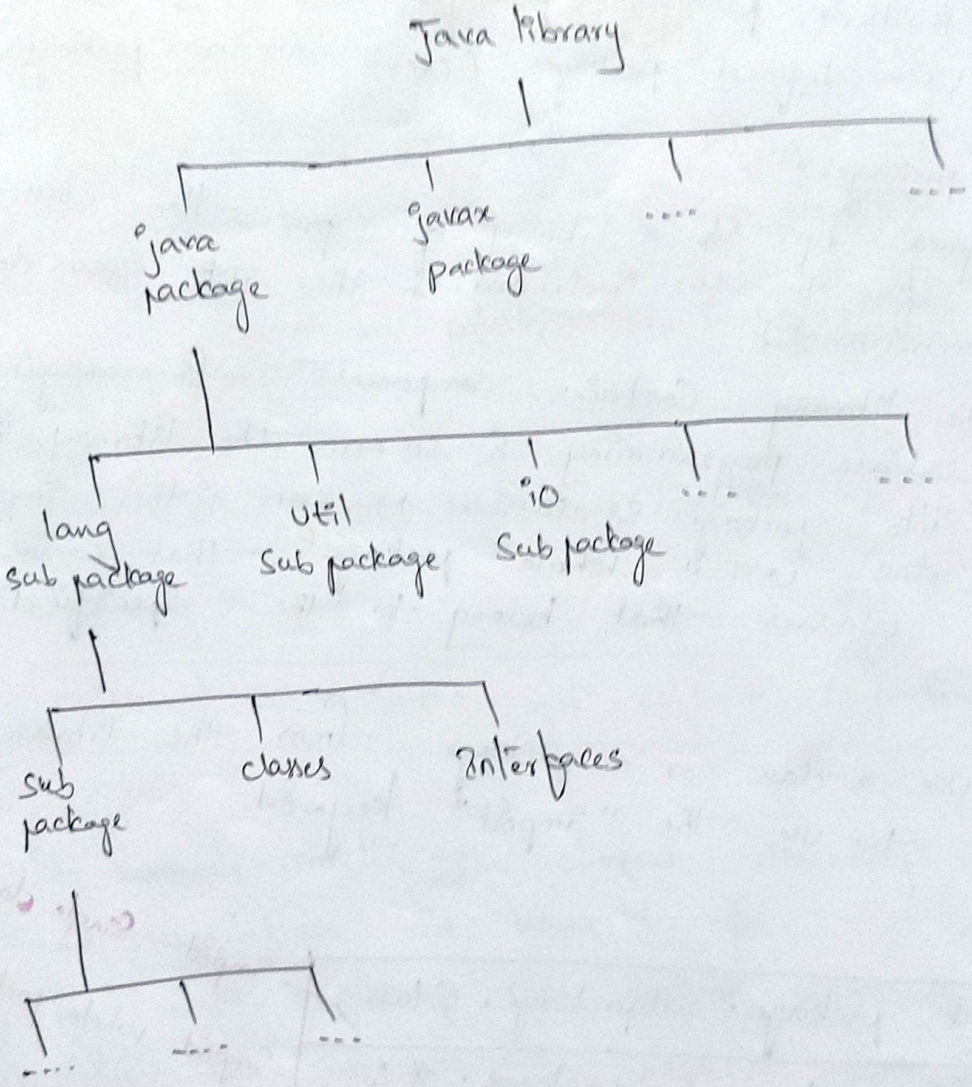
Packages and Java Library

Introduction:

- A package is a collection of Sub packages, classes & Interfaces .. etc Java library organized in the form of packages.



- Java compiler generates ".class" file for every class & every interface & a Source file
- Java compiler generates folder for every package & every Sub package.
- by using "package" keyword we create packages.

Defining a package:

A package in java is used to group related classes. We use packages to avoid name conflicts & to write a better maintable code. Packages are divided into two categories.

1. Built-in packages (packages from the java API)
2. User-defined packages (create our own packages)

Built-in packages:

The java API is a library of prewritten classes, that are free to use, included in the JDE (Java development Environment).

