Java Core+

Inner and local classes

Внутрішні та локальні класи

ПЛАН

Теорія про класи

Розширення теорії

Внутрішні та локальні класи

Логування

Покращення роботи програми

Додавання механізму логування стандартними класами

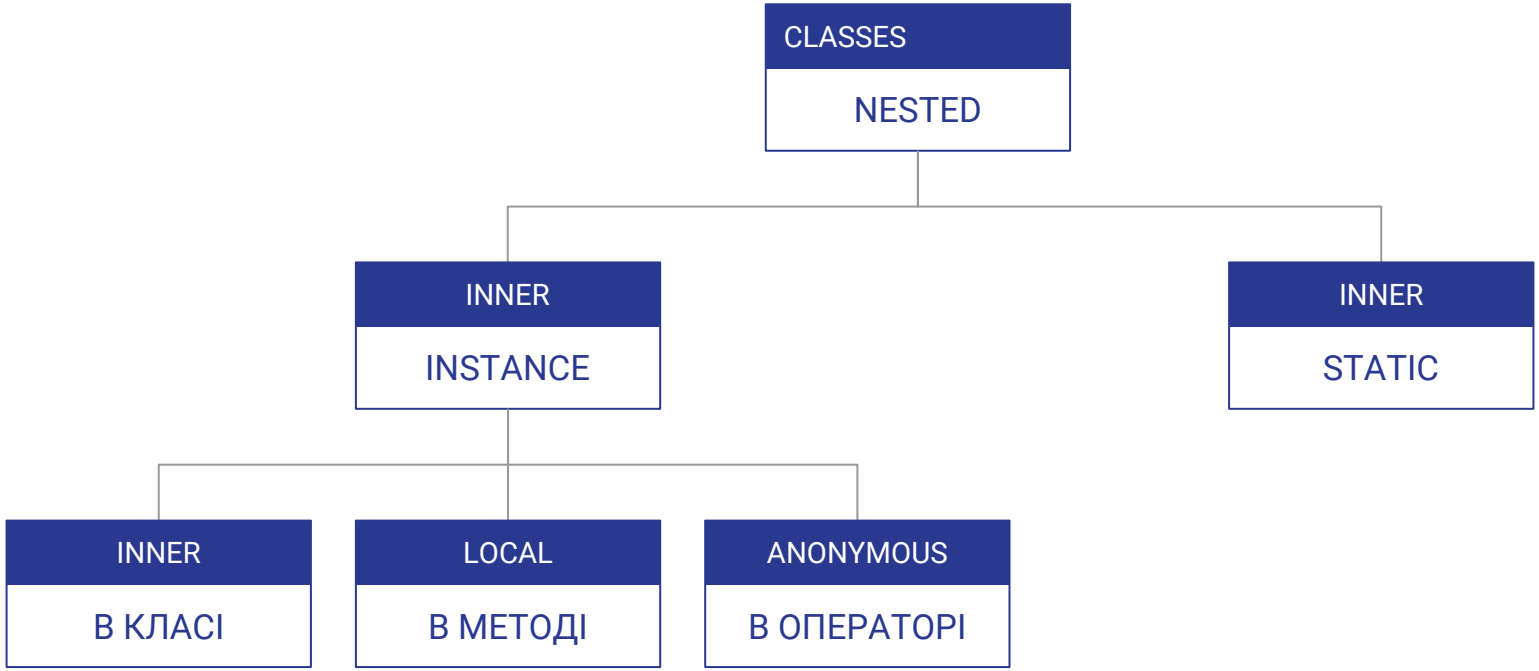
Рзширення БД

Додавання нових таблиць

Додано таблиці щоб додавати кількох користувачів до переглядання заміток та їх маркування

Внутрішні класи

СТРУКТУРА ВНУТРІШНІХ КЛАСІВ



Nested classes

```
class Outer_Demo {
   class Nested_Demo {
   }
}
```

Внутрішні класи можуть бути private.

Також внутрішні класи мають доступ до private змінних головного класу

Коли використовувати внутрішні класи:

- для логічного групування класів в один клас. Коли внутрішній клас корисний для використанні тільки в одному класі
- Для збільшення закритості коду та класів. Коли класи не потрібно виявляти для реалізації функціоналу
- для більш зрозумілого коду, який краще управляється, оскільки такий клас стоїть найближче до місця його використання

ПРИКЛАД

```
class Outer_Demo {
   int num;

   // внутрішній клас
   public class Inner_Demo {
      public void print() {
         System.out.println("This is an inner
class");
      }
   }

   // доступ до внутрішнього класу
   void display_Inner() {
      Inner_Demo inner = new Inner_Demo();
      inner.print();
   }
}
```

