工具類別(2)

TYIC 桃高資訊社

列舉

列舉(enumerate)，顧名思義，就是把東西列出來

在 Java 中，列舉就是一個特殊的類別：

1. 列舉類別為不可繼承類別，也不可以繼承類別，但可實作介面
2. 其中的常數就是公開靜態不可變欄位，為該列舉類別的實例
3. 列舉類別的建構子為 private，外界不可實例化

```java
修飾子 enum 列舉類別名稱 {常數1, 常數2, ..., 常數n}
```

```java
修飾子 enum 列舉類別名稱 {
    常數1(引數...), 常數2(引數...), ..., 常數n(引數...);

    欄位...
    方法...
    建構子...
    類別...
}
```

列舉

列舉若實作介面
可以等到創建實例
才覆寫方法

列舉類別的
公開靜態方法
`Role values()`
可以返回該
列舉類別的常數
組成的陣列

```java
public class Main {
    public static void main(String[] args) {
        for (Role role : Role.values()) {
            new Person(role).printInfo().eat();
        }
    }
}

enum Role implements Eat {
    WORKER("上班族") {
        @Override
        public void eat() {
            System.out.println("吃土");
        }
    }, BABY("嬰兒") {
        @Override
        public void eat() {
            System.out.println("喝奶");
        }
    }, STUDENT("學生") {
        @Override
        public void eat() {
            System.out.println("叫外送");
        }
    };

    final String description;

    Role(String description) {
        this.description = description;
    }
}
```

```java
interface Eat {
    void eat();
}

class Person {
    Role role;

    Person(Role role) {
        this.role = role;
    }

    Person printInfo() {
        System.out.println(
                role.description +
                "：" + switch (role) {
            case WORKER -> "上班";
            case BABY -> "哭";
            case STUDENT -> "上課";
        });
        return this;
    }

    void eat() {
        role.eat();
    }
}
```

java

output

```
上班族：上班
吃土
嬰兒：哭
喝奶
學生：上課
叫外送
```

函式介面

Java 提供許多定義好的函式介面可以使用
大部分位於 java.util.function 套件下，常見的有：
Supplier<T> 生產者，不接收返回 T
Consumer<T> 消費者，接收 T 不返回
Function<T, R> 函式，接收 T 返回 R
Predicate<T> 述詞，接收 T 返回 boolean
以及可以輸入兩個值的變種，如：BiConsumer<T, U>
還有許多不使用泛型而是固定型別的變種，如 IntConsumer

