Zaawansowane programowanie w Javie

Studia zaoczne – Wykład 8

dr hab. Andrzej Zbrzezny, profesor UJD

Katedra Matematyki i Informatyki
Wydział Nauk Ścisłych, Przyrodniczych i Technicznych
Uniwersytet Jana Długosza w Częstochowie

15 czerwca 2024

Zalecana Literatura

Literatura podstawowa

- Cay Horstmann.
  Java. Przewodnik doświadczonego programisty. Wydanie III.
  Wydawnictwo Helion. Gliwice, październik 2023.
- Joshua Bloch.
  Java. Efektywne programowanie. Wydanie III.
  Wydawnictwo Helion. Gliwice, sierpień 2018.
- Cay Horstmann.
  Java. Podstawy. Wydanie XII.
  Wydawnictwo Helion. Gliwice, grudzień 2022.
- https://dev.java/

API daty i czasu

Wprowadzenie

- Wszystkie obiekty `java.time` są niemodyfikowalne.
- Klasa `Instant` opisuje punkt w czasie (podobnie do `Date`).
- W języku Java każdy dzień ma dokładnie `86_400` sekund (nie ma sekund przestępnych).
- Klasa `Duration` opisuje różnicę pomiędzy dwoma obiektami klasy `Instant`.
- Klasa `LocalDateTime` nie ma informacji o strefie czasowej.
- Metody klasy `TemporalAdjuster` wykonują typowe operacje na kalendarzu, takie jak wyszukiwanie pierwszego wtorku miesiąca.

API daty i czasu

Wprowadzenie

- Klasa `ZonedDateTime` opisuje punkt w czasie w określonej strefie czasowej (podobnie do `GregorianCalendar`).
- Gdy chcemy używać czasu z uwzględnieniem stref czasowych, należy korzystać z klasy `Period`, a nie `Duration`, aby uwzględnić zmiany czasu z letniego na zimowy i odwrotnie.
- Do formatowania oraz przetwarzania dat i czasu służy klasa `DateTimeFormatter`.

API daty i czasu

Linia czasu

- Specyfikacja Date and Time API języka Java wymaga, by korzystać w nim ze skali czasu, która:
  - ma `86_400` sekund w ciągu dnia;
  - idealnie zgadza się z czasem oficjalnym codziennie w południe;
  - była bardzo zbliżona do niego w pozostałym czasie w precyzyjnie określony sposób.

  Daje to Javie elastyczność umożliwiającą dostosowanie do przyszłych zmian oficjalnego czasu.
- W języku Java klasa `Instant` reprezentuje punkt na linii czasu.
- Punkt odniesienia, nazywany epoką (ang. epoch), został ustalony na północ 1 stycznia 1970 roku na południku przechodzącym przez Greenwich Royal Observatory w Londynie.
- Została zachowana tutaj konwencja używana do określania czasu w systemie Unix/POSIX.

API daty i czasu

Linia czasu

- Począwszy od tego punktu odniesienia czas jest odmierzany `86_400` sekundami na dzień do przodu i do tyłu z dokładnością do nanosekund.
- Wartości klasy Instant mogą sięgać miliarda lat wstecz. Najmniejsza wartość, `Instant.MIN` jest równa `'-1000000000-01-01T00:00Z'`
- Nie wystarczy to, by zapisać wiek Wszechświata (około `13,799 ± 0,021` miliardów lat), ale powinno wystarczyć dla wszystkich praktycznych zastosowań.
- Największa wartość, `Instant.MAX` jest równa `'1000000000-12-31T23:59:59.999999999Z'`, Jest to 31 grudnia 1 000 000 000 roku.

API daty i czasu

Linia czasu

- Metoda statyczna `Instant.now` zwraca bieżącą chwilę.
- Można porównać dwie chwile za pomocą metod `equals` i `compareTo` w standardowy sposób, dzięki czemu można korzystać z instancji klasy `Instant` w roli znaczników czasu.
- Aby znaleźć różnicę pomiędzy dwoma punktami czasu, należy użyć metody statycznej `Duration.between`. Przykładowo, czas działania algorytmu można zmierzyć w taki sposób:

```
Instant start = Instant.now();
uruchomAlgorytm();
Instant koniec = Instant.now();
Duration zmierzonyCzas =
    Duration.between(start, koniec);
long millis = zmierzonyCzas.toMillis();
```

API daty i czasu

Linia czasu

- Klasa `Duration` opisuje długość odcinka czasu, jaki upłynął pomiędzy dwoma chwilami opisanymi klasą `Instant`.
- Można pobrać długość `Duration` w zwykłych jednostkach, wywołując `toNanos`, `toMillis`, `toSeconds`, `toMinutes`, `toHours` lub `toDays`.
- Klasa `Duration` przechowuje liczbę sekund w zmiennej typu **`long`**, a liczbę nanosekund w dodatkowej zmiennej typu **`int`**.
- Jeżeli chcemy wykonywać obliczenia z dokładnością do nanosekund i potrzebujemy całego zakresu klasy `Duration`, można wykorzystać jedną z metod opisanych w tabeli na następnym slajdzie.
- W innym przypadku można po prostu wywołać `toNanos` i wykonać obliczenia na zmiennych typu **`long`**.

