Lesson 25

14.10.2024

```
public class ex1 {
    public static void main(String[] args) {
        int grade = 60;
        if (grade = 60)
            System.out.println("You passed...");
        else
            System.out.println("You failed...");
    }
}
```

```
public class ex2 {
    public static void main(String[] args) {
        boolean flag = false;
        System.out.println((flag = true) |
                (flag = false) || (flag = true));
    }
}
```

```
public class ex3 {
    public static void main(String [] args) {
        int i = 2 ;
        boolean res = false ;
        res = i++ == 2 | --i == 2 & --i == 2 ;
        System. out .println(i);
        System. out .println(res);
    }
}
```

```
public class ex4 {
    public static void main(String[] args) {
        final int i1 = 1;
        final Integer i2 = 1;
        final String s1 = ":ONE";
        String str1 = i1 + s1;
        String str2 = i2 + s1;
        System.out.println(str1 == "1:ONE");
        System.out.println(str2 == "1:ONE");
    }
}
```

```
public class ex5 {
    public static void main(String[] args) {
        int [] arr1 = { 1 , 2 , 3 };
        int [] arr2 = { 'A' , 'B' };
        arr1 = arr2;
        for ( int i = 0 ; i < arr1.length; i++) {
            System. out .print(arr1[i] + " " );
        }
    }
}
```

```
public class ex6 {
    public static void main(String[] args) {
        System.out.println(0.3 == 0.2 + 0.1);
    }
}
```


5 java questions

1. Які є типи даних Java?
2. Чим відрізняється об'єкт від примітивних типів даних?
3. Що таке JVM, JDK, JRE?
4. Навіщо використовують JVM?
5. Назвіть усі методи класу object?

Паттерн проектування - це вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм.
На відміну від готових функцій або бібліотек, патерн не можна просто взяти та скопіювати у програму. Паттерн є не якимось конкретним кодом, а загальною концепцією вирішення тієї чи іншої проблеми, яку потрібно буде ще підлаштувати під потреби вашої програми.
Паттерни часто плутають із алгоритмами, адже обидва поняття описують типові рішення якихось відомих проблем. Але якщо алгоритм – це точний набір дій, то патерн – це високорівневий опис рішення, реалізація якого може відрізнятись у двох різних програмах.
Якщо навести аналогії, то алгоритм – це кулінарний рецепт із чіткими кроками, а патерн – інженерний креслення, на якому намальовано рішення, але не конкретні кроки його реалізації.

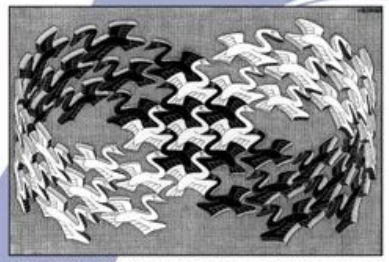
Концепцію патернів вперше описав Крістофер Александер у книзі «Мова шаблонів. Міста. Будинки. Будівництво». У книзі описано «мову» для проектування навколишнього середовища, одиниці якого — шаблони (або патерни, що ближче до оригінального терміну patterns) — відповідають на архітектурні питання: якої висоти зробити вікна, скільки поверхів має бути в будівлі, яку площу у мікрорайоні відвести під дерева та газони.

Ідея здалася привабливою четвірці авторів: Еріху Гамме, Річарду Хелму, Ральфу Джонсону, Джону Вліссідесу. 1995 року вони написали книгу «Прийоми об'єктно-орієнтованого проектування. Паттерни проектування», до якої увійшли 23 патерни, які вирішують різні проблеми об'єктно-орієнтованого дизайну. Назва книги була занадто довгою, щоб хтось зміг всерйоз її запам'ятати. Тому невдовзі всі почали називати її "book by the gang of four", тобто "книга від банди чотирьох", а потім взагалі "GoF book".

