Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра програмної інженерії

Презентація до практичної роботи №1

На тему:

Dart Coding Conventions

Виконала:

Студентка групи ПЗПІ 22-2

Моссур Д. Є.

Перевірив:

Доцент кафедри ПІ

Лещинський Володимир Олександрович

Харків 2025

Dart Coding Conventions

Мета роботи:

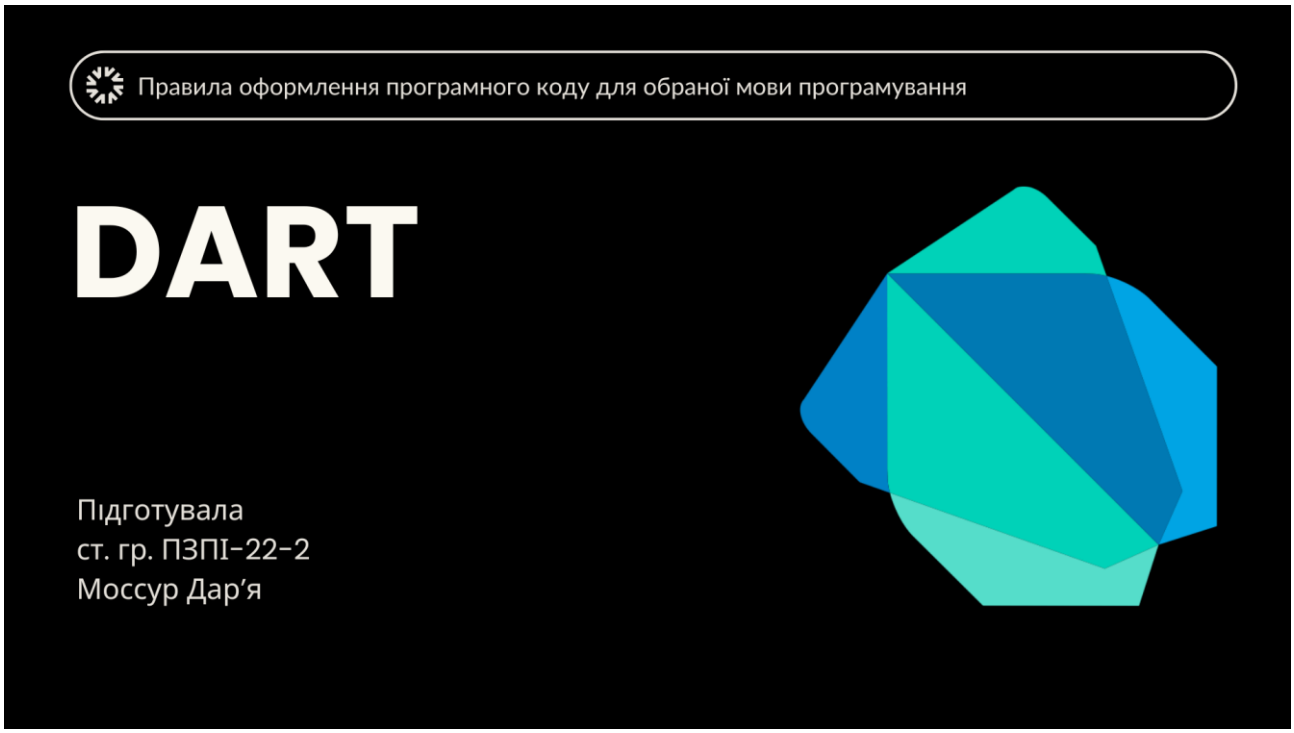
Метою роботи було дослідження основних правил та рекомендацій щодо написання коду, визначення стильових характеристик, структури та організації коду, принципи рефакторингу для мови Dart.

