

Oppimispäiväkirja  
Ohjelmoinnin syventävät tekniikat

Antti Venetjoki

OPPIMISPÄIVÄKIRJA

Tammikuu 2024

SISÄLLYS

[1 Tehtävä 1 3](#_Toc158246131)

[2 Tehtävä 2 4](#_Toc158246132)

[3 Tehtävä 3 5](#_Toc158246133)

[3.1 TimeOnly.java 5](#_Toc158246134)

[3.2 TestTimeOnly.java 6](#_Toc158246135)

[4 Tehtävä 4 7](#_Toc158246136)

[4.1 TimeOnly.java 7](#_Toc158246137)

[4.2 DateOnly.java 8](#_Toc158246138)

[4.3 Timestamp.java 10](#_Toc158246139)

[4.4 TestTimestamp.java 13](#_Toc158246140)

[5 Tehtävä 6 18](#_Toc158246141)

[5.1 Category.java 18](#_Toc158246142)

[5.2 Description.java 19](#_Toc158246143)

[5.3 Event.java 20](#_Toc158246144)

[5.4 TestEvenet.java 22](#_Toc158246145)

[6 Tehtava 7 24](#_Toc158246146)

[7 Tehtava 8 25](#_Toc158246147)

[7.1 Event.java 25](#_Toc158246148)

[7.2 EventManager.java 26](#_Toc158246149)

[7.3 FXListView.java 27](#_Toc158246150)

# Tehtävä 1

import java.util.Random;

public class generate\_random {

public static void main(String[] args) {

Random rand = new Random();

int rand\_int = rand.nextInt(99) + 1;

System.out.printf("Luku, jota ajattelen on " + rand\_int + '\n');

}

}

# Tehtävä 2

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): ");

return scan.nextInt();

}

public static void main(String[] args) {

Random rand = new Random();

int correct = rand.nextInt(99) + 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 3

## TimeOnly.java

public class TimeOnly {

private int hour;

private int minute;

private int second;

public TimeOnly(int hour, int minute, int second) {

setHour(hour);

setMinute(minute);

setSecond(second);

}

public int getHour() {

return hour;

}

public void setHour(int hour) {

if (hour < 0 || hour > 23) {

throw new IllegalArgumentException("Hour must be 0...23");

}

this.hour = hour;

}

public int getMinute() {

return minute;

}

public void setMinute(int minute) {

if (minute < 0 || minute > 59) {

throw new IllegalArgumentException("Minute must be 0...59");

}

this.minute = minute;

}

public int getSecond() {

return second;

}

public void setSecond(int second) {

if (second < 0 || second > 59) {

throw new IllegalArgumentException("Second must be 0...59");

}

this.second = second;

}

@Override

public String toString() {

return String.format("%02d:%02d:%02d", this.hour, this.minute, this.second);

}

}

## TestTimeOnly.java

public class TestTimeOnly {

public static void main(String[] args) {

TimeOnly time = new TimeOnly(12, 34, 50);

try {

time.setSecond(65);

} catch (IllegalArgumentException iae) {

System.out.println("Problem with TimeOnly: " + iae.getMessage());

}

System.out.println(time);

}

}

# Tehtävä 4

## TimeOnly.java

public class TimeOnly {

private int hour;

private int minute;

private int second;

public TimeOnly(int hour, int minute, int second) {

setHour(hour);

setMinute(minute);

setSecond(second);

}

public int getHour() {

return hour;

}

public void setHour(int hour) {

if (hour < 0 || hour > 23) {

throw new IllegalArgumentException("Hour must be 0...23");

}

this.hour = hour;

}

public int getMinute() {

return minute;

}

public void setMinute(int minute) {

if (minute < 0 || minute > 59) {

throw new IllegalArgumentException("Minute must be 0...59");

}

this.minute = minute;

}

public int getSecond() {

return second;

}

public void setSecond(int second) {

if (second < 0 || second > 59) {

throw new IllegalArgumentException("Second must be 0...59");

}

this.second = second;

}

@Override

public String toString() {

return String.format("%02d:%02d:%02d", this.hour, this.minute, this.second);

}

}

## DateOnly.java

public class DateOnly {

private int year;

private int month;

private int day;

public DateOnly(int year, int month, int day) {

setYear(year);

setMonth(month);

setDay(day);

}

private boolean isLeapYear() {

return ((this.year % 4 == 0) && !(this.year % 100 == 0 && this.year % 400 != 0));

}

private boolean invalidYear(int year) {

return (year < 1900 || year > 2100);

}

private boolean invalidMonth(int month) {

return (month < 1 || month > 12);

