عنوان: **شروع کار با Dart - قسمت 3**
نویسنده: میثم خوشبخت
تاریخ: ۱۳۹۳/۰۲/۰۳ ۹:۲۰
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, Web Design, Dart

لطفا قسمت دوم را در اینجا مطالعه بفرمایید

خدمت دوستان عزیز مطلبی را عرض کنم که البته باید در ابتدای این سری مقالات متذکر می‌شدم. این سری مقالات Dart مرجع کاملی برای یادگیری Dart نمی‌باشد. فقط یک Quick Start یا Get Started محسوب می‌شود برای آشنایی مقدماتی با ساختار Dart. از عنوان مقاله هم این موضوع قابل درک و تشخیص می‌باشد. همچنین فرض شده است که دوستان آشنایی مقدماتی با جاوااسکریپت و مباحث شی گرایی را نیز دارند. البته اگر مشغله کاری به بنده این اجازه را بدهد، مطالب جامع‌تری را در این زمینه آماده و منتشر می‌کنم.

گام پنجم: ذخیره سازی اطلاعات در فضای محلی یا Local

در این گام، تغییرات badge را در فضای ذخیره سازی سیستم Local نگهداری می‌نماییم؛ بطوری که اگر دوباره برنامه را راه اندازی نمودید، badge با داده‌های ذخیره شده در سیستم Local مقداردهی اولیه می‌گردد.
کتابخانه dart:convert را به منظور استفاده از کلاس مبدل JSON به فایل piratebadge.dart اضافه نمایید.

```
import 'dart:html';
import 'dart:math' show Random;

import 'dart:convert' show JSON;
```

همچنین یک Named Constructor یا سازنده‌ی با نام را به کلاس PirateName بصورت زیر اضافه کنید.

```
class PirateName {
  ...
  PirateName.fromJSON(String jsonString) {
    Map storedName = JSON.decode(jsonString);
    _firstName = storedName['f'];
    _appellation = storedName['a'];
  }
}
```

توضیحات

- جهت کسب اطلاعات بیشتر در مورد Json به این لینک مراجعه نمایید
- کلاس JSON جهت کار با داده هایی به فرمت Json استفاده می‌شود که امکاناتی را جهت دسترسی سریعتر و راحت‌تر به این داده‌ها فراهم می‌کند.
- سازنده‌ی PirateName.fromJSON، از یک رشته حاوی داده‌ی Json، یک نمونه از کلاس PirateName ایجاد می‌کند.
- سازنده‌ی PirateName.fromJSON، یک Named Constructor می‌باشد. این نوع سازنده‌ها دارای نامی متفاوت از نام سازنده‌های معمول هستند و بصورت خودکار نمونه ای از کلاس مورد نظر را ایجاد نموده و به عنوان خروجی بر می‌گردانند.
- تابع JSON.decode یک رشته‌ی حاوی داده‌ی Json را تفسیر نموده و اشیاء Dart را از آن ایجاد می‌کند.
یک Getter به کلاس PirateName اضافه کنید که مقادیر ویژگی‌های آن را به یک رشته Json تبدیل می‌کند

```
class PirateName {
  ...
  String get jsonString => JSON.encode({"f": _firstName, "a": _appellation});
}
```

جهت ذخیره سازی آخرین تغییرات کلاس PirateName در فضای ذخیره سازی Local، از یک کلید استفاده می‌کنیم که مقدار آن محتوای PirateName می‌باشد. در واقع فضای ذخیره سازی Local داده‌ها را به صورت جفت کلید-مقدار یا Key-Value Pairs نگهداری می‌نماید. جهت تعریف کلید، یک متغیر رشته ای را بصورت top-level و به شکل زیر تعریف کنید.

```
final String TREASURE_KEY = 'pirateName';
```

```
void main() {
  ...
}
```

زمانیکه تغییری در badge name صورت گرفت، این تغییرات را در فضای ذخیره سازی Local، توسط ویژگی window.localStorage ذخیره می‌نماییم. تغییرات زیر را در تابع setBadgeName اعمال نمایید

```
void setBadgeName(PirateName newName) {
  if (newName == null) {
    return;
  }
  querySelector('#badgeName').text = newName.pirateName;
  window.localStorage[TREASURE_KEY] = newName.jsonString;
}
```

