LAB # 01

Introduction to ADT, Wrapper, IO Classes and Filing in JAVA

Object

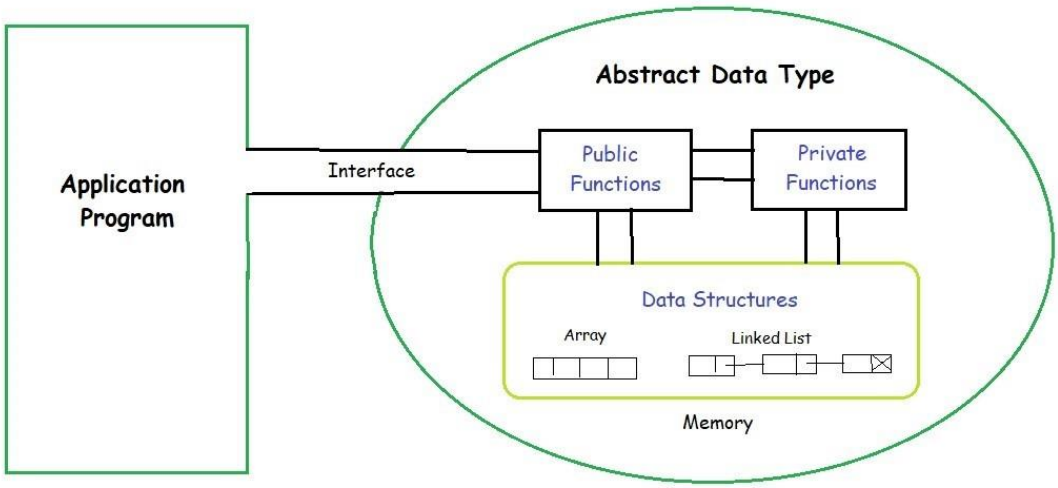
To implement Java Built-In User input Class, usage of wrapper class and filing in java.

Theory

Abstract Data Types

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations.

The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation-independent view. The process of providing only the essentials and hiding the details is known as abstraction.



The user of data type does not need to know how that data type is implemented, for example, we have been using Primitive values like int, float, char data types only with the knowledge that these data type can operate and be performed on without any idea

of how they are implemented. So a user only needs to know what a data type can do, but not how it will be implemented. Think of ADT as a black box which hides the inner structure and design of the data type.ArrayList is one of the example of ADT in java.

How to create an ArrayList?

We can create an ArrayList by writing a simple statement like this:

This statement creates an ArrayList with the name alist with type “String”. The type determines which type of elements the list will have. Since this list is of “String” type, the elements that are going to be added to this list will be of type “String”.

```
ArrayList<String> alist=new ArrayList<String>();
```

Similarly we can create ArrayList that accepts int elements.

```
ArrayList<Integer> list=new ArrayList<Integer>();
```

Demo Program#1

```
import java.util.*;
class JavaExample{
  public static void main(String args[]){
    ArrayList<String> alist=new ArrayList<String>();
    alist.add("Steve");
    alist.add("Tim");
    alist.add("Lucy");
    alist.add("Pat");
    alist.add("Angela");
    alist.add("Tom");

    //displaying elements
    System.out.println(alist);

    //Adding "Steve" at the fourth position
    alist.add(3, "Steve");

    //displaying elements
    System.out.println(alist);
  }
}
```

Java Scanner class

Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings. It is the easiest way to read input in a Java program, though not very efficient if you want an input method for scenarios where time is a constraint like in competitive programming.

- To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them. The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values.

Method	Description
public String next()	it returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

Demo Program#1

```
import java.util.Scanner;
class ScannerTest{

public static void main(String args[]){
   Scanner sc=new Scanner(System.in);
   System.out.println("Enter your rollno");
   int rollno=sc.nextInt();
   System.out.println("Enter your name");
   String name=sc.next();
   System.out.println("Enter your fee");
```

```
   double fee=sc.nextDouble();
   System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);
   sc.close();
         }
}
```

Java Wrapper Classes

Wrapper classes provide a way to use primitive data types (int, boolean, etc..) as objects.

The table below shows the primitive type and the equivalent wrapper class:

Primitive Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
Boolean	Boolean
char	Character

Sample Program#1

```
public class MyClass {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }
}
```

Files and I/O

The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, Object, localized characters

A stream can be defined as a sequence of data. there are two kinds of Streams

- **InPutStream:** The InputStream is used to read data from a source.
- **OutPutStream:** the OutputStream is used for writing data to a destination.

Reading and Writing Files:

FileInputStream:

This stream is used for reading data from the files. Objects can be created using the keyword new.

Following constructor takes a file name as a string to create an input stream object to read the file.:

```
InputStream f = new FileInputStream("C:/java/hello");
```

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows:

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

FileOutputStream:

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Following constructor takes a file name as a string to create an input stream object to write the file:

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows:

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

Sample Program#1

```
import java.io.*;

public class FileStreamTest{

   public static void main(String args[])
    File f;

   try{
      byte bWrite [] = {65,66,67,68,69};
      f =new File("D:/filing/test.txt");
      OutputStream os = new FileOutputStream(f);
      for(int x=0; x < bWrite.length ; x++){
         os.write( bWrite[x] ); // writes the bytes
      }
      os.close();

      InputStream is = new FileInputStream(f);
      int size = is.available();

      for(int i=0; i< size; i++){
         System.out.print((char)is.read() + "  ");
      }
      is.close();
   }
catch(IOException e){
      System.out.print("Exception");
           }
       }
}
```

Date & Time:

Java provides the **Date** class available in **java.util** package, this class encapsulates the current date and time . You can use a simple Date object with toString() method to print current date and time as follows.

Sample Program#2

```
import java.util.Date;

public class DateDemo {
   public static void main(String args[]) {
      // Instantiate a Date object
      Date date = new Date();

      // display time and date using toString()
```

```
    System.out.println(date.toString());
  }
}
```

Task

1. Write a program that take string as an input attempting to guess the secret word enters letters one at a time. After each guess, the guess template is updated (if necessary) to show which letters in the secret word match the letter guessed. This process continues until the guess template matches the secret word choice of 7 wrong attempts. The number of guesses is then output.For example:

    Enter the secret word: test

    Guess a letter: a

    Guess a letter: e
    -e--
    Guess a letter: n
    -e--
    Guess a letter: s
    -es-
    Guess a letter: t
    test=test
    You guessed the word in 5 guesses.

There are three key methods ,Note that because the guess template string will be modifed several times, for effciency it is implemented as a StringBuffer rather than a String. Partial defnitions for these methods are given below:

```
static StringBuffer createTemplate ( String secretWord )
// Returns a new StringBuffer object, which is a template
// containing the same number of dashes as there are letters in the secretWord.

static void updateTemplate ( String secretWord, char guessLetter,StringBuffer
guessTemplate )
// Updates the guessTemplate to include the new letter (guessLetter) guessed
// if it matches a letter in the secretWord. For multiple occurrences of the same
// letter, all letter matches are added to the guessTemplate.

static boolean matchTemplate ( String secretWord, StringBuffer guessTemplate )
// Returns true if the secretWord and guessTemplate match.
// Otherwise, returns false.
```

2. Write a program that read a names from the given file ”Names” and searches for a specific initial letter in the last name and write a new file “NewNames” with those searched names.

3. Write a Java Program to reverse elements in an array list.