API daty i czasu

Działania arytmetyczne na klasach `Instant` i `Duration`

Metoda	Opis
`plus`, `minus`	Dodaje lub odejmuje czas od wartości klasy `Instant` lub `Duration`
`plusNanos`, `plusMillis`, `plusSeconds`, `plusMinutes`, `plusHours`, `plusDays`	Dodaje podaną liczbę wskazanych jednostek czasu do wartości klasy `Instant` lub `Duration`
`minusNanos`, `minusMillis`, `minusSeconds`, `minusMinutes`, `minusHours`, `minusDays`	Odejmuje podaną liczbę wskazanych jednostek czasu od wartości klasy `Instant` lub `Duration`
`multipliedBy`, `dividedBy`, `negated`	Zwraca odcinek czasu uzyskany przez pomnożenie lub podzielenie wartości klasy `Duration` przez podaną wartość typu long lub przez `-1`.
`isZero`, `isNegative`	Sprawdza, czy wartość zapisana w klasie `Duration` jest równa zero lub ujemna

API daty i czasu

Linia czasu – uwagi

- Aby przepełnić typ `long`, trzeba odmierzyć prawie 300 lat w nanosekundach.
- Klasy `Instant` i `Duration` są niemodyfikowalne, a wszystkie metody takie jak `multipliedBy` lub `minus` zwracają nową instancję.

Przykład (Linia czasu)

- Program `r12/r12_01/Timeline.java`

API daty i czasu

Daty lokalne

- Istnieją dwa rodzaje ludzkich miar czasu w Java API – czas lokalny (ang. local date/time) i czas strefowy (ang. zoned time).
- Czas lokalny zawiera informacje o dacie oraz godzinie, ale nie wiąże ich z informacjami o strefie czasowej.
- Przykładowo, 14 czerwca 1903 to data zapisana jako czas lokalny (to dzień, w którym urodził się Alonzo Church, wynalazca rachunku lambda).
- Ponieważ ta data nie zawiera ani godziny, ani informacji o strefie czasowej, nie wskazuje precyzyjnie punktu czasu.
- W odróżnieniu od tego zapis 16 lipca 1969, 09:32:00 EDT (start Apollo 11) opisuje czas strefowy i wskazuje precyzyjnie punkt na linii czasu.

API daty i czasu

Daty lokalne

- Jest wiele obliczeń, w których informacja o strefie czasowej nie jest potrzebna, a w niektórych przypadkach może nawet przeszkadzać.
- Załóżmy, że planujemy cotygodniowe spotkania o godzinie 11.11. Jeśli dodamy siedem dni (czyli 7×24×60×60 sekund) do terminu spotkania odbywającego się przed zmianą czasu letniego na zimowy lub odwrotnie, spotkanie będzie zaplanowane godzinę za późno lub za wcześnie!
- Dlatego projektanci API zalecają, by unikać korzystania z czasu strefowego, jeśli nie jest naprawdę konieczne wskazanie bezwzględnego czasu.
- Urodziny, święta, uzgodnione terminy itp. zazwyczaj najlepiej opisywać w postaci lokalnego czasu i daty.

API daty i czasu

Daty lokalne

- Klasa `LocalDate` reprezentuje datę z rokiem, miesiącem i dniem w miesiącu. Aby utworzyć jej instancję, można użyć metody `now` lub metod statycznych:

```
// Bieżąca data
LocalDate today = LocalDate.now();
LocalDate urodzinyAlonzo = LocalDate.of(1903, 6, 14);
// Korzystamy z typu wyliczeniowego Month
urodzinyAlonzo = LocalDate.of(1903, Month.JUNE, 14);
```

- Inaczej niż w przypadku nietypowych konwencji systemu Unix i klasy `java.util.Date`, w których miesiące są numerowane od 0, a lata zliczane od 1900 roku, miesiące numerowane są standardowo.
- Można też skorzystać z typu wyliczeniowego `Month`.

