

# Przetwarzanie struktur danych Java Standard Edition



# Cześć!

Anna Skulimowska

od zawsze lubię dzielić się wiedzą

[annamskulimowska@gmail.com](mailto:annamskulimowska@gmail.com)

# Powtórka

- Co to jest klasa?
- Co to jest obiekt?
- Co oznacza słowo `this`?
- Co oznacza słowo `static` w zależności od tego w którym miejscu występuje?
- Przy pomocy jakiej metody porównujemy obiekty?
- Jakiej klasy zazwyczaj używamy do precyzyjnego przechowywania liczb (np. salda rachunku)?
- Do czego służą pakiety?

# Powtórka

- Czym jest hermetyzacja?
- Do czego służą gettery i settery? Po co ich używamy?
- Czym jest enum?
- Czym się różnią klasy immutable i mutable?
- Do czego służy rzutowanie? Co się stanie jak użyjemy go w niewłaściwy sposób?
- Czym jest String Pool?
- Jakie mamy instrukcje sterujące?
- Jakie mamy pętle w Javie?

# Powtórka

- Do czego używamy wyjątków?
- Do czego służy try-with-resources?
- Do czego używamy varargs?
- Czym jest dziedziczenie?
- Do czego służą interfejsy?

# Dlaczego struktury danych?

Łatwiej jest korzystać z danych, jeśli przechowujemy je w odpowiedni sposób

# Agenda

- 1.** Czym są struktury danych
- 2.** Czym jest przetwarzanie danych
- 3.** POJO
- 4.** Tablice
- 5.** Typy generyczne
- 6.** Kolekcje
- 7.** Listy
- 8.** Komparatory

# Agenda

- 9.** Iteratory
- 10.** Sety
- 11.** Mapy
- 12.** Kolejki
- 13.** Czas - daty
- 14.** Properties
- 15.** Locale

# Struktura danych

**Struktura danych** - sposób przechowywania danych w pamięci komputera

Przykładowo: rekord, tablica, lista itp.

# Przetwarzanie danych

**Przetwarzanie danych** - przetwarzanie danych wejściowych poprzez wykonywanie różnych operacji w celu otrzymania wyniku

# Algorytm

**Algorytm** - skończony ciąg czynności koniecznych do rozwiązania problemu (ogólny opis jak rozwiązać problem). Algorytmy można implementować w programach.

Prosty przykład algorytmu to przepis kulinarny.

# POJO

Klasa jest POJO jeśli:

- nie zawiera zależności do bibliotek/frameworków innych niż Java SE
- wyjątkiem są adnotacje

<https://spring.io/understanding/POJO>

# POJO

## Plain Old Java Object

```
public class Main
```

POJO

```
public class MyBigDecimal extends BigDecimal
```

POJO

```
public class MyBigDecimal implements  
MyInterface
```

POJO  
(jeśli  
MyInterface  
jest w  
projektie)

# POJO

## Plain Old Java Object

```
public class MyServlet extends HttpServlet
```

-

```
@Entity
```

```
public class MyEntity
```

POJO

# Tablice



# Tablica

- struktura danych gromadząca uporządkowane dane
- można tworzyć jednowymiarowe lub wielowymiarowe
- mają stałą wielkość
- odwołujemy się do elementu po indeksie  
numerujemy od 0

```
int[] array = new int[10];      // wypełniona zerami
int[] array2 = {1, 2, 3, 4, 5}; // od razu zainicjalizowana danymi

int number = array2[2];        // 3
```

# Pętla for each

Po tablicy można “przejść” używając pętli:

```
for (int i : array) {  
}
```

# Klasa Arrays

Klasa Arrays ma różne przydatne metody do operowania na tablicach, przykładowo:

- `toString` - zamienia tablicę na czytelny ciąg znaków
- `copyOf` - kopiuje zawartość tablicy
- `sort` - sortuje zawartość tablicy
- `asList` - przekształca tablicę na listę
- `equals` - porównuje zawartość tablic
- i inne...

