Learning Java - A Foundational Journey

Session: 10

New Date and Time API

Objectives

- Explain classes of the Date and Time API in Java 8 and later versions
- Explain `Enum` and `Clock` types
- Describe the role of time-zones
- Explain support for backward compatibility in the new API
- Explain about Stream of Dates

Introduction

- Each programming language has a unique set of built-in routines to work with date and time.
- A new Date-Time API is introduced from Java 8 onwards, which offers a solution for many unaddressed drawbacks in the earlier API. It is intended to address the following issues:

Thread-safe issue

- As `java.util` is not thread-safe, developers had a tough time in dealing with concurrency issues while using date related data.

Poor design

- Default 'date' in earlier versions of Java starts from 1900; 'month' starts from one and 'day' starts from zero, hence, there is no uniformity.

Time-zone handling issue

- Earlier, developers had to write a lot of code to deal with Time-zone issues. The new API ha been developed keeping a domain-specific design in mind.

Classes in the New Date-Time API

- All classes of the new Date-Time API are located within the `java.time` package.

Class
Clock
Duration
Instant
LocalDate
LocalDateTime
LocalTime
MonthDay
OffsetDateTime
OffsetTime
Period
Year
YearMonth
ZonedDateTime
ZoneId
ZoneOffset

Clock Class 1-2

To get the current instant, date, and time using time-zone:

Developers can use `Clock` in place of `System.currentTimeInMillis()` and `TimeZone.getDefault()`

Duration Calculations

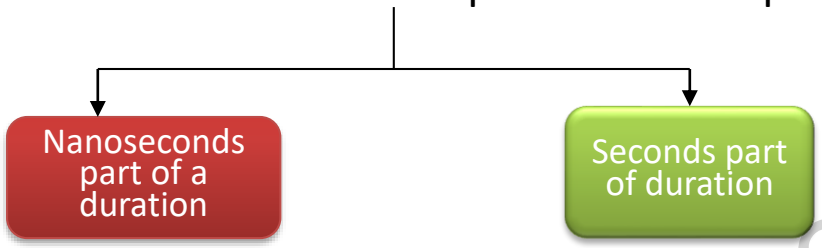
Comprises a set of methods that can be used to perform calculations based on a `Duration` object:

```
plusNanos()
minusNanos()
plusMillis()
minusMillis()
plusSeconds()
minusSeconds()
plusMinutes()
minusMinutes()
plusHours()
minusHours()
plusDays()
minusDays()
```

Clock Class 2-2

Accessing the Time of a Duration

A `Duration` instance comprises two components:



Methods to retrieve these values:

Duration can be converted to time units using these conversion methods:

- `toNanos()`
- `toMillis()`
- `toMinutes()`
- `toHours()`
- `toDays()`

Instant Class (java.time.instant)

An instance of an `Instant` can be generated using one of the `Instant` class factory methods.

Access - Time of an Instant

Following fields contain the time denoted by an Instant object:

Both seconds and nanoseconds can be accessed via following methods respectively:

- `getEpochSecond()`
- `getNano()`

Instant Calculations

Various Date-Time calculations can be performed on the `Instant` class with plus or minus methods.

LocalDate Class (java.time.localdate)

- `LocalDate` class in Date-Time API denotes local date that is a date without time-zone information.
- `LocalDate` instances are immutable, thus, all calculations produce a new `LocalDate`.

Creating a LocalDate

`LocalDate` objects are created using several approaches.