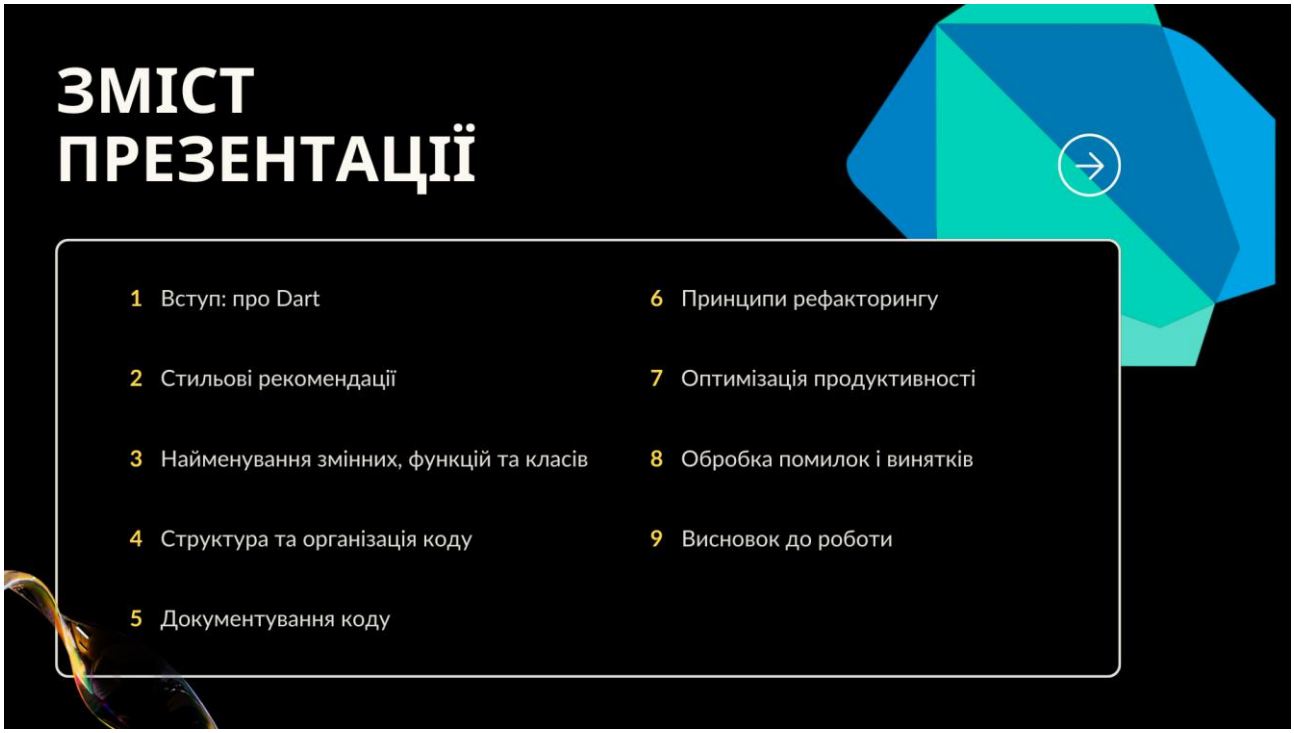
Хід роботи:

Було вивчено ключові моменти щодо Dart Coding Conventions з офіційних джерел, статтей та форумів та викладено основний зміст у презентацію. Презентація додана у додатку А.

Додаток А

Презентація





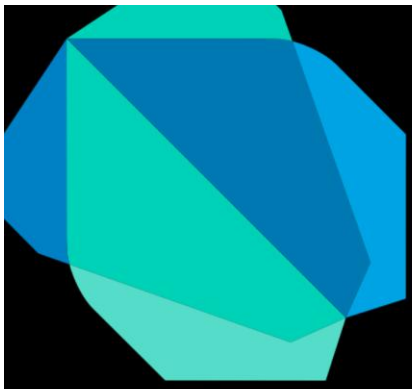
ВСТУП: ПРО DART

Dart — це сучасна мова програмування, розроблена Google у 2011 році. Вона створена для багатоплатформної розробки, що дозволяє працювати з мобільними, веб-, серверними та десктопними застосунками.

- Dart має активну спільноту та підтримується великими компаніями.
- Синтаксис Dart нагадує JavaScript, Java та C#, що робить його легким для вивчення.
- Також Dart є базовою мовою для Flutter — фреймворку для створення кросплатформних мобільних застосунків.

Ключові характеристики Dart:

1. Багатопарадигмова. Підтримує об'єктно-орієнтоване, імперативне та функціональне програмування.
2. Кросплатформна. Працює на мобільних пристроях (Android, iOS), у браузері, на сервері та на десктопі.
3. Продуктивна. Завдяки Just-in-Time (JIT) та Ahead-of-Time (AOT) компіляції забезпечує швидке виконання коду.
4. Асинхронна. Вбудовані механізми для роботи з потоками та асинхронними викликами (Future, Stream).