集合框架

工具類別常常大量的使用泛型

因為其可以讓使用者減少型別檢查、轉型的動作

最常見的泛型類別就是是集合框架中的所有類別

集合框架主要分為兩個介面：

`java.util.Collection<E>` 和 `java.util.Map<K, V>`

`Collection<E>` 代表一些相同型別的物件放在一起

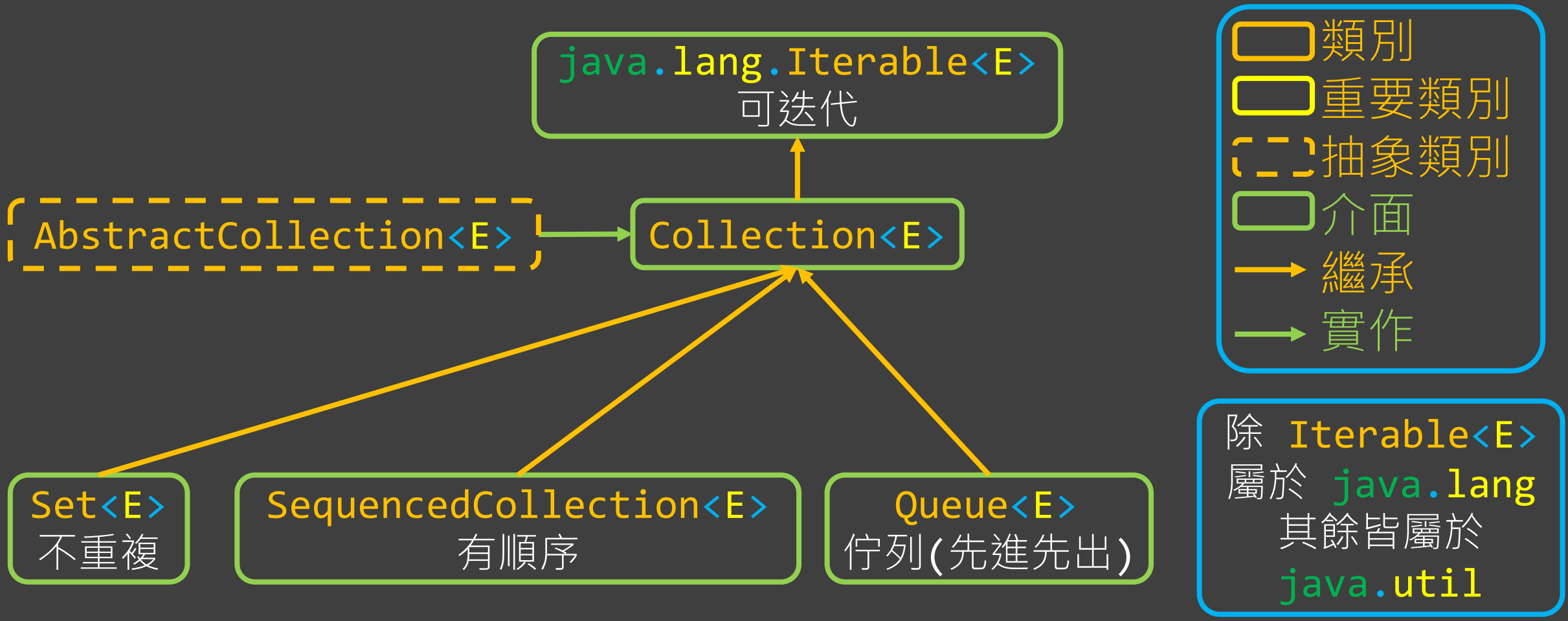
`Map<K, V>` 代表一些相同型別的鍵(key)

各自映射到一個相同型別的值(value)

稱為鍵值映射(key-value mapping)

每組鍵和值稱為鍵值對(key-value pair)

Collection


java.lang.Iterable<E>
可迭代
AbstractCollection<E>
Collection<E>
Set<E>
不重複
SequencedCollection<E>
有順序
Queue<E>
佇列(先進先出)
類別
重要類別
抽象類別
介面
繼承
實作
除 Iterable<E>
屬於 java.lang
其餘皆屬於
java.util

Iterable 與 Iterator

Collection<E> 繼承了 java.lang.Iterable<E> 介面

該介面表示可迭代的，其中有兩個動態方法：

Iterator<E> iterator()

void forEach(Consumer<? super E> action)

Iterator<E> 介面代表迭代器，其中有四個動態方法：

void forEachRemaining(Consumer<? super E> action)