Accessing the Date Information of a LocalDate

Date information of a `LocalDate` can be accessed using following methods:

- `getYear()`
- `getMonth()`
- `getDayOfMonth()`
- `getDayOfWeek()`
- `getDayOfYear()`

LocalDate Calculations

A set of date calculations can be achieved with the `LocalDate` class using one or more of following methods:

```
plusDays()
minusDays()
plusWeeks()
minusWeeks()
plusMonths()
minusMonths()
plusYears()
minusYears()
```

LocalDateTime Class (java.time.LocalDateTime) 1-2

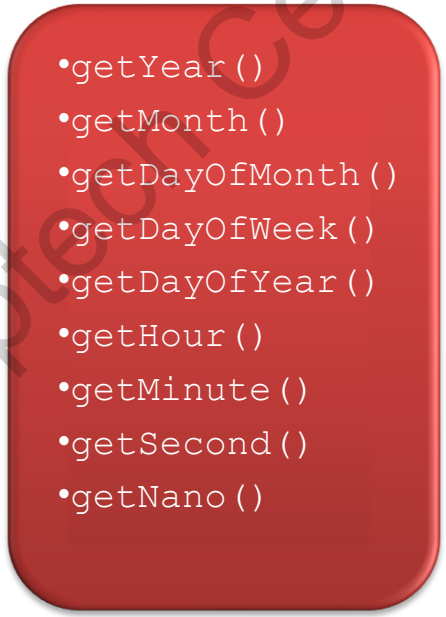
- `LocalDateTime` class in Date-Time API represents a local date and time without any time-zone data.

Creating a LocalDateTime

- `LocalDateTime` object can be created by using one of its static factory methods.

Access - Time of a LocalDateTime

Date-Time information of a `LocalDateTime` object can be accessed using `getValue()` method.



LocalDateTime Class (java.time.LocalDateTime) 2-2

Date-Time Calculations

- Various date and time calculations can be performed on `LocalDateTime` object with plus or minus methods.
  - `plusYears()`
  - `plusMonths()`
  - `plusDays()`
  - `plusHours()`
  - `plusMinutes()`
  - `plusSeconds()`
  - `plusNanos()`
  - `minusYears()`
  - `minusMonths()`
  - `minusDays()`
  - `minusHours()`
  - `minusMinutes()`
  - `minusSeconds()`
  - `minusNanos()`

LocalTime Class

- The `LocalTime` instance can be used to describe real world scenarios such as, the time when the school or work starts in various countries.
- It helps in analyzing interest of different zonal people in the UTC time, with concern to the time-zone of respective countries.

Creating a LocalTime Object

Generated using several approaches. The foremost approach is to create a `LocalTime` instance that denotes the exact time of now.

LocalTime Calculations

`LocalTime` class consists of a set of methods that can perform local time calculations.

MonthDay Class

`MonthDay` is an immutable Date-Time object that represents month as well as day-of-month

For example, a birthday or banking holiday can be derived from a month and day object

```
import java.time.*; // import the package for Date-Time API classes
public class DateDemo {
public static void main(String[] args) {
LocalDate today = LocalDate.now();
LocalDate dateOfBirth = LocalDate.of(1988, 02, 13);
// Code to retrieve the birthday month and day
MonthDay bday = MonthDay.of(dateOfBirth.getMonth(), dateOfBirth. getDayOfMonth());
// Code to retrieve the current month and day
MonthDay currentMonthDay = MonthDay.from(today);
if(currentMonthDay.equals(bday)){
System.out.println("**Colorful Joyful Birthday Buddy**");
}
else{
System.out.println("Nope, today is not your B'day");
}
}
}
```

OffsetDateTime Class

- `OffsetDateTime` is an immutable illustration of date and time with an offset.
- This class stores all date and time fields, to an accuracy of nanoseconds, as well as the offset from UTC/Greenwich.
- For example, the value '23rd November 2016 at 11:34.21.278965143 +05:00' can be stored in an `OffsetDateTime`.

```
LocalDateTime datetime = LocalDateTime.of(2016, Month.FEBRUARY, 15, 18, 20); // to display the result
// using Offset
ZoneOffset sampleoffset = ZoneOffset.of("-07:00");
OffsetDateTime date = OffsetDateTime.of(datetime, sampleoffset);
System.out.println("Sample display of Date and Time using time-zone offset : " + date);
```

OffsetTime and Period Class

- `OffsetTime` is an immutable Date-Time object that denotes a time, frequently observed as hour-minute-second-offset.
- This class stores all time fields, to an exactness of nanoseconds, along with a zone offset.
- `Period` class (`java.time.Period`) represents an amount of time in terms of days, months, and years.
- `Duration` and `Period` are somewhat similar; however, the difference between the two can be seen in their approach towards Daylight Savings Time (DST) when they are added to `ZonedDateTime`.

