Correction Java 8 DateTime API - 0

D - instant .atZone(ZoneId.systemDefault()).toLocalDateTime();

Instant class doesn't have any method to convert to LocalDate, LocalDateTime or LocalTime and vice-versa. Hence, 'instant.toLocalDateTime();' and 'LocalDateTime.of(instant);' cause compilation error.

'(LocalDateTime)instant' also causes compilation failure as LocalDateTime and Instant are not related in multilevel inheritance.

Hence, the only option left is:
'instant.atZone(ZoneId.systemDefault()).toLocalDateTime();'.

Let us understand what is happening with above statement:

ZonedDateTime class has methods to convert to LocalDate, LocalDateTime and LocalTime instances. So, object of Instant is first converted to ZonedDateTime.

An Instant object doesn't store any information about the time zone, so to convert it to ZonedDateTime, the default zone (ZoneId.systemDefault()) is passed to atZone method.

'instant.atZone(ZoneId.systemDefault())' returns an instance of ZonedDateTime and toLocalDateTime() method returns the corresponding instance of LocalDateTime.

Correction Java 8 DateTime API - 1

A - Runtime Exception

Signature of between method defined in Duration class is: ‘Duration between(Temporal startInclusive, Temporal endExclusive)’.

As both LocalTime and LocalDateTime implement ‘Temporal’ interface, hence there is no compilation error.

If the Temporal objects are of different types as in this case, calculation is based on 1st argument and 2nd argument is converted to the type of 1st argument.

1st argument, ‘t2’ is of LocalDateTime and 2nd argument, ‘t1’ is of LocalTime. At runtime it is not possible to convert LocalTime to LocalDateTime and hence exception is thrown at runtime.

Correction Java 8 DateTime API - 2

A - No

DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL); statement returns formatter to format date part.

date is of LocalDateTime type hence it has both date part and time part.

‘formatter.format(date)’ simply formats the date part and ignores time part.

NOTE: You should aware of other formatter related methods for the OCP exam, such as: ‘ofLocalizedTime’ and ‘ofLocalizedDateTime’

Correction Java 8 DateTime API - 3

B - 1970-01-01

0th day in epoch is: 1970-01-01, 1st day in epoch is: 1970-01-02 and so on.

as toString() method of LocalDate class prints the LocalDate object in ISO-8601 format: “uuuu-MM-dd”. Hence output is: ‘1970-01-01’.

Correction Java 8 DateTime API - 4

C - true

Definition of plus method is: ‘public LocalDate plus(long amountToAdd, TemporalUnit unit) {…}’.

TemporalUnit is mentioned in exam objectives and it is being used as method parameters in date time API. enum ChronoUnit implements this interface and it is used at various places in the API.

One such use as an argument in plus and minus methods of LocalDate, LocalDateTime and LocalTime classes.

t1.plus(22, ChronoUnit.HOURS) is same as t1.plusHours(22).

If you check the code of plus(long, TemporalUnit), you will find that it calls plusSeconds, plusMinutes, plusHours etc based on the passed enum value. ‘ChronoUnit.HOURS’ is passed in this case, hence t1.plus(22, ChronoUnit.HOURS) invokes t1.plusHours(22).

Correction Java 8 DateTime API - 5

B - 14:0:0

You should be aware of Day light saving mechanism to answer this question.

Suppose daylight time starts at 2 AM on particular date.

Current time: 1:59:59 [Normal time].

Next second: 3:00:00 [Time is not 2:00:00 rather it is 3:00:00. It is Daylight saving time].

Clock just jumped from 1:59:59 to 3:00:00.

Now Suppose daylight time ends at 2 AM on particular date.

Current time: 1:59:59 [Daylight saving time].

Next second: 1:00:00 [Time is not 2:00:00 rather it is 1:00:00. Clock switched back to normal time].

Clock just went back from 1:59:59 to 1:00:00.