}

private boolean invalidDay(int day) {

if (day < 1 || day > 31) {

return true;

}

switch (this.month) {

case 2:

if (isLeapYear()) {

return day > 29;

} else {

return day > 28;

}

case 4:

case 6:

case 9:

case 11:

return > 30;

default:

return false;

}

}

public int getYear() {

return year;

}

public void setYear(int year) {

if (invalidYear(year)) {

throw new IllegalArgumentException("Year must be 1900...2100");

}

this.year = year;

}

public int getMonth() {

return month;

}

public void setMonth(int month) {

if (invalidMonth(month)) {

throw new IllegalArgumentException("Month must be 1...12");

}

this.month = month;

}

public int getDay() {

return day;

}

public void setDay(int day) {

if (invalidDay(day)) {

throw new IllegalArgumentException("Day must be between 1 and 31");

}

this.day = day;

}

@Override

public String toString() {

return String.format("%04d-%02d-%02d", this.year, this.month, this.day);

}

}

## Timestamp.java

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class Timestamp {

private TimeOnly time;

private DateOnly date;

private Timestamp(int year, int month, int day, int hour, int minute, int second) {

this.time = new TimeOnly(hour, minute, second);

this.date = new DateOnly(year, month, day);

}

private Timestamp(String str) {

this.date = new DateOnly(

Integer.parseInt(str.substring(0, 4)),

Integer.parseInt(str.substring(5, 7)),

Integer.parseInt(str.substring(8, 10))

);

this.time = new TimeOnly(

Integer.parseInt(str.substring(11, 13)),

Integer.parseInt(str.substring(14, 16)),

Integer.parseInt(str.substring(17, 19))

);

}

public static Timestamp parse(String strTime) {

if (strTime.matches("\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:\\d{2}")) {

return new Timestamp(strTime);

}

throw new IllegalArgumentException("Invalid timestamp format");

}

public int getYear() {

return this.date.getYear();

}

public void setYear(int year) {

this.date.setYear(year);

}

public int getMonth() {

return this.date.getMonth();

}

public void setMonth(int month) {

this.date.setMonth(month);

}

public int getDay() {

return this.date.getDay();

}

public void setDay(int day) {

this.date.setDay(day);

}

public int getHour() {

return this.time.getHour();

}

public void setHour(int hour) {

this.time.setHour(hour);

}

public int getMinute() {

return this.time.getMinute();

}

public void setMinute(int minute) {

this.time.setMinute(minute);

}

public int getSecond() {

return this.time.getSecond();

}

public void setSecond(int second) {

this.time.setSecond(second);

}

public TimeOnly getTime() {

return time;

}

public void setTime(TimeOnly time) {

this.time = time;

}

public DateOnly getDate() {

return date;

}

public void setDate(DateOnly date) {

this.date = date;

}

@Override

public String toString() {

return String.format("%sT%s", date.toString(), time.toString());

}

}

## TestTimestamp.java

class TimestampTest {

public static void main(String[] args) {

testParseValidTimestamp();

testParseInvalidTimestamp();

testToString();

testSettersAndGetters();

testInvalidTimestampYear();

testInvalidTimestampMonth();

testInvalidTimestampDay();

testInvalidTimestampHour();

testInvalidTimestampMinute();

testInvalidTimestampSecond();

}

static void assertEquals(int expected, int actual) {

if (expected != actual) {

throw new AssertionError("Assertion failed: expected " + expected + ", but got " + actual);

}

}

static void assertEquals(String expected, String actual) {

if (!expected.equals(actual)) {

throw new AssertionError("Assertion failed: expected \"" + expected + "\", but got \"" + actual + "\"");

}

}

static void assertThrows(Class<? extends Exception> expectedException, Runnable runnable) {

try {

runnable.run();

throw new AssertionError("Assertion failed: Expected exception " + expectedException.getSimpleName() + " not thrown.");

} catch (Exception e) {

if (!expectedException.isInstance(e)) {

throw new AssertionError("Assertion failed: Expected exception " + expectedException.getSimpleName() + ", but got " + e.getClass().getSimpleName());

}

}

}

static void testParseValidTimestamp() {

Timestamp timestamp = Timestamp.parse("2022-01-21T12:34:56");

assertEquals(2022, timestamp.getYear());

assertEquals(1, timestamp.getMonth());

assertEquals(21, timestamp.getDay());

assertEquals(12, timestamp.getHour());

assertEquals(34, timestamp.getMinute());

assertEquals(56, timestamp.getSecond());

}

static void testParseInvalidTimestamp() {

assertThrows(IllegalArgumentException.class, () -> {

Timestamp.parse("invalid\_timestamp");