The library contains components for managing input, database programming & so on. The library is divided into packages & classes. We can either import a single class (or a whole package that contains all the classes that belong to the specified package.

To use a class or a package from the library we need to use the "import" keyword

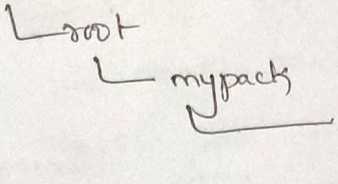
Syntax:

```
import package.subpackage.Class;     // import single class
import package.subpackage.*;         // import whole package.
```

User-defined packages:

To create our own packages, we need to understand that java uses a file system directory to store them. just like folders on our computer.

Ex:



- To create a package, use the "package" keyword

MyPackageClass.java

```
package mypack;
class MyPackageClass
{
    public static void main(String args[])
    {
        System.out.println("this is my package");
    }
}
```

- c:\Users\513> javac MyPackageClass.java

To compile the package

```
c:\Users\ src > javac -d c:\work MyPackageClass.java
```

-d: it omits the current directory & specifies the destination for where to save the class file. If we want to keep the package within the same directory, we can use the "." (dot) sign.

```
c:\Users\ src > javac -d . MyPackageClass.java
```

When we compiled the package in the above example, a new folder was created, called "mypack".

importing packages & classes into programs

import a class:

If we want to use a class for example "Scanner" class, which is used to get input then we have to write the statement like

```
import java.util.Scanner;
```

In the above statement, java → is a main package
util – is a sub package
Scanner – is a class name.

- To use the Scanner class, we can create an Object & use any of the available methods found in the Scanner class.

Ex:

```
import java.util.Scanner;
class A
{
public static void main(String args[])
{
    Scanner s=new Scanner(System.in);
    System.out.println("Enter User name");
    String userName= s.nextLine();
    System.out.println("Username is:"+userName);
}
}
```

In the above program we use Scanner class & nextLine() method in the Scanner class. so we import only Scanner class in util package

import a package:

If we dont know the class name of particular method, then we can import a whole package to our program. then all classes, interfaces, methods are import to our program.

To import a whole package, end the sentence with an asterisk sign (*).

Ex:

```
import java.util.*;
```

by using the above statement we can use all the classes & interfaces of util package

path and class path

PATH & CLASS PATH are environmental variables on microsoft windows

windows 7:

PATH

1. from the desktop, right click on Computer icon
2. choose properties from the menu
3. click the Advanced System Settings link
4. click Environment variables. In the section System variables, find the PATH environment variable & select it. click Edit if the PATH environment variable does not exist click new.
5. In the Edit System variable (or new System Variable) window, specify the value of the PATH Environment variable. click ok. close all remaining windows by clicking ok.

Class path

```
c:\> set path="%path%";c:\program files\java\jdk1.8.0\bin;
```

" (or) `%CLASS PATH%` "

⇝ Access control

private: private members of class are not accessible on other classes of the same package or another package. i.e within the class only accessible.

public: public members of a class available in the other classes of the same package or another package i.e from any where we can access.

protected: protected members are accessible in the sub classes of same package but not in another package

default: Default members are available in other classes of same package but not in another package

⇝ Packages in JAVA SE

1. java.lang → basic language functionality & fundamental types
2. java.util → collection of data structure classes
3. java.io → file operations
4. java.math → multiprecision arithmetics
5. java.nio → non-blocking I/O framework for java
6. java.net → networking operations, sockets, DNS lookups
7. java.security → key generation, encryption & decryption
8. java.sql → java database connectivity (JDBC) to access databases
9. java.awt → basic hierarchy of packages for native GUI components
10. java.text → provides classes & interfaces for handling text, dates, numbers & messages in a manner independent of natural languages.

11. java.rmi → provides the Rmi package
12. java.time → the main APIs for dates, times, instants & durations.
13. java.beans → it contains classes & interfaces related to javaBeans Components
14. java.applet → provides classes & methods to create & communicate with the applets.

java.lang.Object - Super class of all classes

"java.lang" package is a default package. It is available without the use of an import statement.

