Java clases

“this” es refereix a l’objecte que crida la funció, es refereix a l’objecte amb el que s’està treballant. No fa referència a la classe en si, sinó a un objecte de la classe.

si fem .length d’una matriu ens diu la quantitat de files, si fem matriu[x].length ens diu la quantitat de columnes que té la fila x de la matriu

para vincular una clase a otra a java i netbeans fem:

import ruta;

com a vegades no em funciona millor posar els arxius tots en la meteixa carpeta i fer un import amb la carpeta i larxiu:

import newpackage.persona;

public **abstract** void nomfuncio(){} - **abstract** al aplicarlo en el metode d’una classe fa que les classes filles hagin d’implementar i sobreescriure aquest metode si o si, sino no compilara be i donara un error. Nms es posa en les classes que tmb porten **abstract: public abstract class nom{}**

Castear/cambiar el tipo de variable

double kk = 100.04;

//casting de tipo

long qq = (long)kk;

//casting de tipo

int mm = (int)qq;

**EL FOR EXTENDIDO O BUCLES FOR EACH EN JAVA**

 for ( TipoARecorrer nombreVariableTemporal : nombreDeLaColección ) {

                                               Instrucciones

       }

Classes Generiques

<https://javadesdecero.es/avanzado/genericos-ejemplos-java/>

Como se explicó, T es un marcador de posición para el tipo real que se especificará cuando se cree un objeto Gen. Así, ob será un objeto del tipo pasado a T. Por ejemplo, si el tipo String es pasado a T, entonces en ese caso, ob será del tipo ***String***.

La sintaxis genérica que se muestra en los ejemplos anteriores puede generalizarse. Aquí está la sintaxis para declarar una clase genérica:

class nombre-clase<lista-parametros-tipo> { // ...

Aquí está la sintaxis completa para declarar una referencia a una clase genérica y crear una instancia genérica:

nombre-clase<lista-argumentos-tipo> nombre-var =

new nombre-clase<lista-argumentos-tipo>(lista-arg-cons);

Instanceof

Object op = new Object();

Productes p = new Productes();

Object opa = new Object();

Packs pa = new Packs();

op = (Object) p; //prod

opa = (Object) pa; //pack

System.out.println(op instanceof Packs); //false

System.out.println(opa instanceof Packs); //true

System.out.println(op instanceof Productes); //true

System.out.println(opa instanceof Productes); //da true pq aunque sea un Pack, un objeto pack sigue siendo tmb un objeto

//Productes, pero no viceversa!

//instanceof detecta si un objeto tipo Object viene de ser objeto de otra clase (como Packs) o no

//en este caso pa era Packs por lo que opa "es" tmb de la clase Packs

## Tipos de wrappers en Java

**Existen 8 tipos primitivos predefinidos en**[**Java**](https://profile.es/blog/librerias-java/)**,** cada uno de los tipos de tipos de datos primitivos **tienen asociados su correspondiente clase Wrapper para poder realizar las conversiones de tipos primitivos a objetos (tipos no primitivos).**

|  |  |
| --- | --- |
| **Tipos primitivos (no son objetos y por tanto no poseen métodos)** | **Wrappers(son objeto y por tanto poseen métodos)** |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| boolean | Boolean |
| float | Float |
| double | Double |
| char | Character |

### LocalDate, LocalTime y LocalDateTime

https://devs4j.com/2018/10/30/java-8-manejo-de-fechas-y-tiempo-localdate-localtime-y-localdatetime/

Las clases más comunes para el manejo de fechas con java 8 son LocalDate, LocalTime y LocalDateTime, se utilizan cuando la zona horaria no es requerida.



**Fecha:**[**octubre 30, 2018**](https://devs4j.com/2018/10/30/java-8-manejo-de-fechas-y-tiempo-localdate-localtime-y-localdatetime/)**Autor/a:**[**raidentrance**](https://devs4j.com/author/raidentrance/)[**2 Comentarios**](https://devs4j.com/2018/10/30/java-8-manejo-de-fechas-y-tiempo-localdate-localtime-y-localdatetime/#comments)

4 Votes

Uno de los problemas más comunes a los que se enfrentan los desarrolladores es el manejo de fechas, tiempo y zonas horarias, para resolver este problema se solían utilizar api’s como Joda Time, Java 8 incluyó un conjunto de apis que nos ayudan a resolverlo sin incluir dependencias adicionales.

### LocalDate, LocalTime y LocalDateTime

Las clases más comunes para el manejo de fechas con java 8 son LocalDate, LocalTime y LocalDateTime, se utilizan cuando la zona horaria no es requerida.