Year Class

- A `Year (java.time.Year)` object is an immutable `Date-Time` object that denotes a year.
- Every field that is a resultant from a year can be attained.

```
import java.time.Year;// Class to use Year values in calculations
public class SampleYear {
public static void main(String[] args) {
System.out.println(" The Present Year():"+Year.now());
System.out.println("The year 2022 is a Leap year :"+ Year. isLeap(2022)); // to display whether year
// 2022 is a leap year or not
System.out.println("The year 2024 is a Leap year :"+ Year.isLeap(2024));
// to display whether the year 2024 is a leap year or not
}
}
```

Output:

The Present Year (): 2021
The year 2022 is a Leap year: false
The year 2024 is a Leap year: true

YearMonth Class

- `YearMonth (java.time.YearMonth)` is a stable Date-Time object that denotes the combination of year and month.
- Does not store or denote a day, time, or time-zone.
- For example, the value 'November 2011' can be stored in a `YearMonth`.

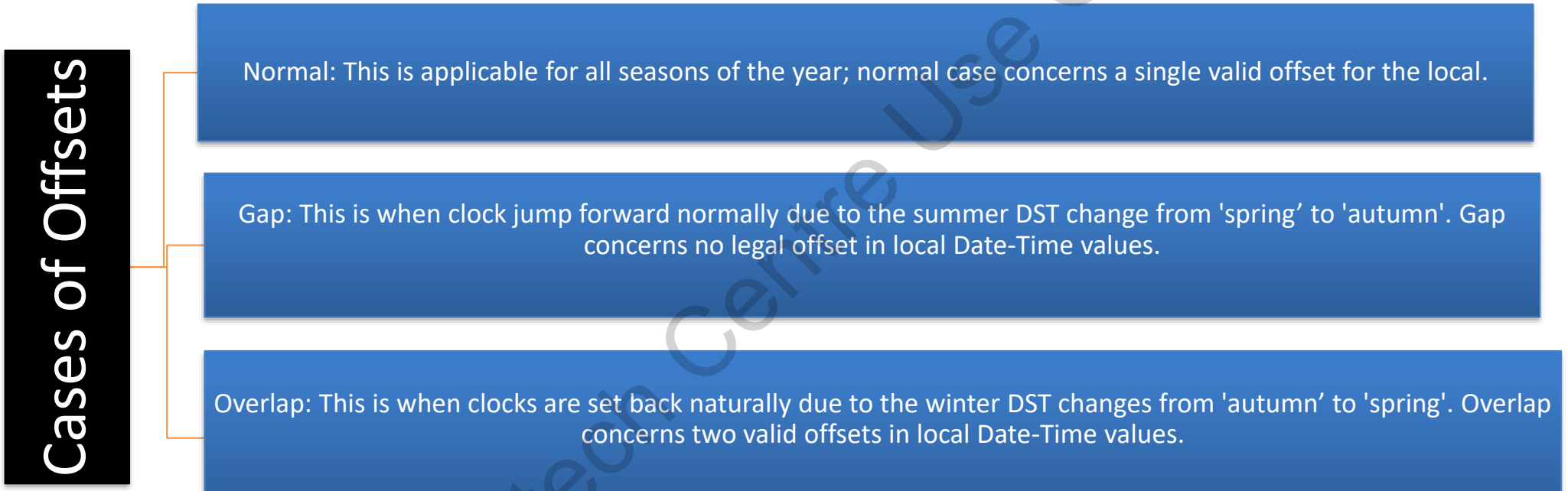
```
import java.time.YearMonth;// to use the Year and Month info
public class YearMonth {
public static void main(String[] args) {
System.out.println("The Present Year Month:"+YearMonth.now());
// To display present year and month
System.out.println("Month alone:"+YearMonth.parse("2021-02") .getMonthValue()); // To display only the
month value
System.out.println("Year alone:"+YearMonth.parse("2021-02").getYear());// to display the year value alone
System.out.println("This year is a Leap year:" +YearMonth.parse("2021-02").isLeapYear());// leap year check
}
}
```

