Java-07

一、Task1.集合

1. **Collection**
    - **List**
        - **ArrayList**

          原理: **动态数组** (因此插入元素会导致后续每个元素移动)

          访问: 由于元素在内存中的储存是连续的，因此通过索引访问元素的 时间复杂度为 **O(1)** 支持快速随机访问

          插入和删除: 由于插入元素会导致后续每个元素移动，插入和删除较 慢，时间复杂度 **O(n)**
        - **LinkedList**

          原理: **双向链表**

          访问: 访问元素需要遍历链表，故时间复杂度 **O(n)**

          插入和删除: 插入和删除元素只需要改变相邻指针，

          理论时间复杂度 **O(1)** ，但是需要先找到插入或删除位置，

          因此实际时间复杂度 **O(n)**
    - **Set**
        - **HashSet**

          原理: 基于 **HashMap**

          元素顺序: 不保证元素顺序

          访问，插入和删除: 由于基于 **HashMap**

          因此时间复杂度通常是 **O(1)**

          但数据量大时 **HashMap** 将使用 **红黑树** 储存，

          此时的时间复杂度为 **O(log n)**

          null: 允许一个null元素
        - **TreeSet**

          原理: 基于 **红黑树**

          元素顺序: 基于输入的自然顺序或 **Comparator**

          访问，插入和删除: 时间复杂度为 **O(log n)**

          null: 不允许null存在 (**原因:** 无法排序)
2. **Map**
    - **HashMap**

      原理:

      元素顺序: 不保证元素顺序(子类 **LinkedHashMap** 可以固定顺序)

      性能: 由储存原理可知，通常时间复杂度是 **O(1)**

      数据量大或特殊时时间复杂度是 **O(log n)**
    - **TreeMap**

原理: 基于**红黑树**

元素顺序: 基于输入的自然顺序或 **Comparator**

性能: 时间复杂度为 **O(log n)**

3. **集合和数组**
    1. 相同点: 可以储存多个元素，可以查找元素
    2. 不同点:

	数组	集合
大小	固定长度，创建时确定	动态大小，可以扩容
性能	内存连续，时间复杂度O(1)	视具体情况而异
功能	功能较少	可以调用接口
存储内容	基本类型和对象	对象(但是可以自动装箱从而存储基本类型)

    3. 集合的优越性:
        - 功能更丰富
        - 大小可以自动扩容

二、Task2.遍历

代码注释如下:

```
//list是一个存在的数组或集合
//list.forEach()即对list的每个元素进行遍历
list.forEach(new Consumer<Integer>()
    //建立一个Consumer的匿名内部类(实现类)以实现方法如下
    {
    //方法accept重写
    @Override
    //接下来会将list的每个元素输入accept方法中
    public void accept(Integer integer) {
        System.out.println(integer);
    }
});
```

- **匿名内部类:**

  为了解决想使用某个接口/类的实现方法，而需要创建实现类/子类去实现/重写

  可以使用 **匿名内部类** 减少代码冗杂

```
classname name = new classname(参数){
 @Override
 void method() {

  }
};
//使用classname.method引用方法
```

- **四大函数式接口:**
  - Consumer 接受一个输入参数并且无返回值

    具体方法: accept()
  - Supplier 无输入参数，返回一个结果

    具体方法: get()

    先定义返回值，再用get()得到返回值
  - Function 接受一个输入参数，返回一个结果

    具体方法: apply()

    有两种实现复合函数的方式，只是角度不同
    - andThen

      A.andThen(B)即执行A后输入B中执行B
    - compose

      A.compose(B)即执行B后输入A中执行A
  - Predicate 接受一个输入参数，返回一个布尔值结果

    示例:

    ```
    Predicate<数据类型> method = s -> s相关表达式;
    ```

- **lambda表达式用法:**

  语法:**(参数列表) -> {方法体}**

  主要应用:**Collection** 和 **Stream流**

使用lambda表达式改写:

```
public class Test07 {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        list.add(1);
        list.add(2);
        list.add(3);
        /*list.forEach(new  Consumer<Integer>() {
            @Override
            public void accept(Integer integer) {
                System.out.println(integer);
            }
        });*/
        list.forEach( Object s -> System.out.println(s));
    }

}
```

输出结果:
```
1
2
3
```

三、Task3.泛型

仓库代码如下:

```
import java.util.*;

public interface Repository{
    void save(Object data);
    Object getById(int Id);
    void putout();
}

class MyRepository implements Repository{
    private Map<Object, Object> Data = new HashMap<>();
    int Id = 0;

    @Override
    public Object getById(int Id) {
        return ("Id为"+Id+"存储的数据为"+Data.get(Id));
    }

    //遍历方法
    @Override
    public void putout() {
        Data.forEach((k,v)->{
            System.out.println("Id为"+k+"存储的数据为"+v);
        });
    }

    @Override
    public void save(Object data) {
        Data.put(Id, data);
        Id++;
    }
}

class User {
    private String name;
    private int age;

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

    @Override
    public String toString() {
        return "User{name='" + name + "', age=" + age + "}";
    }
}

class Test07a{
    public static void main(String[] args) {
```

```
        MyRepository Repository1 = new MyRepository();
        User a = new User("小明",18);
        Repository1.save(18);
        Repository1.save("十八");
        Repository1.save(a);
        //遍历输出示例
        Repository1.putout();
        //getById方法示例
        System.out.println(Repository1.getById(2));
    }
}
```

四、Task4.练习

补全代码如下:

```
import java.util.*;

public class MockSongs {
    public static List<String> getSongStrings(){
        List<String> songs = new ArrayList<>();
        //模拟将要处理的列表
        songs.add("sunrise");
        songs.add("thanks");
        songs.add("$100");
        songs.add("1100");
        songs.add("a100");
        songs.add("havana");
        songs.add("114514");
        //TODO
        songs.sort(Comparator
                .comparingInt(String::length));
        Map<Integer,ArrayList> dividedbylen = new HashMap();

        songs.forEach(a-> dividedbylen.put(a.length(),new ArrayList()));
        songs.forEach(a-> dividedbylen.get(a.length()).add(a));
        songs.forEach(a ->
                dividedbylen.get(a.length()).sort(Comparator
                .reverseOrder()));
        //上面采取了一个取巧的方式，令其自然逆序，刚好符合要求
        songs.clear();
        dividedbylen.forEach((k,v)-> v.forEach(a -> songs.add(a.toString())));
        //END
        return songs;
    }
    public static void main(String[] args){
        MockSongs mockSongs = new MockSongs();
        System.out.println(mockSongs.getSongStrings());
```