# Kiedy używamy tablic?

- z góry znamy rozmiar tablicy i wiemy, że się nie zmieni
- pracujemy z typami prostymi (np. wymuszona przez projekt, API bibliotek itp.)
- varargs
- tworzymy kod krytyczny pod względem szybkości działania lub pamięci  
(rzadko to dobry powód, temat zaawansowany)

# Zadanie 1



W klasie `ArrayTask`:

- 1.** Stwórz 5-cio elementową tablicę intów.
- 2.** Wypisz zawartość tablicy.
- 3.** Stwórz drugą tablicę, zawierającą elementy pierwszej tablicy, ale o dwukrotnie większym rozmiarze.
- 4.** Wypisz zawartość drugiej tablicy.
- 5.** Posortuj drugą tablicę.
- 6.** Wypisz zawartość posortowanej tablicy.

# Typy generyczne (ogólne)



# Typy generyczne

Co można przechowywać w słoiku?

Czy nasypiesz cukru do pojemnika z napisem “sól”?

# Programowanie generyczne

Pisanie kodu, który może być używany z różnymi typami.  
Kompilator sprawdza poprawność typów.

Przykładowo klasa `ArrayList` pozwala gromadzić obiekty typu `String`, ale również inne jak `BigDecimal` itp.

```
List<String> strings = new ArrayList<>();
```

# Klasy ogólne (parametryzowane)

Można tworzyć własne klasy parametryzowane:

```
public class MyClass<T> {
    private T field;

    public T getField() {
        return field;
    }
}
```

# Klasy ogólne (parametryzowane)

Można również zdefiniować ograniczenia:

```
public class MyClass<T extends Number>
```

# Klasy ogólne (parametryzowane)

Metoda ogólna w klasie, która nie jest parametryzowana:

```
public class MyClass {  
  
    public <T> T method(T parameter) {  
        // do something  
        return parameter;  
    }  
}
```

## Zadanie 2



- 1.** Stwórz nową klasę Point
  - a.** z generycznym parametrem `E` ograniczonym do klas dziedziczących po `Number`
  - b.** z dwoma polami `x, y` o typie `E` oznaczającymi współrzędne punktu (o typie `E`)
  - c.** odpowiednim konstruktorem i getterami
- 2.** Przetestuj działanie klasy `Point` w klasie `GenericsTask` na przynajmniej dwóch różnych typach

# Kolekcje



# Collections Framework

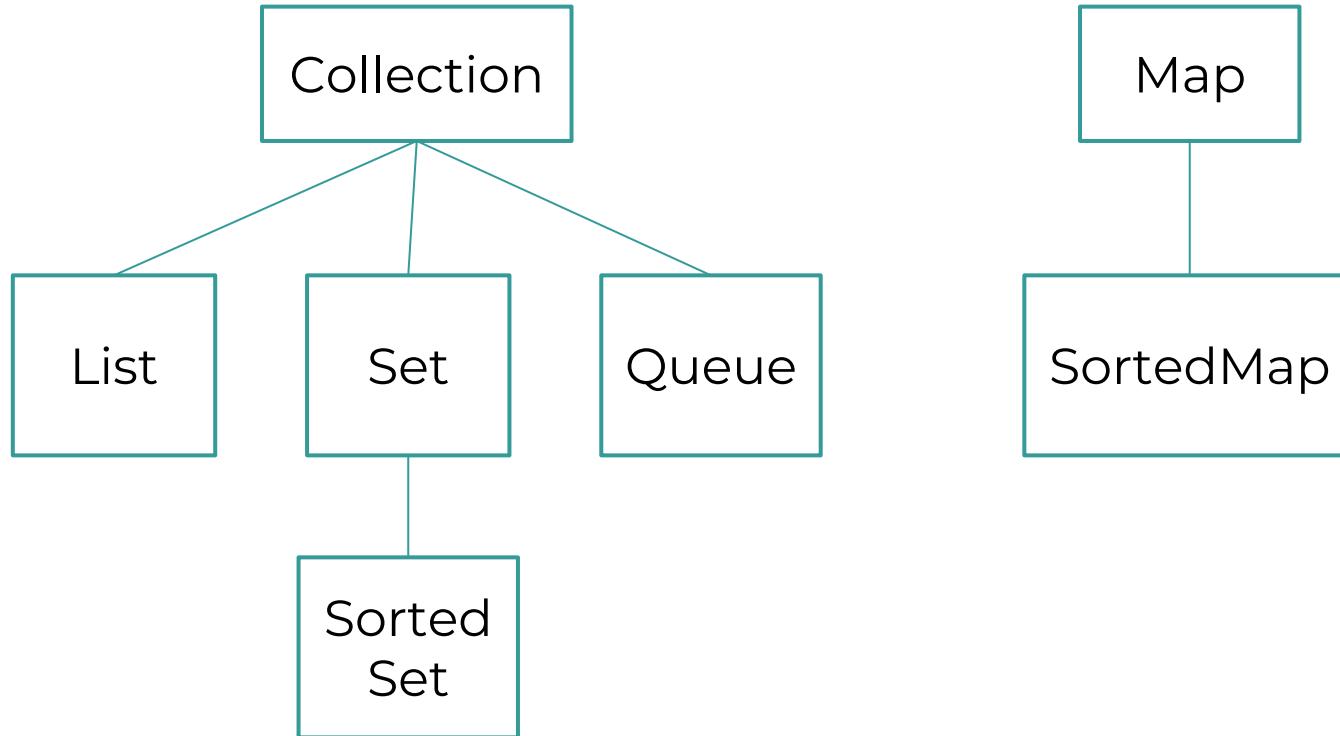
Kolekcje grupują obiekty w kontenery.

