

# Collections and Lambda

Object Oriented Programming  
2022 First Semester  
Shin-chi Tadaki (Saga University)

- 1 Collections
- 2 Utilities for collections and arrays
- 3 Maps
- 4 Streams
- 5 Lambda expressions

# Today's sample programs

- <https://github.com/oop-mc-saga/Lambda>

# Collections of instances

- Ordered objects: List etc.
- Queue: FirstIn-FirstOut
- Set: not allow the same object to contain more than once
- Map: key-value pairs

# Generic

- Class and method definitions can contain class parameters.
  - Collections have class parameters: indicating class instances contained in.
- When using a class with class parameters
  - Compiler can detect type inconsistency by specifying class parameters

# Example

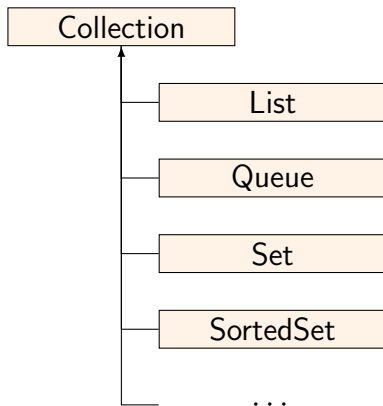
```
1 Student[] students = {  
2     new Student("Tom", 1, 88),  
3     new Student("Jane", 2, 80),  
4     new Student("Ray", 3, 70),  
5     new Student("Kim", 4, 75),  
6     new Student("Jeff", 5, 85),  
7     new Student("Ann", 6, 75),  
8     new Student("Beth", 7, 90)  
9 };  
10 List<Student> studentList = new ArrayList<>();  
11 for (Student s : students) {  
12     studentList.add(s);  
13 }
```

studentList is specified as a list of Student instances.

# java.util.Collection

- The interface for classes containing objects
- Specify the class instances contained
- Fundamental methods are defined as follows:
  - `boolean add()` : add an element
  - `boolean contains()` : check containing the specified element
  - `boolean isEmpty()` : check the collection empty
  - `boolean remove()` : remove the specified element
  - `int size()` : number of elements
  - `Stream stream()` : return stream for iterating elements

# Collection and its extensions



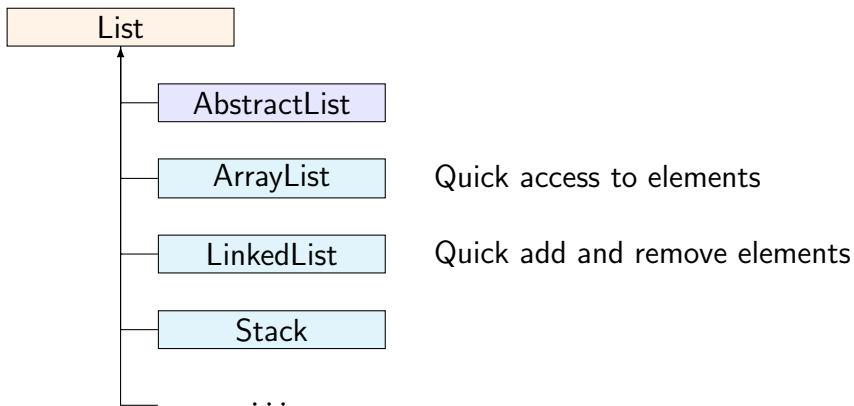
All are defined as interface.



# java.util.List

- Contain ordered elements
- Fundamental methods
  - `boolean add()`: add an element at the end. Throw exception if unsuccessful.
  - `E get()` : return the element at the specified position
  - `int indexOf()` : return the position of the specified element
  - `E set()` : Set the element at the specified position. Returns the element.

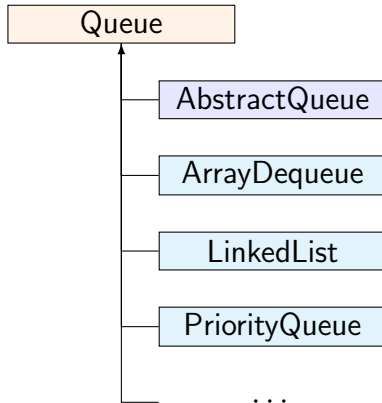
# Implementations of `java.util.List`



# java.util.Queue

- FirstIn-FirstOut
- Methods returning exception at errors
  - `add()`: add element
  - `remove()`: remove specified element
  - `element()`: return the first element
- Methods returning special values at errors
  - `boolean offer()`: add element. return `false` at errors.
  - `E poll()`: remove specified element. return `null` at errors.
  - `E peak()`: return the first element. return `null` at errors.

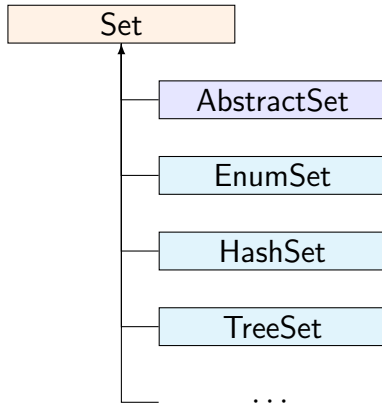
# Implementations of `java.util.Queue`



# java.util.Set

- Set contains elements and not allows the same element to contain more than once.
  - Similarity is decided by `equals()` method.
- fundamental methods
  - `contains()`: return whether the set contains the specified element or not.