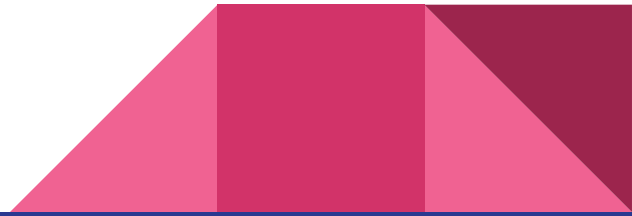
```
public class My_class {

   public static void main(String args[]) {

      // створення об’єкта головного класу
      Outer_Demo outer = new Outer_Demo();

      // доступ до методу внутрішнього класу
      outer.display_Inner();
   }
}

Outer_Demo outer = new Outer_Demo();

Outer_Demo.Inner_Demo inner = outer.new
Inner_Demo()
```

Static Nested Classes

```
class MyOuter {
   static class Nested_Demo {
   }
}
```

Статичний внутрішній клас є статичним полем зовнішнього класу.

В статичному класі не можна використовувати об’єкти і поля зовнішнього класу. Цей клас краще уявляти як верхній клас, що поміщений у інший за міркуваннями сумісності класів



ПРИКЛАД

```
public class Outer {
   static class Nested_Demo {
      public void my_method() {
         System.out.println("This is my nested class");
      }
   }

   public static void main(String args[]) {
      Outer.Nested_Demo nested = new Outer.Nested_Demo();
      nested.my_method();
   }
}
```

Локальні класи

Локальні класи

Ці класи створюються в блоці коду, тобто між фігурними дужками {}.
Як правило вони створюються в тілі методу

```
void my_Method() {
    int num = 23;
    // локальний клас


 class MethodInner_Demo {
      public void print() {
          System.out.println("This is
method inner class "+num);
       }       } // end of inner class

  MethodInner_Demo inner = new
MethodInner_Demo();
   inner.print();     }
```

Локальний клас має доступ

Змінних класу

Має доступ до всіх членів зовнішнього класу, в т. ч. і приватних методів

Змінних методу

З локального класу можна отримати доступ до змінних методу, які оголошені як final. З Java 8 може мати доступ до змінних effectifly final - до простих змінних, які не змінюються

Статичні змінні

Локальні класи, так само як і внутрішні класи не можуть декларувати статичних змінних класу. Вони можуть мати доступ до статичних змінних зовнішнього класу. Можуть мати константи.

Логування

java.util.logging framework

```
public class SomeClass {

        private static Logger log = Logger.getLogger(SomeClass.class.getName());

        public void someMethod()
        {
            log.info("Some message");
        }
…
```

Загальні кроки для роботи з логуванням:

1. Отримати посилання на статичний об’єкт логера. Надати логеру ім’я, яке може бути назвою класу чи пакету.
2. Записати інформацію в лог відповідно до рівня деталізації. Рівень деталізації задає відповідний метод

Рівні деталізації

static Level OFF - is a special level that can be used to turn off logging.
static Level SEVERE - is a message level indicating a serious failure.
static Level WARNING - is a message level indicating a potential problem
static Level INFO - is a message level for informational messages.
static Level CONFIG - is a message level for static configuration messages.
static Level FINE - is a message level providing tracing information.
static Level FINER - indicates a fairly detailed tracing message.
static Level FINEST - indicates a highly detailed tracing message.
static Level ALL - indicates that all messages should be logged.

logger.log(Level.FINER, “Message”);

config(String msg);

fine(String msg);
finer(String msg);
finest(String msg);
info(String msg);
warning(String msg);
severe(String msg);

Початкова конфігурація

```
try {

    Handler fh = new FileHandler(getServletContext()
        .getRealPath("/logs/app.log"));
    Logger.getLogger("").addHandler(fh);
    Logger.getLogger("").addHandler(new ConsoleHandler());
    Logger.getLogger("").setLevel(Level.ALL);

    Logger.getLogger("ua.ivfr.lms.servlets")
        .setLevel(Level.WARNING);
     Logger.getLogger("ua.ivfr.lms.dao.repository")
        .setLevel(Level.FINE);

} catch (IOException e) {
    e.printStackTrace();
}
```