Output:

```
The Present Year Month: 2021-02
Month alone: 2
Year alone: 2021
This year is a Leap year: false
```

ZonedDateTime

- The `ZonedDateTime` class is immutable. This means that all methods executing calculations on a `ZonedDateTime` object yields a new `ZonedDateTime` instance.



Creating a ZonedDateTime Object

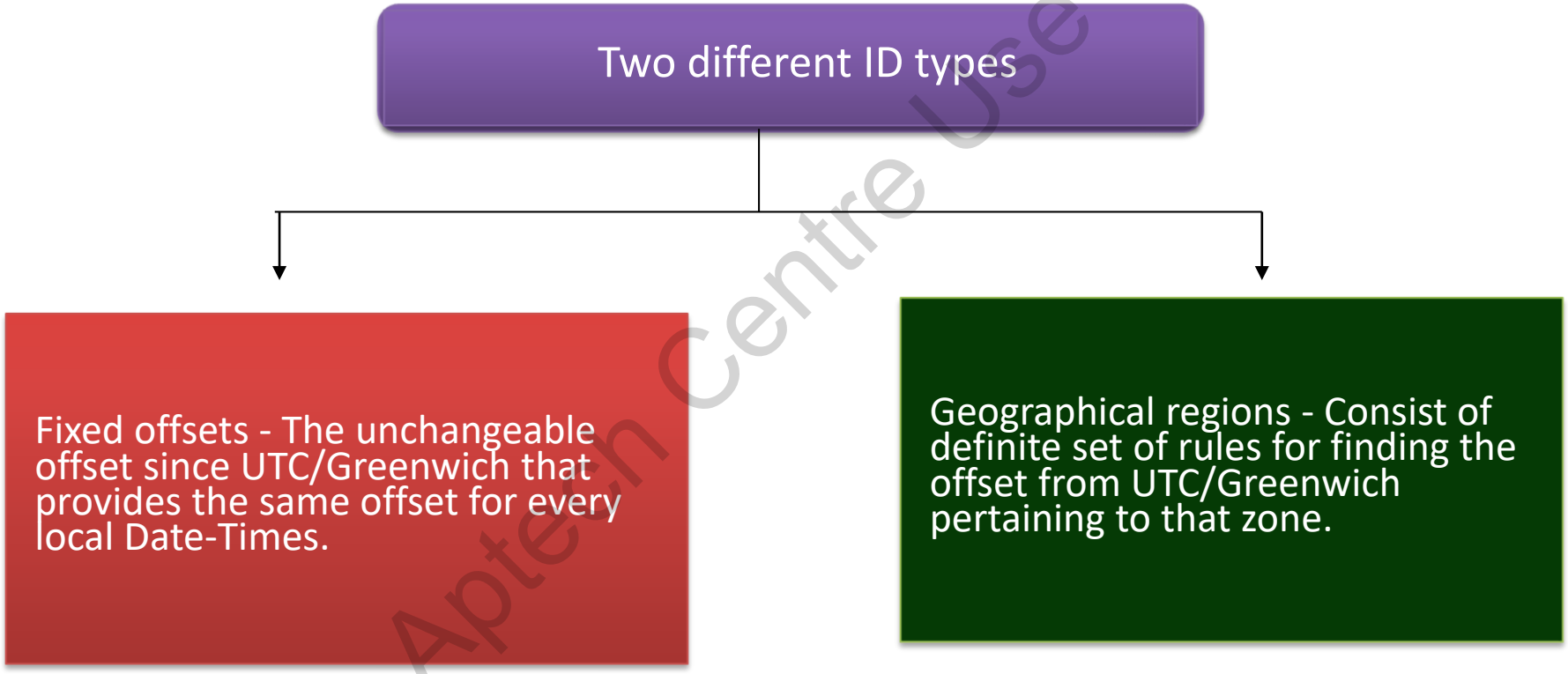
Accessing Date and Time of a ZonedDateTime

Date and Time Calculations

Time-zones

ZoneId Class

A `ZoneId` class is used to recognize rules used to convert between an `Instant` and a `LocalDateTime`.