Collections Framework składa się z:

- interfejsów
- implementacji
- algorytmów

[http://files.zeroturnaround.com/pdf/zt\\_java\\_collections\\_cheat\\_sheet.pdf](http://files.zeroturnaround.com/pdf/zt_java_collections_cheat_sheet.pdf)

# Interfejsy kolekcji



# Interfejs Collection

- add - dodaje element do kolekcji
- addAll - dodaje wszystkie elementy kolekcji
- clear - usuwa zawartość kolekcji
- contains - sprawdza czy kolekcja zawiera dany element
- isEmpty - sprawdza czy kolekcja jest pusta
- iterator - zwraca obiekt umożliwiający przechodzenie po kolekcji
- remove - usuwa dany element kolekcji (jeśli istnieje)
- size - zwraca rozmiar kolekcji
- toArray - przekształca kolekcję w tablicę
- i inne...

# Klasa Collections

Klasa ze statycznymi, pomocniczymi metodami:

- sort - sortuje listę
- frequency - zlicza liczbę wystąpień elementu w kolekcji
- max - zwraca największy element kolekcji
- min - zwraca najmniejszy element kolekcji
- reverse - odwraca kolejność listy
- shuffle - losowo miesza elementy listy
- swap - zamienia elementy listy o podanych indeksach
- i inne...

# Listy (Lists)



# Listy

- zachowują kolejność
- pozwalają na duplikaty elementów
- do elementów można odwoływać się po indeksie

Implementacje:

- ArrayList
- LinkedList

# Listy

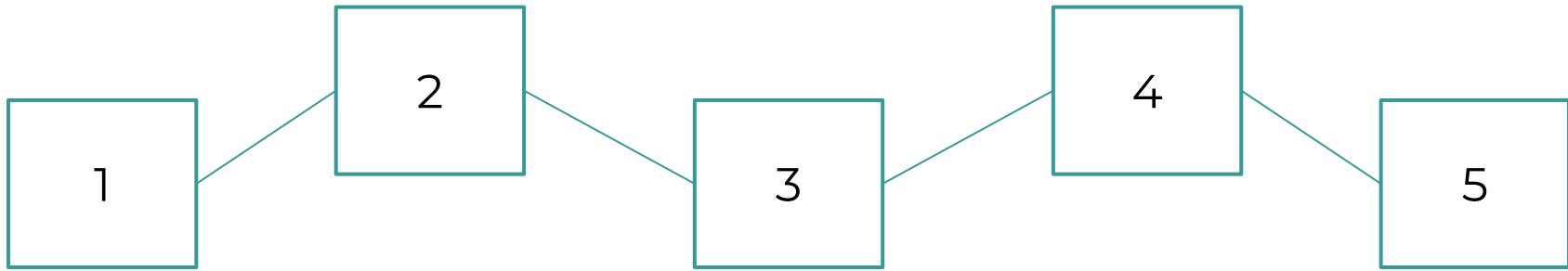
- `get` - zwraca element o danym indeksie
- `indexOf` - zwraca indeks danego elementu
- `set` - zamienia element o danym indeksie na podany w parametrze
- `sort` - sortuje listę
- i inne...