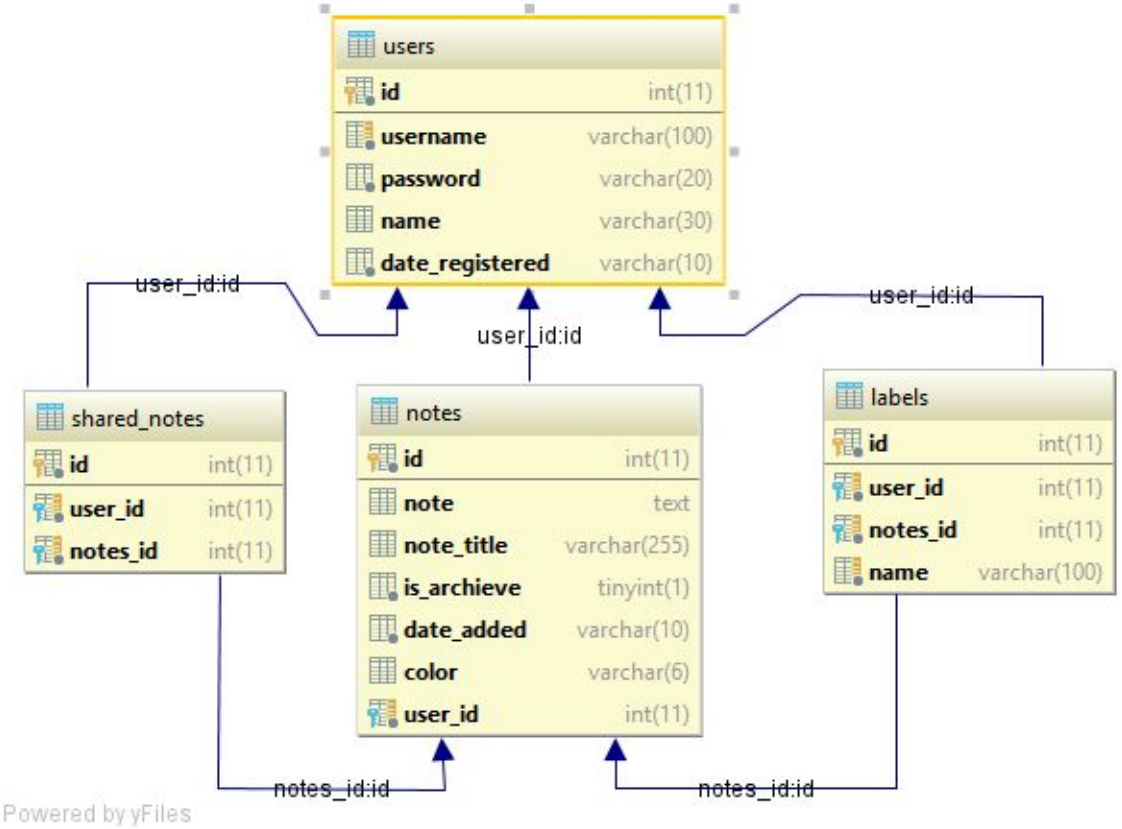
Початкова конфігурація 2

Ліміт на розмір файлу
java.util.logging.FileHandler.limit = 1000000

Кількість файлів до перезаписування
java.util.logging.FileHandler.count = 5

Тип виводу простий, може бути на основі XML
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter

База даних


users
id int(11)
username varchar(100)
password varchar(20)
name varchar(30)
date_registered varchar(10)
user_id:id
user_id:id
user_id:id
shared_notes
id int(11)
user_id int(11)
notes_id int(11)
notes
id int(11)
note text
note_title varchar(255)
is_archieve tinyint(1)
date_added varchar(10)
color varchar(6)
user_id int(11)
labels
id int(11)
user_id int(11)
notes_id int(11)
name varchar(100)
notes_id:id
notes_id:id
Powered by yFiles

База даних 2

```
CREATE TABLE shared_notes
(
 id       INT AUTO_INCREMENT PRIMARY KEY,
 user_id  INT NOT NULL,
 notes_id INT NOT NULL,
 CONSTRAINT shared_notes_users_id_fk
 FOREIGN KEY (user_id) REFERENCES xkeep.users
(id),
 CONSTRAINT shared_notes_notes_id_fk
 FOREIGN KEY (notes_id) REFERENCES xkeep.notes
(id)
)
 COMMENT 'Notes visible to multiple users';

CREATE INDEX shared_notes_notes_id_fk
 ON shared_notes (notes_id);

CREATE INDEX shared_notes_users_id_fk
 ON shared_notes (user_id);
```

```
CREATE TABLE labels
(
 id       INT AUTO_INCREMENT PRIMARY KEY,
 user_id  INT NOT NULL,
 notes_id INT NOT NULL,
 name     VARCHAR(100) NOT NULL,
 CONSTRAINT labels_user_id_notes_id_name_uindex
 UNIQUE (user_id, notes_id, name),
 CONSTRAINT labels_users_id_fk
 FOREIGN KEY (user_id) REFERENCES xkeep.users (id),
 CONSTRAINT labels_notes_id_fk
 FOREIGN KEY (notes_id) REFERENCES xkeep.notes (id)
)
 COMMENT 'labels per user note';

CREATE INDEX labels_notes_id_fk
 ON labels (notes_id);
```