

Q1

List接口：是有序的集合，允许元素重复，可通过索引访问元素

-**ArrayList**：基于动态数组实现，可通过索引直接获取，但插入、删除元素效率相对较低

-**LinkedList**：基于双向链表实现，插入、删除元素效率高，但查询元素需要遍历链表，效率不如ArrayList

Set接口：不允许元素重复，元素无固定顺序

-**HashSet**：基于哈希表实现，存取元素速度快，不保证元素的顺序，且允许null元素。

-**TreeSet**：基于红黑树实现，能对元素进行排序,元素需实现Comparable接口或提供比较器，不允许null元素

Map接口：用于存储键值对（key - value），键不允许重复，每个键对应一个值

-**HashMap**：基于哈希表实现，存取键值对效率高，不保证键的顺序，允许null键和null值

-**TreeMap**：基于红黑树实现，能对键进行排序,键需实现 Comparable 接口或提供比较器，不允许null键

数组

长度固定，不可改变，功能是通过索引访问元素，存储基本数据类型（如 `int`、`char` 等）和引用数据类型

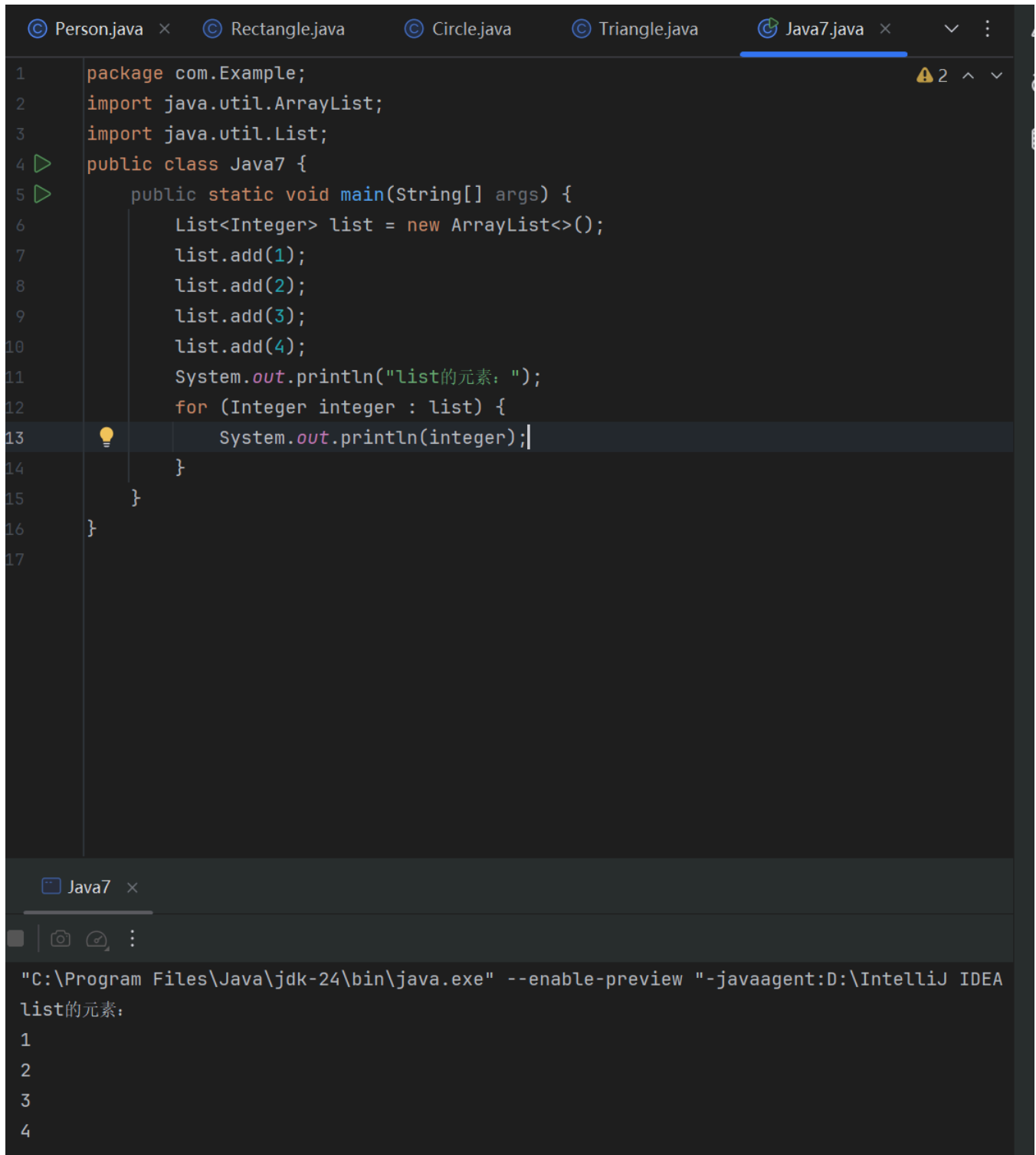
集合

长度动态可变，功能有添加、删除、查找、排序等方法，只能存储引用数据类型，若要存储基本数据类型，需使用对应的包装类（如 `Integer`、`Character` 等）

