第二次实验前的准备工作

了解java的泛型机制

在现代编程语言中，泛型是一项非常重要的特性。Java作为一种面向对象编程语言，也具备了泛型的能力。本文将会介绍Java中泛型的基本概念和用法。
首先，什么是泛型？泛型是指在编程时不预先指定具体的数据类型，而是在使用时再指定具体的类型。这种特性可以让我们编写更加通用的代码，提高代码的复用性和可读性。在Java中，我们可以通过使用尖括号“<>”来声明泛型。一般可以使用大写的字母表示泛型的占位符，常用的大写字母有：T、E、K、V和N，具体使用什么大写字母没有规定。下面的例子都是以字母T为例。

```
public class GenericArray<T> {
    private T[ ] array;
    public GenericArray(int size) {
            array = (T[]) new Object[size];
    }
    public T get(int index) {
            return array[index];
    }
    public void set(int index, T value) {
            array[index] = value;
    }
    }
```

在这个例子中，定义了一个泛型类GenericArray，它使用类型参数T。在构造函数中，使用T类型来创建一个数组。请注意，必须使用Object类来创建一个泛型数组，然后将其强制转换为泛型类型T。这是因为Java不允许直接创建泛型数组。现在可以使用GenericArray类来创建一个泛型数组。以下是一个示例代码，展示了如何使用GenericArray类：

```
GenericArray<Integer> array1 = new GenericArray<>(10);
for (int i = 0; i < array1.length(); i++) {
      array1.set(i, i * i);
}
for (int i = 0; i < array1.length(); i++) {
      System.out.println(array1.get(i));
}
```

```
GenericArray<Double> array2 = new GenericArray<>(10);
for (int i = 0; i < array2.length(); i++) {
      array1.set(i, i * i * 1.0);
}
for (int i = 0; i < array2.length(); i++) {
     System.out.println(array2.get(i));
}
```

在这个例子中，创建了两个GenericArray类型的对象，其中一个用来存放整数类型，一个用来存放浮点类型，大小均为10。使用set方法来设置数组中的值，使用get方法来获取数组中的值。最后，我们使用循环来打印数组中的值。
泛型不仅可以用作方法的定义，也可以用作接口类型和类类型的定义，尤其是用作接口类型和类类型的定义中，可以实现一个更加通用的数据结构和算法。
另外，我们也准备了泛型Stack栈的具体实现，同学们从阅读代码中掌握泛型的使用。

了解Java静态内部类的定义和使用

Java内部类是Java编程语言中的一个重要概念，它允许在一个类中定义另一个类。内部类可以访问外部类的所有成员，包括私有成员。本文将详细解释Java内部类的概念，类型和使用方法。
为了更好地理解Java内部类，需要了解Java中的四种内部类类型：成员内部类，局部内部类，匿名内部类和静态内部类。

- 成员内部类是定义在外部类中的普通类。它可以访问外部类的所有成员，并且可以在外部类的所有方法中创建对象。成员内部类通常用于实现一些特定的功能，例如迭代器类，比较器类等。
- 局部内部类是定义在方法中的内部类。它只能在该方法中访问，并且只能在该方法中创建对象。局部内部类通常用于实现一些特定的功能，例如事件监听器类。
- 匿名内部类是没有名称的内部类。它通常用于实现一些简单的功能，例如事件监听器类。它可以直接在语句中创建，而不需要定义一个具名的类。
- 静态内部类是定义在外部类中的静态类。它不能访问外部类的非静态成员，但可以访问外部类的静态成员。静态内部类通常用于封装某些外部类的静态方法或静态变量。
使用Java内部类的好处在于它可以更好地实现封装和组织代码。它可以隐藏一些实现细节，并且可以更好地实现某些特定功能。此外，内部类还可以实现接口和继承其他类，从而更好地实现多态性。

总之，Java内部类是Java编程语言中的一个重要概念，它具有多种类型和使用方法。了解这些类型和使用方法可以更好地封装和组织代码，并实现更好的多态性。
在使用链表存储技术实现复杂数据结构时，大部分情况中的结点类类型是不需要公开定义

的，只需要使用的数据结构中定义，这个时候就可以使用静态内部类，具体的使用方法请参看LStack.java文件。

了解StreamTokenizer的使用

在计算机编程中，StreamTokenizer是一个常用的类。StreamTokenizer是一个非常有用的工具，可以帮助程序员高效地解析和处理文本文件和网络数据流。在适当的使用和理解下，它可以大大提高程序开发的效率和质量。
它可以将一个输入流分解成一个序列的单词，每个单词可以是一个数字、一个字符串或者一个符号。StreamTokenizer可以处理多种类型的数据，包括整数、浮点数、字符串、注释和符号。它提供了许多方法来读取和解析输入流，包括nextToken、nextNumber、nextString和nextSymbol等等。这个类的使用相对简单，只需要创建一个StreamTokenizer对象，然后将输入流传递给它即可。程序员可以根据需要设置不同的分隔符和特殊字符，以及指定是否忽略注释和空格等。StreamTokenizer的优点在于它可以高效地解析大型文本文件，而且非常灵活。下面将介绍StreamTokenizer的使用方法和一些重要的注意事项。
首先，我们需要创建一个输入流。可以使用FileInputStream类来读取文件，或者使用System.in来读取用户的输入。例如，我们可以使用以下代码创建一个输入流：

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
```

接下来，我们可以创建一个StreamTokenizer对象，并将输入流传递给它。例如，我们可以使用以下代码：

```
StreamTokenizer st = new StreamTokenizer(br);
```

现在，我们已经准备好读取输入流中的标记了。StreamTokenizer类将输入流分解为单个标记，并将它们分类为数字、字母、标点符号等。我们可以使用nextToken()方法来检索下一个标记。例如，以下代码将读取输入流中的下一个标记，并将其存储在变量token中：

```
int token = st.nextToken();
```

在读取标记之后，我们可以使用switch语句来确定标记的类型，并执行相应的操作。例如，以下代码将检查标记是否为数字，并将其打印到控制台：

```
switch(token) {
      case StreamTokenizer.TT_NUMBER:
            double num = st.nval;
            System.out.println("这是一个数字：" + num);
```

```
            break;
        case StreamTokenizer.TT_WORD:
            String word = st.sval;
            System.out.println("这是一个单词：" + word);
            break;
        case StreamTokenizer.TT_EOF:
            System.out.println("已经到达输入流的末尾。");
            break;
        default:
            char ch = (char)token;
            System.out.println("这是一个标点符号：" + ch);
            break;
    }
```

在给同学们准备的文件中，同学们可以阅读LearningStreamTokenizer.java程序，在该程序中解析了input.txt文件中的数据。