

# Exception & Enum & Collection

---

## Exception & Enum & Collection

Exception Handling

Catch

Finally

Throw & Throws

Customize your own Exception

Enum(Enumerator)

Collection & Map & Object

## Exception Handling

---

- **Error** > can't recover --> crush
  - 跟JVM相关
  - **OutOfMemoryError** (e.g. 内存溢出)
  - **StackOverflowError**(e.g. CPU)
- **Checked Exception** - 必须用 **catch** or **throws**
  - **IOException** (e.g. **FileNotFoundException**)
  - **SQLException** (e.g. Id/data does not exists)
- **Unchecked Exception** - Runtime Exception
  - **NullPointerException** (NPE)

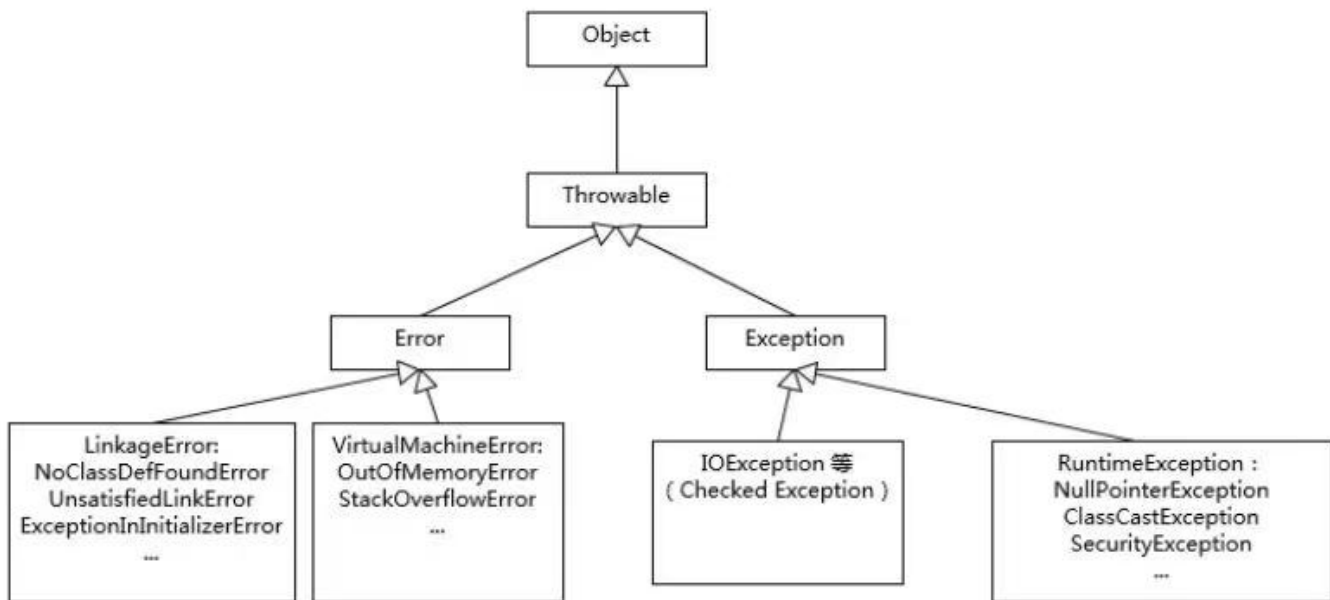
```
1  if (order != null && order.getPrice())
2
3  public Order {
4      private Date date;
5      private Payment payment;
6  }
7
8  if (order != null && order.getPayment() != null &&
    order.getPayment().getPrice()) {
9
10 }
11
12 if (order != null) {
13     return order;
14 } else {
15     try {
16         orderDao.findOrderById(orderId);
```

```

17     } catch (OrderNotFoundException e) {
18         logger.Error("Order # not found");
19         throw new OrderNotFoundException("...");
20     }
21 }

```

- **IndexOutOfBoundsException** (e.g. List/Array)



```

1  List<Integer> list = new ArrayList<Integer>();
2  list={1,2,3,4,5}; //0,1,2,3,4
3  list.get(4); //5
4  list.get(5); //IndexOutOfBoundsException
5
6  // case 1
7  try{
8      //your code
9  }catch(Exception e){
10     // exception is caught here
11     log
12 }finally{ // this is an optional block
13     // will always be executed
14     db.close();
15 }
16
17 // case 2
18 try{
19