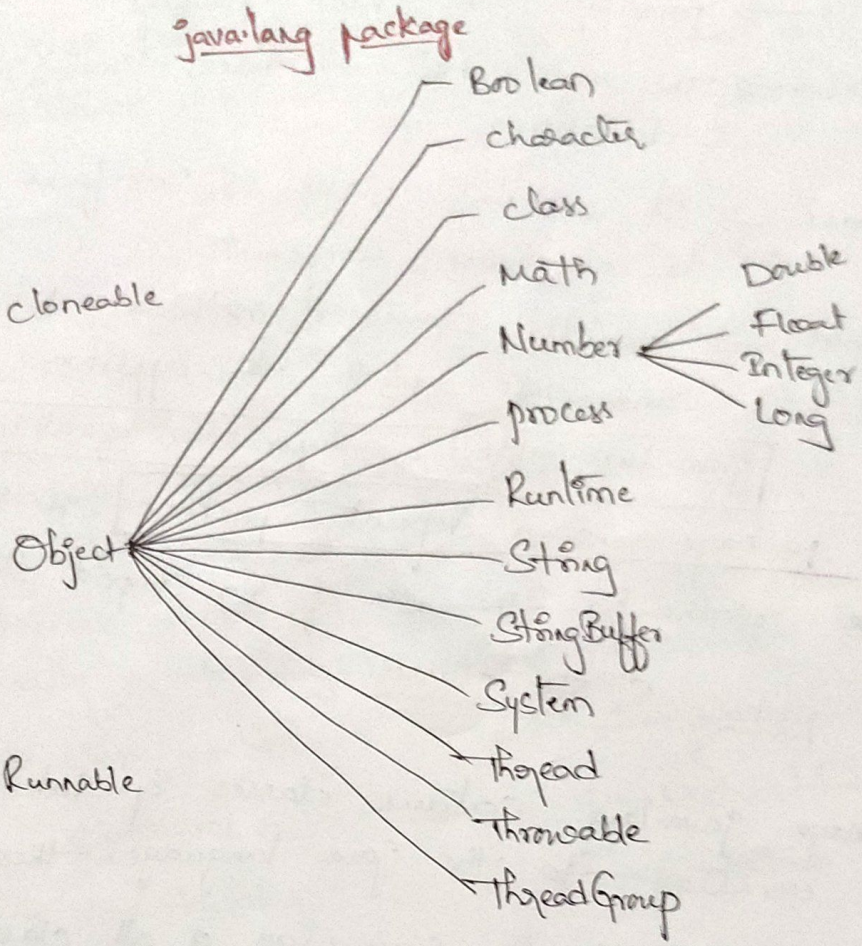
→ java.lang package & its classes

The package java.lang contains classes & interfaces that are essential to the java language. These include

- Object : the ultimate super class of all classes in java.
- Thread : the class that controls Each thread in a multi threaded program.
- Throwable : the Super class of all error & exception classes in java

- classes that encapsulate the primitive data types in java.
- classes for accessing System resources & other low-level entities.
- Math : a class that provides standard mathematical methods
- String : the class that is used to represent strings

- The java.lang package is implicitly imported by Every java Source file.



classes:

Object, Thread, ThreadGroup, Throwable

Interfaces: Cloneable, Runnable,

Abstract class: Number, process

Non-Instantiable class: Class, Runtime, System

final classes: Boolean, character, Math, String, StringBuffer, Double, float, Integer, Long

class Object:

Object class is present in java.lang package. Every class in java is directly (or) indirectly derived from the Object class. If a class does not extend any other class then it is direct child class of Object. & if extends other class then it is an indirectly

derived. therefore the object class methods are available to all java classes. hence object class acts as a root of inheritance heirarchy in any java program.

methods

1. toString():

toString() provides string representation of an object & used to convert an object to String. to override toString() method to get our own String representation of object.