ВСТУП: ПРО DART

Основні переваги Dart:

- Простий у вивченні. Синтаксис легко зрозумілий розробникам, знайомим з JavaScript або Java.
- Гнучкість. Dart дозволяє створювати як прототипи, так і складні системи завдяки своїй багатопарадигмовій природі.
- Інструментами розробника є: Hot Reload - моментальне оновлення мобільних додатків під час розробки; DartPad - онлайн-редактор для вивчення мови та експериментів.

Сфери застосування Dart:

- Мобільні застосунки: через Flutter забезпечує швидку розробку кросплатформних додатків з високою продуктивністю.
- Веб-додатки: Dart компілюється у JavaScript, що дозволяє використовувати його у браузерах.
- Серверна розробка: мова підходить для створення API та бекенд-розробки завдяки таким бібліотекам, як Shelf.
- Десктопні додатки: з Flutter можлива розробка для Windows, macOS і Linux.

СТИЛЬОВІ РЕКОМЕНДАЦІЇ У DART

Підтримуваність

Єдиний стиль полегшує читання та розуміння коду іншими розробниками.

Продуктивність

Уніфікований код зменшує кількість помилок та підвищує ефективність командної роботи.

Автоматизація

Dart має вбудований інструмент dart format, який забезпечує автоматичне дотримання стилю.

ПРИКЛАДИ СТИЛЬОВИХ РЕКОМЕНДАЦІЙ У КОДІ

Осмислені назви змінних, функцій і класів

```dart
// Поганий приклад
int f(int n) {
  return n * n;
}

// Гарний приклад
int calculateSquare(int number) {
  return number * number;
}
```

Використовуйте документаційні коментарі

```dart
/// Обчислює квадрат числа.
/// [number] – число для обчислення.
/// Повертає квадрат числа.
int calculateSquare(int number) {
  return number * number;
}
```

Дотримуйтесь єдиного стилю відступів і структури

```dart
// Dart використовує відступи у 2 пробіли
// Поганий приклад
void main(){
    print("Hello, Dart!");
}

// Гарний приклад
void main() {
  print("Hello, Dart!");
}
```

НАЙМЕНУВАННЯ ЗМІННИХ, ФУНКЦІЙ ТА КЛАСІВ

Важливість структури та організації коду.

Структура коду є важливою. Це забезпечує:

1. Читабельність коду, адже добре організований код легше розуміти, навіть якщо розробник бачить його вперше.
2. Масштабованість. Саме грамотна структура дозволяє легко додавати новий функціонал.
3. Підтримка. Організований код полегшує внесення змін та виправлення помилок.

Як організовують код у Dart:

Dart дозволяє розділяти код на функції, класи та файли/пакети, що сприяє логічному розподілу логіки програми. Основні принципи:

- Використання функцій для повторюваних завдань.
- Розбиття логіки на класи та методи.
- Винесення окремих частин коду в модулі або пакети.

ФУНКЦІЇ У DART

ОРГАНІЗАЦІЯ КОДУ

```dart
// Поганий приклад: Код дублюється.
void main() {
  print("Hello, Alice!");
  print("Hello, Bob!");
}

// Гарний приклад: Використання функції.
void greet(String name) {
  print("Hello, $name!");
}

void main() {
  greet("Alice");
  greet("Bob");
}
```

DART ПІДТРИМУЄ АНОНІМНІ ФУНКЦІЇ (LAMBDA), ЩО РОБИТЬ КОД КОМПАКТНИМ.

```dart
void main() {
  List<int> numbers = [1, 2, 3];
  numbers.forEach((number) => print("Number: $number"));
}
```

КЛАСИ ТА ОБ'ЄКТНО-ОРІЄНТОВАНИЙ ПІДХІД

```dart
// Приклад класу для опису користувача.
class User {
  String name;
  int age;

  User(this.name, this.age);

  void displayInfo() {
    print("User: $name, Age: $age");
  }
}

void main() {
  User user = User("Alice", 25);
  user.displayInfo();
}
```

У Dart класи допомагають структурувати логіку додатку, об'єднуючи дані та функціональність. Використання класів для складної логіки та об'єктів із властивостями є необхідною практикою.

