1. **Create a New Java Project**:
    - Open Eclipse.
    - From the main menu, click on **File > New > Java Project**.
    - Name your project (e.g., `MacroPass1`) and click **Finish**.
2. **Create Java Classes**:
    - In the `src` folder of your new project, create new Java classes for each of the provided files (`MPass1.java`, `mdt.java`, `arglist.java`, `mnt.java`).
    - To create a new class, right-click on `src` > **New** > **Class** and name the class accordingly.
    - Repeat for all the provided classes.
3. **Add Input File**:
    - You need to add the `input.txt` file (the input data you want to process) into your project directory (e.g., under `C:\Eclips\TCOB06\MacroPass1\input.txt` or a similar path).
    - Ensure the input file follows the format provided in your example.
4. **Run the Code**:
    - Right-click on the `MPass1.java` file in the Project Explorer.
    - Select **Run As > Java Application**.
    - The output will be displayed in the **Console** tab and also written into `MDT.txt`, `MNT.txt`, and `ARGLIST.txt` files as defined in the code.

---

## Explanation of the Code and How it Works

The code provided implements a **macro processing system** as part of a **two-pass assembler**. In this case, **Pass 1** is responsible for reading macros from an input file and generating various tables such as the Macro Definition Table (MDT), Macro Name Table (MNT), and Argument List.

Let's break down the code and its components:

### 1. MPass1.java (Main Program)

- **Input File Reading**:
    - It starts by reading the input file (`input.txt`) line by line using `BufferedReader`.
- **Tokenization**:
    - Each line is split into tokens (separated by whitespace) using `split("\\s+")`.
- **Macro Identification**:
    - The program identifies the beginning of a macro definition with the keyword `MACRO`, and the end with `MEND`.
- **Macro Name Table (MNT)**:
    - When a macro is encountered, its name is stored in the `MNT` (Macro Name Table) along with the address of the first statement in the macro.
- **Macro Definition Table (MDT)**:
    - Each statement in the macro is stored in the `MDT`. For arguments in the macro (e.g., `&X`, `&Y`), the macro's arguments are stored in `ARGLIST` and replaced by a placeholder like `#1`, `#2`, etc., in the `MDT`.
- **Output**:
    - After processing the input file, it writes the contents of `MNT`, `MDT`, and `ARGLIST` to corresponding output files (`MNT.txt`, `MDT.txt`, `ARGLIST.txt`), as well as printing them on the console.

### 2. mdt.java (Macro Definition Table)

- This class represents each statement in the macro. It contains a string `stmnt` to store a macro's statement.
- Constructor initializes `stmnt` to an empty string.

### 3. arglist.java (Argument List)

- This class holds an argument (e.g., `&X`, `&Y`) passed to a macro.
- It contains a single member `argname` to store the argument name.

### 4. mnt.java (Macro Name Table)

- This class represents a macro's metadata: its name and address (starting point in the `MDT`).
- It contains `name` (macro name), `addr` (address of the first statement in the macro), and `arg_cnt` (argument count).
- Constructor initializes `name`, `addr`, and `arg_cnt`.

---

## Code Workflow

### Pass 1 Execution:

1. **Macro Start**:
   - The code looks for the `MACRO` keyword. When found, it starts reading the macro definition and adds it to the `MDT` table.
   - It also adds the macro name to the `MNT` with the address of the first statement in `MDT`.
2. **Argument Handling**:
   - The code looks for arguments (like `&X`, `&Y`) in the macro definition and stores them in the `ARGLIST`. Arguments are replaced with placeholders (`#1`, `#2`, etc.) in the `MDT`.
3. **Macro End**:
   - When it encounters `MEND`, it ends the macro processing and moves on to the next macro or code section.
4. **Output**:
   - After processing all macros, the `MNT`, `ARGLIST`, and `MDT` are written to respective output files (`MNT.txt`, `ARGLIST.txt`, `MDT.txt`).

---

## Sample Input and Output

### Input (input.txt):

```
txt
Copy code
MACRO
INCR &X,&Y,&REG1=AREG
MOVER &REG1,&X
ADD &REG1,&Y
MOVEM &REG1,&X
MEND
```

```
MACRO
DECR &A,&B,&REG2=BREG
MOVER &REG2,&A
SUB &REG2,&B
MOVEM &REG2,&A
MEND
START 100
READ N1
READ N2
DECR N1,N2
INCR N1,N2
STOP
N1 DS 1
N2 DS 2
END
```

**Output Files:**

1. **MNT.txt** (Macro Name Table):

   ```
   txt
   Copy code
   INCR    4
   DECR    9
   ```

2. **ARGLIST.txt** (Argument List):

   ```
   txt
   Copy code
   &X
   &Y
   &REG1=AREG
   &A
   &B
   &REG2=BREG
   ```

3. **MDT.txt** (Macro Definition Table):

   ```
   txt
   Copy code
   MOVER &REG1,&X
   ADD &REG1,&Y
   MOVEM &REG1,&X
   MOVER &REG2,&A
   SUB &REG2,&B
   MOVEM &REG2,&A
   ```

**Console Output:**

```
markdown
Copy code
*****MACRO NAME TABLE*****
INDEX  NAME    ADDRESS
    0   INCR    4
    1   DECR    9

*****ARGUMENT LIST*****
INDEX  NAME    ADDRESS
```

```
    0   &X
    1   &Y
    2   &REG1=AREG
    3   &A
    4   &B
    5   &REG2=BREG

*****MACRO DEFINATION TABLE*****
INDEX  STATEMENT
    0    MOVER &REG1,&X
    1    ADD &REG1,&Y
    2    MOVEM &REG1,&X
    3    MOVER &REG2,&A
    4    SUB &REG2,&B
    5    MOVEM &REG2,&A
```

---

## Conclusion

By following the steps above, you can run the Java code in Eclipse and generate macro-related tables (`MDT`, `MNT`, and `ARGLIST`) based on the input data. The code parses the input file, processes macros, and produces the relevant tables, which are stored in text files and displayed on the console.