WORKING WITH JAVA DATES

WORKING WITH JAVA DATES

DATE TIME IN JAVA CAN BE CONFUSING TO WORK WITH, THERE IS A PROLIFERATION OF LIBRARY CLASSES AND IT'S HARD TO SEE WHAT IS THE RIGHT ONE TO USE

THE JAVA 8 CLASSES ARE A LITTLE DIFFERENT FROM PREVIOUS VERSIONS, SOME NEWER CLASSES SUCH AS INSTANT AND DURATION HAVE BEEN INTRODUCED

THIS CLASS WILL GO THROUGH THE MOST COMMONLY USED CLASSES AND BUILD A FOUNDATION OF HOW TO WORK WITH THEM

```
System.currentTimeMillis()
```

EPOCH TIME

```
System.currentTimeMillis()
```

THIS RETURNS A NUMBER WHICH REPRESENTS AN INSTANCE IN TIME

THIS NUMBER REPRESENTS THE EXACT TIME DOWN TO THE MILLISECOND FROM THE BEGINNING OF TIME

?

WHAT IS THE BEGINNING OF TIME?

00:00:00 COORDINATED UNIVERSAL TIME (UTC), THURSDAY, 1 JANUARY 1970

00:00:00 COORDINATED UNIVERSAL TIME (UTC), THURSDAY, 1 JANUARY 1970

UNIX BASED SYSTEMS USE THIS AS THE BEGINNING OF TIME SO ALL TIME CAN BE SPECIFIED AS A NUMBER FROM THIS MOMENT ON WARDS

JAVA USES THIS SAME BASIS IN ORDER TO WORK WITH DATE AND TIME CLASSES

THIS DOES NOT INCLUDE LEAP SECONDS AND IS NOT A TRUE REPRESENTATION, HOWEVER IT IS A STANDARD NOW

ALLOWS AN INSTANT TO BE REPRESENTED USING A SINGLE NUMBER - SUPER SIMPLE!

MEASURING TIME IN JAVA

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

    long startTime = System.currentTimeMillis();
    System.out.println("Operation started at : " + startTime);

    veryLongOperationWhichTakesForever();
    long completedTime = System.currentTimeMillis();

    long operationLengthInMillis = completedTime - startTime;
    System.out.println("Milliseconds for the operation to run: " + operationLengthInMillis);
    System.out.println("Seconds for the operation to run: " + operationLengthInMillis / 60);

}
```

SOME TIME WOULD HAVE ELAPSED BASED ON HOW LONG THE OPERATION TOOK, WE'LL NOW GET THE CURRENT INSTANT ONCE AGAIN

java.util.Date

java.util.Date

THIS IS THE MOST COMMONLY USED CLASS THOUGH ITS NOW DEPRECATED

JAVA.UTIL.CALENDAR IS USED IN ITS PLACE NOW

HOWEVER THIS IS STILL WIDELY USED SO ITS HELPFUL TO LEARN TO WORK WITH IT

TAKES THE CURRENT TIME WHEN NO TIME IN MILLISECONDS IS SPECIFIED

```
public static void someFunction() {
    Date d1 = new Date();

    long currentTime = System.currentTimeMillis();
    Date d2 = new Date(currentTime);

    System.out.println("Date d1: " + d1.getTime() + " date d2: " + d2.getTime());

    if (d1.compareTo(d2) <= 0) {
        System.out.println("d1 was instantiated before d2");
    } else {
        System.out.println("d2 was instantiated before d1");
    }
}
```

IMPLEMENTS THE COMPARABLE INTERFACE

METHODS TO GET YEAR, MONTH AND OTHER DATE CONSTRUCTS ARE DEPRECATED IN DATE, USE THE CALENDAR CLASS INSTEAD

java.util.Calendar

java.util.Calendar

VERY GRANULAR FIELDS OF THE CURRENT DATE AND TIME ARE ACCESSIBLE USING CALENDAR CONSTANTS

```
Calendar c = new GregorianCalendar();

int day = c.get(Calendar.DAY_OF_MONTH);
int month = c.get(Calendar.MONTH);
int year = c.get(Calendar.YEAR);
System.out.println(String.format("Today is the %s day of the %s month of year %s", day, month, year));
```

```
// Set the date
c.set(2012 /* year */, 2 /* month */, 13 /* day */);

Date d = new Date();
c.setTime(d);
```

THE DATE CLASS AND THE CALENDAR CAN WORK TOGETHER

CALENDAR ALSO IMPLEMENTS THE Comparable<> INTERFACE

THINGS TO NOTE:

MONTHS IN THE CALENDAR GO FROM 0 - 11
0 FOR JANUARY AND 11 FOR DECEMBER

DAYS OF THE WEEK GO FROM 1-7 WHERE 1
STANDS FOR SUNDAY

HOW DO YOU DISPLAY DATES NICELY FORMATTED?

```
SimpleDateFormat format = new SimpleDateFormat("dd-MM-yyyy");
```

THE "DD-MM-YYYY" SPECIFIES THE FORMAT THAT THIS INSTANCE UNDERSTANDS AND EXPECTS.
DD = DAY, MM = MONTH, YYYY = YEAR

A WHOLE BUNCH OF FORMATS ARE POSSIBLE, THIS IS JUST ONE EXAMPLE

THIS RETURNS A STRING IN THE FORMAT SPECIFIED

```
Date d1 = new Date();
System.out.println(String.format("Today's date is: %s", format.format(d1)));
```

```
try {
    Date d2 = format.parse("13-2-2012");
} catch (ParseException pe) {
    // Do nothing.
}
```

THIS CAN ALSO TAKE IN A STRING IN THAT FORMAT AND PARSE IT TO CREATE A VALID DATE OBJECT

java.time.Instant

java.time.Instant

REPRESENTS ONE PARTICULAR INSTANT IN TIME

GIVES EASY METHODS TO ADD AND SUBTRACT TIME

INSTANT STORES THE INFORMATION IN THE FORM OF SECONDS AND NANOSECONDS FROM EPOCH

```
Instant thisInstant = Instant.now();

System.out.println("Seconds since epoch: " + thisInstant.getEpochSecond());
System.out.println("Nanoseconds since epoch: " + thisInstant.getNano());

Instant tomorrow = thisInstant.plusSeconds(86400);
Instant yesterday = thisInstant.minusSeconds(86400);

System.out.println("This is true: " + tomorrow.isAfter(thisInstant));
System.out.println("This is true: " + yesterday.isBefore(thisInstant));
```

JUST LIKE DATE THIS HAS SIMPLE METHODS TO COMPARE INSTANTS, IT IMPLEMENTS

Comparable<>

java.time.Duration

java.time.Duration

REPRESENTS A PERIOD OF TIME BETWEEN 2 INSTANTS

DURATION OFFERS ALL KINDS OF HANDY METHODS
TO DEAL WITH PERIODS OF TIME

```
Instant thisInstant = Instant.now();
Instant tomorrow = thisInstant.plusSeconds(86400);
Instant yesterday = thisInstant.minusSeconds(86400);

Duration oneDayDuration = Duration.between(thisInstant, tomorrow);
Duration twoDayDuration = Duration.between(yesterday, tomorrow);

System.out.println("Seconds in 2 days: " + twoDayDuration.getSeconds());

Duration threeDayDuration = twoDayDuration.plus(oneDayDuration);
System.out.println("Seconds in three days: " + threeDayDuration.getSeconds());
```

CAN GET THE PERIOD IN
SECONDS AND NANOSECONDS

ALLOWS MANIPULATING PERIODS
USING SIMPLE ADDITION AND
SUBTRACTION OPERATIONS