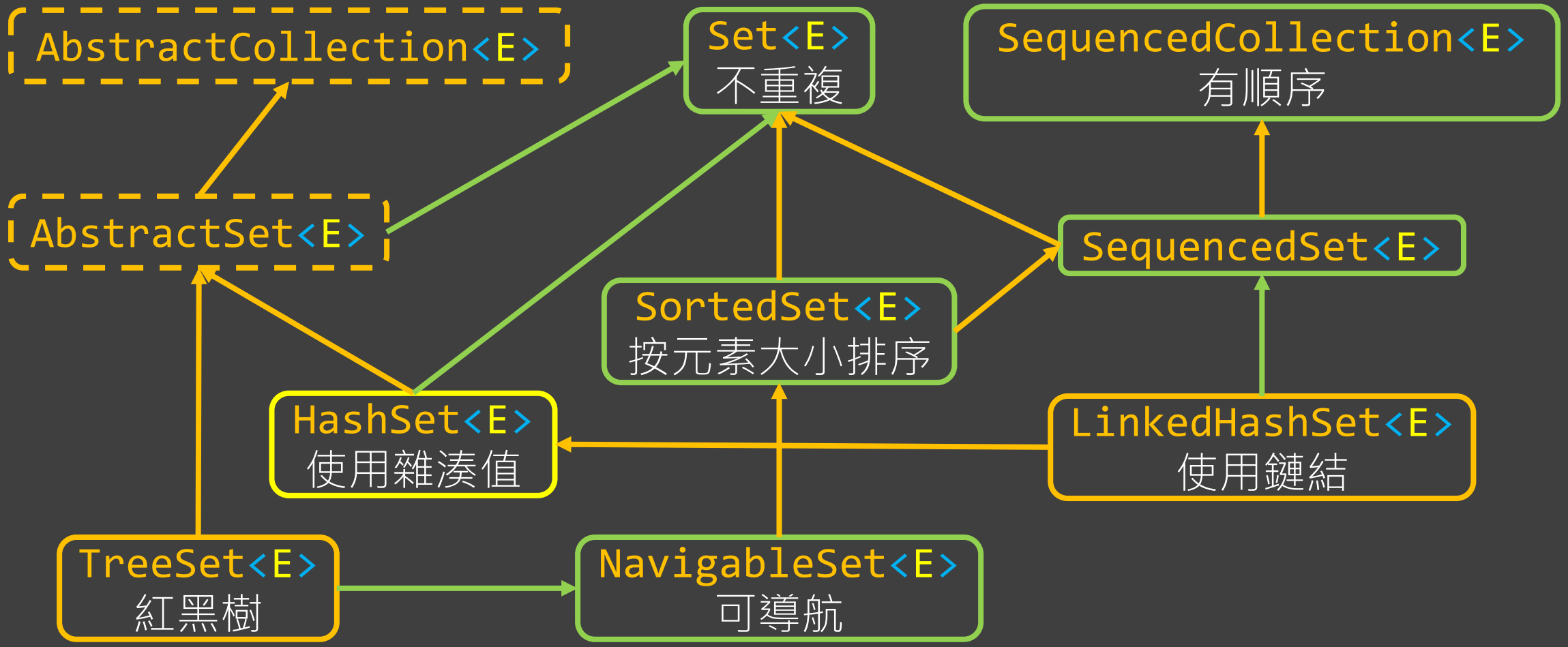
boolean hasNext()、E next()、void remove()

Collection

下方為 Collection<E> 介面部分動態方法：

```
int size()、boolean isEmpty()、void clear()
boolean add(E e)、boolean remove(Object o)
boolean addAll(Collection<? extends E> c)
boolean removeAll(Collection<?> c)
boolean removeIf(Predicate<? super E> filter)
boolean contains(Object o)
boolean containsAll(Collection<?> c)
<T> T[] toArray(T[] a)
```

Set


AbstractCollection<E>
Set<E>
不重複
SequencedCollection<E>
有順序
AbstractSet<E>
SequencedSet<E>
SortedSet<E>
按元素大小排序
HashSet<E>
使用雜湊值
LinkedHashSet<E>
使用鏈結
TreeSet<E>
紅黑樹
NavigableSet<E>
可導航

HashSet

Set<E> 的實作類別儲存的元素不可重複，即數學上的「集合」
其中最重要的實作類別就是 HashSet<E>
其內部使用 HashMap<E, Object> 來實作
HashSet<E> 的元素作為 HashMap<E, Object> 的鍵
而該鍵對應的值皆參考至同一個 Object 物件
HashSet<E> 的子類別 LinkedHashSet<E>
內部則是使用 LinkedHashMap<E, Object>
其餘原理與 HashSet<E> 相同