集合的扩展性更好，可以实现有序等等需求

Task2

Q2



```
1 package com.Example;
2 import java.util.ArrayList;
3 import java.util.List;
4 public class Java7 {
5     public static void main(String[] args) {
6         List<Integer> list = new ArrayList<>();
7         list.add(1);
8         list.add(2);
9         list.add(3);
10        list.add(4);
11        System.out.println("list的元素: ");
12        for (Integer integer : list) {
13            System.out.println(integer);
14        }
15    }
16 }
17
```

Java7 ×

"C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview "-javaagent:D:\IntelliJ IDEA" list的元素:
1
2
3
4

Q3

匿名内部类是没有名字的内部类，它可以在创建对象的同时定义类的内容，通常用于只需要使用一次的类场景

函数式接口是只包含一个抽象方法的接口，允许将“函数”作为参数 / 返回值传递

四大接口:

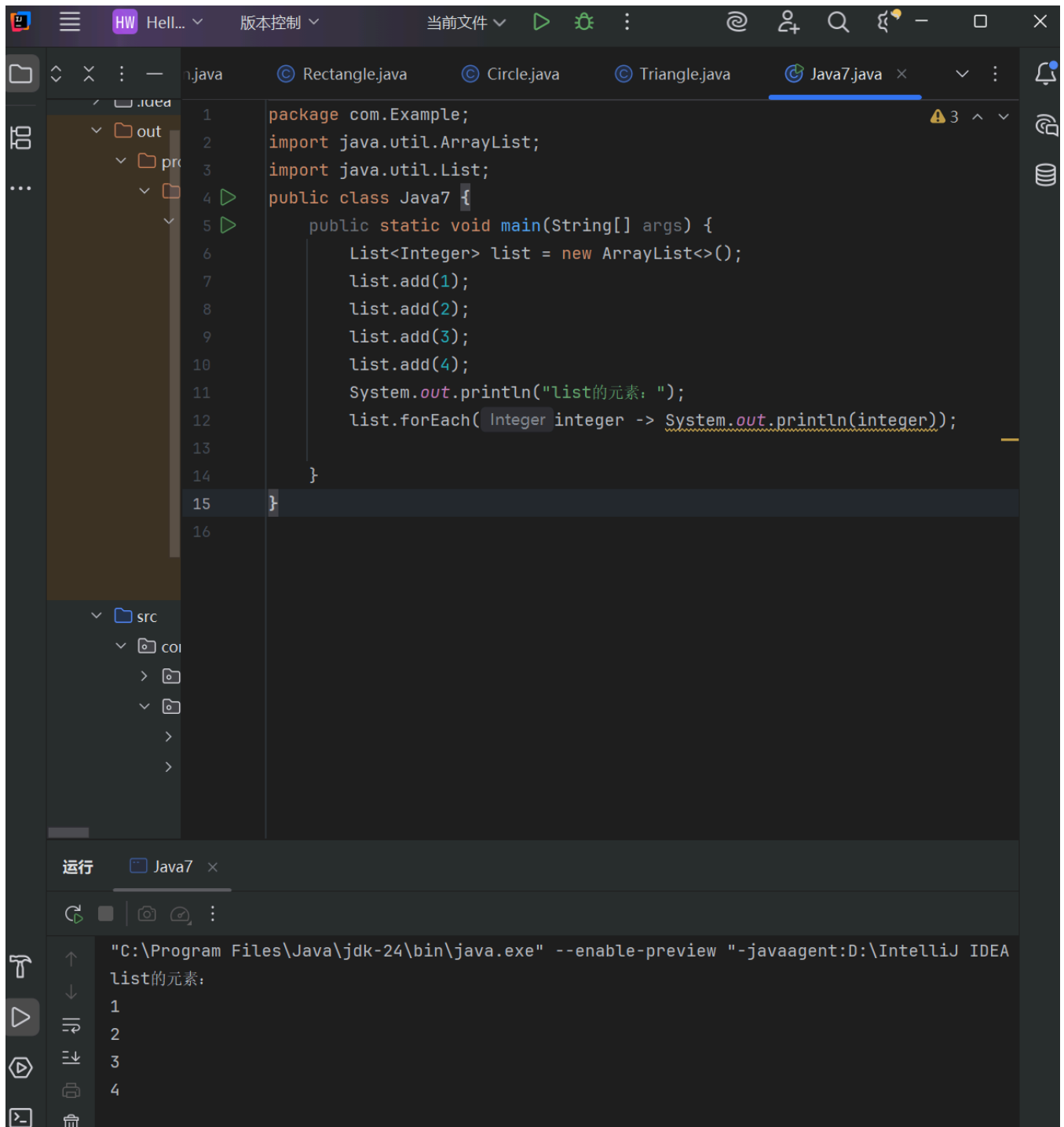
Supplier无参有返回,可用于生成随机数、获取配置值

Consumer有参有返回，可用于打印数据、修改对象属性

Function<T,R>有参无返回，可用于字符串转整数、对象属性提取

Predicate断言判断，可用于过滤集合（如筛选偶数）

OOP 用“对象”封装行为，是“用对象做事情”，FP 用“函数式接口”封装函数，是“用函数做事情”



```
1 package com.Example;
2 import java.util.ArrayList;
3 import java.util.List;
4 public class Java7 {
5     public static void main(String[] args) {
6         List<Integer> list = new ArrayList<>();
7         list.add(1);
8         list.add(2);
9         list.add(3);
10        list.add(4);
11        System.out.println("list的元素: ");
12        list.forEach( Integer integer -> System.out.println(integer));
13    }
14 }
15 }
16
```

运行 Java7 x

"C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview "-javaagent:D:\IntelliJ IDEA

list的元素:

1

2

3

4

形式：(参数列表) -> { 代码块 } 如果参数只有一个，可以省略括号；如果代码块只有一条语句，可以省略大括号、`return`（如果有返回值的话）等。

Task3

```
package com.Example.MyRepository;

public class Test {
    public static void main(String[] args) {
        MyRepository<User> userRepository = new MyRepository<>();
        int id1 = userRepository.save(new User(name="张三", age=20));
        int id2 = userRepository.save(new User(name="李四", age=25));
        System.out.println("id为" + id1 + "的用户: " + userRepository.getById(id1));
        System.out.println("id为" + id2 + "的用户: " + userRepository.getById(id2));
    }
}

class User { 3个用法
    private String name; 2个用法
    private int age; 2个用法
    public User(String name, int age) { 2个用法
        this.name = name;
        this.age = age;
    }