Design Patterns

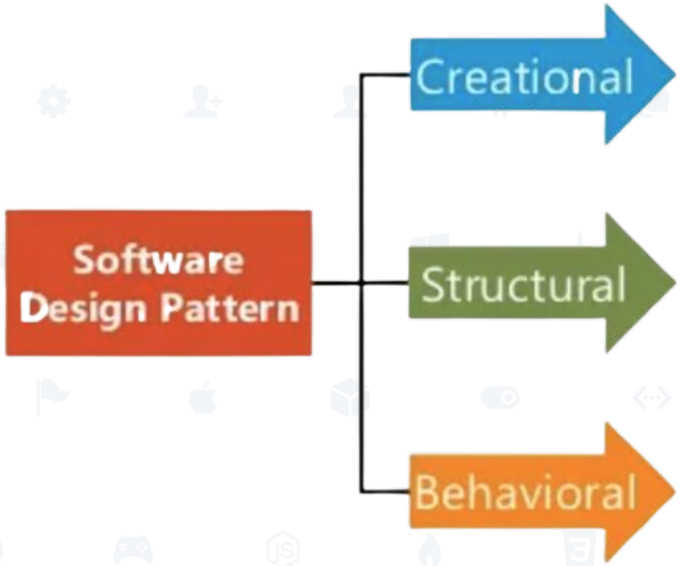
Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Foreword by Grady Booch

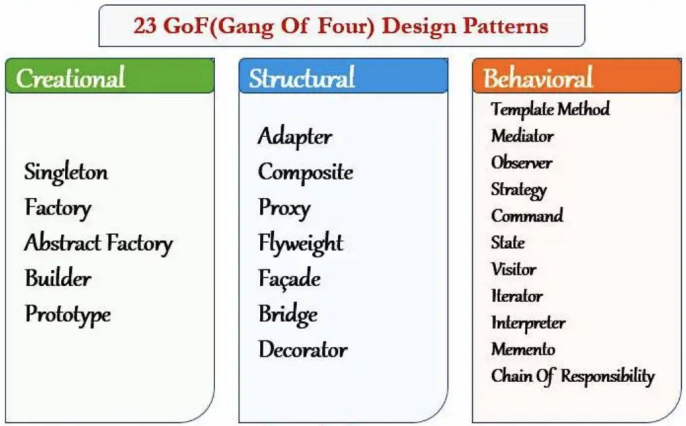
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



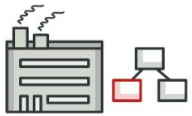
Патерни, що породжують, турбуються про гнучке створення об'єктів без внесення в програму зайвих залежностей.

Структурні патерни показують різні способи побудови зв'язків між об'єктами.

Поведінкові патерни дбають про ефективну комунікацію між об'єктами.

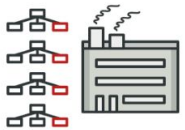


Породжувальні патерни проектування



Фабричний метод
Factory Method

Визначає загальний інтерфейс для створення об'єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об'єктів.

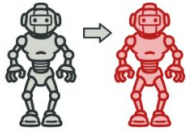


Абстрактна фабрика
Abstract Factory

Дає змогу створювати сімейства пов'язаних об'єктів, не прив'язуючись до конкретних класів створюваних об'єктів.

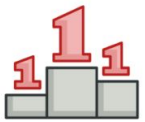
Будівельник
Builder

Дає змогу створювати складні об'єкти крок за кроком. Будівельник дає можливість використовувати один і той самий код будівництва для отримання різних відображень об'єктів.



Прототип
Prototype

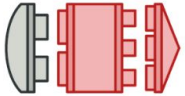
Дає змогу копіювати об'єкти, не вдаючись у подробиці їхньої реалізації.



Одинак
Singleton

Гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього.

Структурні патерни проектування

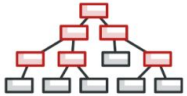


Адаптер
Adapter

Дає змогу об'єктам із несумісними інтерфейсами працювати разом.

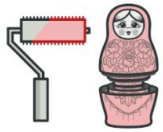
Міст
Bridge

Розділяє один або кілька класів на дві окремі ієрархії — абстракцію та реалізацію, дозволяючи змінювати код в одній гілці класів, незалежно від іншої.



Компонувальник
Composite

Дає змогу згрупувати декілька об'єктів у деревоподібну структуру, а потім працювати з нею так, ніби це одиничний об'єкт.

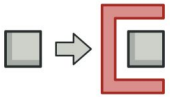


Декоратор
Decorator

Дає змогу динамічно додавати об'єктам нову функціональність, загортаючи їх у корисні «обгортки».

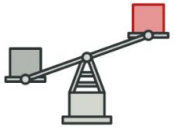
Фасад
Facade

Надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку.



Замісник
Proxy

Дає змогу підставляти замість реальних об'єктів спеціальні об'єкти-замінники. Ці об'єкти перехоплюють виклики до оригінального об'єкта, дозволяючи зробити щось *до* чи *після* передачі виклику оригіналові.



Легковаговик
Flyweight

Дає змогу вмістити більшу кількість об'єктів у відведеній оперативній пам'яті. Легковаговик заощаджує пам'ять, розподіляючи спільний стан об'єктів між собою, замість зберігання однакових даних у кожному об'єкті.

Поведінкові патерни проектування

Ланцюжок обов'язків
Chain of Responsibility

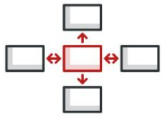
Дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком.

Команда
Command

Перетворює запити на об'єкти, дозволяючи передавати їх як аргументи під час виклику методів, ставити запити в чергу, логувати їх, а також підтримувати скасування операцій.

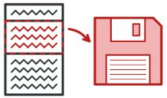
Ітератор
Iterator

Дає змогу послідовно обходити елементи складових об'єктів, не розкриваючи їхньої внутрішньої організації.



Посередник
Mediator

Дає змогу зменшити зв'язаність великої кількості класів між собою, завдяки переміщенню цих зв'язків до одного класу-посередника.

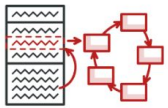


Знімок
Memento

Дає змогу зберігати та відновлювати минулий стан об'єктів, не розкриваючи подробиць їхньої реалізації.

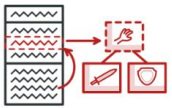
Спостерігач
Observer

Створює механізм підписки, що дає змогу одним об'єктам стежити й реагувати на події, які відбуваються в інших об'єктах.



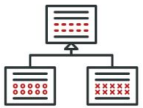
Стан
State

Дає змогу об'єктам змінювати поведінку в залежності від їхнього стану. Ззовні створюється враження, ніби змінився клас об'єкта.



Стратегія
Strategy

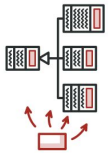
Визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна заміняти один на інший прямо під час виконання програми.



Шаблонний метод
Template Method

Визначає кістяк алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи. Патерн дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури.

Відвідувач
Visitor



Дає змогу додавати до програми нові операції, не змінюючи класи об'єктів, над якими ці операції можуть виконуватися.

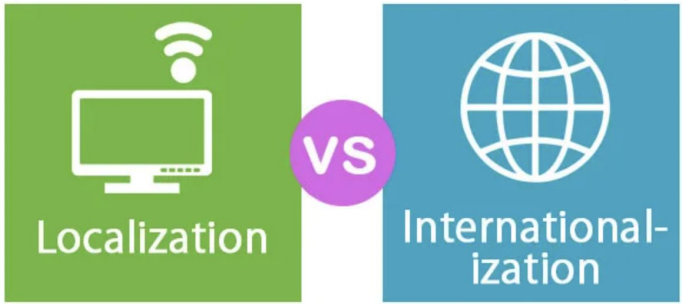
Інтернаціоналізація

Питання інтернаціоналізації інтерфейсу користувача - одне з важливих питань при розробці програми. Для цього недостатньо використовувати Unicode і перекласти на потрібну мову всі повідомлення інтерфейсу користувача. Інтернаціоналізація програми означає щось більше, ніж підтримка Unicode. Дата, час, грошові суми і навіть числа можуть по-різному представлятися різними мовами.

Широкого поширення набули умовні скорочення термінів інтернаціоналізації та локалізації додатків **i18n та l10n**, у яких цифра означає кількість символів між першою та останньою позицією:

i18n - інтернаціоналізація (internationalization);

l10n – локалізація (localization).

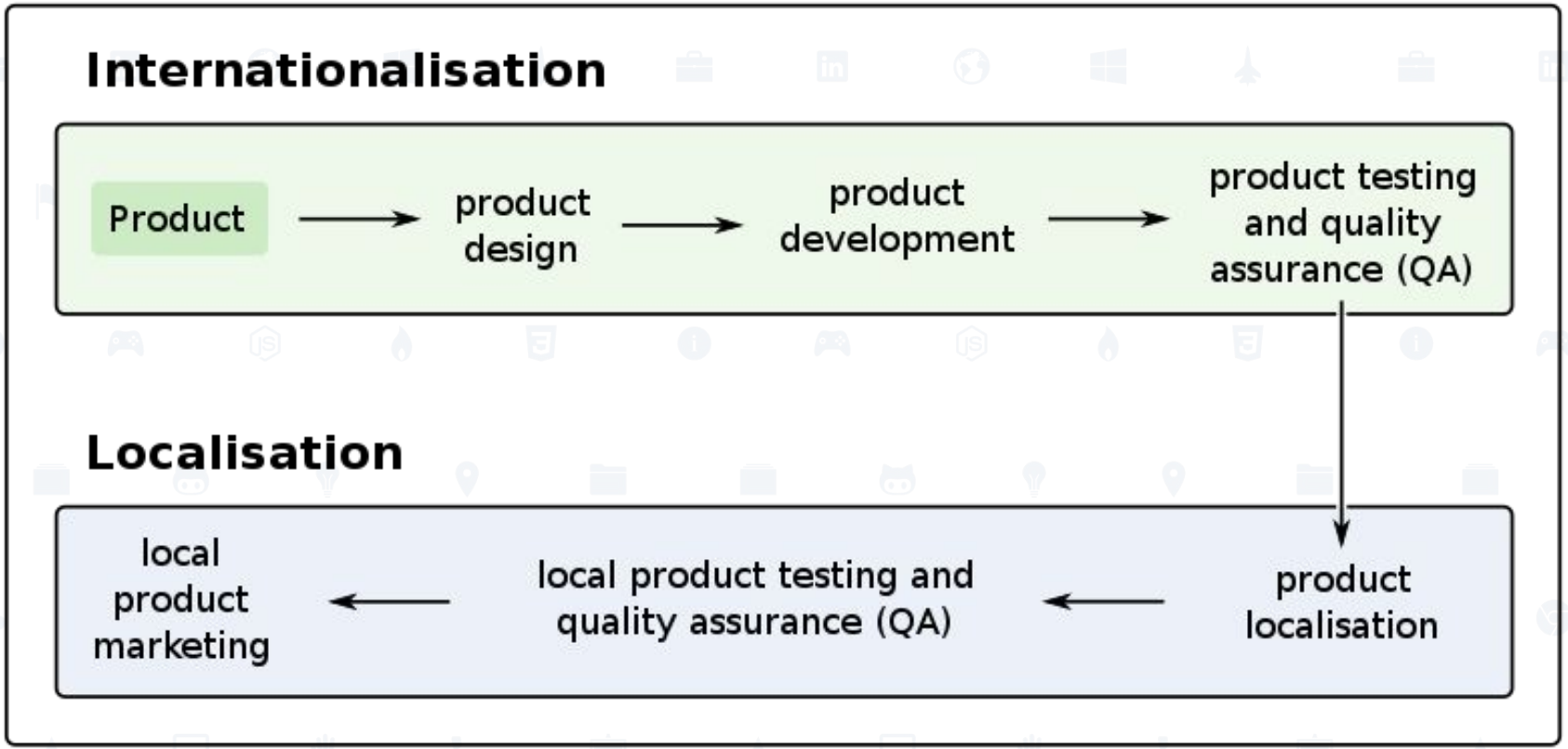


Локалізація (l10n): процес адаптації послуги, продукту чи вмісту до певної місцевості чи ринку, насамперед шляхом перекладу всіх необхідних документів і текстів на місцеву мову, а також з використанням місцевих форматів, символів, стилів подання та інших регіональних особливості, які роблять товар конкурентоспроможним на місцевому ринку.

Інтернаціоналізація (i18n): процес проектування та створення продукту, який гарантує, що його можна легко локалізувати, тобто адаптувати до різних мов і регіонів, не вимагаючи перепроектування та реорганізації змін у вихідних кодах та інших важливих структурах продукт.

Глобалізація (g11n) відноситься до широкого спектру процесів, необхідних для підготовки та запуску продуктів і заходів на міжнародному рівні. Це широкомасштабна концепція, яка стосується багатомовного спілкування та глобальної готовності продуктів і послуг, що дозволяє використовувати продукт у різноманітному середовищі з різними сценаріями, регіональними особливостями та культурами.

