Exception & Enum & Collection

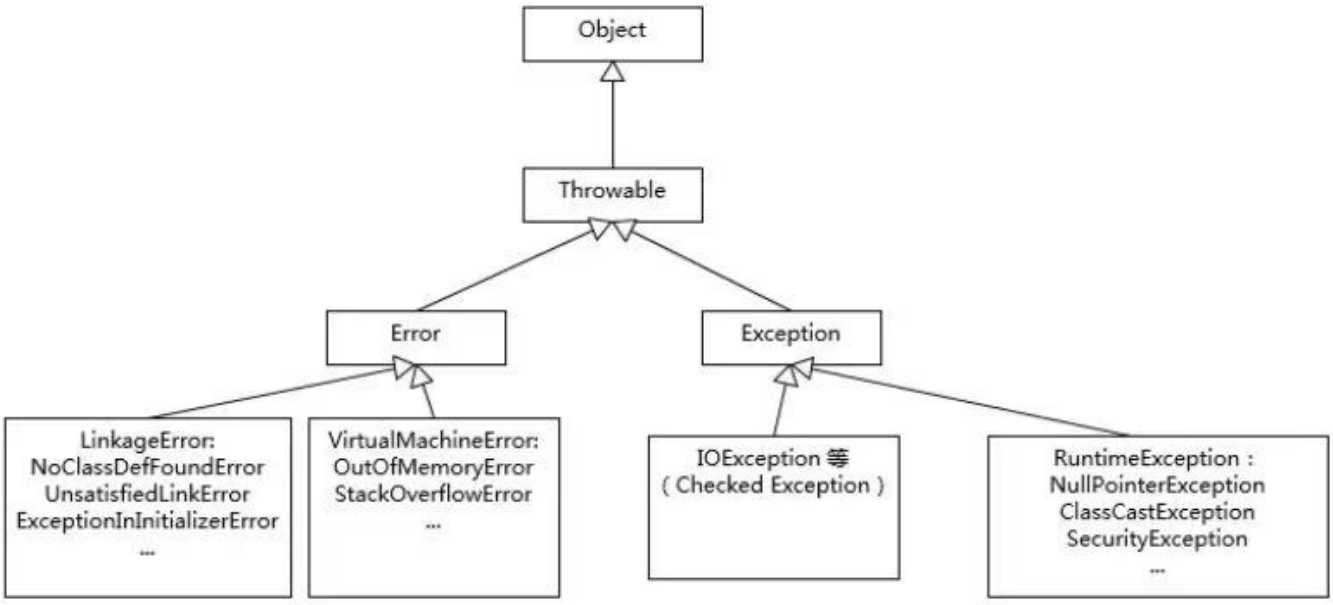
Exception Handling

- **Error**
  - 跟JVM相关
  - **OutOfMemoryError** (e.g. 内存溢出)
  - **StackOverFlowError**(e.g. CPU)
- **Checked Exception** - 必须用 **catch** or **throws**
  - **IO**Exception (e.g. **FileNotFound**)
  - **SQL**Exception (e.g. Id/data does not exits)
- **Unchecked Exception** - Runtime Exception
  - **NullPointerException** (NPE)

```
if (order != null && order.getPrice())

public Order {
  private Date date;
  private Payment payment;
}

if (order != null && order.getPayment() != null &&
order.getPayment().getPrice()) {

}

if (order != null) {
  return order;
} else {
  try {
    orderDao.findOrderById(orderId);
```

```
  } catch (OrderNotFoundException e) {
    logger.Error("Order # not found");
    throw new OrderNotFoundException("...");
  }
}
```

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




```
List<Integer> list = new ArrayList<Integer>();
list={1,2,3,4,5}; //0,1,2,3,4
list.get(4); //5
list.get(5); //IndexOutOfBoundsException

// case 1
try{

}catch(Exception e){
   // exception is caught here
}finally{ // this is an optional block
   // will always be executed
}

// case 2
try{

}catch(Exception e){
   // exception is caught here
```

```
}

// case 3
try{

}finally{ // this is an optional block
   // will always be executed
}

```

Question: Can there be multiple catch blocks? Yes

Catch

Catch scope should be from small to large.

multiple catch:

```
try {

} catch ()
```

e.g.

```
try {
  orderDao.findOrderById(orderId);
} catch(OrderNotFoundException e) { //碟子掉“地上”了.
  // busic logic
  logger.info(e);
} catch(Exception e) {  //碟子掉了 (掉哪里了?地上/水里/天上)
  logger.info(e);
}

// Wrong
try {
  orderDao.findOrderById(orderId);
} catch(Exception e) {
  logger.info(e);
} catch(OrderNotFoundException e) {
  logger.info(e);
}
```

Question: Can there be multiple finally blocks? No

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

```
try {
  orderDao.findOrderById(orderId);
  //return 1;
} catch(OrderNotFoundException e) {
  // busic logic
  return 3;
} finally {
  return 5;
}

// 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

```
throw new RuntimeException();  // throw

public void getOrder(String orderId) throws OrderNotFoundException, DBException,
Exception { //throws