Now let's solve given code:

dt –> {2018-11-04T13:59:59}. Daylight saving time has already ended as it is PM and not AM. This is acturally a normal time now.

dt.plusSeconds(1) => creates a new ZonedDateTime object {2018-11-04T14:00:00} and dt refers to it.

dt.getHour() = 14, dt.getMinute() = 0 and dt.getSecond() = 0.

Correction Java 8 DateTime API - 6

A - Runtime Exception

Signature of between method defined in Duration class is: ‘Duration between(Temporal startInclusive, Temporal endExclusive)’.

As both LocalDate and LocalTime implement ‘Temporal’ interface, hence there is no compilation error.

Time part must be available to calculate duration but as LocalDate object referred by d1 doesn’t have time part, hence an exception is thrown at runtime.

Correction Java 8 DateTime API - 7

A - 1:0:0

You should be aware of Day light saving mechanism to answer this question.

Suppose daylight time starts at 2 AM on particular date.

Current time: 1:59:59 [Normal time].

Next second: 3:00:00 [Time is not 2:00:00 rather it is 3:00:00. It is Daylight saving time].

Clock just jumped from 1:59:59 to 3:00:00.

Now Suppose daylight time ends at 2 AM on particular date.

Current time: 1:59:59 [Daylight saving time].

Next second: 1:00:00 [Time is not 2:00:00 rather it is 1:00:00. Clock switched back to normal time].

Clock just went back from 1:59:59 to 1:00:00.

Now let's solve given code:

dt –> {2018-11-04T01:59:59}. Daylight saving time will end at next second.

dt.plusSeconds(1) => creates a new ZonedDateTime object {2018-11-04T01:00:00} and dt refers to it.

dt.getHour() = 1, dt.getMinute() = 0 and dt.getSecond() = 0.

Correction Java 8 DateTime API - 8

C - 1970-01-02

0th day in epoch is: 1970-01-01, 1st day in epoch is: 1970-01-02 and so on.

As toString() method of LocalDate class prints the LocalDate object in ISO-8601 format: “uuuu-MM-dd”. Hence output is: ‘1970-01-02’.

Correction Java 8 DateTime API - 9

A - false:false

Both the methods public “boolean isEqual(ChronoLocalDate)” and “public boolean equals(Object)” return true if date objects are equal otherwise false.

NOTE: LocalDate implements ChronoLocalDate.

date1 –> {2019-01-02}

date1.minus(Period.ofDays(1)); As LocalDate is immutable, this statement creates another instance of LocalDate class. and date1 still refers to previous object.

Similarly date2 refers to {2018-12-31}

and ‘date2.plus(Period.ofDays(1));’ creates another instance of LocalDate class.

As date1 and date2 are not logically same, equals and isEqual methods return false.

Correction Java 8 DateTime API - 10

D - Period. between(LocalDate.now(), LocalDate.parse(“2022-11-21”))

Signature of Period.between method is: Period between(LocalDate startDateInclusive, LocalDate endDateExclusive) {…}

Both the parameters are of ‘LocalDate’ type.

Correction Java 8 DateTime API - 11

C - Instant.ofEpochMilli(date.getTime()).atZone(ZoneId.systemDefault()).toLocalDate();

D - date. toInstant().atZone(ZoneId.systemDefault()).toLocalDate();

java.util.Date class doesn't have any method to convert to LocalDate, LocalDateTime or LocalTime and vice-versa. Hence, 'date.toLocalDate();' and 'LocalDate.of(date);' cause compilation error.

'(LocalDate)date' also causes compilation failure as LocalDate and Date are not related in multilevel inheritance.

Hence, two correct options left are: 'date.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();' and 'Instant.ofEpochMilli(date.getTime()).atZone(ZoneId.systemDefault()).toLocalDate();'.

Let us understand what is happening with above statements:

java.util.Date class has toInstant() method which converts java.util.Date object to java.time.Instant object. If you check the code of toInstant() method, you will find below one statement inside it:

'return Instant.ofEpochMilli(getTime());'

This means 'date.toInstant()' is same as 'Instant.ofEpochMilli(date.getTime())'. Both statements return the object of Instant class.