Globalisation process

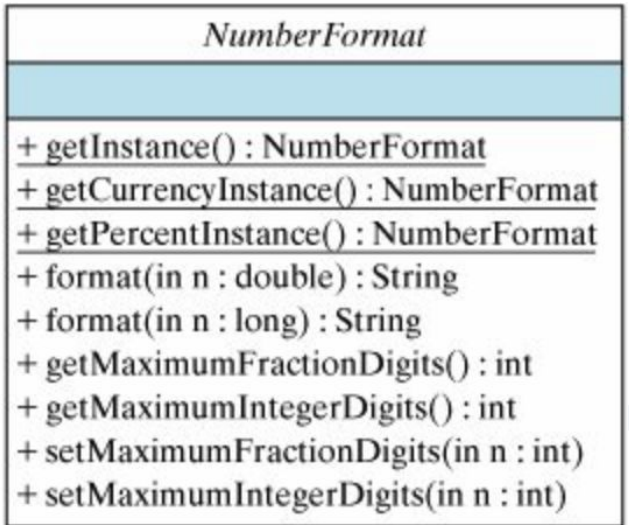


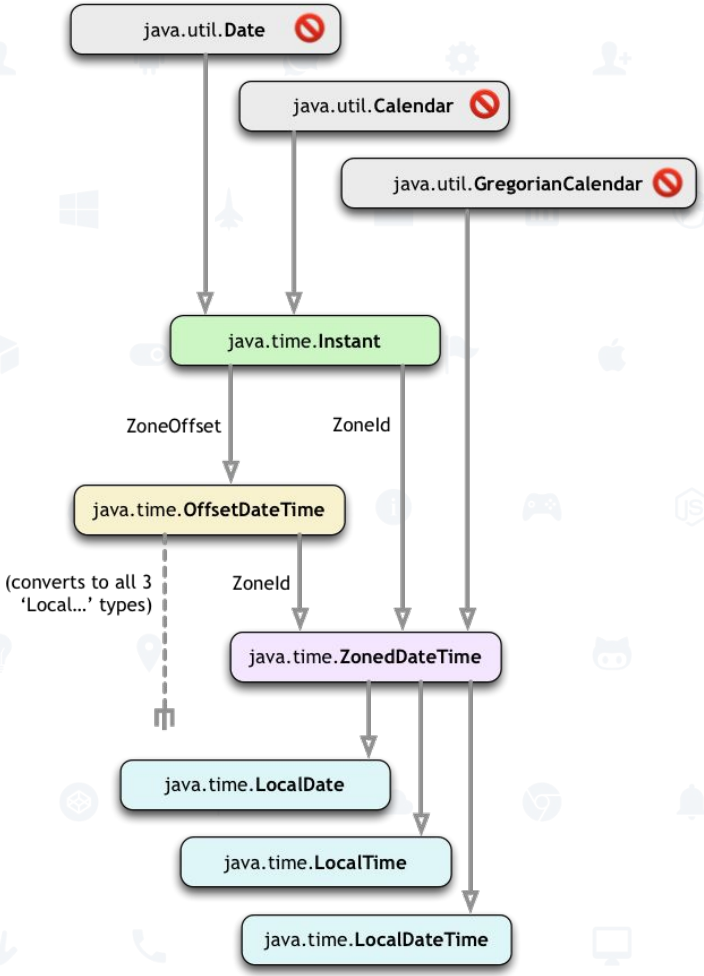
Регіональні стандарти Locale

Додаток, адаптований для міжнародного ринку, легко визначити по можливості вибору мови, для роботи з ним. Але професійно адаптовані програми можуть мати різні регіональні налаштування навіть для тих країн, де використовується однакова мова. У будь-якому випадку команди меню, написи на кнопках та програмні повідомлення мають бути перекладені місцевою мовою, можливо з використанням спеціального національного алфавіту. Але існує ще багато інших більш тонких відмінностей, які стосуються форматів представлення дійсних чисел (розділювачі цілої та дробової частин, роздільників груп тисяч) та грошових сум (включення та місцезнаходження грошового знака), а також формату дати (порядок прямування та символи роздільники днів, місяців та років)

Предопределенные объекты с региональными установками	Объекты, позволяющие указать язык без указания страны
Locale.CHINA	Locale.CHINESE
Locale.FRANCE	Locale.FRENCH
Locale.GERMANY	Locale.GERMAN
Locale.ITALY	Locale.ITALIAN
Locale.JAPAN	Locale.JAPANESE
Locale.US	Locale.ENGLISH