}
```

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

```
try {
  getOrder("123");
} catch (OrderNotFoundException e) {
  ...
} catch (Exception e) {
  ...
}


public void updateOrder(Order order) throws OrderNotFoundException {
  Order order = getOrder(order.getID());
  order.setStatus(Constants.CANCEL);
  ....
}
```

```
try {
  getOrder("123");
} catch (OrderNotFoundException e | UserNotFoundException e1 | SQLException e2 ) {
  ...
}
```

try(String s = "Hello") {} 及时释放资源

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

Customize your own Exception

```
public class OrderNotFoundException extends Exception {
    public OrderNotFoundException(String errorMessage) {
        super(errorMessage);
    }
}
```

Enum(Enumerator)

```
enum Season {
  WINTER,
  SPRING,
  SUMMER,
  FALL
}
```

Every element is in vlaues (Season.values)

```
for (Season s : Season.values()){
    System.out.println(s);
}
// WINTER
// SPRING
// SUMMER
// FALL
```

Every element is a contructor

```
public enum Season2 {
        WINTER(1),
        SPRING(2),
        SUMMER(2),
        FALL(3);

        private int value;

        private Season2(int value) {
                this.value = value;
        }

        public int getValue() {
                return value;
        }
}

public enum Day {
        MONDAY(1),
        TUESDAY(2),
        ....
        SUNDAY(7);

        private int value;

        private Day(int value) {
```

```
                this.value = value;
        }

        public int getValue() {
                return value;
        }
}
```

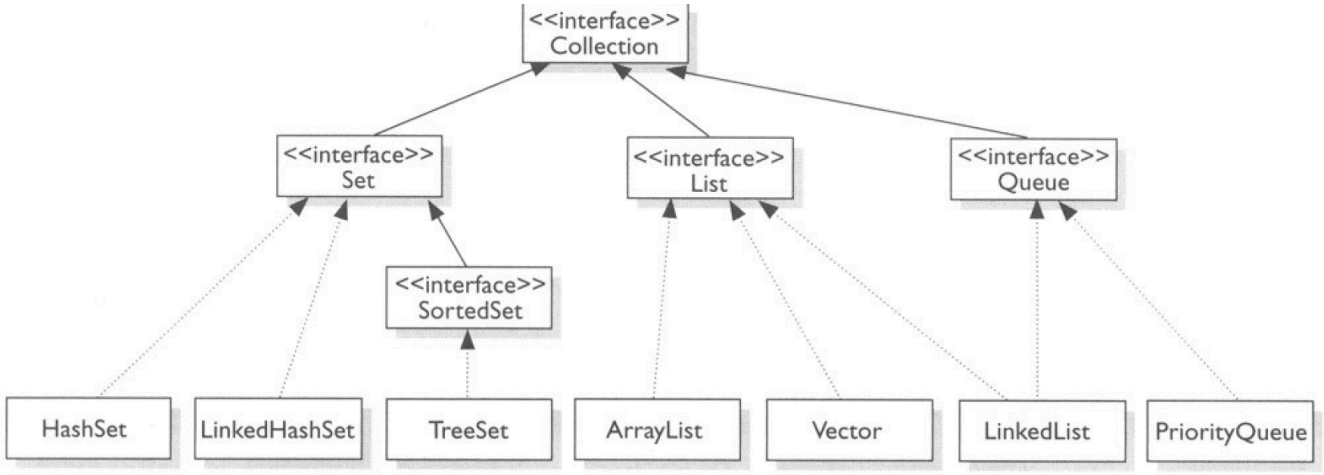
A popular template of enmu

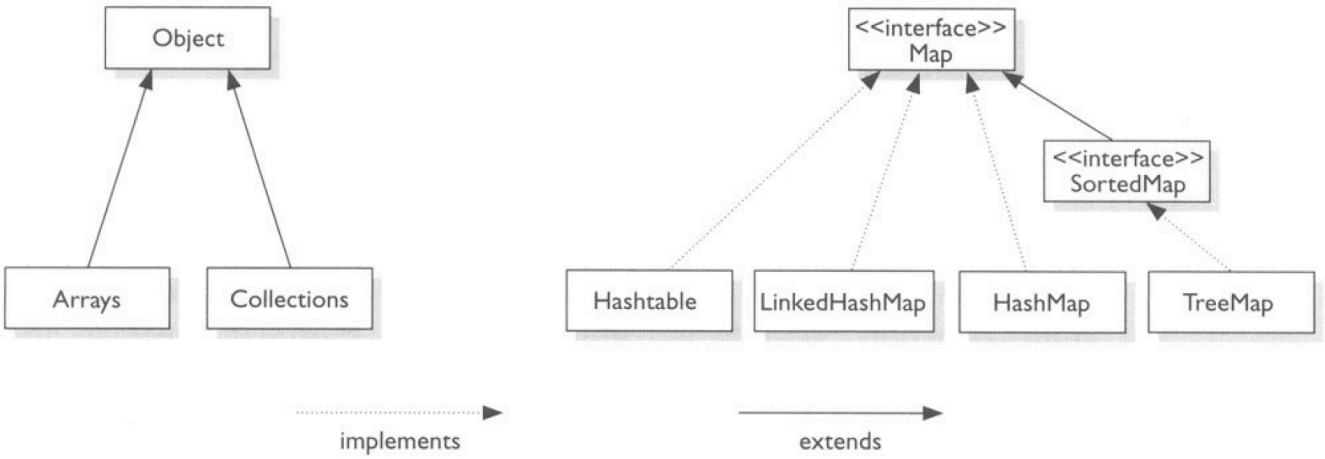
1. **Interface A** -> getCode, getMessage
2. **enum B** implements the **interface A**
3. private enum constructor
4. An exception can aggregate the interface/enum

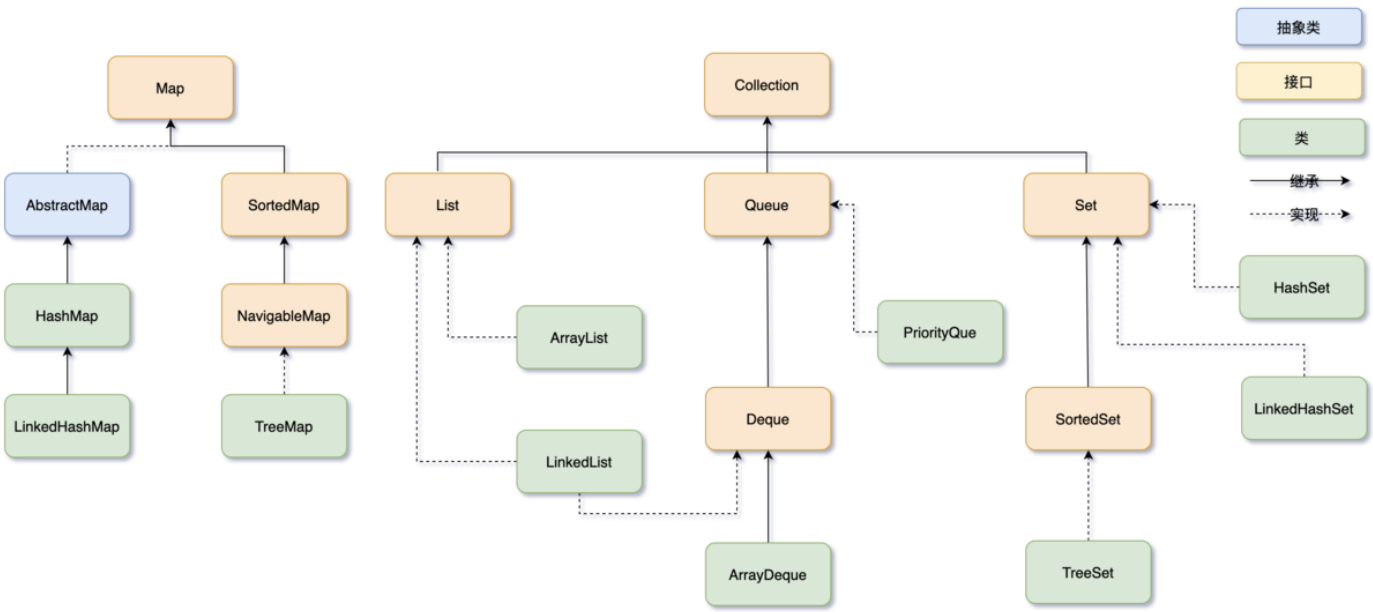
```
public interface IErrorCode {
    long getCode();
    String getMessage();
}