# Implementations of `java.util.Set`



# Collections class

## Methods for operate collections

- Search element
- Maximum and minimum element
- Reverse order
- thread protection
- sort
- swap elements
- protecting modification

See `Lambda.CollectionsSample`

```
1  //Search element in list
2  int k = Collections.binarySearch(studentList, students[3]);
3  System.out.println(students[3] + " is found at " + k);
4
5  //Find the maximum element
6  Student best = Collections.max(studentList);
7  System.out.println(best + " marks the best");
8
9  //Sort list
10 System.out.println("sorted list");
11 Collections.sort(studentList);
12 studentList.forEach(
13     s -> System.out.println(s)
14 );
15 System.out.println("-----");
16 //Copy list to array
17 Student[] studentArray = new Student[studentList.size()];
18 studentArray = studentList.toArray(studentArray);
19 for (Student s : studentArray) {
20     System.out.println(s);
21 }
22 System.out.println("-----");
23
24 //Create immutable view of list
25 List<Student> view = Collections.unmodifiableList(studentList);
26 try {
27     Collections.reverse(view);
28 } catch (UnsupportedOperationException e) {
29     System.err.println("This list is immutable.");
30 }
```



# Arrays class

## methods for operating arrays

- convert to list
- search element
- copy array
- compare arrays
- sort
- convert to string

See `Lambda.arraysSample`

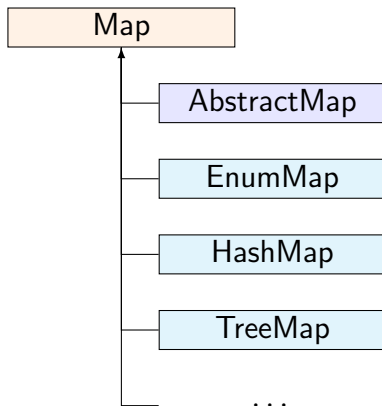
# java.util.Map

- Key-Value pair
- fundamental methods
  - `V get()`: get value specified by Key
  - `Set<K> keySet()`: get a set of key
  - `V put()`: put a key-value pair. Value is update if the key exists.
  - `Collection<V> values()`: get a collection of values.

See `Lambda.mapSample`

```
1 public static void main(String[] args) {
2     String codes[] = {"CTS", "FUK", "HSG", "HND", "KIX"};
3     String names[] = {"Sapporo (New Chitose)", "Fukuoka", "Ariake
4         ↪ Saga",
5         "Haneda", "Kansai"};
6
7     Map<String, String> airports = new HashMap<>();
8     for (int i = 0; i < codes.length; i++) {
9         airports.put(codes[i], names[i]);
10    }
11
12    for (String code : airports.keySet()) {
13        System.out.println(code + "->" + airports.get(code));
14    }
```

# Implementations of `java.util.Map`



# Threads and collections

- Need to prevent multiple threads from accessing collections.
- `Collections.synchronizedList()`
- `Collections.synchronizedSet()`
- etc.

# Operation for all elements in a collection

- extended for

```
1 List<T> list;  
2 for ( T t: list){  
3     do something on t  
4 }
```

- Using Stream and Lambda expressions

# java.util.stream.Stream

- A sequence of element
  - sequential and parallel operations
- fundamental methods
  - `Stream<T> filter()`: filtering elements by predicate
  - `void forEach()`: perform operation for each element
  - `void forEachOrdered()`: perform operation for each element in the order of the stream
  - `Optional<T> reduce()`: Performs a reduction on the elements
  - Arguments are instances of classes in `java.util.function` package.

See `Lambda.lambdaSample`

```
1 public static void main(String[] args) {  
2     int n=100;  
3     List<Double> list = new ArrayList<>();  
4     for(int i=0;i<n;i++){  
5         list.add(Math.random());  
6     }  
7     //print all elements  
8     list.stream().forEach(d -> System.out.println(d));  
9 }
```

- The argument of `forEach()` is an instance of `Consumer` interface.
  - Accept one argument and do some operation without return values.



# Without Lambda

```
1 public static void main(String[] args) {
2     int n = 100;
3     List<Double> list = new ArrayList<>();
4     for (int i = 0; i < n; i++) {
5         list.add(Math.random());
6     }
7     Consumer<Double> c = new Consumer<>(){
8         @Override
9         public void accept(Double d){
10             System.out.println(d);
11         }
12     };
13     //print all elements
14     list.stream().forEach(c);
15 }
```

# Lambda expressions

- Treating functions as an argument of methods
- Using interface mechanisms in Java
- Various functions are defined in `java.util.function`
  - has `apply()` method

# Fundamentals of Lambda expressions

- (arguments)->operation
- type of arguments can be omitted
- () can be omitted for one argument case
- {} can be omitted for one-line operation

# Examples of `java.util.function`

- `BinaryOperator<T>`
  - operation upon two operands of the same type, producing a result of the same type
- `DoubleBinaryOperator`
  - operation upon two double operands, producing a result of `Double`
- `DoubleFunction<R>`
  - operation upon one double operand, producing a result of `R`

```
1 public static void main(String[] args) {  
2     List<Integer> inputList = new ArrayList<>();  
3     for (int i = 0; i < 5; i++) {  
4         inputList.add(i);  
5     }  
6     List<Integer> outputList = listOperation(inputList,  
7         x -> x * x  
8     );  
9     outputList.forEach(  
10        x -> System.out.println(x)  
11    );  
12 }
```

```
1 public static List<Integer> listOperation(List<Integer> inputList,  
2     IntFunction<Integer> func) {  
3     List<Integer> outputList = new ArrayList<>();  
4     for (int i = 0; i < inputList.size(); i++) {  
5         int input = inputList.get(i);  
6         int output = func.apply(input);  
7         outputList.add(output);  
8     }  
9     return outputList;  
10 }
```

# Exercise

Pass a lambda expression for squared sum in `sumAll()` method.