```

```

20 }catch(Exception e){
21     // exception is caught here
22 }
23
24 // case 3
25 try{
26
27 }finally{ // this is an optional block
28     // will always be executed
29 }
30

```

Question: Can there be multiple catch blocks? Yes

## Catch

Catch scope should be from **small to large**.

multiple catch:

```

1  try {
2
3  } catch ( )

```

e.g.

```

1  try {
2      orderDao.findById(orderId);
3  } catch (OrderNotFoundException e) { //碟子掉“地上”了.
4      // basic logic
5      logger.info(e);
6  } catch (DBConnectionException e) {
7      //Can't connect to DB
8      logger.info(e);
9  } catch (Exception e) { //碟子掉了 (掉哪里了?地上/水里/天上)
10     logger.info(e);
11 }
12
13 // Wrong
14 try {
15     orderDao.findById(orderId);
16 } catch (Exception e) {

```

```
17     logger.info(e);
18 } catch (OrderNotFoundException e) {
19     logger.info(e);
20 }
```

Question: Can there be multiple finally blocks? No

Question: When both catch and finally return values, what will be the final result?

```
1  try {
2      orderDao.findOrderById(orderId);
3  } catch (OrderNotFoundException e) {
4      // basic logic
5      return 3;
6  } finally {
7      return 5;
8  }
9
10 // 5
```

If both `catch` and `finally` return, the receiving method will get the returned value from the `finally` block

Question: Why finally always be executed ?

imagine you opened a file, get an exception, then throwed or returned, but never closed. that's the reason why finally always be executed.

## Finally

- `finally` can **only be used once** with a try or try-catch block.
- `finally` is **optional** in the try-catch block.
- `finally` **will be executed whether or not the exception is handled**.
- `finally` will **still be executed** if there is a `return` statement in the `catch` clause.

# Throw & Throws

```
1  throw new RuntimeException(); // throw
2
3  public void getOrder(String orderId) throws OrderNotFoundException { //throws
4
5  }
```

need to throws or try catch when call the method who throws exception

```
1  try {
2      getOrder("123");
3  } catch (OrderNotFoundException e) {
4      ...
5  } catch (Exception e) {
6      ...
7  }
8
9
10 public void updateOrder(Order order) throws OrderNotFoundException {
11     Order order = getOrder(order.getID());
12     order.setStatus(Constants.CANCEL);
13     ....
14 }
```

```
1  try {
2      getOrder("123");
3  } catch (OrderNotFoundException e | UserNotFoundException e1 | SQLException e2 |
4      Exception e) {
5      ...
6  }
```

try(String s = "Hello") {} 及时释放资源

```
1  try(Order order = new order();
2      User user = new User()) {
3  }
```

## Customize your own Exception

```
1 public class OrderNotFoundException extends Exception {
2     public OrderNotFoundException(String errorMessage) {
3         super(errorMessage);
4     }
5 }
```

## Enum(Enumerator)

```
1 enum Season {
2     WINTER,
3     SPRING,
4     SUMMER,
5     FALL
6 }
7
8 enum CarType {
9     SEDAN,
10    TRUCK,
11    PICK_UP,
12    RV
13 }
```

Every element is in vlaues (Season.values)

```
1 for (Season s : Season.values()){
2     System.out.println(s);
3 }
4 // WINTER
5 // SPRING
6 // SUMMER
7 // FALL
```

Every element is a contructor

```
1 public enum Season2 {
2     WINTER(5),
3     SPRING(10),
4     SUMMER(15),
5     FALL(20);
6 }
```

```

7         private int value;
8
9         private Season2(int value) {
10             this.value = value;
11         }
12
13         public int getValue() {
14             return value;
15         }
16     }
17
18     public class Weahter{
19         //can't guarantee the season value
20         public int getTemp(String season){
21             if(season != Season2){
22                 throw new NotSupportedSeasonException()
23             }
24         }
25
26         //make sure it always 1 out of 4 season
27         public int getTemp(Season2 season){
28
29         }
30     }

