Object - Oriented Programming

LAB #14. Collections, Maps and Iterators

Collection

- 여러 원소들을 담을 수 있는 자료구조

- List, Set, Map 인터페이스 존재

- 담을 객체의 수를 동적으로 정할 수 있음

  ex) ArrayList

- Collections : 인터페이스 / Collection : 클래스

Collections Interface

인터페이스	설명	구현 클래스
List<E>	순서가 있는 데이터의 집합으로, 데이터의 중복을 허용함.	Vector, ArrayList, LinkedList, Stack, Queue
Set<E>	순서가 없는 데이터의 집합으로, 데이터의 중복을 허용하지 않음.	HashSet, TreeSet
Map<K, V>	키와 값의 한 쌍으로 이루어지는 데이터의 집합으로, 순서가 없음. 이때 키는 중복을 허용하지 않지만, 값은 중복될 수 있음.	HashMap, TreeMap, Hashtable, Properties

Collections Methods

메소드	설명
boolean add(E e)	해당 컬렉션(collection)에 전달된 요소를 추가함. (선택적 기능)
void clear()	해당 컬렉션의 모든 요소를 제거함. (선택적 기능)
boolean contains(Object o)	해당 컬렉션이 전달된 객체를 포함하고 있는지를 확인함.
boolean equals(Object o)	해당 컬렉션과 전달된 객체가 같은지를 확인함.
boolean isEmpty()	해당 컬렉션이 비어있는지를 확인함.
Iterator<E> iterator()	해당 컬렉션의 반복자(iterator)를 반환함.
boolean remove(Object o)	해당 컬렉션에서 전달된 객체를 제거함. (선택적 기능)
int size()	해당 컬렉션의 요소의 총 개수를 반환함.
Object[] toArray()	해당 컬렉션의 모든 요소를 Object 타입의 배열로 반환함.

List<T>

- 각 요소의 순서 유지

- 같은 요소의 중복 저장 허용

- 종류 : ArrayList<T>, LinkedList<T>, Vector<T>

LinkedList<T>

- 배열을 사용하여 순차적으로 저장해야 하는 ArrayList 보완
- 저장된 요소는 비순차적으로 저장, 연결은 Link로 구성
- 삽입, 삭제가 빠르나 탐색이 느림
- 단일 연결 리스트와 이중 연결 리스트 존재

LinkedList<T>

```java
import java.util.ArrayList;
import java.util.LinkedList;

public class ListTest {

    public static void main(String[] args) {
        String[] months = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
                           "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

        ArrayList<String> subMonths = new ArrayList<>();
        subMonths.add("Sep");
        subMonths.add("Oct");
        subMonths.add("Nov");

        LinkedList<String> monthList = new LinkedList<>();

        for(String month : months)
            monthList.add(month);

        System.out.println("Size of LinkedList : " + monthList.size());
        System.out.println("Is monthList contains 'April' : " + monthList.contains("April"));
        System.out.println("Is monthList contains 'Sep', 'Oct', 'Nov' : "
                           + monthList.containsAll(subMonths));

        System.out.println("Remove 'Sep', 'Oct', 'Nov' from monthList");
        monthList.removeAll(subMonths);

        System.out.println("Values in monthList : " + monthList.toString());

    }

}
```

```
Size of LinkedList : 12
Is monthList contains 'April' : false
Is monthList contains 'Sep', 'Oct', 'Nov' : true
Remove 'Sep', 'Oct', 'Nov' from monthList
Values in monthList : [Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Dec]
```

Vector<T>

- ArrayList와 같은 동작 수행

- Thread-safe : 한번에 한 스레드만 vector 호출 가능
  - 동기화 지원, 속도는 ArrayList보다 느림

Set<T>

- 각 요소의 순서를 유지하지 않음

- 같은 요소의 중복 저장을 허용하지 않음

- 종류 : HashSet<T>, TreeSet<T>

HashSet<T>

- Hash 알고리즘을 사용하여 검색 속도가 빠름
  - Hash Algorithm? 해시 함수를 이용하여 데이터를 해시 테이블에 저장한 후 해시 함수를 이용하여 다시 검색