# ArrayList



- szybki dostęp do elementu jeśli znamy indeks
- wolne usuwanie elementów z początku i środka
- łatwe dodawanie elementu na koniec

# LinkedList



- szybkie dodawanie elementu na początku listy
- szybkie usuwanie elementu ze środka
- wolny dostęp do elementu o danym indeksie

## Zadanie 3



- 1.** W pakiecie lists stwórz nową klasę Person zawierającą imię i nazwisko osoby.
- 2.** W klasie ListTask stwórz listę kilku osób i wypisz ją na konsolę.

# ArrayList vs LinkedList



W jakim przypadku powinniśmy użyć ArrayList,  
a w jakim LinkedList?

# Comparator

Interfejs z parametrem generycznym służący do porównywania obiektów w kolekcjach

Deklarując własny Comparator implementujemy metodę  
compareTo

## Zadanie 4



1. Stwórz PersonComparator sortujący osoby po nazwisku.
2. W klasie ListTask posortuj listę osób używając Collections.sort.

# Iterator

Interfejs z parametrem generycznym służący do definiowania kolejności przetwarzania kolekcji

Ma trzy metody:

- next - zwraca kolejny element
- hasNext - zwraca true jeśli istnieje następny element
- remove - usuwa bieżący element

## Zadanie 5



W klasie ListTask spróbuj usunąć wszystkie osoby z listy używając pętli for each i zobacz co się stanie.

## Zadanie 6



Zrealizuj poprzednie zadanie używając iteratora.

# Zbiory (Sets)



# Zbiory

- elementy zbioru są unikalne
- zachowanie kolejności zależy od implementacji
- umożliwia szybkie sprawdzenie czy dany element znajduje się w zbiorze (metoda `contains`)

# HashSet

Nie gwarantuje kolejności.

```
HashSet<String> backpack = new HashSet<>();  
backpack.add("Jabłko");  
backpack.add("Długopis");
```

## Zadanie 7



W klasie SetTask przetestuj działanie HashSet na obiektach typu Person.

Upewnij się, że zbiór prawidłowo działa w przypadku duplikatów.

# TreeSet

Gwarantuje zachowanie kolejności (wg porządku wyznaczonego przez implementacje interfejsu Comparable lub przekazanego w konstruktorze komparatora).

```
TreeSet<Integer> integers = new TreeSet<>();  
integers.add(3);  
integers.add(2);  
integers.add(4);
```

# LinkedHashSet

Gwarantuje zachowanie kolejności wg dodawania elementów.

```
Set<Integer> set = new LinkedHashSet<>();  
set.add(3);  
set.add(5);  
set.add(7);
```

## Zadanie 8



W klasie SetTask przetestuj działanie TreeSet i LinkedHashSet.

# HashSet vs TreeSet vs LinkedHashSet



W jakim przypadku powinniśmy użyć HashSet,  
TreeSet a w jakim LinkedHashSet?

# Mapy (Maps)



# Mapy

- struktura danych reprezentująca mapę (słownik)
- przechowuje pary klucz-wartość
- do wartości odwołujemy się poprzez klucze
- klucze są unikalne i są obiektami

```
HashMap<Integer, String> hashMap = new HashMap<>();  
hashMap.put(1, "Jeden");  
hashMap.put(2, "Dwa");  
  
String text = hashMap.get(1); // Jeden
```

# Mapy

- containsKey - sprawdza czy w mapie jest element o danym kluczu (szybko działa)
- containsValue - sprawdza czy w mapie jest element o danej wartości (wolno działa)
- put - dodaje pary klucz-wartość do mapy
- get - zwraca wartość dla danego klucza
- keySet - zwraca zbiór kluczy
- entrySet - zwraca zbiór par klucz-wartość
- i inne...

# Mapy

- struktura danych reprezentująca mapę (słownik)
- przechowuje pary klucz-wartość
- do wartości odwołujemy się poprzez klucze
- klucze są unikalne i są obiektami

