Coding Cheat Sheet

This reading provides a reference list of code that you'll encounter as you work with object-oriented coding in Java. Understanding these concepts will help you write and debug your first Java programs. Let's explore the following Java coding concepts:

- Using Java Date and Time Classes
- Formatting Dates in Java
- Using Timezones in Java
- Parsing Dates from Strings in Java

Keep this summary reading available as a reference as you progress through your course, and refer to this reading as you begin coding with Java after this course!

Using Java Date and Time Classes

Using the LocalDate class

Description	Example
Import the `LocalDate` class, which is part of the Java Date and Time API.	`import java.time.LocalDate;`

Define a public class `LocalDateExample` that contains the Java `main` method. Use `LocalDate.now()` to retrieve the current date and print it in the "YYYY-MM-DD" format, which is the default format of `LocalDate.toString()`.	`public class LocalDateExample {`
	`    public static void main(String[] args) {`
	`        LocalDate today = LocalDate.now();`
	`        System.out.println("Today's date: " + today);`

Close curly braces to end the `LocalDateExample` class definition.	`    }`
	`}`

Explanation: This Java program demonstrates the use of the `LocalDate` class from the `java.time` package to get and display the current date.

Using the LocalTime class

Description	Example
Import the `LocalTime` class, which is part of the Java Date and Time API.	`import java.time.LocalTime;`

Define a public class `LocalTimeExample` that contains the Java `main` method. Use `LocalTime.now()` to retrieve the current system time and print it in the "HH:mm:ss.SSS" (hours, minutes, seconds, and milliseconds/nanoseconds) format, which is the default format of `LocalTime.toString()`.	`public class LocalTimeExample {`
	`    public static void main(String[] args) {`
	`        LocalTime currentTime = LocalTime.now();`
	`        System.out.println("Current time: " + currentTime);`

Description	Example

Close curly braces to end the `LocalTimeExample` class definition.	
	```
	    }
	}
	```

Explanation: This Java program demonstrates the use of the `LocalTime` class from the `java.time` package to get and display the current time.

Using the LocalDateTime class

Description	Example
Import the `LocalDateTime` class, which is part of the Java Date and Time API.	`import java.time.LocalDateTime;`

Define a public class `LocalDateTimeExample` that contains the Java `main` method. Use `LocalDateTime.now()` to retrieve the current system date and time. Print the current date and time in the default format "YYYY-MM-DDTHH:MM:SS.SSS" (year, month, day, hours, minutes, seconds, and milliseconds/nanoseconds), which is the default format of `LocalDateTime.toString()`.	
	```
	public class LocalDateTimeExample {
	    public static void main(String[] args) {
	        LocalDateTime now = LocalDateTime.now();
	        System.out.println("Current date and time: " + now);
	```

Close curly braces to end the `LocalDateTimeExample` class definition.	
	```
	    }
	}
	```

Explanation: This Java program demonstrates the use of the `LocalDateTime` class from the `java.time` package to get and display the current date and time. `LocalDateTime` is an immutable class that represents both date and time without a time zone.

Using the ZonedDateTime class

Description	Example
Import the `ZonedDateTime` class, which is part of the Java Date and Time API.	`import java.time.ZonedDateTime;`

Define a public class `ZonedDateTimeExample` that contains the Java `main` method. Use `ZonedDateTime.now()` to retrieve the current system date and time, including the time zone. Print the current date, time, and zone in the default ISO-8601 format.	
	```
	public class ZonedDateTimeExample {
	    public static void main(String[] args) {
	        ZonedDateTime zonedNow = ZonedDateTime.now();
	        System.out.println("Current date and time with zone: " + zonedNow);
	```

Description	Example

Close curly braces to end the `ZonedDateTimeExample` class definition.	
	```
	        }
	    }
	```

Explanation: This Java program demonstrates how to use the `ZonedDateTime` class from the `java.time` package to retrieve and display the current date and time along with the time zone. It is useful when working with time zones in applications such as scheduling, logging, and internationalization.

Real World example of an Event Management System

Description	Example

Import the `LocalDate`, `LocalTime`, `LocalDateTime`, `ZoneId`, `ZonedDateTime`, and `Scanner` classes that are part of the Java Date and Time API.	
	```
	import java.time.LocalDate;
	import java.time.LocalDateTime;
	import java.time.LocalTime;
	import java.time.ZoneId;
	import java.time.ZonedDateTime;
	import java.util.Scanner;
	```

Define an `EventManagement` class to represent an event with `name`, `date`, `time`, and `timeZone`. The method `getEventDateTime()` converts `LocalDate` and `LocalTime` into `LocalDateTime`. Then converts `LocalDateTime` into `ZonedDateTime` using the specified time zone.	
	```
	public class EventManagement {

	    static class Event {
	        String name;
	        LocalDate date;
	        LocalTime time;
	        ZoneId timeZone;

	        public Event(String name, LocalDate date, LocalTime time, ZoneId timeZone) {
	            this.name = name;
	            this.date = date;
	            this.time = time;
	            this.timeZone = timeZone;
	        }
	        public ZonedDateTime getEventDateTime() {
	            LocalDateTime localDateTime = LocalDateTime.of(date, time);
	            return ZonedDateTime.of(localDateTime, timeZone);
	        }
	    }
	```

Define a public class with the Java `main` method and use it to accept user input for event details. This class captures `name`, `date`, `time`, and `timeZone` from user input. The method `Event(name, date, time, timeZone)` creates an event object via user input. The method `getEventDateTime()` displays the event date an time in the specified time zone. The method `ZonedDateTime` converts `eventDateTime` to the system's local time zone. The method `scanner.close()` closes the scanner to free up resources.	
	```
	    public static void main(String[] args) {
	        Scanner scanner = new Scanner(System.in);

	        // Input event details
	        System.out.println("Enter event name:");
	        String name = scanner.nextLine();

	        System.out.println("Enter event date (YYYY-MM-DD):");
	        String dateInput = scanner.nextLine();
	        LocalDate date = LocalDate.parse(dateInput);

	        System.out.println("Enter event time (HH:MM):");
	        String timeInput = scanner.nextLine();
	        LocalTime time = LocalTime.parse(timeInput);

	        System.out.println("Enter time zone (e.g., America/New_York):");
	        String zoneInput = scanner.nextLine();
	        ZoneId timeZone = ZoneId.of(zoneInput);

	        // Create the event
	```

Description	Example

	`Event event = new Event(name, date, time, timeZone);`
	
	`// Display event details`
	`System.out.println("Event created: " + event.name);`
	`ZonedDateTime eventDateTime = event.getEventDateTime();`
	`System.out.println("Event Date and Time: " + eventDateTime);`
	
	`// Display in system's default time zone`
	`ZonedDateTime defaultZonedDateTime = eventDateTime.withZoneSameInstant(ZoneId.systemDefault());`
	`System.out.println("Event Date and Time in your local time zone: " + defaultZonedDateTime);`
	
	`scanner.close();`

Close curly braces to end the `EventManagement` class definition.	`}` `}`

Explanation: This Java program is a simple event management system that allows users to enter an event's details, including its name, date, time, and time zone. It then converts and displays the event time in both the specified time zone and the system's default time zone.

Formatting Dates in Java

Formatting a date using LocalDate

Description	Example

Import the `LocalDate` class to represent a date (year, month, day) without time or a time zone and `DateTimeFormatter` class to define a custom format for displaying dates.	`import java.time.LocalDate;`
	`import java.time.format.DateTimeFormatter;`

Define a public class `DateFormattingExample` that contains the Java `main` method. Use `LocalDate.now()` to retrieve the current date in the "YYYY-MM-DD" format, which is the default format of `LocalDate()`. Define a date format using `DateTimeFormatter.ofPattern("dd/MM/yyyy")`. Format the date using `currentDate.format(formatter)` to convert the current date into the specified format and print the formatted date to the console.	`public class DateFormattingExample {`
	`    public static void main(String[] args) {`
	`        // Get the current date`
	`        LocalDate currentDate = LocalDate.now();`
	
	`        // Define the format`
	`        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");`
	
	`        // Format the date`
	`        String formattedDate = currentDate.format(formatter);`
	
	`        // Print the formatted date`
	`        System.out.println("Formatted Date: " + formattedDate);`

Close curly braces to end the `DateFormattingExample` class definition.	`    }`
	`}`

Description	Example

Explanation: This Java program demonstrates how to format a date using DateTimeFormatter from the java.time package. It formats dates into a human-friendly format.