Також Dart підтримує скорочений синтаксис для конструкторів і ініціалізації.

```dart
class Product {
  final String name;
  final double price;

  Product(this.name, this.price);
}
```

РОЗБИТТЯ КОДУ НА МОДУЛІ

Коли програма стає великою, варто розділяти код на файли або модулі. У великих проєктах зростає кількість функцій, класів і логіки. Поділ на модулі дозволяє легко додавати нові функції, не змінюючи існуючі. Модулі можуть бути використані повторно в інших частинах програми або навіть в інших проєктах. Наприклад, модуль для валідації даних. У великих командах розробників різні люди можуть працювати над різними частинами програми одночасно. Розділення файлів дозволяє уникнути конфліктів у коді.

Приклад:
Уявіть великий проєкт для онлайн-магазину. Якщо зберігати весь код у файлі main.dart, то знайти конкретну логіку буде складно. Але якщо ми створимо окремі модулі для роботи з користувачами - user.dart, продуктами - product.dart та оплатою - payment.dart, проєкт буде більш організованим.

РОЗБИТТЯ КОДУ НА МОДУЛІ

Приклад структури файлів у проєкті:

```
lib/
├── main.dart        // Головна точка входу.
├── models/          // Моделі даних.
│   └── user.dart    // Клас User.
├── services/        // Логіка програми.
│   └── user_service.dart
└── utils/           // Допоміжні функції.
    └── formatter.dart
```

! Організація коду за допомогою функцій, класів та модулів підвищує його читабельність і масштабованість, що критично важливо для великих проєктів.

Dart дозволяє імпортувати потрібні модулі за допомогою ключового слова import:

```dart
// main.dart
import 'models/user.dart';

void main() {
  User user = User("Bob", 30);
  user.displayInfo();
}

// models/user.dart
class User {
  String name;
  int age;

  User(this.name, this.age);

  void displayInfo() {
    print("User: $name, Age: $age");
  }
}
```

ДОКУМЕНТУВАННЯ КОДУ

Чому документація важлива?

1. Розуміння коду. Коментарі та документація дозволяють розробникам швидко зрозуміти логіку програми, особливо якщо вони не працювали над нею раніше.
2. Спрощення підтримки. Зрозумілий код із коментарями легше оновлювати та виправляти.
3. Командна робота: У спільних проєктах документація допомагає всім членам команди бути "на одній сторінці".

Типи коментарів у Dart:

- Однорядкові коментарі - для пояснення невеликих частин коду.
- Багаторядкові коментарі - для складної логіки або опису великих блоків.
- Документаційні коментарі - спеціальні коментарі, що використовуються для автоматичної генерації документації за допомогою DartDoc.

ДОКУМЕНТУВАННЯ КОДУ

Використання DartDoc

DartDoc — це вбудований інструмент для генерації документації з вихідного коду. На основі спеціальних коментарів DartDoc створює HTML-документацію, яку можна переглядати у браузері.
Це дозволяє стандартизувати опис функцій, класів і методів.

Для генерації документації напишіть документаційні коментарі у коді.
Далі виконайте команду: dart doc
Отриману документацію можна знайти у папці doc/ у вигляді HTML-файлів.

Прикладом згенерованої документації можуть бути:
Інформація про класи, методи, параметри з прикладами їх використання, окремі сторінки для кожного модуля або компонента.

ДОКУМЕНТУВАННЯ КОДУ

Приклад документаційних коментарів:

```dart
/// Клас, що представляє користувача.
class User {
  /// Ім'я користувача.
  final String name;

  /// Вік користувача.
  final int age;

  /// Конструктор класу [User].
  ///
  /// [name] – ім'я користувача.
  /// [age] – вік користувача.
  User(this.name, this.age);

  /// Виводить інформацію про користувача.
  void displayInfo() {
    print("User: $name, Age: $age");
  }
}
```

РЕФАКТОРИНГ У DART

Рефакторинг — це процес поліпшення структури існуючого коду без зміни його функціональності.

Чому рефакторинг важливий?

1. Читабельність: поліпшений код стає зрозумілішим для розробників.
2. Підтримка: зменшує ймовірність помилок у майбутньому.
3. Продуктивність: упорядкований код працює швидше і вимагає менше ресурсів.

Основними прийомами рефакторингу у Dart є розбиття великих функцій на менші - необхідно уникати функцій із великою кількістю логіки.
Також винесення повторюваного коду в окремі функції чи класи, що зменшує дублювання.
Та використання сучасних можливостей мови, наприклад, скорочений синтаксис для функцій або колекцій.