- whenever we try to print any object reference, then internally toString() method is called.
- Object class toString() method always return ClassName@HashCode-In-HexaDecimal-Format
- String class toString() method always return content of String object

Ex:

```
class A
{
  public static void main(String args[])
  {
    String s= new String();
    System.out.println(s);
    A a =new A();
    System.out.println(a);
  }
}
```

implicitly s.toString()

o/p: A@3e25a5

implicitly a.toString()

2. hashCode():

for every object, JVM (java virtual machine) generates a unique number which is hashcode. it returns the address of object.

3. Equals(Object obj):

Compares the given object to "this" object (the object on which the method is called). It gives the generic way to compare objects for equality.

4. getClass():

Returns the class object of "this" object & used to get actual run-time class of the object. It can also be used to get metadata of this class.

5. finalize():

This method is called just before an object is garbage collected. It is called by the Garbage Collector. We should override finalize() method to dispose system resources, perform clean-up activities & minimize memory leaks.

finalize() method is called just once in a program.

6. clone():

It returns new object that is exactly the same as this object.

7. wait(), notify(), notifyAll()

These are related to concurrency. For threads these methods are used. These are used in Thread class

⇝ Enumerations

This feature allows to create a new Data type in java. In order to use this feature "enum" keyword introduced in JDK 1.5 version in 2004.

- All enumeration literals are implicitly "static"

Ex:

```
enum Day
{
    Mon, Tue, Wed, Thu, Fri, Sat, Sun
};

class A
{
    public static void main(String args[])
    {
        Day d = Day.Tue;
        System.out.println(d);
    }
}
```

(Mon, Tue, Wed, Thu, Fri, Sat, Sun — enumeration literals)

because all enumeration literals are static by default, so we can call with class Name.

- In the above program Day is the Data Type.

⇝ class Math

Java Math class provides several methods to perform on math calculations like

min()	tan()
max()	round()
avg()	ceil()
sin()	floor()
cos()	abs()

Methods:

The java.lang.Math class contains various methods for performing basic numeric operations such as logarithm, cube root, trignometric functions etc.

Math.abs(): return Absolute value of given value

Math.max(): return largest of two values

Math.min(): return Smallest of two values

Math.round(): round of the decimal numbers to the nearest value.

Math.Sqrt(): return Squareroot of a number

Math.cbrt(): return cube root of a number

Math.pow(): returns power of a number

Math.signum(): find the sign of a given value

Math.ceil(): find Smallest integer value that is greater than or Equal to the mathematical integer

Math.floor(): find largest integer value which is less than & Equal to the argument & Equal to the mathematical integer of double value

Math.random(): returns a double value with a positive sign greater than Equal to 0.0 & less than 1.0

Math.log(): returns natural logarithm of a double value

Math.log10(): return the base 10 logarithm of a double value

Math.exp(): returns Exponent value Equal to 2.71828

Math.Sin(): return trignometric Sine value of a double value

Math.cos(): Cosine value

Math.tan(): tangent value

Math.Asin(): arc Sine value

Wrapper classes

Each of java 8 primitive data types has a class & those classes are called wrapper classes. because they wrap the data into an object.

primitive data types	wrapper class
1. byte	1. Byte
2. short	2. Short
3. int	3. Integer
4. long	4. Long
5. float	5. Float
6. double	6. Double
7. char	7. Character
8. boolean	8. Boolean

the above all wrapper classes are the part of "java.lang" package. All wrapper classes are called Reference data types.

1. java.lang.Integer

Constructor: `public Integer(int);`

it is used to convert primitive data type to reference data type

method: `public int intValue();`

used to convert reference data type to primitive datatype

Ex1:

primitive

```
int x=5;
```

x → 5

```
Integer ob = new Integer(x);
```

reference data type

ob → (5) Integer object

primitive to Reference

Ex2:

```
Integer ob = new Integer(5);
```

reference

ob → (5) Integer object

```
int x = ob.intValue();
```

primitive

x → 5

Reference to primitive

java.lang.Float

`public Float(float);` - constructor - primitive to reference

`public float floatValue();` - method - reference to primitive

Auto boxing

The process of converting primitive data type to the corresponding reference data type is known as Auto boxing.