Real World example of formatting birthdates in a User Registration System

Description	Example

Import the `LocalDate` class to represent a date (year, month, day) without time or a time zone, the `DateTimeFormatter` class to define a custom format for displaying dates, and the `Scanner` class to get user input.	
	```
	import java.time.LocalDate;
	import java.time.format.DateTimeFormatter;
	import java.util.Scanner;
	```

Define a public class `UserRegistration` that contains the Java `main` method. Create a `Scanner` object to read user input. Get the user name and store it in the `name` variable. Get the user birthdate in the "YYYY-MM-DD" format. The input string `birthdateInput` is converted into a `LocalDate` object using `LocalDate.parse()`. Format the birthdate using the "EEEE, MMM dd, yyyy" pattern, where EEE is the full weekday name, such as "Monday"; MMM is the abbreviated month name, such as Mar, dd is the two-digit day, such as 11, and "yyyy" is the four-digit year, such as 2025. Use the `birthdate.format(formartter)` method to convert the date into a readable format. Print a personalized message with the formatted birthdate and close the scanner.	
	```
	public class UserRegistration {
	    public static void main(String[] args) {
	        Scanner scanner = new Scanner(System.in);

	        // Get user's name
	        System.out.print("Enter your name: ");
	        String name = scanner.nextLine();

	        // Get user's birthdate
	        System.out.print("Enter your birthdate (yyyy-MM-dd): ");
	        String birthdateInput = scanner.nextLine();

	        // Parse the input string into a LocalDate object
	        LocalDate birthdate = LocalDate.parse(birthdateInput);

	        // Define the desired output format
	        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("EEEE, MMM dd, yyyy");

	        // Format the birthdate using the defined formatter
	        String formattedBirthdate = birthdate.format(formatter);

	        // Display the result
	        System.out.println("Hello " + name + "! Your birthdate is: " + formattedBirthdate);

	        // Close the scanner
	        scanner.close();
	```

Close curly braces to end the `UserRegistration` class definition.	
	```
	    }
	}
	```

Explanation: This Java program prompts the user to enter their name and birthdate, then formats and displays the birthdate in a more readable format.

Using Timezones in Java

Creating a ZoneId

Description	Example
Import `ZoneId` which is part of the Java Date and Time API class to represent a time zone, such as "America/New_York", "Asia/Tokyo", and "Europe/London".	`import java.time.ZoneId;`

Description	Example

Define a public class `TimeZoneExample` that contains the Java `main` method. Use `ZoneId.of("America/New_York")` to create a `ZoneId` object for New York and display the Time Zone ID to the console.	`public class TimeZoneExample {`
	`    public static void main(String[] args) {`
	`        // Creating a ZoneId for New York`
	`        ZoneId newYorkZone = ZoneId.of("America/New_York");`
	`        System.out.println("Time Zone ID: " + newYorkZone);`

Close curly braces to end the `TimeZoneExample` class definition.	`    }`
	`}`

Explanation: This Java program demonstrates how to create and display a time zone ID using the `java.time` package.

Creating a ZoneDateTime

Description	Example

Import the `ZonedDateTime` and `ZoneId` classes which are part of the Java Date and Time API class to represent a date-time with a time zone.	`import java.time.ZonedDateTime;`
	`import java.time.ZoneId;`

Create a time zone object for New York by calling `ZoneId.of("America/New_York")` and retrieve the current date and time in that time zone. Display the current date and time in New York.	`public class ZonedDateTimeExample {`
	`    public static void main(String[] args) {`
	`        // Getting the current date and time in New York`
	`        ZonedDateTime newYorkTime = ZonedDateTime.now(ZoneId.of("America/New_York"));`
	`        System.out.println("Current Date and Time in New York: " + newYorkTime);`

Close curly braces to end the `ZoneDateTimeExample` class definition.	`    }`
	`}`

Explanation: This Java program demonstrates how to create and display a time zone ID using the `java.time` package.