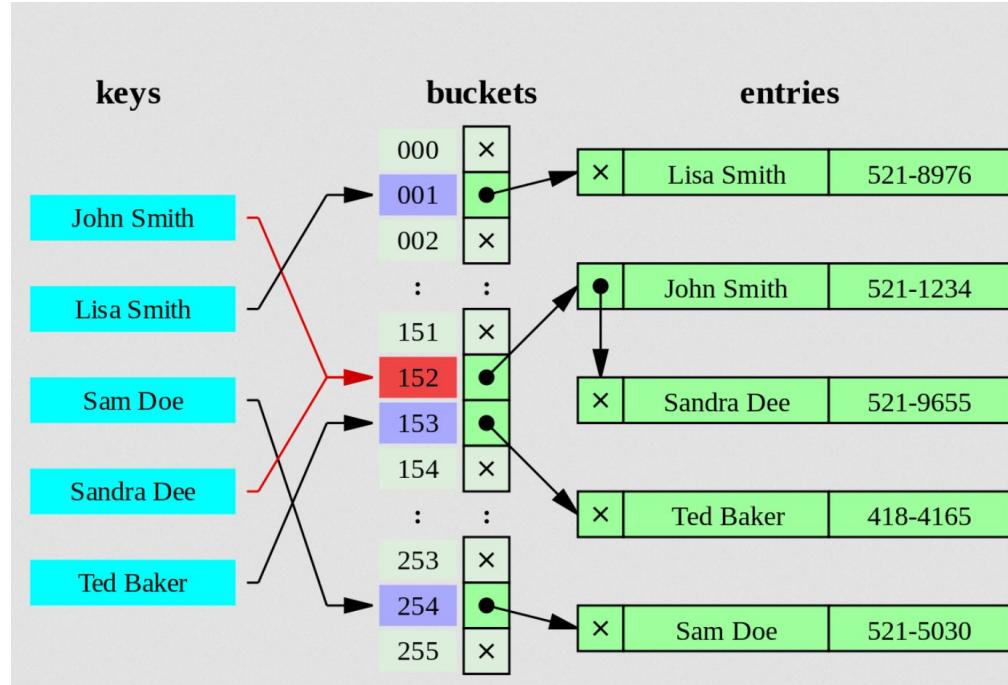
```
HashMap<Integer, String> hashMap = new HashMap<>();  
hashMap.put(1, "Jeden");  
hashMap.put(2, "Dwa");  
  
String text = hashMap.get(1); // Jeden
```

# HashMap

- szybko wyszukuje obiekty po kluczu
- używa tablic hashujących

```
HashMap<Integer, String> hashMap = new HashMap<>();  
hashMap.put(1, "Jeden");  
hashMap.put(2, "Dwa");  
  
String text = hashMap.get(1); // Jeden
```

# Tablice hashujące



By Jorge Stolfi - Own work, CC BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=6471915>

## Zadanie 9



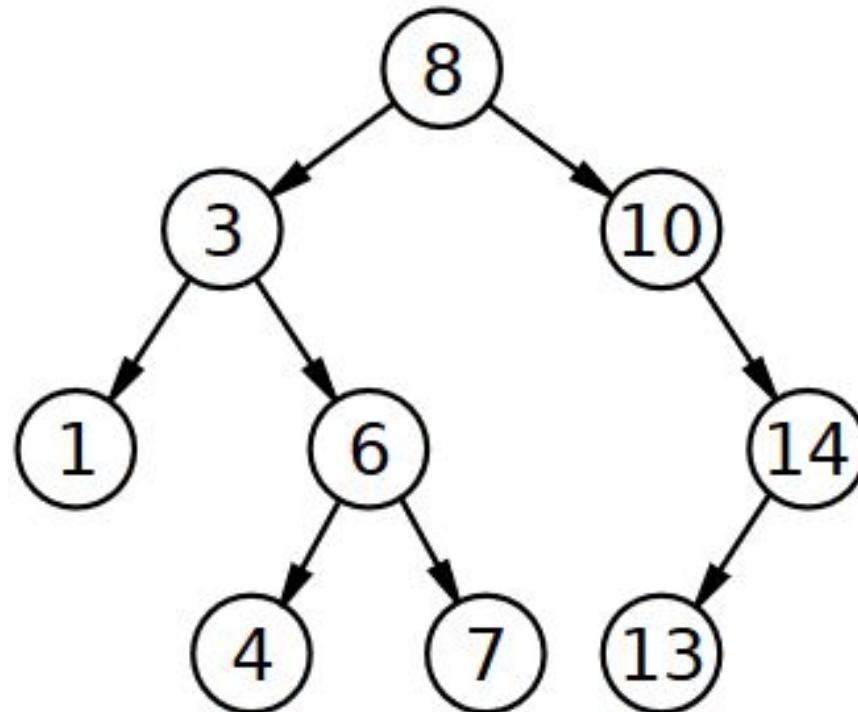
- 1.** W klasie `HashMapTask` dodaj do HashMapy kilka osób, załącz, że kluczem jest login. Dodaj przynajmniej dwie osoby o tym samym loginie.
- 2.** Pobierz jedną osobę używając jej loginu i wypisz jej dane.
- 3.** Użyj metody `entrySet` i iterując po kolekcji wypisz loginy oraz dane osób.