РЕФАКТОРИНГ У DART

Приклади (скриншоти коду)

До рефакторингу:

```dart
void main() {
  print("Circle area: ${3.14 * 10 * 10}");
  print("Square area: ${10 * 10}");
}
```

Після рефакторингу (покращено читабельність, легке повторне використання логіки).

```dart
class ShapeCalculator {
  double calculateCircleArea(double radius) => 3.14 * radius * radius;

  double calculateSquareArea(double side) => side * side;
}

void main() {
  ShapeCalculator calculator = ShapeCalculator();
  print("Circle area: ${calculator.calculateCircleArea(10)}");
  print("Square area: ${calculator.calculateSquareArea(10)}");
}
```

ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ

Важливість продуктивності для програми:

- Продуктивність безпосередньо впливає на користувацький досвід.
- Оптимізовані додатки споживають менше ресурсів пристрою.
- Оптимізація асинхронного коду зменшує час очікування для користувача.

Основні підходи до оптимізації у Dart:

- Виконання операцій (lazy operations): використання Iterable для обробки великих колекцій.
- Асинхронність: використання Future і async/await для роботи з потоками.
- Оптимізація пам'яті: уникнення створення зайвих об'єктів, використання const для незмінних значень.

ПРИКЛАД ОПТИМІЗАЦІЇ КОДУ

```dart
// До оптимізації
void main() {
  List<int> numbers = [1, 2, 3, 4, 5];
  List<int> squares = [];
  for (int number in numbers) {
    squares.add(number * number);
  }
  print(squares);
}
```

```dart
// Оптимізація асинхронності
Future<String> fetchData() async {
  await Future.delayed(Duration(seconds: 2));
  return "Data loaded";
}

void main() async {
  print("Fetching...");
  String data = await fetchData();
  print(data);
}
```

```dart
// Після оптимізації
void main() {
  List<int> numbers = [1, 2, 3, 4, 5];
  List<int> squares = numbers.map((number) => number * number).toList();
  print(squares);
}
```

ОБРОБКА ПОМИЛОК І ВИНЯТКІВ

Обробка помилок і винятків є важливою частиною забезпечення надійності та стабільності програм на Dart. Грамотне управління винятками дозволяє уникнути несподіваних збоїв і полегшує пошук помилок у коді.

МЕХАНІЗМ ОБРОБКИ ПОМИЛОК У DART

- try-catch. Використовується для перехоплення і обробки винятків:

```dart
void main() {
  try {
    int result = 10 ~/ 0; // Ділення на нуль
  } catch (e) {
    print('Помилка: $e');
  }
}
```

- on. Використовується для обробки конкретного типу винятків:

```dart
void main() {
  try {
    List<int> numbers = [1, 2, 3];
    print(numbers[5]); // Вихід за межі списку
  } on RangeError {
    print('Помилка: Ви звертаєтеся до неіснуючого індексу');
  }
}
```

МЕХАНІЗМ ОБРОБКИ ПОМИЛОК У DART

- finally. Код у блоці finally виконується завжди, незалежно від того, чи виникла помилка:

```dart
void main() {
  try {
    int result = 10 ~/ 0;
  } catch (e) {
    print('Помилка: $e');
  } finally {
    print('Завершення виконання програми.');
  }
}
```

- Для покращення читабельності та точності діагностики можна створювати власні класи винятків:

```dart
class CustomException implements Exception {
  final String message;
  CustomException(this.message);

  @override
  String toString() => 'CustomException: $message';
}

void main() {
  try {
    throw CustomException('Це моя власна помилка');
  } catch (e) {
    print(e);
  }
}
```

Загалом, Dart є сучасним інструментом, що поєднує в собі простоту синтаксису, високу продуктивність та широкі можливості для створення кросплатформених застосунків. Завдяки потужним інструментам, таким як Flutter, Dart дозволяє розробникам швидко створювати інтерфейси, які працюють однаково добре на різних пристроях.

Основними перевагами Dart є:

- Простота та зручність у написанні коду, зрозумілого навіть для початківців.
- Висока продуктивність завдяки компіляції у машинний код.
- Гнучкий синтаксис, що підтримує як об'єктно-орієнтований, так і функціональний стиль програмування.
- Широка екосистема бібліотек і інструментів.