});

}

static void testToString() {

Timestamp timestamp = Timestamp.parse("2022-01-21T12:34:56");

assertEquals("2022-01-21T12:34:56", timestamp.toString());

}

static void testSettersAndGetters() {

Timestamp timestamp = Timestamp.parse("2022-01-21T12:34:56");

timestamp.setYear(2023);

timestamp.setMonth(2);

timestamp.setDay(22);

timestamp.setHour(13);

timestamp.setMinute(45);

timestamp.setSecond(57);

assertEquals(2023, timestamp.getYear());

assertEquals(2, timestamp.getMonth());

assertEquals(22, timestamp.getDay());

assertEquals(13, timestamp.getHour());

assertEquals(45, timestamp.getMinute());

assertEquals(57, timestamp.getSecond());

}

static void testInvalidTimestampYear() {

assertThrows(IllegalArgumentException.class, () -> {

Timestamp.parse("10000-01-01T00:00:00");

});

}

static void testInvalidTimestampMonth() {

assertThrows(IllegalArgumentException.class, () -> {

Timestamp.parse("2022-13-01T00:00:00");

});

}

static void testInvalidTimestampDay() {

assertThrows(IllegalArgumentException.class, () -> {

Timestamp.parse("2022-01-32T00:00:00");

});

}

static void testInvalidTimestampHour() {

assertThrows(IllegalArgumentException.class, () -> {

Timestamp.parse("2022-01-01T24:00:00");

});

}

static void testInvalidTimestampMinute() {

assertThrows(IllegalArgumentException.class, () -> {

Timestamp.parse("2022-01-01T00:60:00");

});

}

static void testInvalidTimestampSecond() {

assertThrows(IllegalArgumentException.class, () -> {

Timestamp.parse("2022-01-01T00:00:60");

});

}

}

# Tehtävä 6

## Category.java

import java.lang.Comparable;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Category {

private String category;

public Category(String category) {

setCategory(category);

}

public void setCategory(String category) {

final int maxLength = 50;

int length;

try {

length = category.length();

} catch (NullPointerException npe) {

throw new IllegalArgumentException("Category can't be null");

}

length = category.length();

if (length < 1) {

throw new IllegalArgumentException("Category can't be empty");

}

if (length > maxLength) {

throw new IllegalArgumentException("Category can't be over 50 characters long");

}

Pattern pattern = Pattern.compile("[a-z]\*");

Matcher isValid = pattern.matcher(category);

if (!isValid.find()) {

throw new IllegalArgumentException("Category can only contain lowercase letters");

}

this.category = category;

}

public String getCategory() {

return this.category;

}

@Override

public String toString() {

return this.category;

}

}

## Description.java

public class Description {

private String description;

public Description(String description) {

setDescription(description);

}

public void setDescription(String description) {

int length;

final int maxLength = 500;

try {

length = description.length();

} catch (NullPointerException npe) {

throw new IllegalArgumentException("Description can't be null");

}

length = description.length();

if (length < 1) {

throw new IllegalArgumentException("Description can't be empty");

}

if (length > 500) {

throw new IllegalArgumentException("Description can't be over 500 characters long");

}

this.description = description;

}

public String getDescription() {

return this.description;

}

@Override

public String toString() {

return description;

}

}

## Event.java

import java.time.LocalDate;

import java.lang.Comparable;

public class Event implements Comparable<Event> {

private LocalDate time;

private Description description;

private Category category;

public Event(

LocalDate time,

String description,

String category)

{

this.time = time;

this.description = new Description(description);

this.category = new Category(category);

}

public void setTime(LocalDate time) {

this.time = time;

}

public LocalDate getTime() {

return time;

}

public void setDescription(String description) {

this.description.setDescription(description);

}

public String getDescription() {

return this.description.getDescription();

}

public void setCategory(String category) {

this.category.setCategory(category);

}

public String getCategory() {

return this.category.getCategory();

}

@Override

public int compareTo(Event event) {

return this.time.compareTo(event.getTime());

}

@Override

public String toString() {

return (

"When: " + time.toString() +

"\nDesc: " + description.toString() +

"\nWhat: " + category.toString() + '\n');

}

}

## TestEvenet.java

import java.util.List;

import java.util.ArrayList;

import java.time.LocalDate;

public class TestEvent {

public static void main(String[] args) {

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

try {

events.add(

new Event(

LocalDate.now(),

"Testataan roskaa",

"java"));

} catch (Exception e) {

System.out.println(e.toString());

}

try {

events.add(

new Event(

LocalDate.now(),

"Testataan roskaa part 2",

"java"));