# TreeMap

- zachowuje porządek kluczy
- wykorzystuje drzewa binarne (czerwono-czarne)

```
TreeMap<Integer, Integer> map = new TreeMap<>();  
map.put(1, 2);  
map.put(2, 3);
```

# Drzewo binarne



# HashMap vs TreeMap



W jakim przypadku powinniśmy użyć HashMap, a w jakim TreeMap?

# Kolejki (Queues)



# PriorityQueue

- przyjmuje elementy nieuporządkowane, a zwraca uporządkowane (o najmniejszej wartości)
- nie gwarantuje przechowywania elementów w kolejności

```
PriorityQueue<Integer> queue = new PriorityQueue<>();  
queue.add(8);  
queue.add(4);  
queue.add(10);  
  
Integer leastElement = queue.peek(); // 4
```

# PriorityQueue

- peek - zwraca najmniejszy element
- poll - zwraca i usuwa najmniejszy element
- add - dodaje element do kolejki
- i inne...

## Zadanie 10



W klasie QueueTask przetestuj działanie metod kolejki priorytetowej: add, peek i poll.

# Czas



# java.util.Date

- dla < Java 1.8
- przechowuje datę i czas
- metody:
  - ✓ `setTime` - ustawia czas (parametr to milisekundy od 1 stycznia 1970)
  - ✓ `compareTo` - porównywanie dat
  - ✓ `before/after` - sprawdza czy data jest wcześniejsza/późniejsza od parametru
  - ✓ i inne...

# SimpleDateFormat

- klasa do formatowania `java.util.Date`
- konstruktor może przyjmować Pattern
- metody:
  - ✓ `format` - zwraca sformatowany tekst
  - ✓ `parse` - przekształca tekst na datę
  - ✓ i inne...

<https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html>

# Calendar

- klasa reprezentująca kalendarz
- pozwala manipulować czasem

```
Calendar now = Calendar.getInstance();
now.add(Calendar.MONTH, amount: 1);
Date nextMonth = now.getTime();
```

## Zadanie 11



W klasie `DateTask` wypisz na konsolę sformatowany obiekt daty, który wskazuję na 2 tygodnie i 3 dni do przodu od dzisiaj. Ustaw godzinę 11:23.

Wybierz czytelny dla użytkownika format daty.

# java.util.Date

- dla < Java 1.8
- klasa problematyczna przy współbieżności
- mało praktyczna w codziennych operacjach
- sporo metod oznaczonych jako przestarzałe
- niełatwwe operowanie na strefach czasowych
- często zastępowane biblioteką Joda Time  
<https://www.joda.org/joda-time/>