ZoneOffset Class

A time-zone offset is the quantity of time that a time-zone differs from Greenwich/UTC. This is fixed in most cases.

Time-zone offsets differ from place to place across the planet.

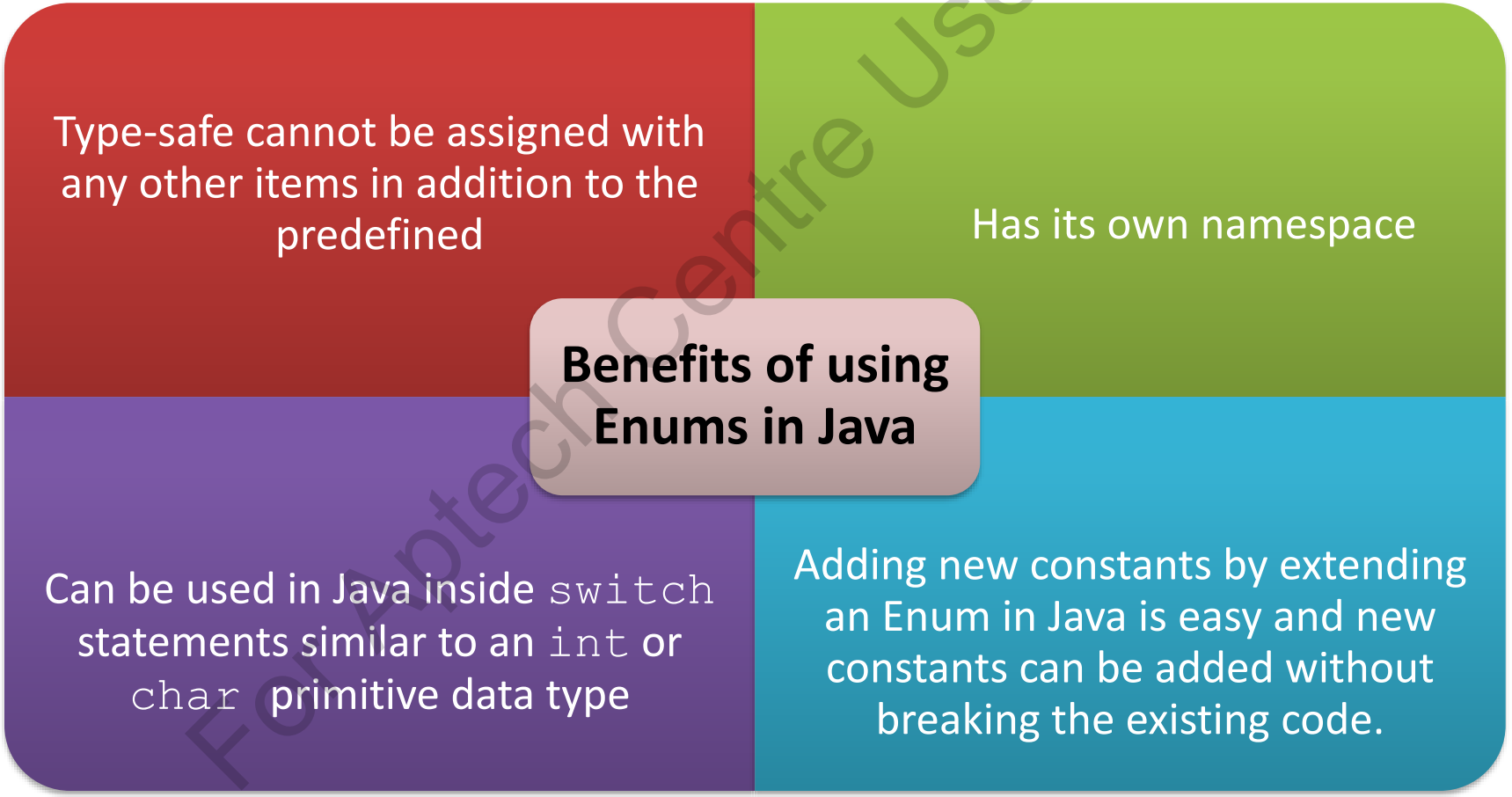
The rules for how offsets vary by place and time of year are specified in the `ZoneId` class.