#### LocalDate

Un LocalDate representa una fecha en formato ISO (yyyy-MM-dd) **sin tiempo.** Veamos algunos ejemplos:

|  |  |
| --- | --- |
| 1  2 | LocalDate date = LocalDate.now();  System.out.println(date); |

El código anterior tendrá la siguiente salida(Entendiendo que el post se escribió el 30-10-2018):

|  |  |
| --- | --- |
| 1 | 2018-10-30 |

Como vemos es una fecha **sin tiempo y sin zona horaria.**Veamos algunas otras formas de crear un LocalDate:

|  |  |
| --- | --- |
| 1  2 | LocalDate date2 = LocalDate.of(2018, 10, 30);  LocalDate date3 = LocalDate.parse("2018-10-30"); |

* Manipulación de fechas (Sumar o restar días, meses, años, etc ):

|  |  |
| --- | --- |
| 1  2  3  4 | LocalDate date = LocalDate.parse("2018-10-30");  LocalDate newDate = date.plusDays(10);  System.out.println(date);  System.out.println(newDate); |

Salida:

|  |  |
| --- | --- |
| 1  2 | 2018-10-30  2018-11-09 |

|  |
| --- |
| LocalDate date = LocalDate.parse("2018-10-30");  LocalDate newDate = date.plusMonths(3);  System.out.println(date);  System.out.println(newDate); |

Salida:

|  |  |
| --- | --- |
| 1  2 | 2018-10-30  2019-01-30 |

Valida si una fecha es antes que otra:

System.out.println(LocalDate.parse("2018-10-30").isBefore(LocalDate.parse("2018-10-31")));

Salida

|  |  |
| --- | --- |
| 1 | True |

Valida si un año es bisiesto:

System.out.println(LocalDate.parse("2018-10-30").isLeapYear());

#### LocalTime

LocalTime representa una hora sin la fecha, del mismo modo que con LocalDate podemos crearlo haciendo uso de los métodos now(), parse(..) y of(..), veamos algunos ejemplos:

|  |
| --- |
| LocalTime time = LocalTime.now();  LocalTime time2 = LocalTime.parse("11:00:59.759");  LocalTime time3 = LocalTime.of(11, 00, 59);  System.out.println(time);  System.out.println(time2);  System.out.println(time3); |

Salida:

|  |  |
| --- | --- |
| 1  2  3 | 11:02:06.198  11:00:59.759  11:00:59 |

La primera operación que veremos es como modificar un LocalTime:

|  |  |
| --- | --- |
| 1  2  3 | LocalTime time = LocalTime.parse("11:00:59.759");  LocalTime time2 = time.plusHours(1);  System.out.println(time2); |

Salida:

|  |  |
| --- | --- |
| 1 | 12:00:59.759 |

El siguiente punto será hacer validaciones sobre un LocalTime:

|  |  |
| --- | --- |
| 1  2  3 | LocalTime time = LocalTime.parse("11:00:59.759");  LocalTime time2 = LocalTime.parse("12:00:59.759");  System.out.println(time.isBefore(time2)); |

Salida:

|  |  |
| --- | --- |
| 1 | true |

El siguiente paso será extraer solo una parte del objeto LocalTime:

|  |  |
| --- | --- |
| 1  2 | LocalTime time = LocalTime.parse("11:00:59.759");  System.out.println(time.getHour()); |

#### LocalDateTime

La siguiente clase a analizar será **LocalDateTime**la cual representa una combinación entre LocalDate y LocalTime, veamos como crearlo:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | LocalDateTime dateTime = LocalDateTime.now();  LocalDateTime dateTime1=LocalDateTime.of(2018, 10, 10, 11, 25);  LocalDateTime dateTime2=LocalDateTime.parse("2018-10-10T11:25");    System.out.println(dateTime);  System.out.println(dateTime1);  System.out.println(dateTime2); |

Salida:

|  |  |
| --- | --- |
| 1  2  3 | 2018-10-30T11:46:58.274  2018-10-10T11:25  2018-10-10T11:25 |

De igual forma que con los anteriores podemos realzar manipulaciones sobre el LocalDateTime:

|  |  |
| --- | --- |
| 1  2  3  4 | LocalDateTime dateTime=LocalDateTime.parse("2018-10-10T11:25");  LocalDateTime newDateTime = dateTime.plusDays(1).plusHours(2);    System.out.println(newDateTime); |

**Lo anterior creará un objeto LocalDateTime le agregará 1 día y después 2 horas, recordemos que debemos asignar el resultado a una nueva referencia ya que el objeto original no se modificará sino que se devolverá uno nuevo.**

Como lo vimos en los ejemplos anteriores podemos realizar validaciones sobre el LocalDateTime:

|  |  |
| --- | --- |
| 1  2  3  4 | LocalDateTime dateTime=LocalDateTime.parse("2018-10-10T11:25");  LocalDateTime dateTime2=LocalDateTime.parse("2019-10-10T11:25");    System.out.println(dateTime.isBefore(dateTime2)); |