} catch (Exception e) {

System.out.println(e.toString());

}

try {

events.add(

new Event(

LocalDate.now(),

"Testataan roskaa part 3",

"java"));

} catch (Exception e) {

System.out.println(e.toString());

}

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

System.out.println(events.get(i).toString());

}

}

}

# Tehtava 7

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.layout.StackPane;

import javafx.scene.text.Font;

import javafx.scene.text.Text;

import javafx.stage.Stage;

public class SimpleJavaFXApp extends Application {

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("OhSyTe 2024");

Text text = new Text("OhSyTe 2024");

text.setFont(new Font(20));

StackPane root = new StackPane();

root.getChildren().add(text);

Scene scene = new Scene(root, 300, 200);

primaryStage.setScene(scene);

primaryStage.show();

}

}

A screenshot of a computer

Description automatically generated

# Tehtava 8

## Event.java

import java.time.LocalDate;

import java.lang.Comparable;

public class Event implements Comparable<Event> {

private LocalDate time;

private Description description;

private Category category;

public Event(

LocalDate time,

String description,

String category)

{

this.time = time;

this.description = new Description(description);

this.category = new Category(category);

}

public void setTime(LocalDate time) {

this.time = time;

}

public LocalDate getTime() {

return time;

}

public void setDescription(String description) {

this.description.setDescription(description);

}

public String getDescription() {

return this.description.getDescription();

}

public void setCategory(String category) {

this.category.setCategory(category);

}

public String getCategory() {

return this.category.getCategory();

}

@Override

public int compareTo(Event event) {

return this.time.compareTo(event.getTime());

}

@Override

public String toString() {

return (

"When: " + time.toString() +

"\nDesc: " + description.toString() +

"\nWhat: " + category.toString() + '\n');

}

}

## EventManager.java

import java.util.List;

import java.util.ArrayList;

//

// The "singleton" design pattern.

// Get the only instance by calling getInstance() in your program.

//

public class EventManager {

private List<Event> events;

// Private constructor, nobody else can create instances

private EventManager() {

this.events = new ArrayList<Event>();

}

// Private instance, statically created.

private static final EventManager INSTANCE = new EventManager();

// When someone wants a reference, they get it through this method.

public static EventManager getInstance() {

return INSTANCE;

}

// Any other methods can be public.

public void addEvent(Event e) {

this.events.add(e);

}

// It may not be a good idea to give a reference to the whole

// event list. Maybe clone the list instead, and return the clone?

public List<Event> getEvents() {

return this.events;

}

}

## FXListView.java

import javafx.application.Application;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.scene.Scene;

import javafx.scene.control.ListView;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import java.time.LocalDate;

public class FXListView extends Application {

private ListView<Event> listView;

@Override

public void start(Stage primaryStage) {

primaryStage.setTitle("Event List View");

// Create a ListView to display events

listView = new ListView<>();

// Add corrected events to the EventManager

EventManager eventManager = EventManager.getInstance();

eventManager.addEvent(new Event(LocalDate.parse("2020-11-12"), "macOS 11 Big Sur released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2015-09-30"), "OS X 10.11 El Capitan released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2019-10-07"), "macOS 10.15 Catalina released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2017-09-25"), "macOS 10.13 High Sierra released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2007-10-26"), "Mac OS X 10.5 Leopard released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2011-07-20"), "Mac OS X 10.7 Lion released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2013-10-22"), "OS X 10.9 Mavericks released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2018-09-24"), "macOS 10.14 Mojave released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2021-10-25"), "macOS 12 Monterey released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2012-07-25"), "OS X 10.8 Mountain Lion released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2016-09-20"), "macOS 10.12 Sierra released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2009-08-28"), "Mac OS X 10.6 Snow Leopard released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2023-09-26"), "macOS 14 Sonoma released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2005-04-29"), "Mac OS X 10.4 Tiger released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2022-10-24"), "macOS 13 Ventura released", "apple"));

eventManager.addEvent(new Event(LocalDate.parse("2014-10-16"), "OS X 10.10 Yosemite released", "apple"));

// Rerun the sorting logic to update the ListView in descending order (newest first)

ObservableList<Event> events = FXCollections.observableArrayList(eventManager.getEvents());

events.sort((event1, event2) -> event2.compareTo(event1)); // Compare in descending order

listView.setItems(events);

// Create a VBox layout and add the ListView

VBox vbox = new VBox(listView);

// Create the scene and set it to the stage

Scene scene = new Scene(vbox, 400, 300);

primaryStage.setScene(scene);

// Show the stage

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

A screenshot of a computer

Description automatically generated