public enum ResultCode implements IErrorCode {
    SUCCESS(200, "操作成功"),
    FAILED(500, "操作失败"),
    VALIDATE_FAILED(404, "参数检验失败"),
    UNAUTHORIZED(401, "暂未登录或token已经过期"),
    FORBIDDEN(403, "没有相关权限");

    LOVEDAY(520, "Love Day Message");

    private long code;
    private String message;

    private ResultCode(long code, String message) {
        this.code = code;
        this.message = message;
    }

    @Override
    public long getCode() {
        return code;
    }

    @Override
    public String getMessage() {
        return message;
```

```
    }
}

public class SomeOtherErrorCode implements IErrorCode {
  //
}

public class ApiException extends RuntimeException {
    private IErrorCode errorCode;

    public ApiException(IErrorCode errorCode) {
        super(errorCode.getMessage());
        this.errorCode = errorCode;
    }

    public ApiException(String message) {
        super(message);
    }

    public ApiException(Throwable cause) {
        super(cause);
    }

    public ApiException(String message, Throwable cause) {
        super(message, cause);
    }

    public IErrorCode getErrorCode() {
        return errorCode;
    }
}
```

Collection & Map & Object







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

12 * e.g.


src
main
java
com.chuwa
exercise
collection
AdditionalMapExerciseTest
ArrayListExerciseTest
ArraysExerciseTest
CollectionsExerciseTest
CopyOnWriteArrayListExerciseTest
HashMapExerciseTest
HashSetExerciseTest
LinkedHashMapExerciseTest
LinkedListExerciseTest
README.md
TreeMapExerciseTest
TreeSetExerciseTest
database
tutorial
* 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() {
}