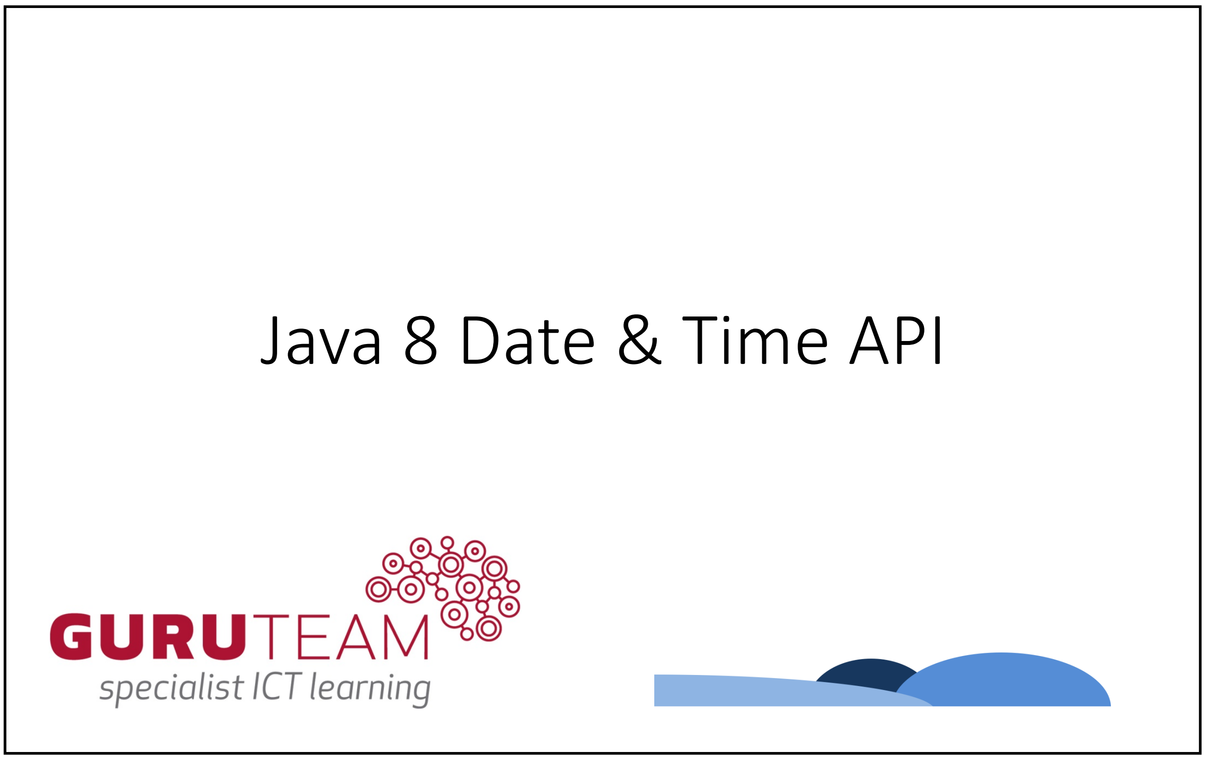
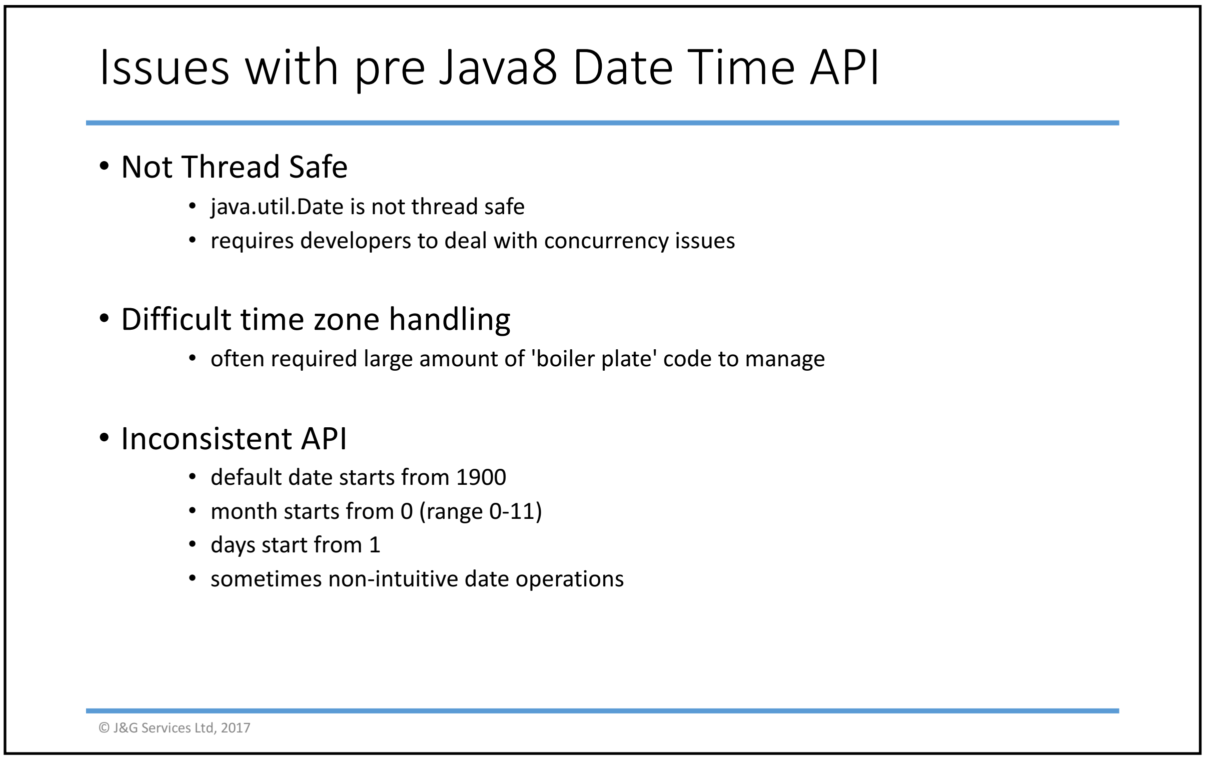