Real World example of Scheduling Meeting across Time Zones

Description	Example
Import the `ZonedDateTime`, `ZoneId`, and `DateTimeFormatter` classes which are part of the Java Date and Time API class to represent a date-time with a time zone and format the date-time in a custom pattern.	`import java.time.ZonedDateTime;` `import java.time.ZoneId;` `import java.time.format.DateTimeFormatter;`
Define the meeting time in UTC. `ZonedDateTime.parse("2024-12-30T15:00:00Z")` parses the fixed UTC time (2024-12-30 15:00:00 UTC) into a `ZonedDateTime` object. Create an array of time zones for participants in New York, London, Kolkata, and Sydney. These time zones are later used to convert the UTC time to each participant's local time. Create a custom formatter for displaying the date and time in the pattern: `DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss z")`. Print the meeting time in UTC using the defined format. For each time zone, use `meetingTimeUTC.withZoneSameInstant(ZoneId.of(timeZone))` to convert the meeting time from UTC to the local time of that participant's time zone and print the meeting time in the participant's local time zone using the custom formatter.	`public class ConferenceScheduler {` `public static void main(String[] args) {` `// Define the meeting time in UTC` `ZonedDateTime meetingTimeUTC = ZonedDateTime.parse("2024-12-30T15:00:00Z");` `// Define participant time zones` `String[] participantTimeZones = {` `"America/New_York", // Eastern Standard Time (EST)` `"Europe/London",    // Greenwich Mean Time (GMT)` `"Asia/Kolkata",     // Indian Standard Time (IST)` `"Australia/Sydney"  // Australian Eastern Daylight Time (AEDT)` `};` `// Format for displaying the date and time` `DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:s` `// Print the meeting time in each participant's local time zone` `System.out.println("Meeting Time in UTC: " + meetingTimeUTC.format(formatter)` `for (String timeZone : participantTimeZones) {` `ZonedDateTime localTime = meetingTimeUTC.withZoneSameInstant(ZoneId.of(ti` `System.out.println("Meeting Time in " + timeZone + ": " + localTime.forma` `}`
Close curly braces to end the `ConferenceScheduler` class definition.	`}` `}`

Explanation: This Java program simulates scheduling a meeting across different time zones. It converts a fixed UTC meeting time to the local times of participants in various time zones and displays it in a formatted way.

Parsing Dates from Strings in Java

Parsing dates with DateTimeFormatter

Description	Example
Import the `LocalDate` and `DateTimeFormatter` classes, which are part of the Java Date and Time API class and used to represent dates without a time zone and define a pattern for parsing and formatting dates.	`import java.time.LocalDate;` `import java.time.format.DateTimeFormatter;`
Create a public class `DateParsingExample` that contains the Java `main` method and define a string variable `dateString` to represent date in the format "yyyy-MM-dd". Create a date formatter using the `DateTimeFormatter.ofPattern("yyyy-MM-dd")` method. Use `LocalDate.parse(dateString, formatter)` to convert the `dateString` into a `LocalDate` object and print the parsed date.	`public class DateParsingExample {` `public static void main(String[] args) {` `// Define a date string to parse` `String dateString = "2025-01-23";` `// Create a DateTimeFormatter to define the expected format` `DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");` `// Parse the string into a LocalDate object` `LocalDate date = LocalDate.parse(dateString, formatter);`

Description	Example
	```
	        // Output the parsed date
	        System.out.println("Parsed date: " + date);
	```

Close curly braces to end the `DateParsingExample` class definition.	
	```
	    }
	}
	```

Explanation: This Java program demonstrates how to parse a date string into a `LocalDate` object using the `DateTimeFormatter` class.

