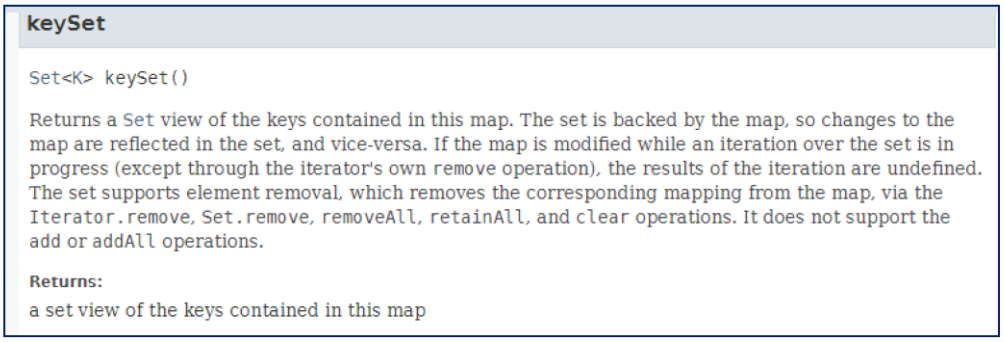
Operations of Abstract Type

Creators create new objects of the type.
Producers create new objects from old objects of the type.
Observers take objects of the abstract type and return objects of a different type.
Mutators change objects.

- `Map.keySet()` is a method from the Java library
  - below is the method's Javadoc documentation, look at its signature
- What kind of operation of an abstract data type is it?



keySet

`Set<K> keySet()`

Returns a `Set` view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator's own `remove` operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the `Iterator.remove`, `Set.remove`, `removeAll`, `retainAll`, and `clear` operations. It does not support the `add` or `addAll` operations.

Returns:
a set view of the keys contained in this map

  - creator
  - producer
  - observer
  - mutator

A good abstract data type (ADT) should be representation independent. (和它是由什么方式实现的，list还是array无关)

Inheritance

Interface inheritance: subclass inherits signatures, but NOT implementation
Implementation inheritance: subclass inherits signatures AND implementation

Static Type vs Dynamic Type

static type specified at declaration and it never changes.
dynamic type specified at instantiation (equal to the type of the object being pointed at).

- What will be printed by those four lines?

```
public interface Animal {
  default void hello(Animal a) {
    print("hello animal"); }
  default void sniff(Animal a) {
    print("sniff animal"); }
  default void cool(Animal a) {
    print("cool animal"); }
}
```

```
public class Dog implements Animal {
  void sniff(Animal a) {
    print("sniff dog"); }
  void cool(Dog d) {
    print("cool dog"); }
}
```

```
Animal a = new Dog();
Dog d = new Dog();
a.hello(d);
a.sniff(d);
d.cool(d);
a.cool(d);
```

What is the output of this line ?

hello animal

- What will be printed by those four lines?

```
public interface Animal {
  default void hello(Animal a) {
    print("hello animal"); }
  default void sniff(Animal a) {
    print("sniff animal"); }
  default void cool(Animal a) {
    print("cool animal"); }
}
```

```
public class Dog implements Animal {
  void sniff(Animal a) {
    print("sniff dog"); }
  void cool(Dog d) {
    print("cool dog"); }
}
```

```
Animal a = new Dog();
Dog d = new Dog();
a.hello(d);
a.sniff(d);
d.cool(d);
a.cool(d);
```

What is the output of this line ?

sniff dog

- What will be printed by those four lines?

```
public interface Animal {
  default void hello(Animal a) {
    print("hello animal"); }
  default void sniff(Animal a) {
    print("sniff animal"); }
  default void cool(Animal a) {
    print("cool animal"); }
}
```

```
public class Dog implements Animal {
  void sniff(Animal a) {
    print("sniff dog"); }
  void cool(Dog d) {
    print("cool dog"); }
}
```

```
Animal a = new Dog();
Dog d = new Dog();
a.hello(d);
a.sniff(d);
d.cool(d);
a.cool(d);
```

What is the output of this line ?

cool dog

- What will be printed by those four lines?

```
public interface Animal {
  default void hello(Animal a) {
    print("hello animal"); }
  default void sniff(Animal a) {
    print("sniff animal"); }
  default void cool(Animal a) {
    print("cool animal"); }
}
```

```
public class Dog implements Animal {
  void sniff(Animal a) {
    print("sniff dog"); }
  void cool(Dog d) {
    print("cool dog"); }
}
```

```
Animal a = new Dog();
Dog d = new Dog();
a.hello(d);
a.sniff(d);
d.cool(d);
a.cool(d);
```

What is the output of this line ?

cool animal
a is Animal, mothed 'cool' in Dog is overload