Java 8 Date & Time API

GURUTEAM
specialist ICT learning



Issues with pre Java8 Date Time API

- Not Thread Safe
  - java.util.Date is not thread safe
  - requires developers to deal with concurrency issues
- Difficult time zone handling
  - often required large amount of 'boiler plate' code to manage
- Inconsistent API
  - default date starts from 1900
  - month starts from 0 (range 0-11)
  - days start from 1
  - sometimes non-intuitive date operations

New Date Time API

- In java.time package
- Thread safe
  - new date time API is immutable
  - does not have setter methods
- Consistent API with numerous utility methods
- Provides a Local API
  - a simplified data time API avoiding time zone issues
  - makes it easier to work with dates when time zones are not required
- Also provides a Zoned API
  - specialized data time API for time zones

New Date Time API

- Date time packages
  - consists of the primary java.time package
  - plus four *sub* packages
- java.time
  - core of the new API
- java.time.chrono
  - API for representing calendar systems
- java.time.format
  - supports formatting and parsing dates and times
- java.time.temporal
  - extended API
- java.time.zone
  - support different time zones and offsets

Method Naming Conventions

- Method names follow consistent pattern

Prefix	Method Type	Use
of	static factory	Object creation
from	static factory	Converts from input to target type
parse	static factory	Parse the input string
get	instance	Returns a part of the state
is	instance	Queries the state of the target
with	instance	Returns a copy with one element changed
plus	instance	Returns copy of object with amount of time added
minus	instance	Returns copy of object with amount of time subtracted
to	instance	Converts receiver to another type

Local Date Time API

- Comprises LocalDate, LocalTime, LocalDateTime
- Termed 'Local' as they represent date and time
  - from the context of the local runtime
  - i.e. from the perspective of the observer
- Time Zones not supported by the Local Date Time API
  - simplifies API
- Objects created from factory classes
  - follow Date Time API factory name conventions
  - `LocalDateTime.now();`
  - `LocalDate.of(2016, Month.JANUARY, 12);`
  - `LocalTime.of(17, 18);`
  - `LocalTime.parse("20:15:30");`