Using custom date formats

Description	Example

Import the `LocalDate` and `DateTimeFormatter` classes, which are part of the Java Date and Time API class and used to represent dates without a time zone and define a pattern for parsing and formatting dates.	
	```
	import java.time.LocalDate;
	import java.time.format.DateTimeFormatter;
	```

Create a public class `CustomDateParsing` that contains the Java `main` method and define a string variable `dateString` to represent date in the format "dd/MM/yyyy". Create a date formatter using the `DateTimeFormatter.ofPattern("dd/MM/yyyy")` method. Use `LocalDate.parse(dateString, formatter)` to convert the `dateString` into a `LocalDate` object and print the parsed date.	
	```
	public class CustomDateParsing {
	    public static void main(String[] args) {
	        String dateString = "23/01/2025";

	        // Define the pattern for parsing
	        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");

	        LocalDate date = LocalDate.parse(dateString, formatter);

	        System.out.println("Parsed date: " + date);
	```

Close curly braces to end the `CustomDateParsing` class definition.	
	```
	    }
	}
	```

Explanation: This Java program demonstrates how to parse a date string with a custom format into a `LocalDate` object using the `DateTimeFormatter` class.

Parsing LocalDateTime

Description	Example

Import the `LocalDateTime` and `DateTimeFormatter` classes, which are part of the Java Date and Time API class and used to represent dates without a time zone and define a pattern for parsing and formatting dates.	`import java.time.LocalDateTime;`
	`import java.time.format.DateTimeFormatter;`

Create a public class `DateTimeParsingExample` that contains the Java `main` method and define a string variable `dateString` to represent date in the "yyyy-MM-dd" format. Create a date formatter using the `DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm")` method. Use `LocalDateTime.parse(dateTimeString, formatter)` to convert the `dateTimeString` into a `LocalDateTime` object using the formatter and print the parsed date.	`public class DateTimeParsingExample {`
	`    public static void main(String[] args) {`
	`        String dateTimeString = "2025-01-23 15:30";`
	
	`        // Define the pattern for date and time`
	`        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");`
	
	`        LocalDateTime dateTime = LocalDateTime.parse(dateTimeString, formatter);`
	
	`        System.out.println("Parsed date and time: " + dateTime);`

Close curly braces to end the `DateTimeParsingExample` class definition.	`    }`
	`}`

Explanation: This Java program demonstrates how to parse a date string with a custom format into a `LocalDateTime` object using the `DateTimeFormatter` class.

Example of extracting date from a simple sentence

Description	Example

Import the `LocalDate`, `DateTimeFormatter`, and `DateTimeParseException` classes, which are part of the Java Date and Time API class and used to represent dates without a time zone, define a pattern for parsing and formatting dates, and handle errors if the date format is incorrect.	`import java.time.LocalDate;`
	`import java.time.format.DateTimeFormatter;`
	`import java.time.format.DateTimeParseException;`

Create a public class `ExtractDateFromSentence` that contains the Java `main` method and define a sentence containing a date formatted as "yyyy-MM-dd". Extract the date substring using `sentence.substring(sentence.indexOf("on") + 3, sentence.indexOf("."))`. The `sentence.indexOf("on") + 3` method finds the position of "on" and moves three characters forward to skip "on " (with the space), `sentence.indexOf(".")` identifies the position of the period (".") at the end of the date, and `substring(...)` extracts the portion of the string that contains the date. Parse the extracted date using `LocalDate.parse(dateString, formatter)` and convert the extracted string into a `LocalDate` object. The `try-catch` block prints the extracted date if successful. If parsing fails due to an incorrect format, the block catches `DateTimeParseException` and displays an error message.	`public class ExtractDateFromSentence {`
	`    public static void main(String[] args) {`
	`        String sentence = "The event will take place on 2025-01-23.";`
	
	`        // Define the date pattern`
	`        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");`
	
	`        // Extract the date part from the string`
	`        String dateString = sentence.substring(sentence.indexOf("on") + 3, sentence.indexOf("."));`
	
	`        try {`
	`            LocalDate date = LocalDate.parse(dateString, formatter);`
	`            System.out.println("Extracted date: " + date);`
	`        } catch (DateTimeParseException e) {`
	`            System.out.println("Error parsing date: " + e.getMessage());`
	`        }`

Description	Example

Close curly braces to end the `ExtractDateFromSentence` class definition.	
	```
	    }
	}
	```

Explanation: This Java program extracts a date from a given sentence, parses it into a `LocalDate` object, and displays it in a structured format. It also gracefully handles potential parsing errors.