ZonedDateTime class has methods to convert to LocalDate, LocalDateTime and LocalTime instances. So, object of Instant is first converted to ZonedDateTime.

An Instant object doesn't store any information about the time zone, so to convert it to ZonedDateTime, the default zone (ZoneId.systemDefault()) is passed to atZone method.

'instant.atZone(ZoneId.systemDefault())' returns an instance of ZonedDateTime and toLocalDate() method returns the corresponding instance of LocalDate.

Correction Java 8 DateTime API - 12

A - PT-48H

As Period is expressed in terms of Years, Months and Days, Duration is expressed in terms of Hours, Minutes and Seconds.

String representation of Period starts with “P” whereas String representation of Duration starts with “PT”.

You should be aware of following ‘of’ methods for the exam:

Duration.ofDays(long days) => Returns a Duration instance with specified number of days converted to hours. -2 days equals to -48 hours.

Duration.ofHours(long hours) => Returns a Duration instance with specified number of hours.

Duration.ofMinutes(long minutes) => Returns a Duration instance with specified number of minutes.

Duration.ofSeconds(long seconds) => Returns a Duration instance with specified number of seconds, nanos field is set to 0. NOTE: if nanos field is 0, toString ignores it.

Duration.ofMillis(long millis) => Returns a Duration instance with passed value converted to seconds and nano seconds.

Duration.ofNanos(long nanos) => Returns a Duration instacne with passed value converted to seconds and nono seconds.

Correction Java 8 DateTime API - 13

C - Period, Duration

Correct statement is:

Period represents date-based amount of time whereas Duration represents time-based amount of time.

Period is date-based, such as ‘10 years, 5 months and 3 days’.

Duration is time-based, such as ‘5 hours, 10 minutes and 44 seconds’.

Instant is to just represent an instantaneous point on the time-line.

Check the Javadoc of above classes.

Correction Java 8 DateTime API - 14

A - 1970-01-01T00:00:00Z

DateTimeFormatter.ISO_INSTANT is used by the toString() method of Instant class and it formats or parses an instant in UTC, such as ‘2018-10-01T15:02:01.231Z’.

EPOCH instant is: 1970-01-01T00:00:00Z and it is printed to the console.

NOTE: nano portion is 0 in EPOCH instant, hence it is ignored by toString() method.

Correction Java 8 DateTime API - 15

C - true

DateTimeFormatter.ofPattern(String) accepts String argument. Format string is also correct.

d represents day-of-month, M represents month-of-year and u represents year.

New Date/Time API has format method in DateTimeFormatter as well as LocalDate, LocalTime, LocalDateTime, ZonedDateTime classes.

formatter.format(date) returns “04-11-2018” and date.format(formatter) returns “04-11-2018” and hence ‘true’ is printed on to the console.

Correction Java 8 DateTime API - 16

B - 1969-12-31

0th day in epoch is: 1970-01-01, 1st day in epoch is: 1970-01-02 and so on.

And if you pass a negative number, then it represents earlier days.

LocalDate.ofEpochDay(-1); –> {1969-12-31}.

Correction Java 8 DateTime API - 17

A - ZIndia.withZoneSameLocal(us)

zIndia –> {2019-01-01T00:00+05:30[Asia/Kolkata]}.

zIndia.withZoneSameLocal(us) => {2019-01-01T00:00-08:00[America/Los_Angeles]}. It just changes the Zone but keeps the date and time same and this is what zUS supposed to refer to.

Duration.between(zIndia, zUS) returns 'PT13H30M' in this case. So, people of Los Angeles have to wait for 13 hours 30 minutes to celebrate the new year.

zIndia.withZoneSameInstant(us) => {2018-12-31T10:30-08:00[America/Los_Angeles]}. It just converts India time to Los Angeles time.

As both zIndia and zUS will refer to same instance of time, hence 'Duration.between(zIndia, zUS)' would return 'PT0S'. And this is not expected.