For example

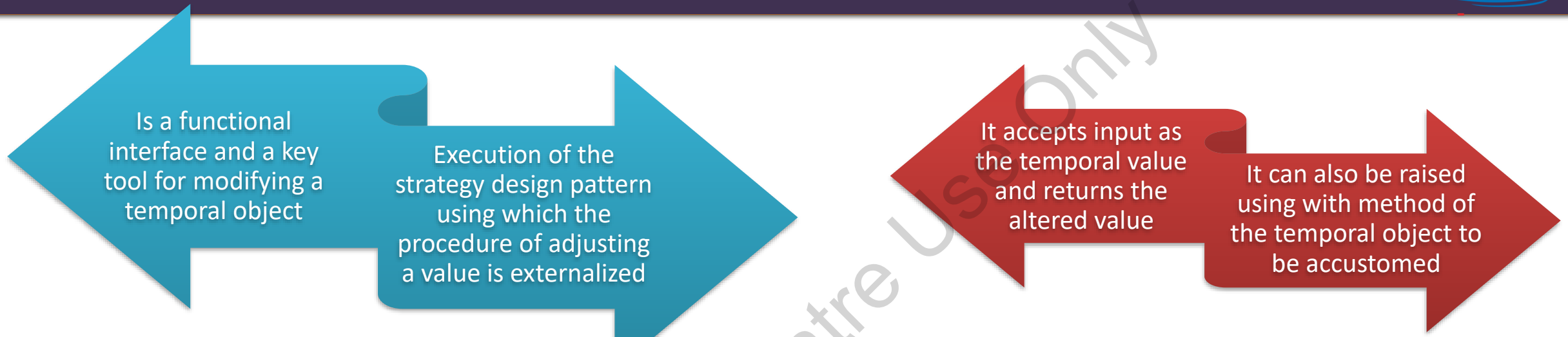
Berlin is two hours ahead of Greenwich/UTC in Spring and four hours ahead during Autumn.

Enums

An enumeration or enum is a type in Java that helps to denote the fixed number of well-known values in Java.



Temporal Adjusters 1-2



Following are some of the methods:

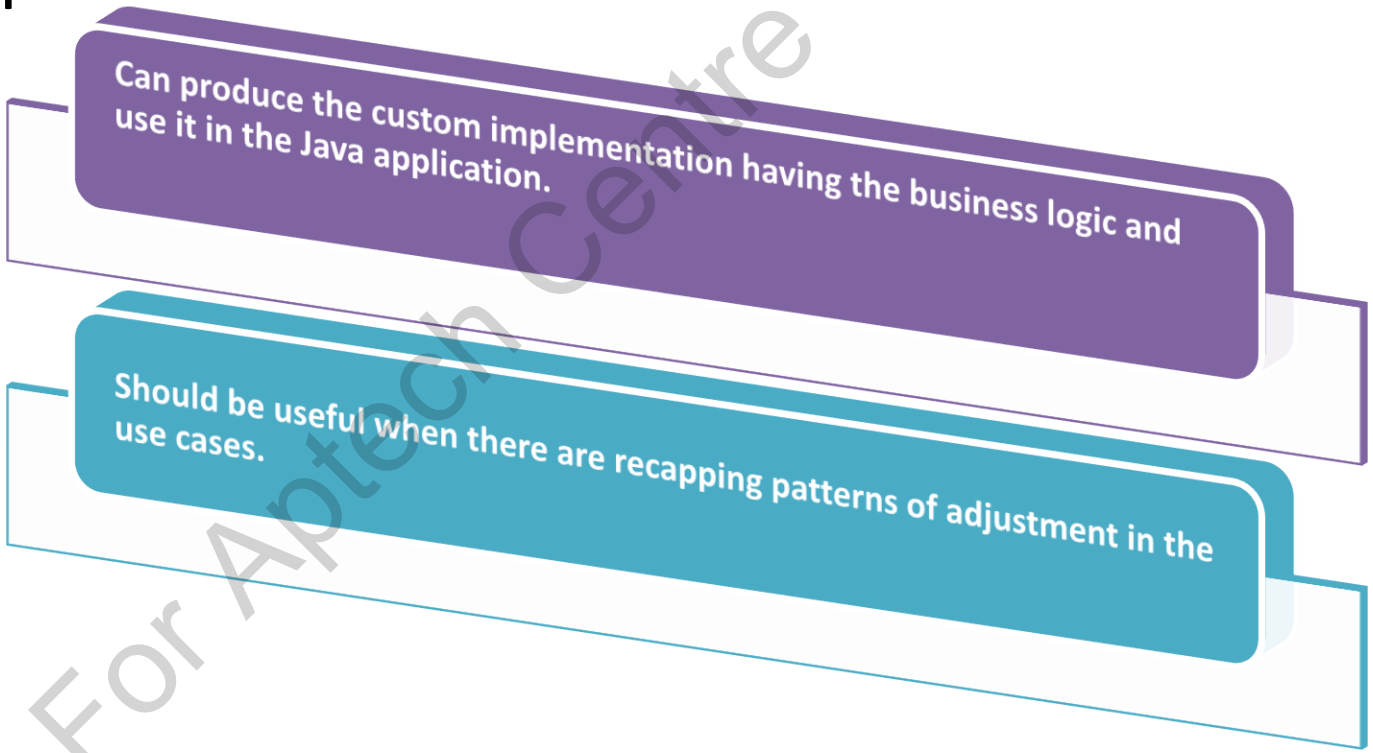
- `FirstDayOfMonth()`
- `FirstDayOfNextMonth()`
- `FirstInMonth(DayOfWeek)`
- `LastDayOfMonth()`
- `Next(DayOfWeek)`
- `NextOrSame(DayOfWeek)`
- `Previous(DayOfWeek)`
- `PreviousOrSame(DayOfWeek)`

Temporal Adjusters 2-2

TemporalAdjusters class

- `TemporalAdjusters` offers many `TemporalAdjuster` implementations. These can be used to adjust `Date-Time` objects.

Custom TemporalAdjuster



Backward Compatibility with Older Versions

- The original `Date` and `Calendar` objects contains the `toInstant()` method to convert them to new Java Date-Time API.
- It can then use an `ofInstant(Instant, ZoneId)` method to return a `LocalDateTime` or `ZonedDateTime` object.

```
import java.time.LocalDateTime;// to initiate local date and time
import java.time.ZonedDateTime; // to initiate zoned time
import java.util.Date;
import java.time.Instant;
import java.time.ZoneId;
public class BWCompatibility {
public static void main(String args[]){
BWCompatibility bwcompatibility = new BWCompatibility();
bwcompatibility.sampleBW();
}
public void sampleBW(){
// To display the current date
Date sampleCurDay = new Date();
System.out.println(" Desired Current date= " + sampleCurDay); // to display result
// To display the instant of current date
Instant samplenow = sampleCurDay.toInstant();
ZoneId samplecurZone = ZoneId.systemDefault();
// To display the current local date
LocalDateTime sampleLoDaTi = LocalDateTime.ofInstant(samplenow, samplecurZone);
System.out.println(" Desired Current Local date= " + sampleLoDaTi);
// To display the desired current zoned date
ZonedDateTime sampleZoDaTi = ZonedDateTime.ofInstant(samplenow, samplecurZone);
System.out.println(" Desired Current Zoned date= " + sampleZoDaTi);
// To display result
}
}
```