Ex:

primitive data type

```
int x=5
Integer ob=x;
```

- Auto boxing

reference data type

here we can assign a int variable to reference data type object reference.

Auto unboxing

The process of converting reference data type to the corresponding primitive data type is known as Auto unboxing

```
Integer ob = new Integer(5);
int x= ob;
```

reference data type

- Auto unboxing

primitive data type

here we can assign a object reference of reference data type to primitive data type variable.

- both Auto boxing & Auto unboxing features are introduced in JDK 1.5 version in 2004.

Ex:

```
float f=1.5;
Float ob=f;          } Auto boxing

Float ob=new Float(1.5);
float f=ob;          } Auto un boxing
```

⇒ Java Util classes & Interfaces

java.util package classes & interfaces are divided into 2 categories.

1. collection frame work collections
2. legacy collections.

- collections frame work collections are divided into 3 Sub categories

1. Core collection interfaces
2. General purpose implementations
3. More utility collections.

1. Core collection interfaces.

These are the foundations of collections frame work.

1. Collection
2. List
3. Set
4. Map
5. SortedSet
6. SortedMap
7. NavigableSet
8. NavigableMap
9. Queue
10. Deque

1. Collection Interface: it is a root interface in the heirarchy
2. List interface: it extends collection interface & it maintain sequences
3. Set interface: it extends collection interface & it maintain sets
4. Map interface: it is a 2-dimensional collection & it maintain data as a "key/value" pair
5. SortedSet interface: it extends Set Interface & it maintains sorted sets

6. SortedMap interface: It extends Map interface & it maintains sorted maps. here only keys are sorted

7. NavigableSet interface: It extends SortedSet interface & it is used to navigate elements of a set

8. NavigableMap interface: It extends SortedMap interface & it is used to navigate key/value pair of a Map

9. Queue interface: It is called as First In First Out list

10. Dequeue interface: It is called as Double Ended Queue.

2. General purpose implementations

The core collection interfaces implementations classes are called General purpose implementations

1. ArrayList
2. LinkedList
3. HashSet
4. LinkedHashSet
5. TreeSet
6. HashMap
7. LinkedHashMap
8. TreeMap
9. PriorityQueue
10. ArrayDeque

1. ArrayList: It is an implementation of Linear list data Structure & it supports all operations of linear list data structure.

2. LinkedList: It is an implementation of "Double linked list data Structure". It supports all operations of double linked list data structure. here data stored in Nodes.

3. HashSet: It is an implementation of "Hashing technique" with array representation.

4. LinkedHashSet: It is an implementation of hashing technique with linked representation. here data stored in Nodes.

5. TreeSet: It is an implementation of "binary search tree technique with linked representation.

6. HashMap: It is the two dimensional collection & it maintains data as a key/value pair. implements hashing with Array representation.

7. LinkedHashMap: It is two dimensional collection & implement with hashing technique with Linked representation.

8. TreeMap: It is two dimensional collection & implementation of Binary Search tree technique with linked representation.

9. PriorityQueue: It is called as Priority In & priority out list It allows insertions at rear end & deletion at front end.

10. ArrayDeque: It is an implementation of double ended Queue data structure with array representation. It allows both insertion & deletion at both ends.

3. More Utility Collections:

1. Iterator (Interface)
2. ListIterator (Interface)
3. Arrays (class)
4. Collections (class)
5. Scanner (class)

1. Iterator: used to iterate elements of a collection
2. ListIterator: used to iterate elements of a collection
3. Arrays: It contains several useful methods, used to perform on Arrays
4. Collections: It contains several useful methods, used to perform on collections
5. Scanner: used to accept data from keyboard, file & network.