تابع getBadgeNameFromStorage را بصورت top-level تعریف نمایید. این تابع داده‌های ذخیره شده را از Local Storage بازیابی نموده و یک شی از نوع کلاس PirateName ایجاد می‌نماید.

```
void setBadgeName(PirateName newName) {
  ...
}

PirateName getBadgeNameFromStorage() {
  String storedName = window.localStorage[TREASURE_KEY];
  if (storedName != null) {
    return new PirateName.fromJSON(storedName);
  } else {
    return null;
  }
}
```

در پایان نیز تابع setBadgeName را به منظور مقدار دهی اولیه به badge name، در تابع main، فراخوانی می‌نماییم.

```
void main() {
  ...
  setBadgeName(getBadgeNameFromStorage());
}
```

حال به مانند گامهای قبل برنامه را اجرا و بررسی نمایید.

گام ششم: خواندن نام‌ها از فایل‌های ذخیره شده به فرمت Json

در این گام کلاس PirateName را به گونه‌ای تغییر می‌دهیم که نام‌ها را از فایل Json بخواند. این عمل موجب می‌شود تا به راحتی اسامی مورد نظر را به فایل اضافه نمایید تا توسط کلاس خوانده شوند، بدون آنکه نیاز باشد کد کلاس را مجددا دستکاری کنید.

به منوی File > New File... مراجعه نموده و فایل piratenames.json را با محتوای زیر ایجاد نمایید. این فایل را در پوشه 1-blankbadge و در کنار فایلهای HTML و Dart ایجاد کنید.

```
{ "names": [ "Anne", "Bette", "Cate", "Dawn",
        "Elise", "Faye", "Ginger", "Harriot",
        "Izzy", "Jane", "Kaye", "Liz",
        "Maria", "Nell", "Olive", "Pat",
        "Queenie", "Rae", "Sal", "Tam",
        "Uma", "Violet", "Wilma", "Xana",
        "Yvonne", "Zelda",
        "Abe", "Billy", "Caleb", "Davie",
        "Eb", "Frank", "Gabe", "House",
        "Icarus", "Jack", "Kurt", "Larry",
        "Mike", "Nolan", "Oliver", "Pat",
        "Quib", "Roy", "Sal", "Tom",
        "Ube", "Val", "Walt", "Xavier",
        "Yvan", "Zeb"],
  "appellations": [ "Awesome", "Captain",
        "Even", "Fighter", "Great", "Hearty",
        "Jackal", "King", "Lord",
```

```
        "Mighty", "Noble", "Old", "Powerful",
        "Quick", "Red", "Stalwart", "Tank",
        "Ultimate", "Vicious", "Wily", "aXe", "Young",
        "Brave", "Eager",
        "Kind", "Sandy",
        "Xeric", "Yellow", "Zesty"]}
```

این فایل شامل یک شی Json با دو لیست رشته ای می‌باشد.

به فایل piratebadge.html مراجعه نمایید و فیلد input و المنت button را غیر فعال نمایید.

```
...
  <div>
    <input type="text" id="inputName" maxlength="15" disabled>
  </div>
  <div>
    <button id="generateButton" disabled>Aye! Gimme a name!</button>
  </div>
...
```

این دو المنت پس از اینکه تمامی نام‌ها از فایل Json با موفقیت خوانده شدند فعال می‌گردند.

کتابخانه dart:async را در ابتدای فایل دارت import نمایید

```
import 'dart:html';
import 'dart:math' show Random;
import 'dart:convert' show JSON;

import 'dart:async' show Future;
```

توضیحات

- کتابخانه dart:async برنامه نویسی غیر همزمان یا asynchronous را فراهم می‌کند

- کلاس Future روشی را ارئه می‌کند که در آن مقادیر مورد نیاز در آینده ای نزدیک و به صورت غیر همزمان واکشی خواهند شد.