```

### A popular template of enmu

1. **Interface A** -> getCode, getMessage
2. **enum B** implements the **interface A**
3. private enum constructor
4. An exception can aggregate the interface/enum

```

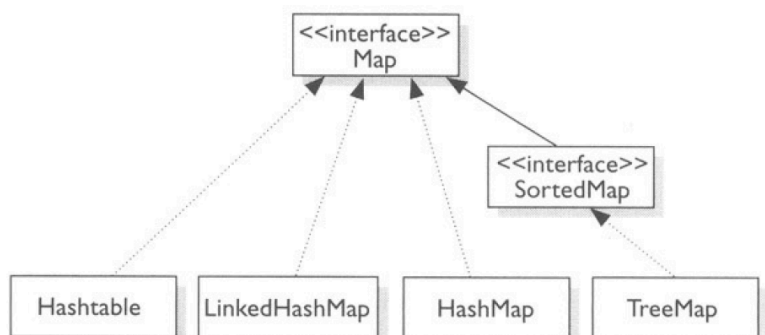
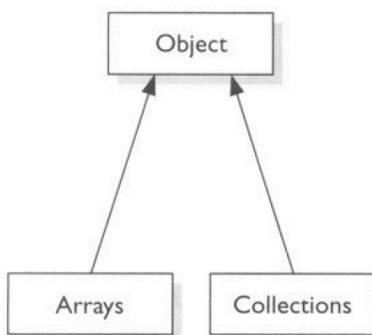
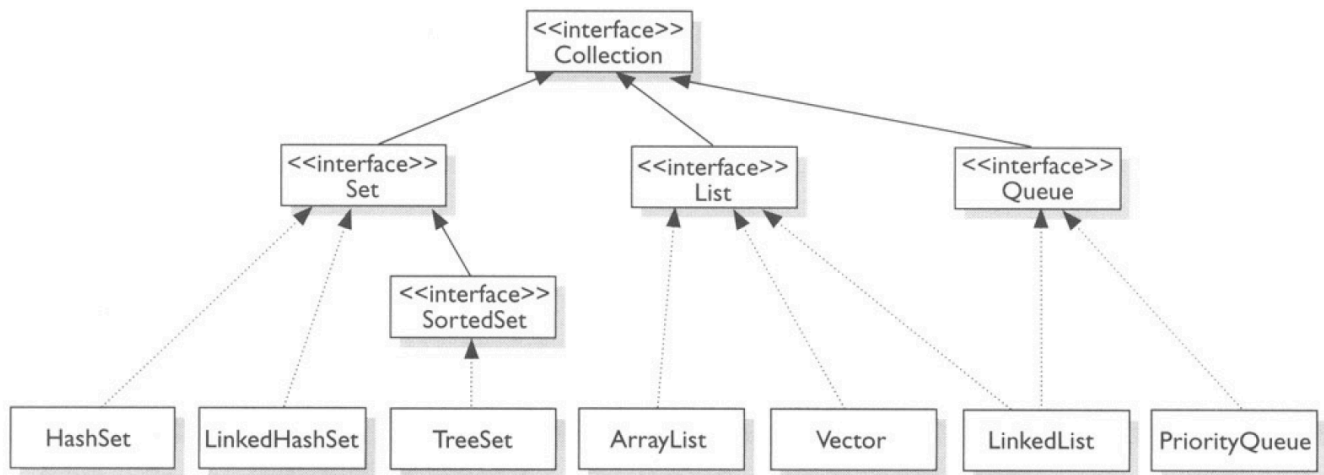
1     public interface IErrorCode {
2         long getCode();
3         String getMessage();
4     }
5
6     public enum ResultCode implements IErrorCode {
7         SUCCESS(200, "操作成功"),
8
9         FAILED(500, "操作失败"),
10        VALIDATE_FAILED(404, "参数检验失败"),
11        UNAUTHORIZED(401, "暂未登录或token已经过期"),
12        FORBIDDEN(403, "没有相关权限");
13
14        // ...

```

```
14     LOVEDAY(520, "Love Day Message");
15
16     private long code;
17     private String message;
18
19     private ResultCode(long code, String message) {
20         this.code = code;
21         this.message = message;
22     }
23
24     @Override
25     public long getCode() {
26         return code;
27     }
28
29     @Override
30     public String getMessage() {
31         return message;
32     }
33 }
34
35 public class ApiException extends RuntimeException {
36     private IErrorCode errorCode;
37
38     public ApiException(IErrorCode errorCode) {
39         super(errorCode.getMessage());
40         this.errorCode = errorCode;
41     }
42
43     public ApiException(String message) {
44         super(message);
45     }
46
47     public ApiException(Throwable cause) {
48         super(cause);
49     }
50
51     public ApiException(String message, Throwable cause) {
52         super(message, cause);
53     }
54
55     public IErrorCode getErrorCode() {
56         return errorCode;
57     }
58 }
```

