Q1

接口

Collection：所有集合的根接口，定义了基本操作如添加、删除、遍历等
List：有序集合，允许重复元素，可通过索引访问
Set：不允许重复元素的集合
Map：存储键值对的结构，键唯一，值可重复

实现类

ArrayList：基于动态数组的List实现，查询快，增删慢
LinkedList：基于链表的List实现，增删快，查询慢
HashSet：基于哈希表的Set实现，无序但查找效率高
TreeSet：基于红黑树的Set实现，自动排序
HashMap：基于哈希表的Map实现，键值对存储
TreeMap：基于红黑树的Map实现，按键排序
数组与集合的相同点和不同点:

相同点:

都用于存储多个数据元素
都可以存储引用数据类型
都可以通过某种方式访问元素
不同点:
集合长度是动态的，可以自动扩容;但是数组的长度是固定的，需要手动扩容很麻烦
集合只能存储引用类型，数组可以存储基本数据类型和引用类型
集合提供了丰富的操作方法（如增删查改：排序）数组只提供基础的存储功能

观点:

我认为集合相较于数组具有更好的封装性，更多的方法：更灵活的大小，使用起来更简便实用

Q2

Q3

匿名内部类是没有没有名字的内部类，通常用于只需使用一次的类创建，可以使程序更简洁更容易维护
函数式接口是只包含一个抽象方法的接口，用于支持Lambda表达式

Lambda表达式的用法总结：

基本语法结构
标准形式: (参数列表) -> { 方法体 }
简化形式:
单个参数可省略括号: 参数 -> {方法体 }
单行语句可省略大括号: (参数) -> 方法调用
单行返回语句可省略return关键字: (参数) -> 表达式
使用条件:
只能用于实现函数式接口（只包含一个抽象方法的接口）
常见应用场景
集合遍历
事件处理
比较
线程创建

Q4:

Repository接口：

```
1. public interface Repository <E>{
2.     void save(E Data);
3.      E getById(int id);
4.
5. }
```

MyRepository类

```
1. import java.util.HashMap;
2. import java.util.Map;
```

```
public class MyRepository<E> implements Repository<E>{
    private Map<Integer,E> storage = new HashMap<>();
    private int Id=0;

     @Override
     public void save(E Data) {
         storage.put(Id,Data);
         Id++;

     }
     @Override
     public E getById(int id) {
         return storage.get(id);
     }
     public void print(){
         for (Map.Entry<Integer, E> entry : storage.entrySet()) {
             System.out.println("ID " + entry.getKey() + ": " + entry.getValue());
         }
     }
}
```

User类

```
public 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 + "}";
    }
}
```

Use测试类：

```
public class Use {
    public static void main(String[] args) {
        MyRepository<String> storage1 = new MyRepository<>();
```

```
        storage1.save("Hello");
        storage1.save("World");
        storage1.save("Java");
        System.out.println("storage1:");
        storage1.print();


        MyRepository<Integer> storage2 = new MyRepository<>();
        storage2.save(100);
        storage2.save(200);
        storage2.save(300);
        System.out.println("storage2");
        storage2.print();









        MyRepository<User> storage3 = new MyRepository<>();
        storage3.save(new User("Alice", 25));
        storage3.save(new User("Bob", 30));
        storage3.save(new User("Charlie", 35));
        System.out.println("storage3");
        storage3.print();

    }
}
```

运行截图

Q5:

代码:

```
import java.util.ArrayList;
```

```java
import java.util.Collections;
import java.util.List;

public class MockSongs {
    public static List<String> getSongStrings(){
        //模拟将要处理的列表
        List<String> songs = new ArrayList<>();
        songs.add("sunrise");
        songs.add("thanks");
        songs.add("$100");
        songs.add("havana");
        songs.add("114514");
        //TODO
        //在这里完成你的代码
        Collections.sort(songs, (s1, s2) -> {
            // 首先按字符串长度排序
            if (s1.length() != s2.length()) {
                return s1.length() - s2.length();
            }
            // 长度相同时，按字符类型排序：字母 < 数字 < 其他符号
            char c1 = s1.charAt(0);
            char c2 = s2.charAt(0);
            int type1 = getCharType(c1);
            int type2 = getCharType(c2);

            if (type1 != type2) {
                return type1 - type2;
            }
            // 类型相同则按字典序排序
            return s1.compareTo(s2);
        });
        //END
        return songs;
    }
    private static int getCharType(char c) {
        if (Character.isLetter(c)) {
            return 0;
        } else if (Character.isDigit(c)) {
            return 1;
        } else {
            return 2;
        }
    }
    // 添加测试方法
    public static void main(String[] args) {
        List<String> Songs = getSongStrings();
```

```
48.         System.out.println("排序后的歌曲列表:");
49.         for (String song : Songs) {
50.             System.out.println(song);
51.         }
52.     }
53. }
```

运行截图: