

Oppimispäiväkirja  
Ohjelmoinnin syventävät tekniikat

Antti Venetjoki

OPPIMISPÄIVÄKIRJA

Tammikuu 2025

SISÄLLYS

[1 Tehtävä 1 3](#_Toc190071187)

[2 Tehtävä 2 4](#_Toc190071188)

[3 Tehtävä 3 5](#_Toc190071189)

[4 Tehtävä 4 7](#_Toc190071190)

[4.1 Today.java 7](#_Toc190071191)

[4.2 TodayRelatable.java 10](#_Toc190071192)

[4.3 Event.java 10](#_Toc190071193)

[4.4 Category.java 15](#_Toc190071194)

# Tehtävä 1

import java.util.Scanner;

import java.util.Random;

public class guess\_random {

static int guess() {

Scanner scan = new Scanner(System.in);

System.out.println("Enter a guess (1 - 100): ");

scan.close();

return scan.nextInt();

}

public static void main(String[] args) {

Random rand = new Random();

int correct = rand.nextInt(100) + 1;

for (int i = 0; i < 7; i++) {

int guess = guess();

if (guess == correct) {

System.out.println("You guessed correctly!");

return;

}

}

System.out.println("You ran out of attempts.");

System.out.println("Correct answer was " + correct);

return;

}

}

# Tehtävä 2

import java.time.LocalDate;

public class Birthday {

public static void main(String[] args) {

String birthdateValue = System.getenv("BIRTHDATE");

LocalDate birthdate = LocalDate.parse(birthdateValue);

LocalDate today = LocalDate.now();

if (isSameDate(today, birthdate)) {

System.out.println("Happy birthday!");

}

long daysOld = today.toEpochDay() - birthdate.toEpochDay();

System.out.println(String.format("You are %d days old.", daysOld));

if (daysOld % 1000 == 0 && daysOld > 0) {

System.out.println("That's a nice round number!");

}

}

public static boolean isSameDate(LocalDate date1, LocalDate date2) {

return date1.getDayOfMonth() == date2.getDayOfMonth() &&

date1.getMonth() == date2.getMonth();

}

}

# Tehtävä 3

import java.time.LocalDate;

import java.time.format.TextStyle;

import java.util.Locale;

import java.util.Arrays;

public class EventTest {

public static void main(String[] args) {

Category appleMacOS = new Category("apple", "macos");

Event[] events = {

new Event(LocalDate.parse("2024-09-16"),"macOS 15 Sequioa released",appleMacOS),

new Event(LocalDate.parse("2023-09-26"),"macOS 14 Sonoma released",appleMacOS),

new Event(LocalDate.parse("2022-10-24"),"macOS 13 Ventura released",appleMacOS),

new Event(LocalDate.parse("2021-10-25"),"macOS 12 Monterey released",appleMacOS),

new Event(LocalDate.parse("2020-11-12"),"macOS 11 Big Sur released",appleMacOS),

};

for (Event event : events) {

String dayName = event

.getDate()

.getDayOfWeek()

.getDisplayName(

TextStyle.FULL,

Locale.getDefault()

);

String releaseName = event

.getDescription()

.substring(

0,

event.getDescription().length() - 9

);

System.out.print(releaseName + " was released on a " + dayName + "\n");

}

String[] osNames = new String[events.length];

for (int i = 0; i < events.length; i++) {

osNames[i] = events[i]

.getDescription()

.substring(

9,

events[i]

.getDescription()

.length() - 9

);

}

Arrays.sort(osNames);

System.out.printf("In aplhabetical order: %s\n", Arrays.toString(osNames));

}

}

# Tehtävä 4

## Today.java

import java.time.LocalDate;

import java.time.format.DateTimeParseException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class Today {

private List<Event> events;

public Today() {

this.events = new ArrayList<>();

}

public static void main(String[] args) {

if (args.length != 2) {

System.err.println("Usage: java Today --MM-DD primary/secondary");

System.exit(1);

}

LocalDate queryDate = parseDate(args[0]);

if (queryDate == null) {

System.err.println("Invalid date format. Use --MM-DD.");

System.exit(1);

}

Category queryCategory = Category.parse(args[1]);

if (queryCategory == null) {

System.err.println("Invalid category format. Use primary/secondary.");

System.exit(1);

}

Today app = new Today();

app.addEvents();

app.report(queryDate, queryCategory);