Local Date Time API

- LocalDate, LocalTime, LocalDateTime

```
import java.time.*;
...
LocalDateTime currentTime = LocalDateTime.now();
System.out.println("Current DateTime: " + currentTime);

LocalDate d1 = currentTime.toLocalDate();
System.out.println("d1: " + d1);

LocalDate d2 = LocalDate.of(2016, Month.JANUARY, 12);
System.out.println("d2: " + d2);

LocalTime d3 = LocalTime.parse("20:15:30");
System.out.println("d3: " + d3);
```

```
Current DateTime: 2016-01-
29T17:41:37.591
d1: 2016-01-29
d2: 2016-01-12
d3: 20:15:30
```

Local Date Time API

- Getter conventions used to obtain values

```
int year = currentTime.getYear();
Month month = currentTime.getMonth();
int day = currentTime.getDayOfMonth();
DayOfWeek dayOfWeek = currentTime.getDayOfWeek();
int dayOfYear = currentTime.getDayOfYear();
int hours = currentTime.getHour();
int minutes = currentTime.getMinute();
int seconds = currentTime.getSecond();


System.out.println(
    "Year: " + year +
    "\nMonth: " + month +
    "\nDay: " + day +
    "\nDay Of Week: " + dayOfWeek + ", Day of Year: " + dayOfYear +
    "\nHours: " + hours + ", Minutes: " + minutes +
    "\nSeconds: " + seconds);
```

```
Year: 2016
Month: FEBRUARY
Day: 8
Day Of Week: MONDAY, Day of Year:
39
Hours: 11, Minutes: 5
Seconds: 9
```

Local Date Time API

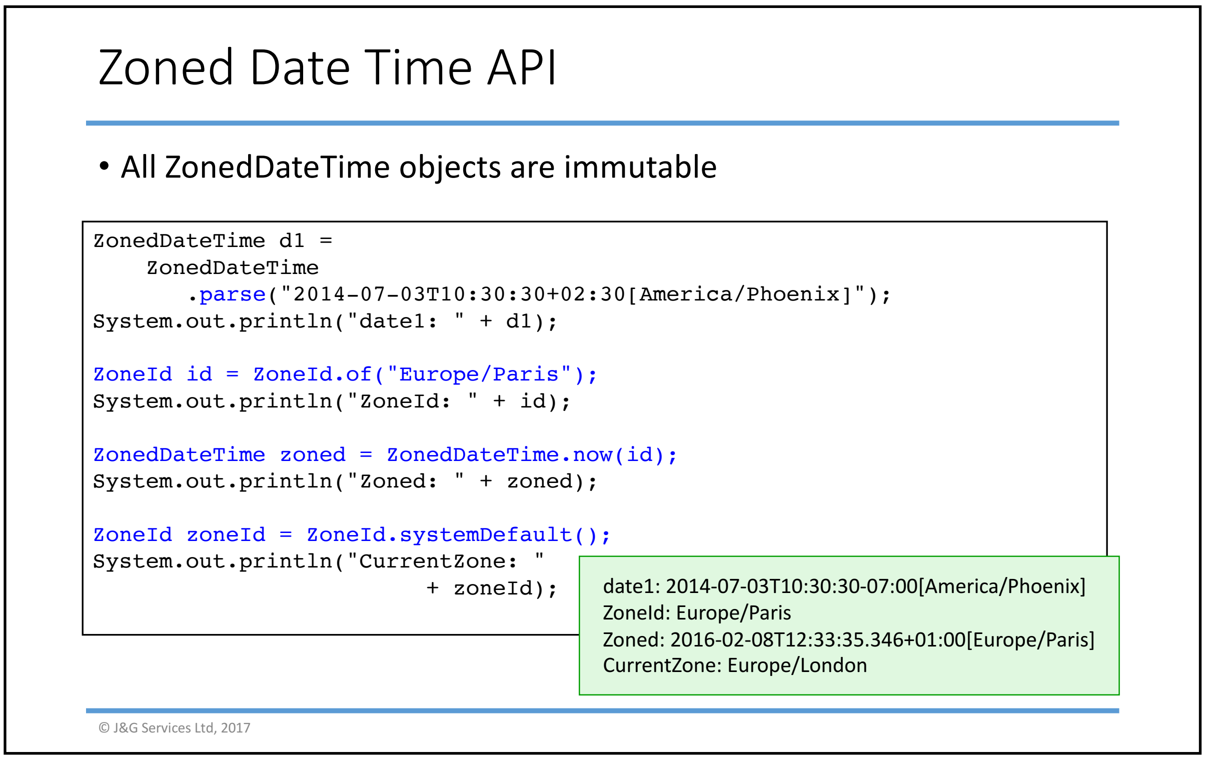
- Can perform operations on dates / times
  - as immutable these operations create new objects

```
System.out.println("Current DateTime: " + currentTime);
LocalDateTime dx1 = currentTime.withYear(2020);
System.out.println("dx1: " + dx1);

LocalDateTime dx2 = currentTime.plusWeeks(4);
System.out.println("dx2: " + dx2);

LocalDateTime dx3 = currentTime.withDayOfMonth(25);
System.out.println("dx3: " + dx3);

LocalDateTime dx4 = currentTime.withHour(15);
System.out.println("dx4: " + dx4);
```

```
Current DateTime: 2016-02-08T11:13:38.886
dx1: 2020-02-08T11:13:38.886
dx2: 2016-03-07T11:13:38.886
dx3: 2016-02-25T11:13:38.886
dx4: 2016-02-08T15:13:38.886
```