Example of extracting multiple dates from a text string

Description	Example

Import the `LocalDate`, `DateTimeFormatter`, and `DateTimeParseException` classes, which are part of the Java Date and Time API class and used to represent dates without a time zone, define a pattern for parsing and formatting dates, and handle errors if the date format is incorrect.	
	```
	import java.time.LocalDate;
	import java.time.format.DateTimeFormatter;
	import java.time.format.DateTimeParseException;
	```

Create a public class `ExtractMultipleDates` that contains the Java `main` method and define a `text` string containing three dates in the "yyyy-MM-dd" format. These dates are separated by commas and the word "and". Define the date format using `DateTimeFormatter.ofPattern("yyyy-MM-dd")`. Use regular expressions (`", | and "`) to split the string by comma followed by a space (", ") and the word "and" followed by a space ("and "). This extracts the date strings from the text. Iterate over the extracted parts and parse dates. For each extracted part, `trim()` removes any leading or trailing spaces and `LocalDate.parse(part.trim(), formatter)` converts the string into a `LocalDate` object. If parsing is successful, it prints the extracted date. If parsing fails, the catch block handles the error and prints an error message.	
	```
	public class ExtractMultipleDates {
	    public static void main(String[] args) {
	        String text = "Important dates: 2025-01-23, 2025-02-14, and 2025-03-01.";

	        // Define the date pattern
	        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

	        // Split the string to find dates
	        String[] parts = text.split(", | and ");

	        for (String part : parts) {
	            try {
	                LocalDate date = LocalDate.parse(part.trim(), formatter);
	                System.out.println("Extracted date: " + date);
	            } catch (DateTimeParseException e) {
	                System.out.println("Error parsing date: " + part.trim());
	            }
	        }
	```

Close curly braces to end the `ExtractMultipleDates` class definition.	
	```
	    }
	}
	```

Explanation: This Java program extracts multiple dates from a given text, parses them into `LocalDate` objects, and prints them in a structured format. It also handles potential errors if any part of the text is not in the expected date format.

Example of extracting dates from mixed content

Description	Example

Import the `LocalDate`, `DateTimeFormatter`, and `DateTimeParseException` classes, which are part of the Java Date and Time API class and used to represent dates without a time zone, define a pattern for parsing and	
	```
	import java.time.LocalDate;
	import java.time.format.DateTimeFormatter;
	import java.time.format.DateTimeParseException;
	```

Description	Example
formatting dates, and handle errors if the date format is incorrect.	

Create a public class `ExtractDatesFromMixedContent` that contains the Java `main` method and define a string named `mixedContent` containing a mixture of text and two dates (2025-01-23 and 2025-02-28). The dates are in the "yyyy-MM-dd" format. These dates are separated by commas and the word "and". Define the date format using `DateTimeFormatter.ofPattern("yyyy-MM-dd")`. Splits the input string by spaces into individual words. The resulting `words[]` array contains both text and possible date strings. Iterate over each word using the regex `word.matches("\\d{4}-\\d{2}-\\d{2}")` and check if it matches the date pattern (yyyy-MM-dd). If a word matches the pattern, attempt to parse it into a LocalDate using the previously defined formatter. If parsing is successful, prints the extracted date. If there is a parsing error (invalid date), the `try-catch` block handles it and prints an error message.	
	```
	public class ExtractDatesFromMixedContent {
	    public static void main(String[] args) {
	        String mixedContent = "Please note that our deadlines are on 2025-01-23 and 2025-02-28.";

	        // Define the date pattern
	        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

	        // Split based on spaces and check each part
	        String[] words = mixedContent.split(" ");

	        for (String word : words) {
	            if (word.matches("\\d{4}-\\d{2}-\\d{2}")) { // Check if it matches a date pattern
	                try {
	                    LocalDate date = LocalDate.parse(word, formatter);
	                    System.out.println("Extracted date: " + date);
	                } catch (DateTimeParseException e) {
	                    System.out.println("Error parsing date: " + word);
	                }
	            }
	        }
	```

Close curly braces to end the `ExtractDatesFromMixedContent` class definition.	
	```
	    }
	}
	```

Explanation: This Java program extracts dates from a string containing mixed content (text and dates), parses them into `LocalDate` objects, and prints the valid dates. If any date format is invalid, it gracefully handles the error.

Author(s)

Ramanujam Srinivasan
Lavanya Thiruvali Sunderarajan