NOTE: If you intend to use 'zIndia.withZoneSameInstant(us);', then Line 15 should be changed as below:

System.out.println(Duration.between(zUS.toLocalDateTime(), zIndia.toLocalDateTime())); // would print 'PT13H30M'.

To get positive duration value, first parameter should be earlier date/time.

Correction Java 8 DateTime API - 18

D - Runtime Exception

LocalDate ofYearDay(int year, int dayOfYear): Valid values for dayOfYear for non-leap year is 1 to 365 and for leap year is 1 to 366.

For other values, java.time.DateTimeException is thrown.

Correction Java 8 DateTime API - 19

A - Program terminates successfully after displaying the output

Signature of between method defined in Duration class is: ‘Duration between(Temporal startInclusive, Temporal endExclusive)’.

As both LocalTime and LocalDateTime implement ‘Temporal’ interface, hence there is no compilation error.

If the Temporal objects are of different types as in this case, calculation is based on 1st argument and 2nd argument is converted to the type of 1st argument. It is easy to convert LocalDateTime to LocalTime.

Program executes successfully and terminates successfully after displaying the Duration object on to the console.

Correction Java 8 DateTime API - 20

B - 14:14:59.111100

LocalTime.parse(text); => text must represent a valid time and it is parsed using DateTimeFormatter.ISO_LOCAL_TIME.

ISO_LOCAL_TIME represents time in following format:

HH:mm (if second-of-minute is not available),

HH:mm:ss (if second-of-minute is available),

HH:mm:ss.SSS (if nano-of-second is 3 digit or less),

HH:mm:ss.SSSSSS (if nano-of-second is 4 to 6 digits),

HH:mm:ss.SSSSSSSSS (if nano-of-second is 7 to 9 digits).

Valid values for hour-of-day (HH) is: 0 to 23.

Valid values for minute-of-hour (mm) is: 0 to 59.

Valid values for second-of-minute (ss) is: 0 to 59.

Valid values for nano-of-second is: 0 to 999999999.

In the given expression, 'LocalTime.parse("14:14:59.1111");' all the values are within range and as nano-of-second is of 4 digit, hence toString() method appends 2 zeros to it.

Output is: '14:14:59.111100'.

Correction Java 8 DateTime API - 21

D - P700D

For Period.ofWeeks(int), the resulting period will be day-based, with the amount of days equal to the number of weeks multiplied by 7.

Period is represented in terms of Year, Month and Day only and toString() method uses upper case characters.

NOTE: Other ‘of’ methods of Period class are:

Period.of(int years, int months, int days) => Returns a Period instance with specified number of years, months and days.

Period.ofDays(int days) => Returns a Period instance with specified number of days.

Period.ofMonths(int months) => Returns a Period instance with specified number of months.

Period.ofYears(int years) => Returns a Period instance with specified number of years.

Correction Java 8 DateTime API - 22

B - 10-Sep-2018

M -> Represents actual digit for the month (1 to 12).

MM -> Represents 2 digits for the month (01 to 12).

MMM -> Represents short name for the month, with first character in upper case, such as Jun, Sep

MMMM -> Represents full name for the month, with first character in upper case, such as June, September

DateTimeFormatter.ofPattern(String) method uses the default Locale and in this case default Locale is set to en_US, hence English month names will be printed in this case.

“2018-09-10” will be formatted to “10-Sep-2018”.

Correction Java 8 DateTime API - 23

A - Compilation error

dt.toLocalDate() returns an instance of LocalDate and dt.toLocalTime() returns an instance of LocalTime.

But '+' operator is not overloaded for LocalDate and LocalTime objects OR 2 LocalDate objects OR 2 LocalTime objects OR in general for 2 Java Objects.

It would work if one of the operand is of String type.