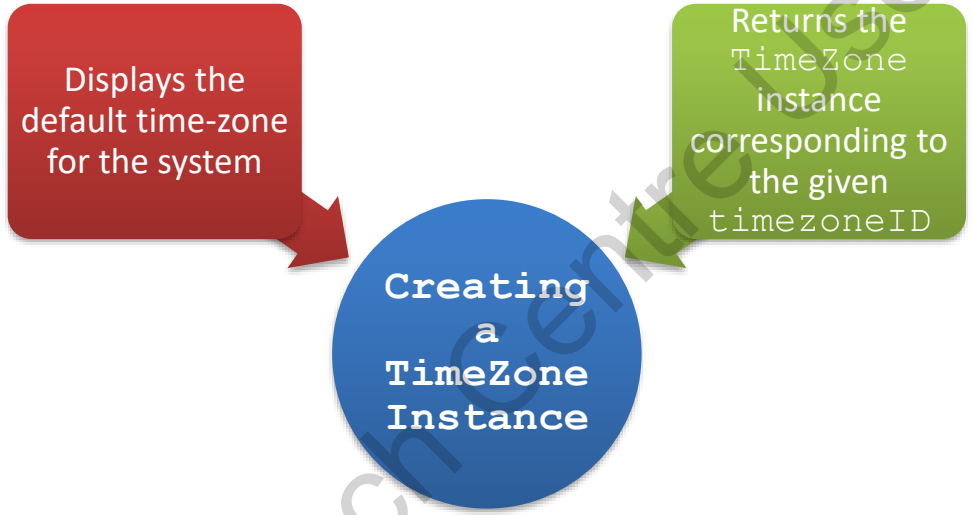
Parsing and Formatting Dates

- Parsing dates from strings and formatting dates to strings are possible with the `java.text.SimpleDateFormat` class.

```
SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
String dateString = format.format( new Date() );
Date samplDate = format.parse ("2011-03-25");
```

TimeZone (java.util.TimeZone)

- `TimeZone` class in Java represents the time-zones. This class is used in time-zone bound calculations.
- Retrieving a Time-Zone from a Calendar.



Output:

```
dateA: 2021-02-16T10:15:30+08:00[Asia/Singapore]
ZoneId: Asia/Singapore
CurrentZone: Etc/UTC
```

Stream of Dates

- Java 9 introduced a new method `LocalDate.datesUntil()` which returns an ordered sequential stream of dates.
- The returned stream begins from the specified date (inclusive) up to the end (exclusive) by an incremental step of one day.
- Easy to create dates streams with fixed offset.

```
package session10;
import java.time.LocalDate;
import java.time.Period;
import java.time.Month;
import java.util.stream.Stream;
public class DatesUntilMethodDemo {
public static void main(String args[]) {
// Print the days between today and 01 March 2021
Stream<LocalDate> dates = LocalDate.now().datesUntil(LocalDate. parse("2021-03-01"));
dates.forEach(System.out::println);
}
}
```

Summary

- The new Date-Time API introduced from Java 8 onwards is a solution for many unaddressed drawbacks of the previous API.
- Date-Time API contains many classes to reduce coding complexity and provides various additional features to work date and time.
- `Enum` in Java is a keyword, a feature that is used to denote the fixed number of well-known values in Java.
- `TemporalAdjuster` is a functional interface and a key tool for altering a temporal object.
- Java `TimeZone` class is a class that denotes time-zones and is helpful when doing calendar arithmetic across time-zones.
- A time-zone offset is the quantity of time that a time-zone differs from Greenwich/UTC. This is fixed in most cases.
- `TemporalAdjuster` is a functional interface and a key tool for modifying a temporal object.