در مرحله بعد لیست‌های names و appellations را با کد زیر بصورت یک لیست خالی جایگزین نمایید.

```
class PirateName {
  ...
  static List<String> names = [];
  static List<String> appellations = [];
  ...
}
```

توضیحات

- مطمئن شوید که کلمه کلیدی final را از تعاریف فوق حذف نموده اید

- [] معادل new List() می‌باشد

- کلاس List یک نوع Generic می‌باشد که می‌تواند شامل هر نوع شی ای باشد. اگر می‌خواهید که لیست شما فقط شامل داده هایی از نوع String باشد، آن را بصورت List<String> تعریف نمایید.

دو تابع static را بصورت زیر به کلاس PirateName اضافه نمایید

```
class PirateName {
  ...

  static Future readyThePirates() {
    var path = 'piratenames.json';
    return HttpRequest.getString(path)
        .then(_parsePirateNamesFromJSON);
  }

  static _parsePirateNamesFromJSON(String jsonString) {
    Map pirateNames = JSON.decode(jsonString);
    names = pirateNames['names'];
    appellations = pirateNames['appellations'];
  }
}
```

توضیحات - کلاس HttpRequest یک Utility می‌باشد که داده‌ها را از یک آدرس یا URL خاص واکشی می‌نماید.

- تابع getString یک درخواست را به صورت GET ارسال می‌نماید و رشته ای را بر می‌گرداند
- در کد فوق از کلاس Future استفاده شده است که موجب می‌شود درخواست GET بصورت غیر همزمان ارسال گردد.
- زمانیکه Future با موفقیت خاتمه یافت، تابع then فراخوانی می‌شود. پارامتر ورودی این تابع، یک تابع می‌باشد که پس از خاتمه درخواست GET اجرا خواهد شد. به این نوع توابع که پس از انجام یک عملیات خاص بصورت خودکار اجرا می‌شوند توابع CallBack می‌گویند.
- زمانیکه Future با موفقیت خاتمه یافت، اسامی از فایل Json خوانده خواهند شد
- تابع readyThePirates دارای نوع خروجی Future می‌باشد بطوری که برنامه اصلی در زمانی که فایلها در حال خوانده شدن هستند، به کار خود ادامه میدهد و متوقف نخواهد شد

یک متغیر top-level از نوع SpanElement در کلاس PirateName ایجاد کنید.

```
SpanElement badgeNameElement;

void main() {
  ...
}
```

تغییرات زیر را در تابع main ایجاد کنید.

```
void main() {
  InputElement inputField = querySelector('#inputName');
  inputField.onInput.listen(updateBadge);
  genButton = querySelector('#generateButton');
  genButton.onClick.listen(generateBadge);

  badgeNameElement = querySelector('#badgeName');
  ...
}
```

کد زیر را نیز به منظور خواندن نام‌ها از فایل Json اضافه کنید. در این کد اجرای موفقیت آمیز درخواست و عدم اجرای درخواست، هر دو به شکلی مناسب مدیریت شده اند.

```
void main() {
  ...

  PirateName.readyThePirates()
    .then((_) {
      //on success
      inputField.disabled = false; //enable
      genButton.disabled = false;  //enable
      setBadgeName(getBadgeNameFromStorage());
    })
    .catchError((arrr) {
      print('Error initializing pirate names: $arrr');
      badgeNameElement.text = 'Arrr! No names.';
    });
}
```

توضیحات

- تابع readyThePirates فراخوانی شده است که یک Future بر می‌گرداند.
- زمانی که Future با موفقیت خاتمه یافت تابع CallBack موجود در تابع then فراخوانی می‌شود.

- (_) به عنوان پارامتر ورودی تابع then ارسال شده است، به این معنا که از پارامتر ورودی صرف نظر شود.
- تابع then المنت‌های صفحه را فعال می‌کند و داده‌های ذخیره شده را بازیابی می‌نماید
- اگر Future با خطا مواجه شود، توسط تابع catchError که یک تابع CallBack می‌باشد، پیغام خطایی را نمایش می‌دهیم.

برنامه را به مانند گامهای قبل اجرا نموده و نتیجه را مشاهده نمایید