}

private static LocalDate parseDate(String dateStr) {

if (!dateStr.matches("--\\d{2}-\\d{2}")) {

return null;

}

try {

String[] parts = dateStr.substring(2).split("-");

int month = Integer.parseInt(parts[0]);

int day = Integer.parseInt(parts[1]);

if (month < 1 || month > 12) {

System.err.println("Error: Invalid month. Must be between 01 and 12.");

return null;

}

LocalDate tempDate = LocalDate.of(LocalDate.now().getYear(), month, 1);

int maxDays = tempDate.lengthOfMonth();

if (day < 1 || day > maxDays) {

System.err.println("Error: Invalid day for the given month. Must be between 01 and " + maxDays + ".");

return null;

}

return LocalDate.of(LocalDate.now().getYear(), month, day);

} catch (NumberFormatException e) {

System.err.println("Error: Invalid number format. Use --MM-DD.");

return null;

}

}

private void report(LocalDate queryDate, Category queryCategory) {

List<Event> filteredEvents = new ArrayList<>();

for (Event event : events) {

if (isSameDate(queryDate, event.getDate()) && event.getCategory().matches(queryCategory)) {

filteredEvents.add(event);

}

}

if (filteredEvents.isEmpty()) {

System.out.println("No events found for given criteria.");

return;

}

Collections.sort(filteredEvents, (e1, e2) -> e2.getDate().compareTo(e1.getDate()));

for (Event event : filteredEvents) {

System.out.println(event.getDate().getYear() + ": " + event.getDescription());

}

}

private boolean isSameDate(LocalDate queryDate, LocalDate eventDate) {

return queryDate.getMonth() == eventDate.getMonth() && queryDate.getDayOfMonth() == eventDate.getDayOfMonth();

}

private void addEvents() {

events.add(new Event(LocalDate.of(2023, 9, 19), "Java SE 21 released", new Category("oracle", "java")));

events.add(new Event(LocalDate.of(2022, 9, 20), "Java SE 19 released", new Category("oracle", "java")));

events.add(new Event(LocalDate.of(2021, 9, 14), "Java SE 17 released", new Category("oracle", "java")));

events.add(new Event(LocalDate.of(2020, 11, 12), "macOS 11 Big Sur released", new Category("apple", "macos")));

events.add(new Event(LocalDate.now(), "Test event for today", new Category("test", "test")));

}

}

## TodayRelatable.java

public interface TodayRelatable {

public enum Relation {

BEFORE\_TODAY, TODAY, AFTER\_TODAY

}

Relation getTodayRelation();

long getTodayDifference();

}

## Event.java

import java.time.LocalDate;

import java.time.temporal.ChronoUnit;

import java.util.Objects;

import java.util.Comparator;

/\*\*

\* Represents an event in history.

\*/

public class Event implements TodayRelatable, Comparable<Event> {

private LocalDate date;

private String description;

private Category category;

/\*\*

\* Constructs a new event.

\*

\* @param date the date of the event

\* @param description the description of the event

\* @param category the category of the event

\*

\* @see Category

\*/

public Event(LocalDate date, String description, Category category) {

this.date = date;

this.description = description;

this.category = category;

}

/\*\*

\* Gets the date of the event.

\*

\* @return the date

\*/

public LocalDate getDate() {

return this.date;

}

/\*\*

\* Gets the event description.

\*

\* @return the description

\*/

public String getDescription() {

return this.description;

}

/\*\*

\* Gets the category of the event.

\*

\* @return the category

\* @see Category

\*/

public Category getCategory() {

return this.category;

}

/\*\*

\* Returns a string representation of this event.

\*

\* @return the event as a string

\*/

@Override

public String toString() {

return String.format(

"%s: %s (%s)",

this.date, this.description, this.category);

}

/\*\*

\* Tests for equality with another event.

\*

\* @return true if events are equal, false otherwise

\*/

@Override

public boolean equals(Object o) {

// Identical references?

if (o == this) return true;

// Correct type and non-null?

if (!(o instanceof Event)) return false;

// Cast to our type:

Event that = (Event) o;

if (Objects.equals(this.date, that.date) &&

Objects.equals(this.description, that.description) &&

Objects.equals(this.category, that.category)) {

return true;

}

return false;

}

/\*\*

\* Returns a hash code for this event.