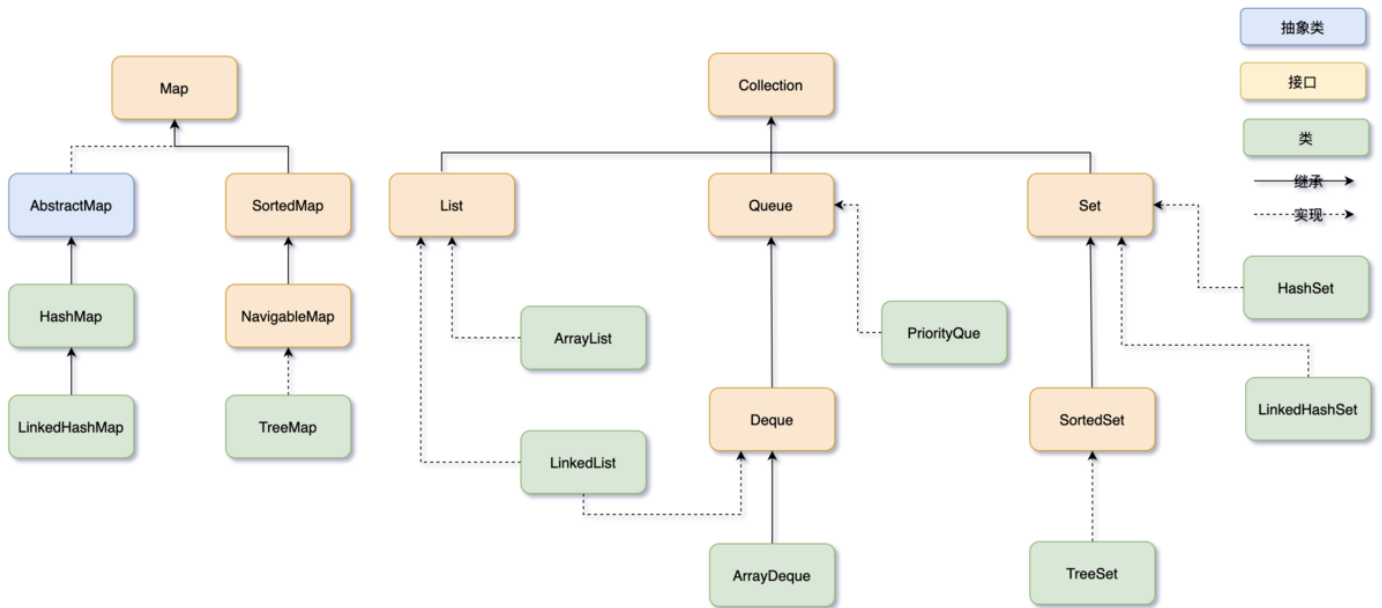


# Collection & Map & Object



implements

extends



assignments -> 根据列出的方法名，Google学习怎么用，别在下方的methods里写出相关的例子。

- ▼ java-core
  - ▼ src
    - ▼ main
      - ▼ java
        - ▼ com.chuwa
          - ▼ exercise
            - ▼ collection
 

- 🔗 AdditionalMapExerciseTest
              - 🔗 ArrayListExerciseTest
              - 🔗 ArraysExerciseTest
              - 🔗 CollectionsExerciseTest
              - 🔗 CopyOnWriteArrayListExerciseTest
              - 🔗 HashMapExerciseTest
              - 🔗 HashSetExerciseTest
              - 🔗 LinkedHashMapExerciseTest
              - 🔗 LinkedListExerciseTest
              - 📄 README.md
              - 🔗 TreeMapExerciseTest
              - 🔗 TreeSetExerciseTest
          - > database
          - > tutorial

```

12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

    * e.g.
    * List<Integer> list = new LinkedList<Integer>();
    * Inserting:
    * add(E e) or addLast(E e)
    * addFirst(E e)
    * add(int index, E element)
    * addAll(Collection c)
    * addAll(int index, Collection c)
    *
    * Retrieving:
    * getFirst()
    * getLast()
    * get(int index)
    *
    */
@Test
public void learn_Inserting_And_Retrieving() {

```