Hence 'dt.toLocalDate() + " " + dt.toLocalTime()' doesn't cause any compilation error as '+' operator is Left to Right associative.

dt.toLocalDate() + " " + dt.toLocalTime()

= (dt.toLocalDate() + " ") + dt.toLocalTime()

= "2018-03-16" + dt.toLocalTime()

= "2018-03-16 10:15:30.220"

Correction Java 8 DateTime API - 24

B - Yes

To know whether 2 instances represent same time or not, convert the time to GMT time by subtracting the offset.

2018-02-01T10:30 -(+05:30) = 2018-02-01T05:00 GMT.

and

2018-01-31T21:00 -(-08:00) = 2018-02-01T05:00 GMT.

So both represents same instance of time.

Correction Java 8 DateTime API - 25

C - P-10D

Signature of Period.between method is: Period between(LocalDate startDateInclusive, LocalDate endDateExclusive) {...}

Difference between 1st March 2018 and 11th March 2018 is 10 days and the result of this method is negative period as 1st argument is later date.

Correction Java 8 DateTime API - 26

C - 2018-02-01

LocalDate ofYearDay(int year, int dayOfYear) returns an instance of LocalDate from a year and day-of-year.

January has 31 days, so 32nd day of the year means 1st Feb of the given year.

Output is: ‘2018-02-01’ as toString() method of LocalDate class prints the LocalDate object in ISO-8601 format: “uuuu-MM-dd”.

Correction Java 8 DateTime API - 27

B - 32nd day of 2018

In the parse string, anything between opening and closing quote is displayed as it is.

DD -> 2 digit representation for the day-of-year.

‘nd day of’ -> nd day of is displayed.

uuuu -> 4 digit representation for the year.

Output is: 32nd day of 2018.

NOTE: To display single quote, escape it with one more single quote. For example,

‘DateTimeFormatter.ofPattern(“‘It’‘s’ DD’nd day of’ uuuu”);’ this will help to print: [It’s 32nd day of 2018] whereas ‘DateTimeFormatter.ofPattern(“‘It’s’ DD’nd day of’ uuuu”);’ causes runtime exception.

Correction Java 8 DateTime API - 28

D - 2018-03-16 10:15:30.220

LocalDateTime.parse(text); => text is parsed using DateTimeFormatter.ISO_LOCAL_DATE_TIME.

ISO_LOCAL_DATE_TIME is a combination of ISO_LOCAL_DATE and ISO_LOCAL_TIME, and it parses the date-time in following format:

[ISO_LOCAL_DATE][T][ISO_LOCAL_TIME]: T is case-insensitive.

ISO_LOCAL_DATE represents date in following format:

uuuu-MM-dd

Valid values for year (uuuu) is: 0000 to 9999.

Valid values for month-of-year (MM) is: 01 to 12.

Valid values for day-of-month (dd) is: 01 to 31 (At runtime this value is validated against month/year).

ISO_LOCAL_TIME represents time in following format:

HH:mm (if second-of-minute is not available),

HH:mm:ss (if second-of-minute is available),

HH:mm:ss.SSS (if nano-of-second is 3 digit or less),

HH:mm:ss.SSSSSS (if nano-of-second is 4 to 6 digits),

HH:mm:ss.SSSSSSSSS (if nano-of-second is 7 to 9 digits).

Valid values for hour-of-day (HH) is: 00 to 23.

Valid values for minute-of-hour (mm) is: 00 to 59.

Valid values for second-of-minute (ss) is: 00 to 59.

Valid values for nano-of-second is: 0 to 999999999.

dt.toLocalDate() returns an instance of LocalDate, whose toString() method prints date part: ‘2018-03-16’.

dt.toLocalTime() returns an instance of LocalTime, whose toString() method prints time part: ‘10:15:30.220’. As nano-of-second is of 2 digit, hence single zero is appended to its value.

Correction Java 8 DateTime API - 29

A - Runtime exception

Duration works with time component and not Date component.

plus(TemporalAmount) method of LocalDate class accepts TemporalAmount type and Duration implement TemporalAmount, so no compilation error. But at runtime date1.plus(d) causes exception.