- 중복 값 검사 : hashCode() 사용 후 equals() 사용

- 순서를 저장해야 할 시에는 LinkedHashSet<T> 이용 가능

HashSet<T>

```java
public class Fruit {
    private String name;
    private double brix;

    Fruit(String name, double brix) {
        this.name = name;
        this.brix = brix;
    }

    public int hashCode() {
        return (name.toLowerCase() + brix).hashCode();
    }

    public boolean equals(Object o) {
        if(o instanceof Fruit) {
            Fruit f = (Fruit) o;
            return this.name.equalsIgnoreCase(f.name) && (this.brix == f.brix);
        }

        return false;
    }

    public String toString() {
        return "이름 : " + name + "\n당도 : " + brix;
    }
}
```

```java
import java.util.HashSet;
import java.util.Iterator;

public class HashSetDemo {

    public static void main(String[] args) {
        HashSet<Fruit> fruitSet = new HashSet<>();

        fruitSet.add(new Fruit("orange", 3.11));
        fruitSet.add(new Fruit("ORANGE", 3.11));
        fruitSet.add(new Fruit("orange", 4.11));
        fruitSet.add(new Fruit("Apple", 2.17));

        System.out.println("FruitSet's size : " + fruitSet.size());
        Iterator<Fruit> iter = fruitSet.iterator();

        while(iter.hasNext())
            System.out.println(iter.next().toString());

        System.out.println("Does fruitSet contains 'orange' with 3.11 brix? : "
                           + fruitSet.contains(new Fruit("orange", 3.11)));

    }
}
```

TreeSet<T>

- 데이터가 정렬된 상태로 저장
  - 이진 검색 트리의 형태로 저장 : 데이터 추가/제거 속도 빠름

Map<K, V>

- 각 요소의 순서를 유지하지 않음

- 같은 값을 가지는 key는 중복을 허용하지 않으나 같은 값의 value는 중복 허용

- Key는 value를 찾기 위한 이름의 역할

- 종류 : HashMap<K, V>, HashTable<K, V>, TreeMap<K, V>

HashMap<K, V>

- Hash 알고리즘을 사용하여 검색 속도가 빠름

메소드	설명
void clear()	해당 맵(map)의 모든 매핑(mapping)을 제거함.
boolean containsKey(Object key)	해당 맵이 전달된 키를 포함하고 있는지를 확인함.
boolean containsValue(Object value)	해당 맵이 전달된 값에 해당하는 하나 이상의 키를 포함하고 있는지를 확인함.
V get(Object key)	해당 맵에서 전달된 키에 대응하는 값을 반환함. 만약 해당 맵이 전달된 키를 포함한 매핑을 포함하고 있지 않으면 null을 반환함.
boolean isEmpty()	해당 맵이 비어있는지를 확인함.
Set<K> keySet()	해당 맵에 포함되어 있는 모든 키로 만들어진 Set 객체를 반환함.
V put(K key, V value)	해당 맵에 전달된 키에 대응하는 값으로 특정 값을 매핑함.
V remove(Object key)	해당 맵에서 전달된 키에 대응하는 매핑을 제거함.
boolean remove(Object key, Object value)	해당 맵에서 특정 값에 대응하는 특정 키의 매핑을 제거함.
V replace(K key, V value)	해당 맵에서 전달된 키에 대응하는 값을 특정 값으로 대체함.
boolean replace(K key, V oldValue, V newValue)	해당 맵에서 특정 값에 대응하는 전달된 키의 값을 새로운 값으로 대체함.
int size()	해당 맵의 매핑의 총 개수를 반환함.

HashMap<K, V>

```
public class HashMapDemo {

    public static void main(String[] args) {
        HashMap<String, Integer> fruitData = new HashMap<>();
        fruitData.put("pineapple", 5);
        fruitData.put("orange", 3);
        fruitData.put("pineapple", 7);
        fruitData.put("apple", 2);
        fruitData.put("carrot", 0);

        System.out.println("fruitData's size : " + fruitData.size());
        System.out.println("Does fruitData has 'orange' in key? " + fruitData.containsKey("orange"));
        System.out.println("Does fruitData has 10 in value? " + fruitData.containsValue(10));
        System.out.println("Does fruitData has 2 in value? " + fruitData.containsValue(2));

        fruitData.replace("orange", fruitData.get("orange") * 2);
        fruitData.remove("carrot");

        for(String fruit : fruitData.keySet())
            System.out.println("과일 : " + fruit + " 당도 : " + fruitData.get(fruit));

        fruitData.clear();
        System.out.println("Is fruitData empty? " + fruitData.isEmpty());

    }

}
```

Iterator<T> Interface

- Java가 제공하는 인터페이스

- 컬렉션에 저장되어 있는 요소들을 읽어오는 방법을 표준화한 것

- List 같은 집합의 요소에 순차적으로 접근하기 위해 사용

Iterator<T> Interface

메소드	설명
boolean hasNext()	해당 이터레이션(iteration)이 다음 요소를 가지고 있으면 true를 반환하고, 더 이상 다음 요소를 가지고 있지 않으면 false를 반환함.
T next()	이터레이션(iteration)의 다음 요소를 반환함.
default void remove()	해당 반복자로 반환되는 마지막 요소를 현재 컬렉션에서 제거함. (선택적 기능)