Legacy Collections

1. Enumeration (interface)
2. StringTokenizer (class)
3. Vector (class)
4. Dictionary (abstract class)
5. HashTable (class)
6. Random (class)
7. Stack (class)
8. Date (class)

1. Enumeration (interface): Similar to Iterator interface
2. StringTokenizer : allows to an application to break a String into Tokens
3. Vector : Similar to ArrayList class
4. Dictionary : Similar to Map interface
5. HashTable : Similar to HashMap class
6. Random : This class generates random Integers, floating point numbers & boolean values
7. Stack : called as Last In ~~Fast~~ out list & allows insertion & deletion at top end only
8. Date : used to get System date & Time

⇝ Formatter class

The Formatter is a built-in class in Java used for layout justification & alignment, common formats for numeric, String, date/time data. The Formatter class is defined as final class inside the java.util package.

- The Formatter class implements Closeable & Flushable interface.

Constructors:

1. Formatter () : creates new Formatter
2. Formatter(Appendable a) : Creates new formatter with specified destination.
3. Formatter (File file) : creates new formatter with specified file
4. Formatter (File file, String charset) : creates new formatter with specified file & character set
5. Formatter (Locale l) : creates new formatter with specified locale.
6. Formatter (OutputStream os) : Creates new formatter with specified output stream.

7. Formatter(PrintStream ps): creates new formatter with specified print stream

8. Formatter(String fileName): creates new formatter with specified file name.

methods:

1. Formatter format(Locale l, String format, Object ... args)
   - it writes a formatted string to the invoking objects destination using the specified locale, format string & arguments

2. void flush(): it flushes the invoking formatter
3. Appendable out(): it returns the destination of the output
4. Locale locale(): returns locale set by the construction of the invoking formatter
5. String toString(): converts invoking objects to String
6. IOException ioException(): returns the IOException last thrown by the invoking formatters Appendable
7. void close(): closes the invoking formatter.

⇝ Random class

It is in java.util package. It generates "random integers, floating point numbers & boolean values"

java.util.Random

Constructor: public Random();

methods: 1. public int nextInt();
it returns one number within int range

2. public int nextInt(int);
it returns one number between 0 & specified number

3. public boolean nextBoolean();
4. public float nextFloat();

Ex:

```
import java.util.*;
class A
{
public static void main(String args[])
{
  Random r = new Random();
  boolean b = r.nextBoolean();
  System.out.println(b);
  for(int i=1; i<=10; i++)
  {
    System.out.println(r.nextInt(100));
  }
}
}
```

o/p:
```
1  80
99  9
87  11
44  97
55
67
```

Time package — java.time

This package is used for all the date, time related operations. The various classes in this package are.

1. clock :- provides access to current date, time according to the zone specified.
2. Duration :- it is the amount of time spent/recorded etc.
3. Instant :- current time.
4. LocalDate :- it is a date without any specified time zone.
5. LocalDateTime :- it is a date-time without a time zone.
6. LocalTime :- it represents a time without a time zone.
7. MonthDay :- it represents a day of the month in the calendar
8. OffsetDateTime :- the class represents a date time with an offset
9. OffsetTime :- it represents the time with an offset

10. period :- represents a amount of time
11. year :- it represents a year in the calendar system
12. YearMonth :- it represents a combination of the month & the year with respect to the calendar system
13. ZonedDateTime :- represents a date-time with the specified time zone.
14. ZoneId :- gives the idea of any time zone
15. ZoneOffset :- represents the offset of any time zone

=> class Instant

It is in java.times package. it is used to represent the specific moment on the timeline. it inherits the Object class & implements the Comparable interface

java.time.Instant

public final class Instant extends Object implements Temporal, TemporalAdjuster, Comparable, Serializable

methods:

1. Temporal adjustInto(Temporal temporal):- used to adjust the specified temporal object to have this instant.
2. int get(TemporalField field):- used to get the value of the specified field from this instant as an int
3. boolean isSupported(TemporalField field):- used to check if the specified field is supported
4. Instant minus(TemporalAmount amountToSubtract):- used to return a copy of this instant with the specified amount subtracted
5. static Instant now():- used to obtain the current instant from the system clock
6. static Instant parse(CharSequence text):- used to obtain an instance of Instant from a text String such as 2007-12-03T10:15:30.002.