API daty i czasu

Daty lokalne – metody klasy `LocalDate`

- `now`, `of` – te metody statyczne tworzą, na podstawie bieżącego czasu lub przekazanych wartości opisujących rok, miesiąc i dzień, instancję `LocalDate`.
- `plusDays`, `plusWeeks`, `plusMonths`, `plusYears` – dodaje przekazaną liczbę dni, tygodni, miesięcy lub lat do bieżącej instancji `LocalDate`.
- `minusDays`, `minusWeeks`, `minusMonths`, `minusYears` – odejmuje przekazaną liczbę dni, tygodni, miesięcy lub lat od bieżącej instancji LocalDate
- `plus`, `minus` – dodaje lub odejmuje `Duration` lub `Period`
- `withDayOfMonth`, `withDayOfYear`, `withMonth`, `withYear` – zwraca nową instancję `LocalDate` z dniem miesiąca, dniem roku, miesiącem lub rokiem zastąpionym przekazaną wartością

API daty i czasu

Daty lokalne – metody klasy `LocalDate`

- `getDayOfMonth` – pobiera dzień miesiąca (od 1 do 31)
- `getDayOfYear` – pobiera dzień roku (od 1 do 366)
- `getDayOfWeek` – pobiera dzień tygodnia, zwracając wartość typu wyliczeniowego `DayOfWeek`
- `getMonth`, `getMonthValue` – pobiera miesiąc jako wartość typu wyliczeniowego `Month` lub jako liczbę z zakresu od 1 do 12
- `getYear` – pobiera rok z zakresu od -999999999 do 999999999
- `until` – pobiera `Period` lub liczbę przekazanych `ChronoUnits` pomiędzy dwoma datami
- `isBefore`, `isAfter` – porównuje bieżącą instancję `LocalDate` z inną

API daty i czasu

Daty lokalne – metody klasy `LocalDate`

- `isLeapYear` – zwraca `true`, jeśli wskazywany jest rok przestępny: czyli jeśli rok jest podzielny przez 4 i nie jest podzielny przez 100 lub jeśli jest podzielny przez 400.
- Algorytm ten jest wykorzystywany do wszystkich przeszłych lat, nawet jeśli jest to historycznie niepoprawne.
- Lata przestępne wprowadzono w roku -46, a reguły dotyczące podzielności przez 100 i 400 zostały wprowadzone w reformie kalendarza gregoriańskiego w 1582 roku.
- Zmiana ta została powszechnie przyjęta po ponad 300 latach.

API daty i czasu

Daty lokalne

- Przykładowo, Dzień Programisty to 256. dzień roku. Oto sposób, aby go łatwo obliczyć:

```
// 13 września, ale w roku przestępnym 12 września
var dzieńProgramisty =
    LocalDate.of(2019, 1, 1).plusDays(255);
```

- Różnicę pomiędzy dwoma chwilami czasu opisuje klasa `Duration`. Jej odpowiednikiem dla dat lokalnych jest klasa `Period`, która zawiera liczbę lat, miesięcy i dni.
- Można wywołać `birthday.plus(Period.ofYears(1))`, by uzyskać datę urodzin za rok. Oczywiście można też po prostu wywołać `birthday.plusYears(1)`.
- Ale wyrażenie `birthday.plus(Duration.ofDays(365))` nie da poprawnego wyniku w roku przestępnym.

API daty i czasu

Daty lokalne

- Metoda `until` zwraca różnicę pomiędzy dwoma datami lokalnymi.
- Przykładowo, `dzieńNiepodległości.until(bożeNarodzenie)` zwraca okres 5 miesięcy i 21 dni.
- Nie jest to zbyt przydatne, ponieważ liczba dni w miesiącu może być różna. Aby ustalić liczbę dni, należy użyć `dzieńNiepodległości.until(bożeNarodzenie, ChronoUnit.DAYS)` co daje w wyniku 174 dni.

API daty i czasu

Daty lokalne

- Niektóre metody klasy `LocalDate` mogą potencjalnie utworzyć nieistniejące daty.
- Przykładowo, dodanie jednego miesiąca do 31 stycznia nie powinno zwracać 31 lutego.
- Zamiast wyrzucać wyjątek, metody te zwracają ostatni poprawny dzień miesiąca. Na przykład
  `LocalDate.of(2016, 1, 31).plusMonths(1)`
  oraz
  `LocalDate.of(2016, 3, 31).minusMonths(1)`
  zwracają 29 lutego 2016.

API daty i czasu

Daty lokalne

- Metoda `getDayOfWeek` zwraca dzień tygodnia w postaci wartości typu wyliczeniowego `DayOfWeek`.
- Element `DayOfWeek.MONDAY` ma wartość 1, a `DayOfWeek.SUNDAY` ma wartość 7.
- Na przykład `LocalDate.of(1900, 1, 1).getDayOfWeek().getValue()` zwraca 1.
- Typ wyliczeniowy `DayOfWeek` ma wygodne metody `plus` oraz `minus` do obliczania dni tygodnia z dzieleniem modulo 7.
- Na przykład `DayOfWeek.SATURDAY.plus(3)` zwraca `DayOfWeek.TUESDAY`.