Iterator<T> Interface

```java
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;

public class IteratorTest {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        String[] weeks = {"Monday", "Tuesday", "Wednesday", "Thursday",
                          "Friday", "Saturday", "Sunday"};
        ArrayList<String> weekList = new ArrayList<>();
        for(String week : weeks)
            weekList.add(week);

        System.out.print("Write word that you want to find in list: ");
        int index = find(weekList, scan.nextLine());
        if(index == -1)
            System.out.println("Your word doesn't exist in list.");
        else
            System.out.println("The word is found at index " + index);

    }

    private static int find(ArrayList<String> list, String search) {
        Iterator<String> iter = list.iterator();
        int count = 0;
        while(iter.hasNext()) {
            String word = iter.next();
            if(word.equals(search))
                return count;
            count++;
        }
        return -1;
    }
}
```

*오픈소스 소프트웨어 소개

Memo

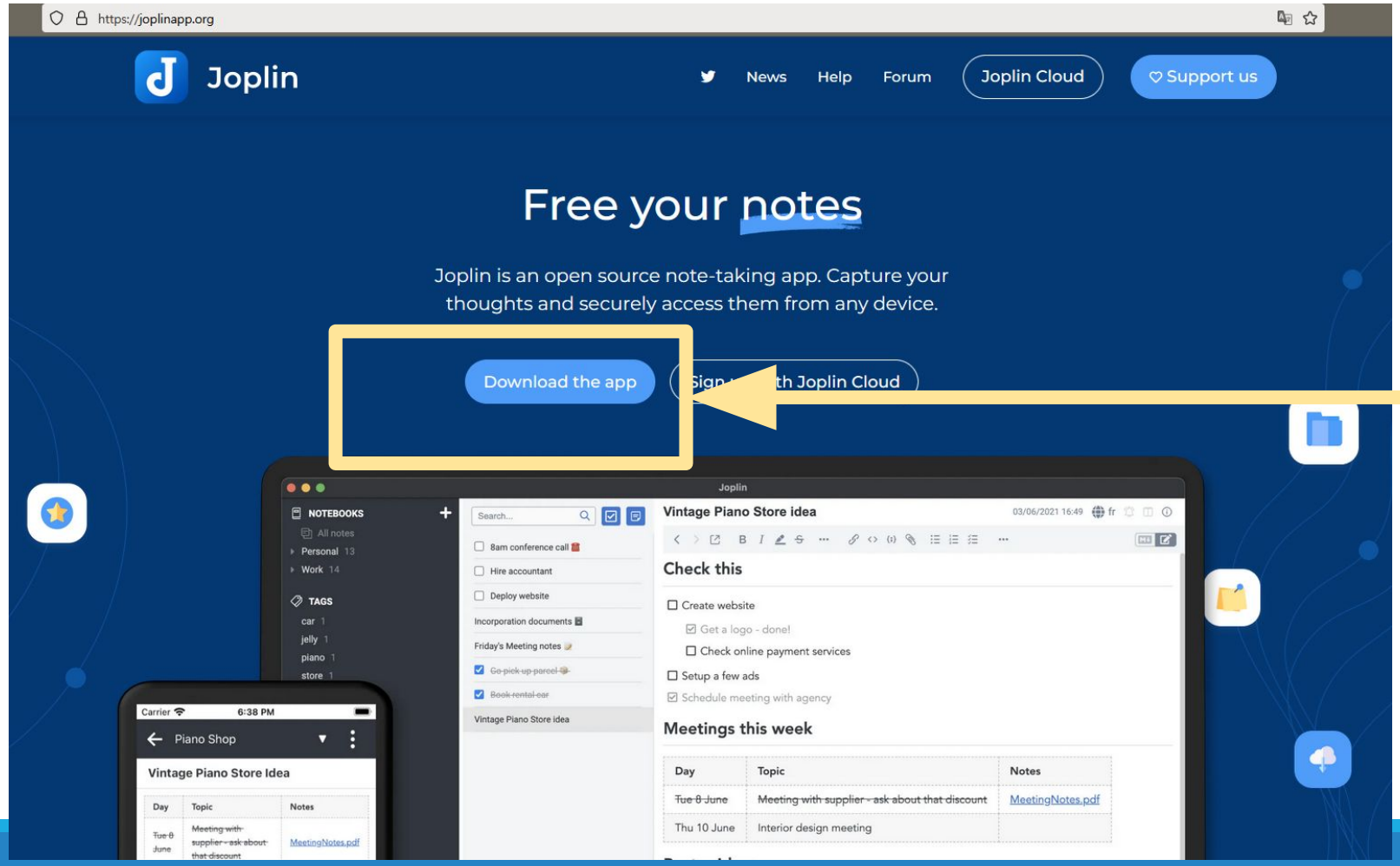
1. joplin

Video Editor

2. losslesscut + video-remove-silence
3. Olive ~~0.1 (old version)~~

joplin

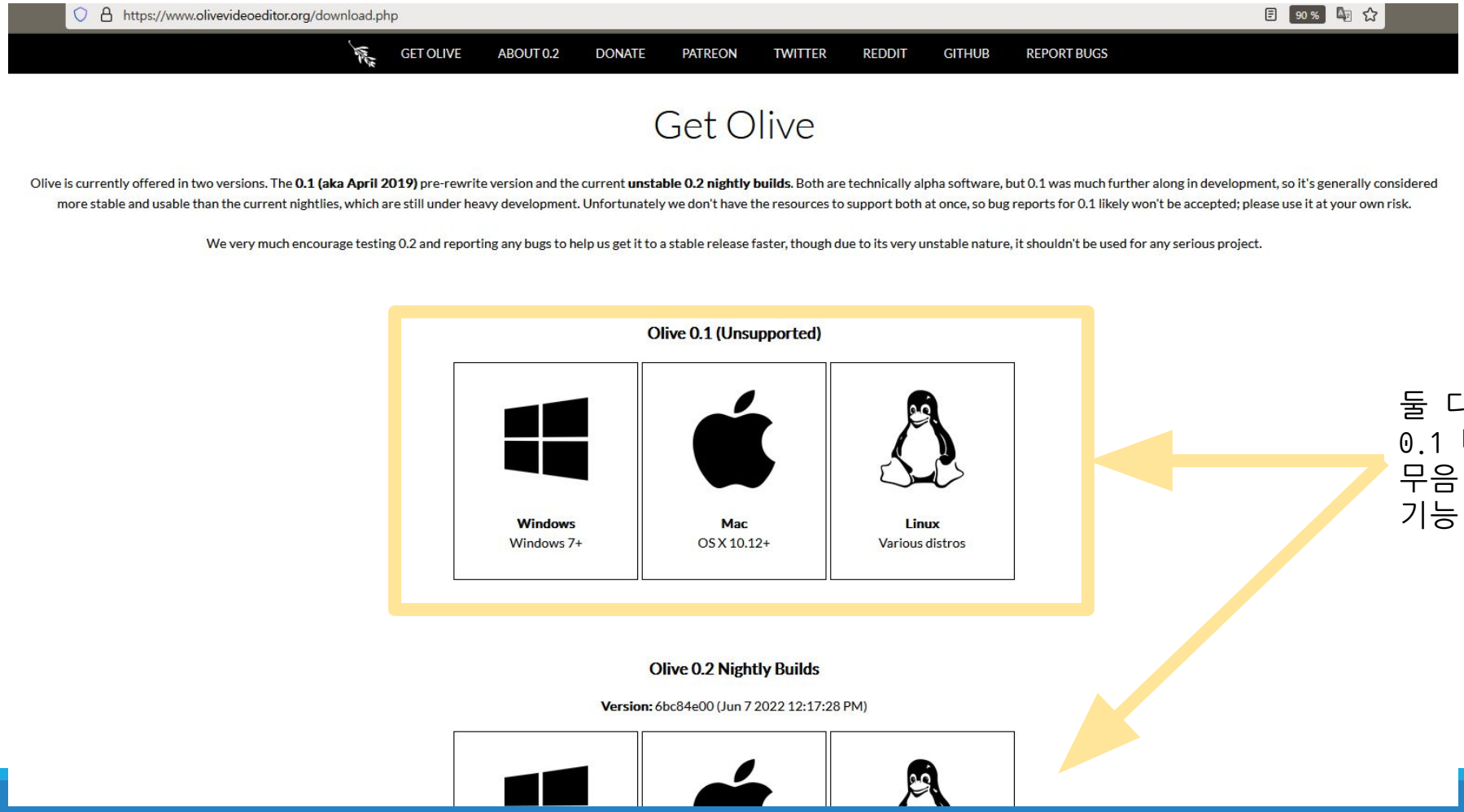
https://github.com/laurent22/joplin
https://joplinapp.org/