    @Override
    public String toString() {
        return "User{name='" + name + "', age=" + age + "}";
    }
}
```

运行 Test

```
"C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview "-javaagent:D:\IntelliJ IDEA 2025.2.1\lib\idea_rt.jar=62495" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8
id为0的用户: User{name='张三', age=20}
id为1的用户: User{name='李四', age=25}

进程已结束，退出代码为 0
```

```
package com.Example.MyRepository;
import java.util.HashMap;
import java.util.Map;

public class MyRepository<T> implements Repository<T> { 0个用法
    private Map<Integer, T> dataMap = new HashMap<>(); 2个用法
    private int currentId = 0; 2个用法
    @Override 0个用法
    public int save(T data) {
        dataMap.put(currentId, data);
        return currentId++;
    }
    @Override 0个用法
    public T getById(int id) {
        return dataMap.get(id);
    }
}
```

```
package com.Example.MyRepository;

public interface Repository<T> { 0个用法 1个实现
    int save(T data); 0个用法 1个实现
    T getById(int id); 0个用法 1个实现
}
```

Task4

Q5

```
package com.Example;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class MockSongs {
    public static List<String> getSongStrings() {
        List<String> songs = new ArrayList<>();
```

```

songs.add("sunrise");
songs.add("thanks");
songs.add("$100");
songs.add("havana");
songs.add("114514");

songs.sort((s1, s2) -> {

    if (s1.length() != s2.length()) {
        return s1.length() - s2.length();
    } else {

        boolean s1IsLetter = s1.matches("[a-zA-Z]+");
        boolean s2IsLetter = s2.matches("[a-zA-Z]+");
        boolean s1IsDigit = s1.matches("[0-9]+");
        boolean s2IsDigit = s2.matches("[0-9]+");

        if (s1IsLetter && s2IsDigit) {
            return -1;
        } else if (s1IsDigit && s2IsLetter) {
            return 1;
        } else if (s1IsDigit && !s2IsDigit && !s2IsLetter) {
            return -1;
        } else if (!s1IsDigit && !s1IsLetter && s2IsDigit) {
            return 1;
        } else {
            return s1.compareTo(s2);
        }
    }
});
return songs;
}

public static void main(String[] args) {
    List<String> sortedSongs = getSongStrings();
    for (String song : sortedSongs) {
        System.out.println(song);
    }
}
}

```