API daty i czasu

Daty lokalne

- Dni weekendu są umieszczone na końcu tygodnia. Różni się to od `java.util.Calendar` – tu niedziela ma wartość 1, a sobota wartość 7.
- Poza klasą `LocalDate` istnieją też klasy: `MonthDay`, `YearMonth` oraz `Year` – do opisywania dat częściowych.
- Przykładowo, 25 grudnia (bez określania roku) może być reprezentowany przez `MonthDay`.

Przykład (Daty lokalne)

- Program `r12/r12_02/LocalDates.java`

Modyfikatory daty

- W przypadku aplikacji do planowania często potrzebujemy obliczać daty takie jak „pierwszy wtorek każdego miesiąca”.
- Klasa `TemporalAdjusters` udostępnia kilka statycznych metod umożliwiających typowe modyfikacje.
- Przekazujemy wynik działania metody modyfikującej do metody `with`.
- Przykładowo, pierwszy wtorek miesiąca można obliczyć w następujący sposób:

```
LocalDate pierwszyWtorek = LocalDate
    .of(year, month, 1)
    .with(TemporalAdjusters
        .nextOrSame(DayOfWeek.TUESDAY));
```

API daty i czasu

Modyfikatory daty

- `next`(dzieńTygodnia), `previous`(dzieńTygodnia) – następna lub poprzednia data przypadająca na wskazany dzieńTygodnia
- `nextOrSame`(dzieńTygodnia), `previousOrSame`(dzieńTygodnia) – następna lub poprzednia data przypadająca na wskazany dzieńTygodnia, począwszy od bieżącej daty
- `dayOfWeekInMonth`(n, dzieńTygodnia) – n-ty dzieńTygodnia w danym miesiącu
- `lastInMonth`(dzieńTygodnia) – ostatni dzieńTygodnia w danym miesiącu

API daty i czasu

Modyfikatory daty

- `firstDayOfMonth`(), `firstDayOfNextMonth`(), `firstDayOfNextYear`(), `lastDayOfMonth`(), `lastDayOfPreviousMonth`(), `lastDayOfYear`() – zwraca odpowiednią datę zgodnie z nazwą metody: pierwszy dzień miesiąca, pierwszy dzień kolejnego miesiąca, pierwszy dzień następnego roku, ostatni dzień miesiąca, ostatni dzień poprzedniego miesiąca, ostatni dzień roku.
- Można też wykonać swój własny modyfikator, implementując interfejs `TemporalAdjuster`.

API daty i czasu

Modyfikatory daty

- Poniżej znajduje się modyfikator do obliczania następnego dnia tygodnia:

```
TemporalAdjuster NASTĘPNY_DZIEŃ_PRACY = w -> {
    LocalDate result = (LocalDate) w;
    do
    {
        result = result.plusDays(1);
    } while (result.getDayOfWeek().getValue() >= 6);
    return result;
};
LocalDate backToWork =
    today.with(NASTĘPNY_DZIEŃ_PRACY);
```

Modyfikatory daty

- Zauważmy, że parametr wyrażenia lambda ma typ `Temporal` i musi być rzutowany na `LocalDate`.
- Można uniknąć tego rzutowania, korzystając z metody `ofDateAdjuster`, która oczekuje wyrażenia lambda typu `UnaryOperator<LocalDate>`:

```
TemporalAdjuster NASTĘPNY_DZIEŃ_PRACY =
    TemporalAdjusters.ofDateAdjuster(w -> {
        LocalDate result = w; // Bez rzutowania
        do {
            result = result.plusDays(1);
        } while (result.getDayOfWeek().getValue() >= 6);
        return result;
});
```

API daty i czasu

Przykład (Modyfikatory daty)

- Program `r12/r12_03/DateAdjusters.java`

API daty i czasu

Czas lokalny

- Klasa `LocalTime` reprezentuje godzinę taką jak 14:30:00. Można utworzyć jej instancję za pomocą metody `now` lub `of`:

```
LocalTime teraz = LocalTime.now();
LocalTime spanie = LocalTime.of(14, 30);
// Lub LocalTime.of(14, 30, 0)
```

API daty i czasu

Czas lokalny

- `now`, `of` – te metody statyczne tworzą instancję `LocalTime`, opierając się na bieżącym czasie lub przekazanych wartościach opisujących godziny, minuty oraz, opcjonalnie, sekundy i nanosekundy
- `plusHours`, `plusMinutes`, `plusSeconds`, `plusNanos` – dodaje godziny, minuty, sekundy lub nanosekundy do bieżącej instancji `LocalTime`
- `minusHours`, `minusMinutes`, `minusSeconds`, `minusNanos` – odejmuje godziny, minuty, sekundy lub nanosekundy od bieżącej instancji `LocalTime`
- `plus`, `minus` – dodaje lub odejmuje `Duration`