# java.util.Date

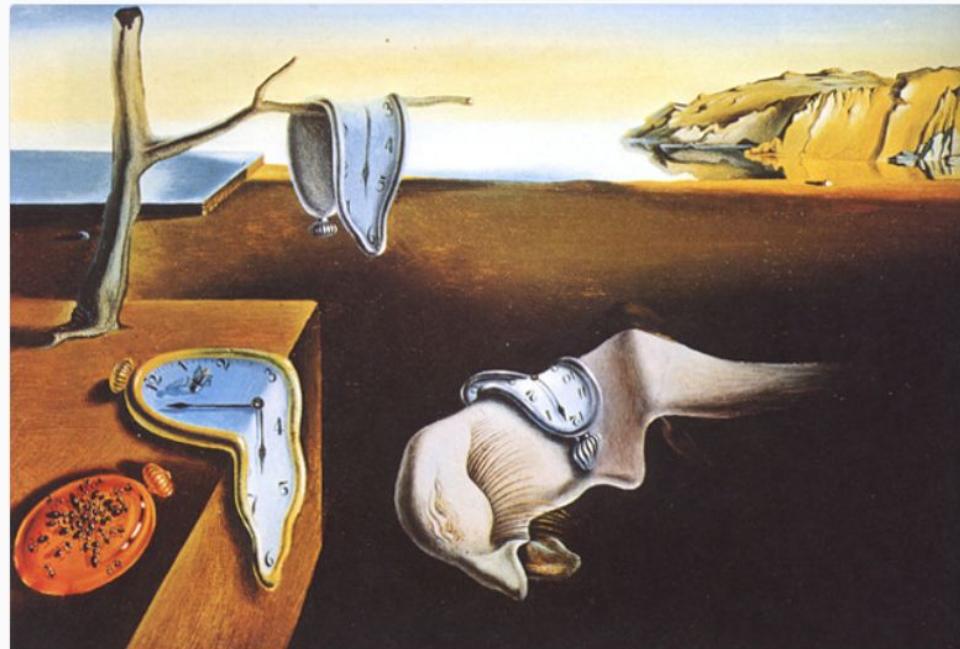
```
Date date = new Date();
```

```
m  notify()                      void
m  notifyAll()                    void
m  wait()                        void
m  wait(long timeout)           void
m  wait(long timeout, int nanos) void
m  getDate()                     int
m  getDay()                      int
m  getHours()                    int
m  getMinutes()                  int
m  getMonth()                    int
m  getSeconds()                  int
```

Use Ctrl+Shift+Enter to syntactically correct your code after completing (balance parentheses etc.) >> 

```
date.
```

# java.util.Date



"java.util.Date"

Salvador Dali

Oil on canvas, 1931

494 notes

... ↗ ❤

<http://classicprogrammerpaintings.com>

# LocalDate/LocalDateTime

- dla Java 1.8+
- nowe, wygodniejsze API Javy
- więcej na zajęciach z Java 8+...

```
LocalDate now = LocalDate.now();  
LocalDate tommorow = now.plusDays(1);
```

# Properties



# Properties

- przechowuje pary klucz-wartość (tylko Stringi)
- unikalne klucze
- przydatna do przechowywania ustawień aplikacji
- można podać dodatkowe, domyślne Properties

```
Properties properties = new Properties();
properties.setProperty("key", "value");
String value = properties.getProperty("key");
```

## Zadanie 12



- 1.** Stwórz plik config.properties w resources, dodaj do niego kilka par klucz-wartość rozdzielone znakiem =
- 2.** W klasie PropertiesTask wczytaj plik (użyj metody load z obiektu Properties oraz klasy FileInputStream)
- 3.** Wypisz na konsolę wszystkie klucze z pliku
- 4.** Wypisz na konsolę wartość dla jednego z kluczy

# Locale



# Locale

- klasa obsługująca lokalizację geograficzną użytkownika
- często potrzebna do prawidłowego wyświetlania numerów (np. w Polsce z przecinkami, w USA z kropkami), dat
- dzięki niej wiadomo jaki dzień tygodnia jest pierwszy, jaki jest kod kraju, języka itp.

```
Locale constantLocale = Locale.JAPAN;  
Locale createdLocale = new Locale( language: "pl", country: "PL");
```

# Java Koans

Seria ćwiczeń służąca do nauki Javy:

<https://github.com/infoshareacademy/java-koans>

Pobierz projekt i na swojej gałęzi uzupełnij wskazane w terminalu koany.

# Bonus



# Tablice wielowymiarowe

W Javie możemy deklarować tablice wielowymiarowe, np:

```
int[][][] array2d = new int[10][10];  
  
int[][][] differentOne = {  
    {1, 2, 3},  
    {4, 5, 6, 9},  
    {7},  
};
```

## Zadanie 13



- 1.** Stwórz listę liczb.
- 2.** Przy użyciu zbioru sprawdź czy lista zawiera duplikaty, jeśli tak to wypisz je na konsolę.

## Zadanie 14



- 1.** Stwórz listę liczb.
- 2.** Odwróć listę, przykładowo:

1, 2, 3 przekształć na 3, 2, 1

## Zadanie 15



- 1.** Stwórz listę Stringów.
- 2.** Odwróć wielkość liter w co drugim stringu.

## Zadanie 16



- 1.** Stwórz listę znaków.
- 2.** Sprawdź czy znaki w liście tworzą palindrom.



# Dzięki

## Pytania?

Zawsze możesz do mnie napisać:  
[annamkulimowska@gmail.com](mailto:annamkulimowska@gmail.com)