1. **String**: A classe **String** é usada para representar cadeias de caracteres e oferece métodos para manipulação de texto.
2. **StringBuilder**: **StringBuilder** é uma classe que permite construir strings mutáveis de forma eficiente, especialmente quando você precisa concatenar muitas strings.
3. **Date**: A classe **Date** representa um ponto no tempo, mas está desencorajada em favor das classes **java.time** introduzidas no Java 8.
4. **Calendar**: A classe **Calendar** é usada para realizar operações com datas e horários mais complexas do que a classe **Date**.
5. **DateFormat**: **DateFormat** é uma classe abstrata usada para formatar e analisar datas e horários.
6. **Locale**: **Locale** representa uma configuração regional, como idioma e região, e é usada para formatação e internacionalização.
7. **SimpleDateFormat**: **SimpleDateFormat** é uma implementação concreta de **DateFormat** para formatar e analisar datas e horários de forma simples.
8. **LocalDate, LocalDateTime, LocalTime**: Essas classes fazem parte do pacote **java.time** introduzido no Java 8 e são usadas para representar datas, horários e data/horário, respectivamente.
9. **Instant**: **Instant** é usada para representar um ponto instantâneo no tempo, com precisão de nanossegundos.
10. **Duration e Period**: **Duration** representa um período de tempo em horas, minutos, segundos, etc., enquanto **Period** representa um período em anos, meses e dias.
11. **ChronoUnit**: Enumeração que define unidades de tempo usadas para cálculos com datas e horários.
12. **TemporalAdjuster**: Uma interface que permite ajustar objetos de data e hora, por exemplo, para encontrar o próximo dia útil.
13. **ZonedDateTime, ZoneId, OffsetDateTime**: Classes relacionadas à representação de datas e horários com informações de fuso horário.
14. **DateTimeFormatter**: Usado para formatar e analisar datas e horários no formato desejado.
15. **ResourceBundle**: Usado para internacionalização, permite carregar recursos específicos de idioma.
16. **Regex Pattern e Matcher**: Classes para trabalhar com expressões regulares, onde **Pattern** compila as expressões e **Matcher** encontra correspondências.
17. **Meta caracteres em Regex**: São caracteres especiais em expressões regulares que têm significados especiais, como **\***, **+**, **?**, etc.
18. **Range em Regex**: Usado para especificar um intervalo de caracteres em expressões regulares, por exemplo, **[a-z]** corresponde a qualquer letra minúscula.
19. **Quantificadores em Regex**: São usados para especificar quantidades, como **\*** para zero ou mais, **+** para um ou mais, **{n}** para exatamente n vezes, etc.
20. **Anchor em Regex**: São usados para ancorar a correspondência no início ou no final de uma linha, por exemplo, **^** e **$**.
21. **Tokens e Delimitadores**: Geralmente usados em análise de texto para dividir uma string em partes menores com base em delimitadores.
22. **IO File, FileWriter, FileReader**: Classes para operações de entrada e saída de arquivos.
23. **IO BufferedWriter, BufferedReader**: Classes que oferecem um buffer para operações mais eficientes de leitura e gravação de arquivos.
24. **IO File para Diretórios**: Usado para manipular diretórios em Java.
25. **NIO Path, Paths e Files**: Classes para manipulação de caminhos e operações de arquivos mais avançadas, introduzidas no Java 7.
26. **NIO Normalização**: A normalização de caminhos envolve a resolução de caminhos relativos e a remoção de referências a diretórios pai.
27. **NIO relativize**: Calcula um caminho relativo entre dois caminhos.
28. **NIO BasicFileAttributes**: Usado para acessar atributos básicos de um arquivo, como tamanho e data de modificação.
29. **Serialization**: É um mecanismo em Java para serializar (converter em bytes) objetos para que possam ser armazenados ou transmitidos e deserializados posteriormente para reconstruir o objeto original.