API daty i czasu

Czas lokalny

- `withHour`, `withMinute`, `withSecond`, `withNano` – zwraca nową instancję `LocalTime` z godziną, minutą, sekundą lub nanosekundą zmienioną na podaną wartość
- `getHour`, `getMinute`, `getSecond`, `getNano` – pobiera godzinę, minutę, sekundę lub nanosekundę bieżącej instancji `LocalTime`
- `toSecondOfDay`, `toNanoOfDay` – zwraca liczbę sekund lub nanosekund od północy do czasu zapisanego w bieżącej instancji `LocalTime`
- `isBefore`, `isAfter` – porównuje bieżącą instancję `LocalTime` z inną.

API daty i czasu

Czas lokalny

- Istnieje klasa `LocalDateTime`, reprezentująca datę i czas.
- Ta klasa jest odpowiednia do zapisywania punktów w czasie w ustalonej strefie czasowej – na przykład przy planowaniu zajęć lub wydarzeń.
- Jednak jeśli musimy wykonywać obliczenia, które biorą pod uwagę zmiany czasu z letniego na zimowy, lub obsługiwać użytkowników znajdujących się w różnych strefach czasowych, powinniśmy użyć klasy `ZonedDateTime` – omówimy ją jako kolejną.

API daty i czasu

Przykład (Czas lokalny)

- Program `r12/r12_04/LocalTimes.java`

API daty i czasu

Czas strefowy

- Strefy czasowe, prawdopodobnie dlatego, że są w całości wynikiem ludzkiej kreatywności, wprowadzają jeszcze większy bałagan niż komplikacje wynikające z nieregularności obrotów Ziemi.
- W świecie racjonalnym wszyscy korzystalibyśmy z czasu Greenwich i niektórzy z nas jedliby śniadanie o 2:00, inni o 22:00. Nasze żołądki by to ustaliły.
- Tak dzieje się w Chinach, które leżą w czterech zwykłych strefach czasowych.
- Niestety, mamy strefy czasowe z nieregularnymi i przesuwającymi się granicami oraz, aby jeszcze pogorszyć sprawę, zmiany czasu z letniego na zimowy i odwrotnie.

API daty i czasu

Czas strefowy

- Chociaż strefy czasowe mogą wydawać się niepotrzebnym kaprysem, ich istnienie jest faktem i trzeba brać je pod uwagę.
- Gdy implementujemy aplikację kalendarza, musi ona działać poprawnie, jeżeli ludzie będą przemieszczać się między krajami.
- Jeżeli mamy połączenie konferencyjne o dziesiątej w Nowym Jorku, ale w tej chwili jesteśmy w Berlinie, oczekujemy, że przypomnienie pojawi się o odpowiedniej godzinie.

API daty i czasu

Czas strefowy

- Internet Assigned Numbers Authority (IANA) przechowuje bazę danych wszystkich znanych stref czasowych świata (`https://www.iana.org/time-zones`), która jest aktualizowana kilka razy do roku.
- Większość aktualizacji dotyczy zmieniających się reguł zmian z czasu letniego na zimowy i odwrotnie. Java korzysta z bazy danych IANA.
- Każda strefa czasowa ma identyfikator, taki jak `America/New_York` czy `Europe/Warsaw`.
- Aby pobrać wszystkie dostępne strefy czasowe, należy wywołać `ZoneId.getAvailableIds`.
- W dniu 31 stycznia 2018 roku istniało dokładnie 600 identyfikatorów. Obecnie są 603 identyfikatory stref czasowych.

API daty i czasu

Czas strefowy

- Dla danego identyfikatora strefy czasowej statyczna metoda `ZoneId.of(id)` zwraca obiekt `ZoneId`.
- Można go wykorzystać, aby przekształcić obiekt `LocalDateTime` w obiekt `ZonedDateTime`, wywołując `local.atZone(idStrefy)`, lub można utworzyć `ZonedDateTime`, wywołując metodę statyczną `ZonedDateTime.of(rok, miesiąc, dzień, godzina, minuta, sekunda, nano, idStrefy)`.
- Przykładowo:

```
// 1969-07-16T09:32-04:00[America/New_York]
ZonedDateTime startApollo11 =
    ZonedDateTime.of(1969, 7, 16, 9, 32, 0, 0,
        ZoneId.of("America/New_York"));
```