Salida:

|  |  |
| --- | --- |
| 1 | true |

**Otras funciones con date:**

//compara los dias y devuelve la diferencia en dias, meses o años. Dependiendo de la diferencia, mostrara dias, meses o años redondeandolos

System.out.println(day2.compareTo(day1));

if(day1.compareTo(day2) == 0) {

System.out.println("day1 = day2");

}else if(day1.compareTo(day2) < 0) {

System.out.println("day1 < day2");

}else if(day1.compareTo(day2) > 0) {

System.out.println("day1 > day2");

}

LocalDate day1 = askDate(sc);

LocalDate day2 = askDate(sc);

//metodo que nos devuelve la diferencia entre dos fechas, pudiendo filtrar los resultados en dias, meses y años

Period p = Period.between(day1, day2);

System.out.println(p.getYears()+ " year, "+p.getMonths()+" months, "+p.getDays()+" days");

//metodo que nos permite ver la diferencia entre 2 fechas y dependiendo del enum nos devolvera el resultado en dias, meses, años, centenarios, etc.

long daysDifference = ChronoUnit.DAYS.between(day1, day2);

System.out.println("Diference days "+daysDifference);

//podemos separar una fecha en dias, meses, años, etc.

LocalDate today = LocalDate.now();

System.out.println("Day: "+today.getDayOfMonth()+" Month: "+today.getMonthValue()+ " Year: "+today.getYear());

**Otras funciones con time:**

ZoneId applePie = ZoneId.of("America/New\_York");

LocalTime murica = LocalTime.now(applePie);

ZoneId relaxingCupOfCafeConLecheInPlazaMayor = ZoneId.of("Europe/Madrid"); //debemos obtener el id de la zona que queremos

LocalTime spain = LocalTime.now(relaxingCupOfCafeConLecheInPlazaMayor); //si le pasamos el id, nos dira la hora actual en dicha region

System.out.println(spain);

System.out.println(murica);

**datetimeformatter:**

Learn to use DateTimeFormatter for formatting ZonedDateTime, LocalDateTime, LocalDate and LocalTime to string with predefined and custom patterns.

We can create a very simple DateTimeFormatter instance in 3 ways:

1. Using predefined constants, such as ISO\_LOCAL\_DATE
2. Creating **custom patterns** using pattern latters and ofPattern() method
3. Using localized styles with FormatStyle, such as LONG or MEDIUM

*//Use inbuilt pattern constants*

DateTimeFormatter inBuiltFormatter1 = DateTimeFormatter.ISO\_DATE\_TIME;

DateTimeFormatter inBuiltFormatter2 = DateTimeFormatter.ISO\_LOCAL\_DATE\_TIME;

*//Define your own custom patterns*

DateTimeFormatter customFormatter = DateTimeFormatter.ofPattern("MM/dd/yyyy 'at' hh:mma z");

*//Using FormatStyle*

DateTimeFormatter customFormatter = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.LONG);

Exemples:

### Formatting ZonedDateTime

*//Create formatter*

DateTimeFormatter FOMATTER = DateTimeFormatter.ofPattern("MM/dd/yyyy 'at' hh:mm a z");

*//Zoned datetime instance*

ZonedDateTime zdt = ZonedDateTime.now();

*//Get formatted String*

String zdtString = FOMATTER.format(zdt);

System.out.println(zdtString); *// 07/15/2018 at 02:51 PM IST*

### Formatting LocalDateTime

LocalDate does not have a timezone part so create the pattern accordingly.

*//Create formatter*

DateTimeFormatter FOMATTER = DateTimeFormatter.ofPattern("MM/dd/yyyy 'at' hh:mm a");

*//Local date time instance*

LocalDateTime localDateTime = LocalDateTime.now();

*//Get formatted String*

String ldtString = FOMATTER.format(localDateTime);

System.out.println(ldtString); *// 07/15/2018 at 02:49 PM*

### Formatting LocalDate

LocalDate does not have time and timezone parts. So create the pattern accordingly.

*//Create formatter*

DateTimeFormatter FOMATTER = DateTimeFormatter.ofPattern("MM/dd/yyyy");

*//Local date instance*

LocalDate localDate = LocalDate.now();

*//Get formatted String*

String dateString = FOMATTER.format(localDate);

System.out.println(dateString); *//07/15/2018*

### Formatting LocalTime

LocalTime does not have the date and timezone parts so create the pattern accordingly.

*//Create formatter*

DateTimeFormatter FOMATTER = DateTimeFormatter.ofPattern("hh:mm a");

*//Local time instance*

LocalTime localTime = LocalTime.now();

*//Get formatted String*

String localTimeString = FOMATTER.format(localTime);

System.out.println(localTimeString); *// 02:53 PM*

**Internacionalització i localització**