HashSet

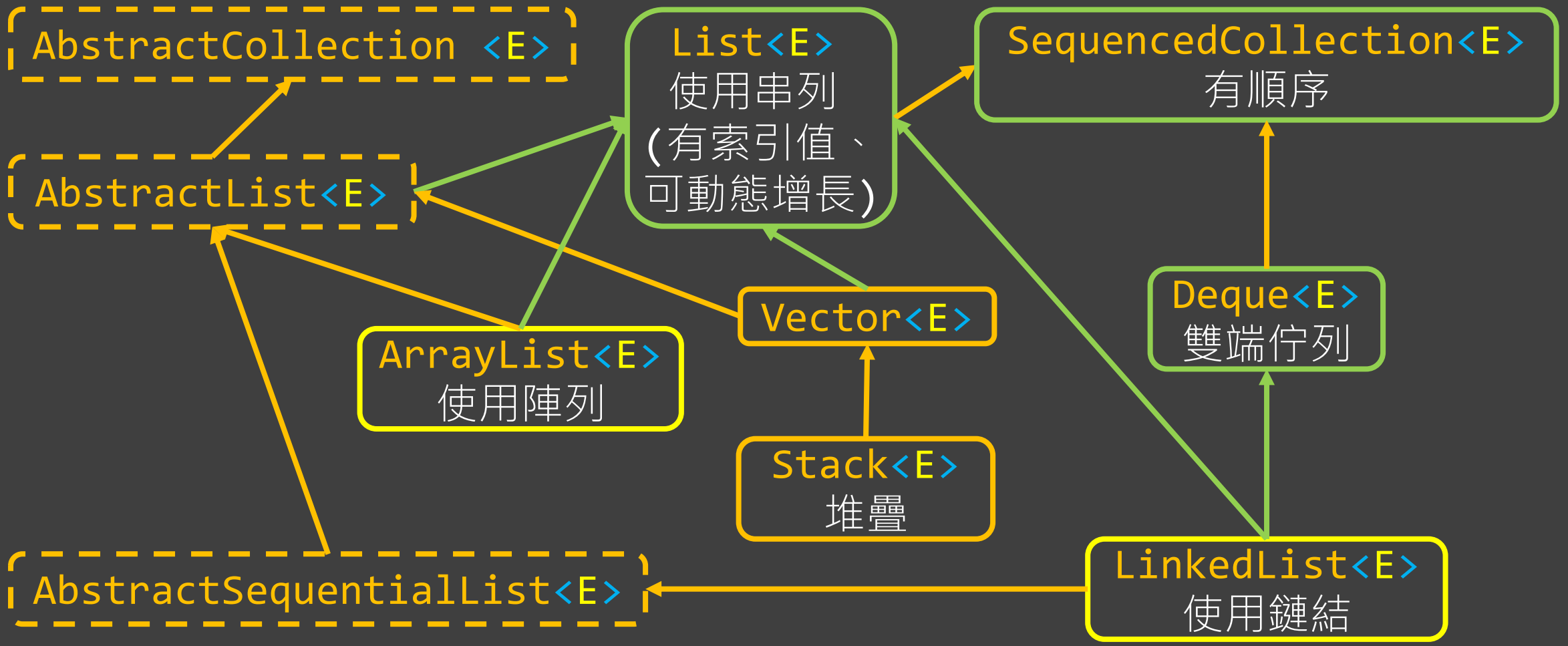
```
[姓名：鄭宜 年齡：30]
[姓名：鄭宜 年齡：30, 姓名：張棟良 年齡：50, 姓名：陳生 年齡：20, 姓名：迪鶯 年齡：40]
[姓名：鄭宜 年齡：30, 姓名：陳生 年齡：20]
姓名：鄭宜 年齡：30
姓名：陳生 年齡：20
[]
```

output

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

public class Main {
    public static void main(String[] args) {
        HashSet<Person> hashSet = new HashSet<>();
        hashSet.add(new Person(30, "鄭宜"));
        System.out.println(hashSet);
        hashSet.addAll(Set.of(
                new Person(20, "陳生"),
                new Person(30, "鄭宜"),
                new Person(40, "迪鶯"),
                new Person(50, "張棟良")
        ));
        System.out.println(hashSet);
        int ageSum = 0;
        for (Person person : hashSet) {
            ageSum += person.age;
        }
        int ageAvg = ageSum / hashSet.size();
        hashSet.removeIf(person -> person.age >= ageAvg);
        System.out.println(hashSet);
        Iterator<Person> it = hashSet.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
            it.remove();
        }
        System.out.println(hashSet);
    }
}
```

```java
class Person {
    protected int age;
    String name;

    Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    @Override
    public String toString() {
        return "姓名：%s 年齡：%d".formatted(name, age);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Person person = (Person) o;
        return age == person.age && Objects.equals(name, person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(age, name);
    }
}
```

List


AbstractCollection <E>
List<E>
使用串列
(有索引值、
可動態增長)
SequencedCollection<E>
有順序
AbstractList<E>
Vector<E>
Deque<E>
雙端佇列
ArrayList<E>
使用陣列
Stack<E>
堆疊
AbstractSequentialList<E>
LinkedList<E>
使用鏈結

List

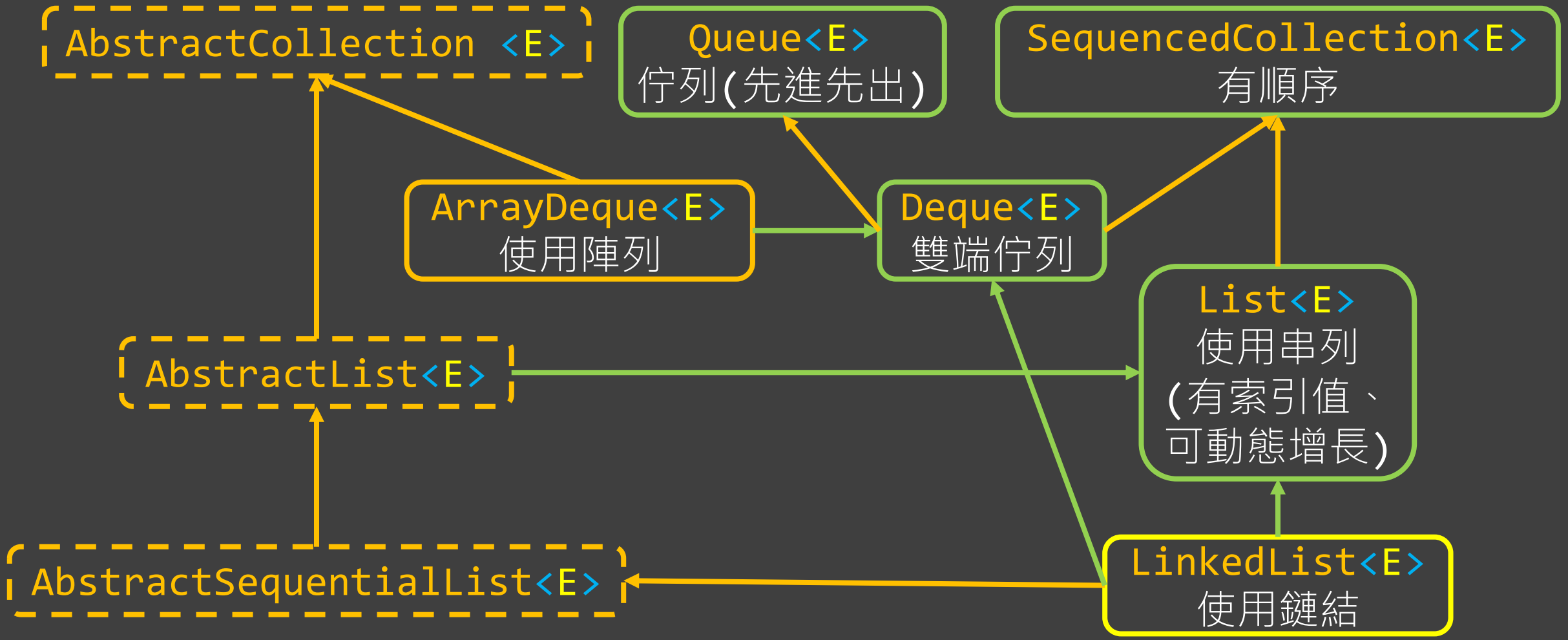
`List<E>` 的實作類別可以像陣列一樣儲存元素
但是儲存容量可以動態增長，也就是說元素的數量可以不固定
下方為該介面的部分動態方法：

該介面最重要的實作類別是 `ArrayList<E>` 和 `LinkedList<E>`

ArrayList

ArrayList<E> 內部使用陣列去實作

Queue


AbstractCollection <E>
Queue<E>
佇列(先進先出)
SequencedCollection<E>