API daty i czasu

Czas strefowy

- Jest to konkretny punkt w czasie. Aby pobrać obiekt `Instant` należy zastosować wywołanie `startApollo11.toInstant()`.
- I odwrotnie, jeżeli mamy punkt w czasie, należy użyć wywołania `instant.atZone(ZoneId.of("UTC"))`, aby pobrać obiekt `ZonedDateTime` dla Greenwich Royal Observatory.
- Aby pobrać czas dla dowolnego innego punktu na Ziemi należy użyć innego `ZoneId`.
- UTC oznacza Coordinated Universal Time, czyli uniwersalny czas koordynowany, a sam akronim powstał w wyniku kompromisu pomiędzy skrótem angielskiego określenia oraz francuskiego Temps Universel Coordiné i wyróżnia się tym, że nie jest poprawny w żadnym z języków.
- UTC jest czasem dla Greenwich Royal Observatory bez uwzględnienia sezonowej zmiany czasu.

API daty i czasu

Czas strefowy

- Klasa `ZonedDateTime` ma wiele takich samych metod jak klasa `LocalDateTime` Większość z nich jest oczywista, ale sezonowe zmiany czasu wprowadzają kilka komplikacji.
- Po rozpoczęciu sezonowej zmiany czasu zegar jest przesuwany o godzinę do przodu. Co się stanie, jeśli utworzymy czas wypadający w ominiętej godzinie?
- Przykładowo, w 2013 roku zmieniono czas 31 marca o godzinie 2:00. Jeżeli spróbujemy utworzyć nieistniejącą godzinę 2:30 31 marca, w rzeczywistości otrzymamy godzinę 3:30.
- I odwrotnie, przy zmianie czasu w przeciwną stronę zegary są cofane o godzinę i pojawiają się dwa różne punkty opisane tym samym czasem lokalnym!
- Konstruując czas w tym zakresie, uzyskamy wskazanie na wcześniejszą z dwóch chwil.

API daty i czasu

Czas strefowy

- Godzinę później czas ma taką samą godzinę i minutę, ale przesunięcie czasu dla strefy czasowej się zmieniło.
- Musimy też zwrócić uwagę przy modyfikacjach daty przekraczających granicę zmiany czasu.
- Przykładowo, jeżeli ustalamy spotkanie na następny tydzień, nie należy dodawać siedmiu dni:

```
ZonedDateTime meeting = ZonedDateTime.of(
    LocalDate.of(2013, 10, 31),
    LocalTime.of(14, 30),
    ZoneId.of("America/Los_Angeles"));
// Uwaga! Nie zadziała
// przy sezonowej zmianie czasu
ZonedDateTime nextMeeting =
    meeting.plus(Duration.ofDays(7));
```

API daty i czasu

Czas strefowy

- Zamiast tego należy użyć klasy `Period`:

```
ZonedDateTime nextMeeting =
    meeting.plus(Period.ofDays(7)); // OK
```

- Istnieje też klasa `OffsetDateTime`, reprezentująca czas z przesunięciem w stosunku do UTC, ale bez reguł związanych ze strefami czasowymi.
- Ta klasa jest przeznaczona do specjalnych zastosowań, które rzeczywiście wymagają braku tych reguł, takich jak niektóre protokoły sieciowe.
- Do opisu czasu dla ludzi należy wykorzystywać klasę `ZonedDateTime`.

API daty i czasu

Przykład (Czas strefowy)

- Program `r12/r12_05/ZonedTimes.java`

API daty i czasu

Formatowanie i przetwarzanie

- Klasa `DateTimeFormatter` udostępnia trzy rodzaje formaterów do wyświetlania wartości czasu i daty:
  - predefiniowane standardowe formatery,
  - formatery specyficzne dla bieżącej lokalizacji,
  - formatery dostosowanymi wzorcami.
- Aby wykorzystać jeden ze standardowych formaterów, należy wywołać jego metodę `format`:

```
// 1969-07-16T09:32:00-05:00[America/New_York]
String sformatowany =
    DateTimeFormatter.ISO_DATE_TIME.format(
        startApollo11
    );
```

API daty i czasu

Formatowanie i przetwarzanie

- Standardowe formatery są stosowane do oznaczania czasu odczytywanego przez automaty.
- Aby wyświetlać daty i czas ludziom, należy używać formaterów specyficznych dla lokalizacji.
- Istnieją cztery style: `SHORT`, `MEDIUM`, `LONG` i `FULL` – zarówno dla daty, jak i czasu.