Don’t get confused with ‘Duration.ofDays(1);’, it is a convenient method to work with large time.

Correction Java 8 DateTime API - 30

D - Coupon expiry date: 2018-03-11

In LocalDate.of(int, int, int) method, 1st parameter is year, 2nd is month and 3rd is day of the month.

ocpCouponPurchaseDate –> {2018-03-01} and ocpCouponPurchaseDate.plusDays(10) –> {2018-03-11}.

toString() method of LocalDate class prints the LocalDate object in ISO-8601 format: “uuuu-MM-dd”.

Correction Java 8 DateTime API - 31

C - ZonedDateTime

LocalDate, LocalTime and LocalDateTime don’t have concepts of time zones.

ZonedDateTime class is used for time zones.

Correction Java 8 DateTime API - 32

D - 2018-03-16T10:15:30.220

LocalDateTime.parse(text); => text is parsed using DateTimeFormatter.ISO_LOCAL_DATE_TIME.

ISO_LOCAL_DATE_TIME is a combination of ISO_LOCAL_DATE and ISO_LOCAL_TIME, and it parses the date-time in following format:

[ISO_LOCAL_DATE][T][ISO_LOCAL_TIME]: T is case-insensitive.

ISO_LOCAL_DATE represents date in following format:

uuuu-MM-dd

Valid values for year (uuuu) is: 0000 to 9999.

Valid values for month-of-year (MM) is: 01 to 12.

Valid values for day-of-month (dd) is: 01 to 31 (At runtime this value is validated against month/year).

ISO_LOCAL_TIME represents time in following format:

HH:mm (if second-of-minute is not available),

HH:mm:ss (if second-of-minute is available),

HH:mm:ss.SSS (if nano-of-second is 3 digit or less),

HH:mm:ss.SSSSSS (if nano-of-second is 4 to 6 digits),

HH:mm:ss.SSSSSSSSS (if nano-of-second is 7 to 9 digits).

Valid values for hour-of-day (HH) is: 00 to 23.

Valid values for minute-of-hour (mm) is: 00 to 59.

Valid values for second-of-minute (ss) is: 00 to 59.

Valid values for nano-of-second is: 0 to 999999999.

toString() method of LocalDateTime class has following definition: return date.toString() + ‘T’ + time.toString();

NOTE: ‘T’ is in upper case and as nano-of-second is of 2 digit, hence single zero is appended to its value.

Correction Java 8 DateTime API - 33

A - Runtime Exception

D represents day-of-year and DD is for printing 2 digits. If you calculate, day of the year for 4th Nov 2018, it will be of 3 digit value as year has 365 days and date is in November month.

Pattern DD (for 2 digits) will cause Runtime Exception in this case.

If you get confused with pattern letters in lower case and upper case, then easy way to remember is that Bigger(Upper case) letters represent something bigger.

M represents month & m represents minute, D represents day of the year & d represents day of the month.

Correction Java 8 DateTime API - 34

B - 2019-03-06

Period.parse(CharSequence) method accepts the String parameter in “PnYnMnD” format, over here P,Y,M and D can be in any case.

“P9M” represents period of 9 months.

Signature of plus method of LocalDate is: ‘LocalDate plus(TemporalAmount)’, Period implements ChronoPeriod and ChnonoPeriod extends TemporalAmount.

Hence Period type can be passed as an argument of plus method. Adding 9 months to 6th June 2018 returns 6th March 2019.

Correction Java 8 DateTime API - 35

D - 45-02-2018

New Date/Time API has format method in DateTimeFormatter as well as LocalDate, LocalTime, LocalDateTime, ZonedDateTime classes, so valDay.format(formatter) doesn't cause compilation error.

D represents day-of-year and DD is for printing 2 digits. If you calculate, day of the year for 14th Feb 2018, then it is 45th day of the year (as January has 31 days).

45 is a 2-digit number and hence can be easily parsed by 'DD'.

Output in this case is: 45-02-2018.