有順序
ArrayDeque<E>
使用陣列
Deque<E>
雙端佇列
List<E>
使用串列
(有索引值、
可動態增長)
AbstractList<E>
AbstractSequentialList<E>
LinkedList<E>
使用鏈結

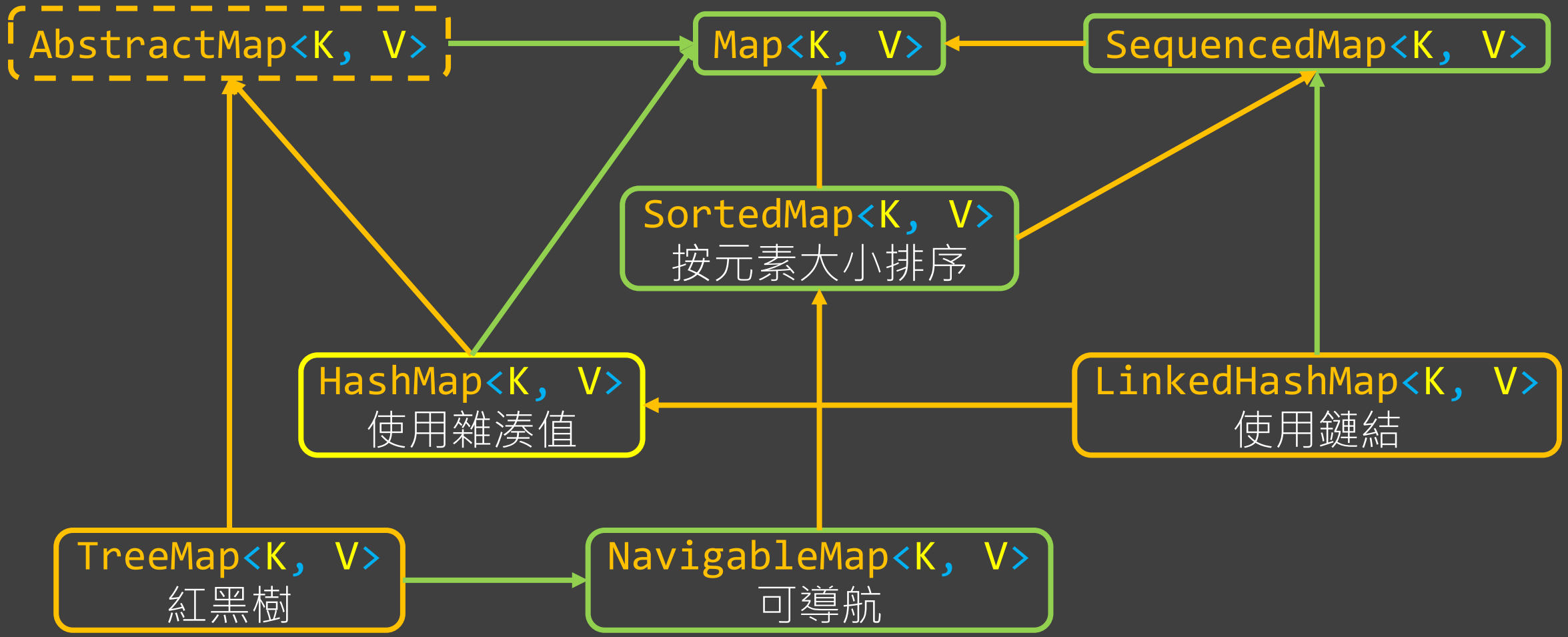
LinkedList

Map

下方為 Map<K, V> 介面部分動態方法：

```
int size()、boolean isEmpty()、void clear()
Collection<V> values()、Set<V> keyset()、Set<Map.Entry<K, V>> entrySet()
V put(K key, V value)、V remove(Object key)
void putAll(Map<? extends E, ? extends V> m)
void replaceAll(BiFunction<? super K, ? super V, ? extends V> function)
boolean containsKey(Object key)、boolean containsValue(Object value)
V get(Object key)、V getOrDefault(Object key, V defaultValue)
void forEach(BiConsumer<? super K, ? super V> action)
```

Map


AbstractMap<K, V>
Map<K, V>
SequencedMap<K, V>
SortedMap<K, V>
按元素大小排序
HashMap<K, V>
使用雜湊值
LinkedHashMap<K, V>
使用鏈結
TreeMap<K, V>
紅黑樹
NavigableMap<K, V>
可導航