Zoned Date Time API

- Comprises ZonedDateTime, ZoneId
- ZonedDateTime for a fully qualified time zone
  - represents date and time with respect to a time zone
- ZoneId is an identifier for a region
  - use instead of literal strings
  - can be identified by short form "PLT"
  - or long from "Europe/London"
- ZoneOffset is the period of time representing
  - the difference between Greenwich/UTC and
  - a specific time zone

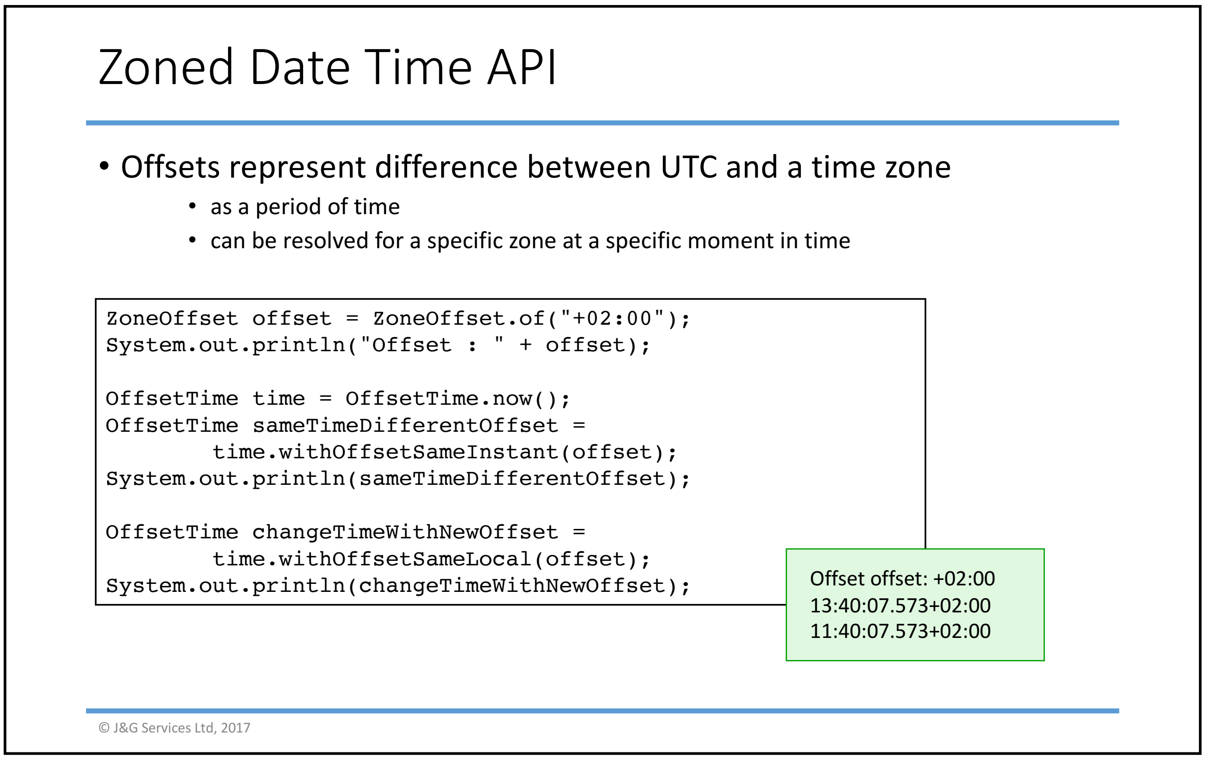


Zoned Date Time API

- All ZonedDateTime objects are immutable

```
ZonedDateTime d1 =
    ZonedDateTime
       .parse("2014-07-03T10:30:30+02:30[America/Phoenix]");
System.out.println("date1: " + d1);

ZoneId id = ZoneId.of("Europe/Paris");
System.out.println("ZoneId: " + id);

ZonedDateTime zoned = ZonedDateTime.now(id);
System.out.println("Zoned: " + zoned);

ZoneId zoneId = ZoneId.systemDefault();
System.out.println("CurrentZone: "
                           + zoneId);
```

```
date1: 2014-07-03T10:30:30-07:00[America/Phoenix]
ZoneId: Europe/Paris
Zoned: 2016-02-08T12:33:35.346+01:00[Europe/Paris]
CurrentZone: Europe/London
```



Zoned Date Time API

- Offsets represent difference between UTC and a time zone
  - as a period of time
  - can be resolved for a specific zone at a specific moment in time

```
ZoneOffset offset = ZoneOffset.of("+02:00");
System.out.println("Offset : " + offset);

OffsetTime time = OffsetTime.now();
OffsetTime sameTimeDifferentOffset =