\*

\* @return hash code computed based on the fields

\*/

@Override

public int hashCode() {

return Objects.hash(this.date, this.description, this.category);

}

/\*\*

\* Gets the relation of this event with today.

\*

\* @return the relation

\* @see TodayRelatable.Relation

\*/

public Relation getTodayRelation() {

long days = getDays();

if (days > 0) {

return TodayRelatable.Relation.AFTER\_TODAY;

} else if (days < 0) { // end is before start

return TodayRelatable.Relation.BEFORE\_TODAY;

} else {

return TodayRelatable.Relation.TODAY;

}

}

// Helper method to get days between now and this event.

private long getDays() {

return ChronoUnit.DAYS.between(LocalDate.now(), this.date);

}

/\*\*

\* Returns the difference between today and this event

\* in days. The return value is always positive; use

\* getRelation() to determine the relative position.

\*

\* @return number of days between today and this event

\*/

public long getTodayDifference() {

return Math.abs(this.getDays());

}

/\* ==== java.lang.Comparable implementation ==== \*/

/\*\*

\* Compares this event to another.

\*

\* @return negative, zero, or positive

\* @see java.lang.Comparable

\*/

@Override

public int compareTo(Event other) {

int result = Objects.compare(

this.date,

other.date,

Comparator.naturalOrder());

if (result != 0) {

return result;

}

result = Objects.compare(

this.description,

other.description,

Comparator.naturalOrder());

if (result != 0) {

return result;

}

return Objects.compare(

this.category,

other.category,

Comparator.naturalOrder());

}

}

## Category.java

import java.util.Comparator;

import java.util.Objects;

/\*\*

\* The category of an event, with primary and secondary categories.

\*/

public class Category implements Comparable<Category> {

private String primary;

private String secondary;

/\*\*

\* Constructs a category with primary and secondary values.

\*

\* @param primary the primary category

\* @param secondary the secondary category

\*/

public Category(String primary, String secondary) {

this.primary = primary;

this.secondary = secondary;

}

/\*\*

\* Parse a category string in the format "primary"

\* or "primary/secondary" and make a category of them.

\* Folds the category parts to lower case.

\* Throws java.lang.IllegalArgumentException if the

\* string is of the wrong format.

\*

\* @param categoryString the string to parse

\* @return new category

\*/

public static Category parse(String categoryString) {

if (categoryString == null ||

categoryString.isEmpty() ||

categoryString.isBlank()) {

throw new IllegalArgumentException("invalid category string");

}

String[] categoryParts = categoryString.split("/");

String primary = categoryParts[0].toLowerCase();

String secondary = null;

if (categoryParts.length == 2) {

secondary = categoryParts[1].toLowerCase();

}

return new Category(primary, secondary);

}

/\*\*

\* Gets the primary category.

\*

\* @return primary

\*/

public String getPrimary() {

return this.primary;

}

/\*\*

\* Gets the secondary category.

\*

\* @return secondary

\*/

public String getSecondary() {

return this.secondary;

}

/\*\*

\* Returns a string representation of this category.

\*

\* @return category as string

\*/

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(this.primary);

if (this.secondary != null) {

sb.append("/");

sb.append(this.secondary);

}

return sb.toString();

}

/\*\*

\* Tests for equality with another category.

\*

\* @return true if categories are equal, false otherwise

\*/

@Override

public boolean equals(Object o) {

// Identical references?

if (o == this) return true;

// Correct type and non-null?

if (!(o instanceof Category)) return false;

// Cast to our type:

Category that = (Category) o;

if (this.primary.equals(that.primary) &&

this.secondary.equals(that.secondary)) {

return true;

}

return false;

}

public boolean matches(Category other) {

if (other == null) return false;

return this.primary.equals(other.primary) &&

(this.secondary == null || other.secondary == null || this.secondary.equals(other.secondary));

}

/\*\*

\* Returns a hash code for this category.

\*

\* @return hash code computed based on primary and secondary categories

\*/

@Override

public int hashCode() {

return Objects.hash(this.primary, this.secondary);

}

@Override

public int compareTo(Category other) {

int result = Objects.compare(

this.primary,

other.getPrimary(),

Comparator.naturalOrder());

if (result != 0) {

return result;

}

return Objects.compare(

this.secondary,

other.getSecondary(),

Comparator.naturalOrder());

}

}