API daty i czasu

Formatowanie i przetwarzanie – predefiniowane formatery

- `BASIC_ISO_DATE`: rok, miesiąc, dzień, przesunięcie strefy czasowej bez znaków rozdzielających
  `19690716-0500`
- `ISO_LOCAL_DATE`, `ISO_LOCAL_TIME`, `ISO_LOCAL_DATE_TIME`: separatory `-` `:` `T`
  `1969-07-16, 09:32:00, 1969-07-16T09:32:00`
- `ISO_OFFSET_DATE`, `ISO_OFFSET_TIME`, `ISO_OFFSET_DATE_TIME`: jak `ISO_LOCAL_XXX`, ale z przesunięciem strefy czasowej
  `1969-07-16-05:00, 09:32:00-05:00, 1969-07-16T09:32:00-05:00`
- `ISO_ZONED_DATE_TIME`: z przesunięciem strefy czasowej i identyfikatorem strefy
  `1969-07-16T09:32:00-05:00[America/New_York]`

API daty i czasu

Formatowanie i przetwarzanie

- `ISO_INSTANT`: w UTC oznaczanym identyfikatorem strefy Z
  `1969-07-16T14:32:00Z`
- `ISO_DATE`, `ISO_TIME`, `ISO_DATE_TIME`: jak `ISO_OFFSET_DATE`, `ISO_OFFSET_TIME` i `ISO_ZONED_DATE_TIME`, (informacja o strefie jest opcjonalna)
  `1969-07-16-05:00, 09:32:00-05:00, 1969-07-16T09:32:00-05:00[America/New_York]`
- `ISO_ORDINAL_DATE`: rok i dzień roku dla `LocalDate`
  `1969-197`
- `ISO_WEEK_DATE`: rok, tydzień i dzień tygodnia dla `LocalDate`
  `1969-W29-3`

API daty i czasu

Formatowanie i przetwarzanie

- `RFC_1123_DATE_TIME`: standard dla znaczników czasu w e-mailach, opisany w RFC 822 i zaktualizowany do czterech cyfr dla roku w RFC
  `1123 Wed, 16 Jul 1969 09:32:00 -0500`

Specyficzne dla lokalizacji style formatowania

Styl	Data	Czas
SHORT	7/16/69	9:32 AM
MEDIUM	Jul 16, 1969	9:32:00 AM
LONG	July 16, 1969	9:32:00 AM EDT
FULL	Wednesday, July 16, 1969	9:32:00 AM EDT

API daty i czasu

Formatowanie i przetwarzanie

- Takie formatery tworzą metody statyczne `ofLocalizedDate`, `ofLocalizedTime` i `ofLocalizedDateTime`. Na przykład:

```
DateTimeFormatter formatter = DateTimeFormatter
    .ofLocalizedDateTime(FormatStyle.LONG);
String formatted =
    formatter.format(startApollo11);
// July 16, 1969 9:32:00 AM EDT
```

- Te metody korzystają z domyślnych ustawień lokalizacji. Aby zmienić lokalizację, należy użyć metody `withLocale`:

```
formatted = formatter.withLocale(Locale.FRENCH)
    .format(startApollo11);
// 16 juillet 1969 09:32:00 EDT
```

API daty i czasu

Formatowanie i przetwarzanie

- Wyliczenia `DayOfWeek` oraz `Month` mają metody `getDisplayName`, zwracające nazwy dni tygodnia i miesięcy w różnych lokalizacjach i formatach.

```
for (DayOfWeek w : DayOfWeek.values()) {
    System.out.print(w.getDisplayName(
        TextStyle.SHORT, Locale.ENGLISH) + " ");
}
// Wyświetla Mon Tue Wed Thu Fri Sat Sun
```

- Klasa `java.time.format.DateTimeFormatter` zastępuje `java.util.DateFormat`.
- Jeżeli potrzebujemy instancji tej drugiej klasy dla zachowania kompatybilności wstecznej, należy zastosować wywołanie `formatter.toFormat()`.

API daty i czasu

Formatowanie i przetwarzanie

- Można utworzyć swój własny format daty, określając wzorzec. Przykładowo,

```
formatter = DateTimeFormatter
    .ofPattern("E yyyy-MM-dd HH:mm");
```

  formatuje datę w postaci `Wed 1969-07-16 09:32`.
- Każda litera opisuje inne pole czasu, a liczba powtórzeń litery wybiera konkretny format zgodnie ze specyficznymi regułami wypracowanymi w praktyce.
- Tabela na kolejnym slajdzie pokazuje najbardziej użyteczne elementy wzorców.