7. Instant plus(TemporalAmount amountToAdd):- used to return a copy of this instant with the specified amount added

8. Instant with(TemporalAdjuster adjuster):- used to return an adjusted copy of this instant

Ex:

```
import java.time.Instant;
public class Instant
{
  public static void main(String args[])
  {
    Instant inst = Instant.parse("2017-02-03T10:37:30.00Z");
    System.out.println(inst);
  }
}
```

o/p: 2017-02-03T10:37:30Z

Ex:

```
import java.time.*;
public class Instant
{
  public static void main(String args[])
  {
    Instant inst = Instant.parse("2017-02-03T11:25:30.00Z");
    Instant inst2 = inst.plus(Duration.ofDays(125));
    System.out.println(inst2);
  }
}
```

o/p: 2017-06-08T11:25:30Z

⇒ Formatting for Date/Time in Java

Formatting is used to separate the date from the time. So we can use DateTimeFormatter class with the ofPattern() method in the same package to format date-time objects. In the above example "T" & nano-seconds are display but by using ofPattern() method we will remove both T & nano-seconds. from the date-Time.

Ex:

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class A
{
public static void main(String args[])
{
LocalDateTime obj = LocalDateTime.now();
System.out.println("before formatting:" + obj);
DateTimeFormatter obj2 = DateTimeFormatter.ofPattern
("dd-MM-YYYY HH:mm:ss");
String formattedDate = obj.format(obj2);
System.out.println("After formatting:" + formattedDate);
}}
```

O/P: Before formatting : 2021-04-27T11:54:30.526283
After formatting : 27-04-2021 11:54:30.

The ofPattern() method accepts all sorts of values.

Ex:

YYYY-MM-dd	- 1988-09-29
dd/MM/YYYY	- 29/09/1988
dd-MMM-YYYY	- 29-Sep-1988
E, MMM dd YYYY	- Thu, Sep 29 1988

Temporal Adjusters class

java.time.temporal.TemporalAdjusters

This class provides Adjusters, which are a key tool for modifying temporal objects. i.e date like "second Saturday of the month" or "next Tuesday" etc.

This class can be used in two ways

1. invoking the method on the interface directly
2. by using Temporal.with(TemporalAdjuster):

but 2nd way is recommended approach.

methods:

1. dayOfWeekInMonth(int ordinal, DayOfWeek dayOfWeek)
2. firstDayOfMonth()
3. firstDayOfNextMonth()
4. firstDayOfNextYear()
5. firstDayOfYear()
6. firstInMonth(DayOfWeek dayOfWeek)
7. lastDayOfMonth()
8. lastDayOfYear()
9. lastInMonth(DayOfWeek dayOfWeek)
10. next(DayOfWeek dayOfWeek)
11. nextOrSame(DayOfWeek dayOfWeek)
12. previous(DayOfWeek dayOfWeek)
13. previousOrSame(DayOfWeek dayOfWeek)

Ex:

```
import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.temporal.TemporalAdjusters;

public class A
{
public static void main(String args[])
{
    A a=new A();
    a.testAdjusters();
}
public void testAdjusters()
{
LocalDate date1=LocalDate.now();
System.out.println("Today's date:" + date1);
LocalDate nextTuesday = date1.with(TemporalAdjusters.
        next(DayOfWeek.MONDAY));
System.out.println("Next Monday is" + nextTuesday);
LocalDate firstInYear=LocalDate.of(date1.getYear(),
        date1.getMonth(), 1);
LocalDate SecondSaturday = firstInYear.with(TemporalAdjusters.
    nextOrSame(DayOfWeek.SATURDAY)).with(TemporalAdjusters.
    next(DayOfWeek.SATURDAY));
System.out.println("Second saturday is" + secondSaturday);
```

O/P:

```
Today's date : 2021-02-24
Next Monday is : 2021-03-01
Second Saturday is : 2021-02-13.
```