        time.withOffsetSameInstant(offset);
System.out.println(sameTimeDifferentOffset);

OffsetTime changeTimeWithNewOffset =
        time.withOffsetSameLocal(offset);
System.out.println(changeTimeWithNewOffset);
```

```
Offset offset: +02:00
13:40:07.573+02:00
11:40:07.573+02:00
```

ChronoUnit

- `java.time.temporal.ChronoUnit` enumerated type
  - replaces integers used in old API
  - represents hours, days, weeks, months, years, decades etc.

```
import java.time.*;
import java.time.temporal.ChronoUnit;
...
LocalDate today = LocalDate.now();
System.out.println("Current date:  " + today);

// add 1 week to the current date
LocalDate nextWeek = today.plus(1, ChronoUnit.WEEKS);
System.out.println("Next week:  " + nextWeek);
```

```
Current date: 2016-01-29
Next week: 2016-02-05
```

Period and Duration

- Period represents date based temporal periods
- Duration represents time based periods

```
import java.time.*;
import java.time.temporal.ChronoUnit;
...
LocalDate today = LocalDate.now();
LocalDate nextWeek = today.plus(1, ChronoUnit.WEEKS);
Period period = Period.between(today, nextWeek);
System.out.println("Period:  " + period);

System.out.println("---");

LocalTime time1 = LocalTime.now();
Duration twoHours = Duration.ofHours(2);
LocalTime time2 = time1.plus(twoHours);
Duration duration = Duration.between(time1, time2);
System.out.println("Duration:  " + duration);
```

```
Period: P7D
---
Duration: PT2H
```