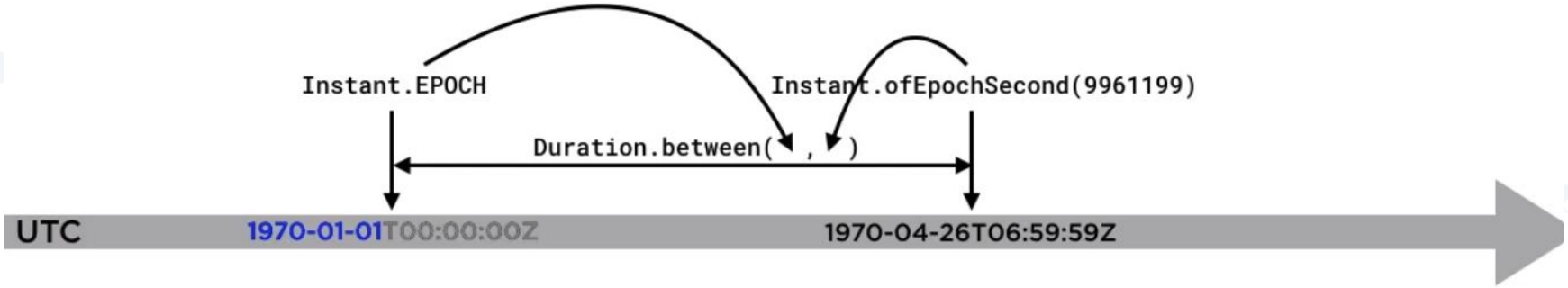
Крім виклику конструктора або вибору обумовлених об'єктів, існує ще два шляхи отримання об'єктів з регіональними налаштуваннями. Статичний метод **getDefault()** класу Locale дозволяє визначити регіональне налаштування, яке використовується в операційній системі за замовчуванням. Змінити налаштування за замовчуванням можна викликавши метод **setDefault()**. Однак слід пам'ятати, що даний метод впливає лише Java-програму, а чи не на операційну систему загалом




java.util.Date
java.util.Calendar
java.util.GregorianCalendar
java.time.Instant
ZoneOffset
ZoneId
java.time.OffsetDateTime
(converts to all 3
'Local...' types)
ZoneId
java.time.ZonedDateTime
java.time.LocalDate
java.time.LocalTime
java.time.LocalDateTime

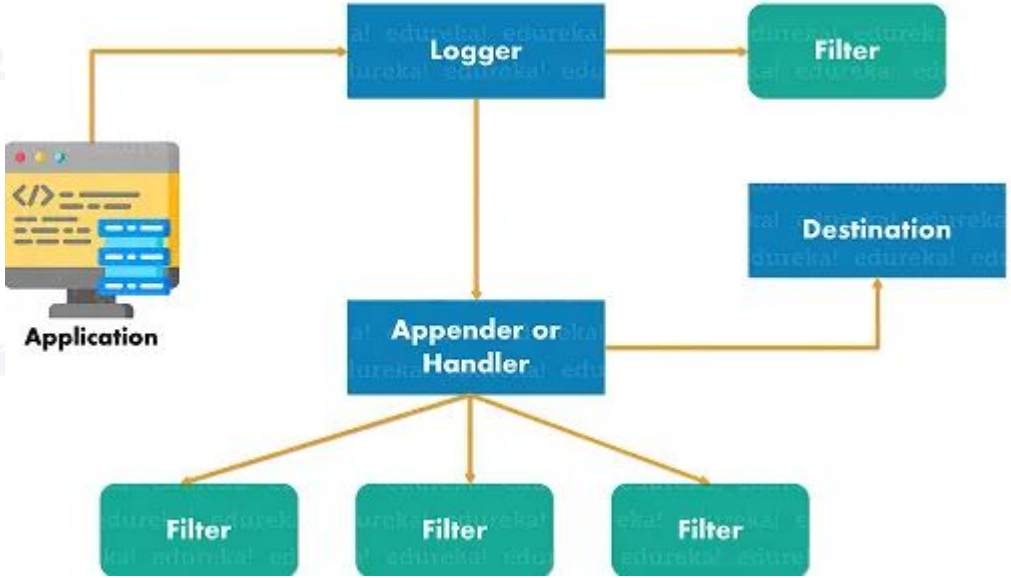
Java Date and Time

Machine/Solar Time



Java™

Logging API



Loggers — відповідають за захоплення записів журналу та передачу їх відповідному Appender.