API daty i czasu

Wybrane symbole formatujące dla formatów daty i czasu

Opis	**Pole czasu**	**Przykłady**
Era	ERA	G: AD, GGGG: Anno Domini, GGGGG: A
Rok	YEAR_OF_ERA	yy: 69, yyyy: 1969
Miesiąc	MONTH_OF_YEAR	M: 7, MM: 07, MMM: Jul, MMMM: July, MMMMM: J
Dzień	DAY_OF_MONTH	d: 6, dd: 06
Dzień tygodnia	DAY_OF_WEEK	e: 3, E: Wed, EEEE: Wednesday, EEEEE: W
Godzina	HOUR_OF_DAY	H: 9, HH: 09
Godzina zegarowa	CLOCK_HOUR_OF_AM_PM	K: 9, HH: 09
Pora dnia	AMPM_OF_DAY	a: AM
Minuta	MINUTE_OF_HOUR	mm: 02
Sekunda	SECOND_OF_MINUTE	ss: 00
Nanosekunda	NANO_OF_SECOND	nnnnnn: 000000

API daty i czasu

Wybrane symbole formatujące dla formatów daty i czasu

Opis	**Pole czasu**	**Przykłady**
Identyfikator strefy czasowej		W: America/New_York
Nazwa strefy czasowej		z: EDT, zzzz: Eastern Daylight Time
Przesunięcie czasu strefy		x: -04, xx: -0400, xxx: -04:00, XXX : tak samo, ale z Z zamiast zera
Przesunięcie czasu strefy z lokalizacją		0: GMT-4, 0000: GMT-04:00 0: GMT-4, 0000: GMT-04:00

API daty i czasu

Formatowanie i przetwarzanie

- Aby przetworzyć wartości opisujące datę i czas z ciągu znaków, nalezy wykorzystać jedną ze statycznych metod `parse`. Na przykład:

```
LocalDate otwarcieKościoła =
    LocalDate.parse("1903-06-14");
ZonedDateTime startApollo11 =
    ZonedDateTime.parse("1969-07-16 03:32:00-0400",
        DateTimeFormatter
            .ofPattern("yyyy-MM-dd HH:mm:ssxx"));
```

- Pierwsze wywołanie wykorzystuje standardowy formater `ISO_LOCAL_DATE`, drugie – dostosowany formater.

API daty i czasu

Przykład (Formatowanie i przetwarzanie)

- Program `r12/r12_06/Formatting.java`

API daty i czasu

Współpraca z przestarzałym kodem

- Ponieważ API obsługujące datę i czas w języku Java jest nowością, będzie musiało współpracować z istniejącymi klasami, w szczególności ze wszechobecnymi `java.util.Date`, `java.util.GregorianCalendar` oraz `java.sql.Date/Time/Timestamp`.
- Klasa `Instant` jest bliskim odpowiednikiem `java.util.Date`.
- W Java 8 klasa ta ma dodane dwie metody: `toInstant`, konwertującą `Date` na `Instant`, i statyczną metodę `from`, wykonującą konwersję w przeciwną stronę.
- Podobnie `ZonedDateTime` jest bliskim odpowiednikiem `java.util.GregorianCalendar` i ta klasa również zyskała metody do konwersji w Java 8.

API daty i czasu

Współpraca z przestarzałym kodem

- Metoda `toZonedDateTime` konwertuje `GregorianCalendar` do `ZonedDateTime`, a statyczna metoda `from` wykonuje konwersję w przeciwną stronę.
- Inny zestaw konwersji jest dostępny dla klas opisujących datę i czas w pakiecie `java.sql`.
- Można też przekazać `DateTimeFormatter` do przestarzałego kodu korzystającego z `java.text.Format`.
- Tabela na następnym slajdzie podsumowuje te konwersje.

API daty czasu – współpraca z przestarzałym kodem

`Instant` ↔ `java.util.Date`	`Date.from(instant)`	`date.toInstant()`
`ZonedDateTime` ↔ `java.util.GregorianCalendar`	`GregorianCalendar.from(zonedDateTime)`	`cal.toZonedDateTime()`
`Instant` ↔ `java.sql.Timestamp`	`TimeStamp.from(instant)`	`timeStamp.toInstant()`
`LocalDateTime` ↔ `java.sql.Timestamp`	`Timestamp.valueOf(localDateTime)`	`timeStamp.toLocalDateTime()`
`LocalDate` ↔ `java.sql.Date`	`Date.valueOf(localDate)`	`date.toLocalDate()`
`LocalTime` ↔ `java.sql.Time`	`Time.valueOf(localTime)`	`time.toLocalTime()`
`DateTimeFormatter` → `java.text.DateFormat`	`formatter.toFormat()`	Brak
`java.util.TimeZone` ↔ `ZoneId`	`Timezone.getTimeZOne(id)`	`timeZone.toZoneId()`
`java.nio.file.attribute.FileTime` ↔ `Instant`	`FileTime.from(instant)`	`fileTime.toInstant()`

API daty i czasu

Przykład (Współpraca z przestarzałym kodem)

- Program `r12/r12_07/Legacy.java`