Temporal Adjusters

- Perform date based calculations
  - e.g. get the next Monday

```
import java.time.*;
import java.time.temporal.TemporalAdjusters;
...
LocalDate today = LocalDate.now();
System.out.println("Current date: " + today);

// get the next Monday
LocalDate nextMonday =
    today.with(
        TemporalAdjusters.next(DayOfWeek.MONDAY));

System.out.println("Next Monday on : " + nextMonday);
```

```
Current date: 2016-01-29
Next Monday on : 2016-02-01
```

Temporal Query

- TemporalQuery interface
  - implementations can be used to retrieve information
  - from temporal objects
  - e.g. query a date
- Can be used to answer questions such as
  - is the market open today?
  - what is the current quarter?
  - is daylight saving being used?
- Predefined set of queries available
  - see TemporalQueries (note the name) class
  - set of static methods to perform common queries
  - e.g. obtain the current precision of a date

Temporal Query

- TemporalQueries utility class
  - with several static query methods

```
ZonedDateTime d = ZonedDateTime.now(ZoneId.of("Europe/Paris"));
System.out.println("Date object: " + d);
TemporalQuery<ZoneId> q1 = TemporalQueries.zone();
System.out.println("Zone: " + d.query(q1));
TemporalQuery<TemporalUnit> q2 = TemporalQueries.precision();
System.out.println("Precision: " + d.query(q2));
TemporalQuery<ZoneOffset> q3 = TemporalQueries.offset();
System.out.println("Offset: " + d.query(q3));
TemporalQuery<Chronology> q4 = TemporalQueries.chronology();
System.out.println("Chronology: " + d.query(q4));
```

```
Date object: 2016-12-09T11:10:30.585+01:00[Europe/Paris]
Zone: Europe/Paris
Precision: Nanos
Offset: +01:00
Chronology: ISO
```

Temporal Query

- Can define custom queries
  - implement TemporalQuery interface

```
class FestiveSeasonQuery implements TemporalQuery<Boolean> {
    public Boolean queryFrom(TemporalAccessor temporal) {
        LocalDate date = LocalDate.from(temporal);

        MonthDay first =
                  MonthDay.of(Month.DECEMBER.getValue(), 1);
        MonthDay last =
                  MonthDay.of(Month.DECEMBER.getValue(), 30);

        return (date.isAfter(first.atYear(date.getYear())) &&
                date.isBefore(last.atYear(date.getYear())));
        }
}
```

```
LocalDateTime date = LocalDateTime.now();
System.out.println("Date object: " + date);
System.out.println(date.query(new FestiveSeasonQuery()));
```

```
Date object: 2016-12-09T10:15:37.042
true
```

Truncation

- API supports different precision time points
  - supported by the `truncatedTo` method

```
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;
...
LocalTime time = LocalTime.now();
System.out.println("Time: " + time);

LocalTime truncatedTime =
    time.truncatedTo(ChronoUnit.SECONDS);

System.out.println("truncated Time: " + truncatedTime);
```

```
Time: 11:23:13.734
truncated Time: 11:23:13
```

Clock

- Several methods take a clock as an argument
  - e.g. now(Clock)
- Clocks are used to ensure that the date/time
  - is created with respect to the correct time-zone
- Can be useful when *testing* code
- Clock class is abstract
  - use factory methods to create clocks
  - e.g. Clock.fixed(Instant, ZoneId)

Backwards Compatibility

- `toInstant` added to Date and Calendar for conversions
- `ofInstant` to get local or Zoned objects

```
import java.time.*;
import java.util.Date;
...
Date date = new Date();
System.out.println("Current date: " + date);
// Get the instant of current date in terms of milliseconds
Instant now = date.toInstant();
ZoneId currentZone = ZoneId.systemDefault();
LocalDateTime local =
    LocalDateTime.ofInstant(now, currentZone);
System.out.println("Local date: " + local);
ZonedDateTime zoned =
    ZonedDateTime.ofInstant(now, currentZone);
System.out.println("Zoned date: " + zone
```

```
Current date: Fri Jan 29 17:53:05 GMT 2016
Local date: 2016-01-29T17:53:05.275
Zoned date: 2016-01-29T17:53:05.275Z[Europe/London]
```