Appenders or Handlers — вони відповідають за запис подій журналу до місця призначення. Додатки форматують події за допомогою Layouts перед надсиланням виходів

Layouts or Formatters — відповідальні за визначення того, як виглядають дані, коли вони з'являються в записі журналу.

DEBUG fine-grained informational events that are most useful to debug an application

INFO informational messages that highlight the progress of the application at coarse-grained level

WARN potentially harmful situations

ERROR error events that might still allow the application to continue running

FATAL very severe error events that will presumably lead the application to abort

Logging Level	Detail Included in Logs				
	Debug statements	Informational messages	Warning statements	Error statements	Fatal statements
ALL	Included	Included	Included	Included	Included
DEBUG	Included	Included	Included	Included	Included
INFO	Excluded	Included	Included	Included	Included
WARN	Excluded	Excluded	Included	Included	Included
ERROR	Excluded	Excluded	Excluded	Included	Included
FATAL	Excluded	Excluded	Excluded	Excluded	Included
OFF	Excluded	Excluded	Excluded	Excluded	Excluded

slf4j	Log4j	Log4j2	Logback	java.util.logging
FATAL	FATAL	FATAL		
ERROR	ERROR	ERROR	ERROR	SEVERE
WARN	WARN	WARN	WARN	WARNING
DEBUG	INFO	INFO	DEBUG	INFO
INFO	DEBUG	DEBUG	INFO	CONFIG
				FINE
TRACE	TRACE	TRACE	TRACE	FINER
				FINEST

Розглянемо рівні на прикладі log4j, тут вони в порядку спадання:

FATAL: помилка, після якої програма більше не зможе працювати і буде зупинена, наприклад, помилка JVM бракує пам'яті;

ERROR: рівень помилок, коли є проблеми, які потрібно вирішити. Помилка не зупиняє роботу програми в цілому. Інші запити можуть працювати правильно;

WARN: об'явлений логі, який містить попередження. Сталася неочікувана дія, хоча система продовжувала працювати та виконала запит;

INFO: журнал, який записує важливі дії в додатку. Це не помилка, це не попередження, це очікувана дія системи;

DEBUG: журнали, необхідні для налагодження програми. Для впевненості, що система робить саме те, що від неї очікується, або опис дій системи: «метод1 почав роботу»;

TRACE: менш пріоритетні журнали для налагодження з найнижчим рівнем журналювання;

ALL: рівень, на якому будуть записані всі журнали з системи.

Що потрібно логувати

Зрозуміло, логувати все поспіль не варто. Іноді це не потрібно, і навіть небезпечно. Наприклад, якщо залогувати чиїсь особисті дані і це якимось чином спливе на поверхню, то будуть реальні проблеми, особливо на проектах, орієнтованих на Захід. Але є й те, що обов'язково логувати:

- Початок/кінець роботи програми. Потрібно знати, що програма дійсно запустилася, як ми і очікували, і завершилася так само очікувано.
- Питання безпеки. Тут добре б логувати спроби підбору пароля, логування входу важливих користувачів тощо.
- Деякі стани програми. Наприклад, перехід із одного стану в інший у бізнес процесі.
- Деяка інформація для дебага, з відповідним рівнем логування.
- Деякі SQL скрипти. Є реальні випадки, коли це потрібне. Знов-таки, вмілим чином регулюючи рівні, можна досягти відмінних результатів.
- Нитки (Thread), що виконуються, можуть бути логовані у випадках з перевіркою коректної роботи.

Популярні помилки у логуванні

Нюансів багато, але можна виділити кілька частих помилок:

- Надлишок логування. Не варто логувати кожен крок, який суто теоретично може бути важливим. Є правило: логи можуть навантажувати працездатність трохи більше, ніж 10%. Інакше будуть проблеми із продуктивністю.
- Логування всіх даних на один файл. Це призведе до того, що в певний момент читання/запис у нього буде дуже складним, не кажучи про те, що є обмеження за розміром файлів у певних системах.
- Використання неправильних рівнів логування. Кожен рівень логування має чіткі межі, і їх варто дотримуватися. Якщо межа розпливчаста, можна домовитися який із рівнів використовувати.