여기서 다운로드

olive

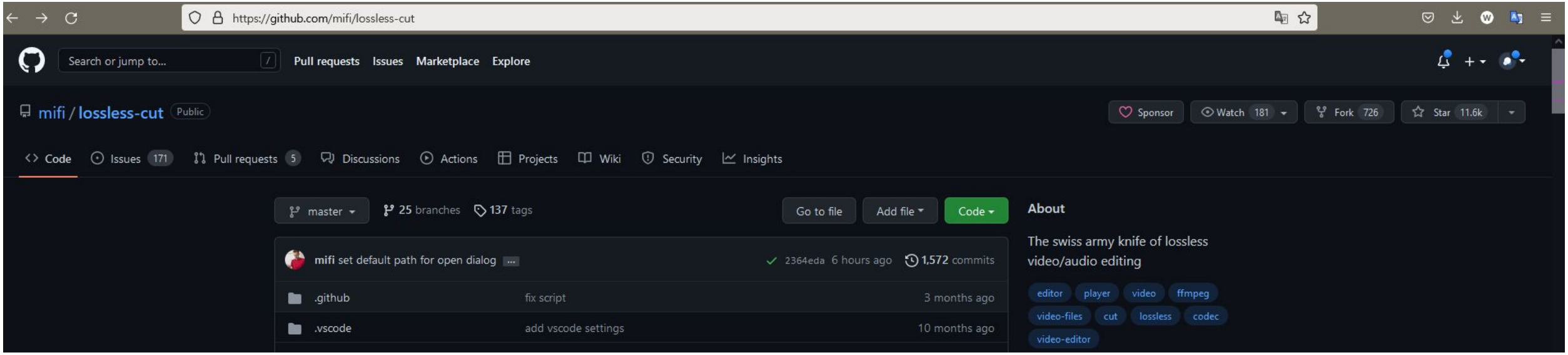
https://github.com/olive-editor/olive
https://olivevideoeditor.org/



둘 다 상관없음
0.1 버전에
무음 구간 자동 구분
기능 있음

losslesscut

https://github.com/mifi/lossless-cut



https://github.com/excitoon/video-remove-silence -> 무음 화면 제거

losslesscut download

Download

If you want to support my continued work on LosslessCut, and you want the advantage of a secure and simple installation process with automatic updates, consider getting it from your favorite store:

Download on the Mac App Store | Get it from Microsoft | Get it from the Snap Store

If you prefer to download the executables manually, this will of course always be free:

- Mac OS X: DMG
- Window: EXE (portable/self-extracting) / ZIP (portable)
- Linux: x64 tar.bz2 / x64 AppImage / arm64 tar.bz2
- More releases

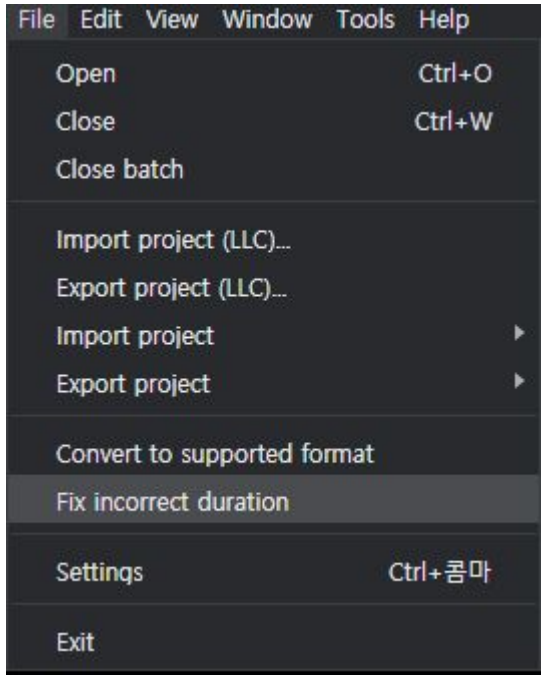
If you find LosslessCut useful, I'm very thankful for donations.

Difference between App Stores and Github download

They have exactly the same in-app features, except for a few platform limitations. Apple doesn't allow opening VOB files with App Store apps. Apple App Store apps need to prompt for output directory. LosslessCut version in the App Stores is a few versions behind the GitHub version, because I want to be sure that the new versions work perfectly before releasing in the App Stores. GitHub version can contain new, untested features and may contain some bugs. I consider the newest GitHub versions to be a public "beta" test.

여기서 다운로드

주의



영상 편집 전 누르기를 추천