Chapter - 01, Introduction to Java

J S Packiaraj, Chitra Raj

May 30, 2022

1 JAVA Programming Exercises

These are a few of the programming exercises that we have worked on. We will attempt in the next few resources have the concepts of Java presented.

1.1 The following is a brief introduction to Java.

Every java program lives in a file which is called as a class. The data which can be stored are called *fields* and the operations on the fields is defined by *methods*. The main method is the first method which will be executed by a compiler, thereby it provides the entry point for the compiler to run the lines of code which lives in the class. Ofcourse, if we know the method which we want to use from the code, the main method is not required. Only when a class is expected to execute as an individual entity, the main method is required.

Every bit of data which is worked on is an *object*. Other programming languages might call them as *variables*, we will call every such instance of data bit as an *object*. Because Java is based on objects, it is termed as an *Object Oriented Program*(OOP).

From the object, we can create many *instance* of the class. Every instance of object will be following the blueprint of what data can be stored in it is stored and the operation which is permitted to be done on or with the data. Throughout the course, you will be using objects and methods. In this particular example, the methods which other people have made available to us are going to be used. The package that we will be using is the *java.util.Scanner*, by using __java.util.*__ all the objects created in java.util can be made available for the class which we create.

```
As an example, the java.util has an object called Scanner.  We will be creating
an instance of the scanner class and use the method next() which it has to get an
input from the user and print the same.
```

[1]:
```
import java.util.Scanner;


    Scanner inputObject = new Scanner(System.in);
    String strTextInput = inputObject.next();
    System.out.print("Hello " + strTextInput);
```

```
 JSP
```

```
Hello JSP
```

We have created our first line of code in Java. The line number 4 creates an named object *inputObject* which is an instance of the object *Scanner*. The *System.in* within *()* is the *parameter* which we pass to the method that creates a new object of the class Scanner, instructing that the instance of Scanner named inputObject will accept the input from the standard input device of the System on which the Java code will run.

Now, the new instance of the object Scanner that we created inherits all the methods of the Class Scanner that relates to input from the standard input device. In line number 5, we create a variable named *strTextInput* of the data type *String*. `Do note that String starts with a capital 'S', we will be coming to it a little later. We assign whatever is typed into the keyboard and store it in the variable strTextInput.`

We also store the value which is made available by running the method *next()* of the instance of the Scanner class named inputObject. The method next(), essentially takes whatever is available from the keyboard after the method is invoked till the *enter* key is pressed. When we run the lines of code, line 3 and 4 create memory locations. The line 4 also has the Scanner objects instance whic takes any thing typed on the keyboard and stores it in the variable strTextInput. After storing the input text, we print the output.

Line 6 introduces *Concatenation*. Concatenation is an operation where strings are combined.

So, when the compiler reaches the line 5, it will tell to take what is store in line number 5 into the variable strTextInput and append it after the text "Hello " and print it. Please note that we have given the space after the text Hello.

While you will not be able to run the above code as is, it essentially tells how the different bits of code behave.

To make the code run, it should live in a *class file*. By convention, the name of a class will start with an *uppercase* (capital letter, A-Z) and variables with a *lowercase* (small letters, a-z). By convention, the name of a class will start with an *uppercase* (capital letter, A-Z) and variables with a *lowercase* (small letters, a-z). Create an empty file called *Intro.java* and modify the code as given below:

1.1.1 Final Code

```java
import java.util.Scanner;
public class Intro{
    public static void main(String [] args){
        Scanner inputObject = new Scanner(System.in);
        String strTextInput = inputObject.next();
        System.out.print("Hello " + strTextInput);
    }
}
```

2 Exercise - 01

In the following exercise, we will be write a simple program which calculates the age of an individual. In engineering problems, calculating the time to an event or the time that has elapsed since an event has happened is important. You would have seen small lables on lifts which detail the time when a fire extinguisher or lift is due for the next maintenance. For these cases, it is important to have the intervals of time calculated.

2.1 Exercise - 01 - Problem Statement

Write a program which will take the name of a user and print the age of the person as on 01 June 2022. The input shall be the Name in the first line and the date of birth in ”DD-MM-YYYY” in the second line. The program shall test the following cases, and print the text within the quoted characters (The ” should NOT be printed):

1) If the person is born after 01 June 2022

For a sample input of

JSP

21-09-2022

The output shall be

”NOT ELIGIBLE”

2) if the person is born before or on 01 June 2022:

For a sample input of

JSP

21-09-1988

The output shall be

”Hello JSP, You are 33 years old”

2.2 Exercise - 01 (Solution)

The following will be the required data and the output. ### Input:

a) The Name

b) Date of birth in ”DD-MM-YYYY” format

2.2.1 Importing the required packages:

Input has to be got from the user. For this purpose, we need the Scanner class, which is part of the util package. Date time in Java require the use of the package time. The date time also needs to be formatted. So, we need to import the time package.

[2]:
```
import java.util.*;
import java.time.LocalDate;
import java.time.Period;
import java.time.temporal.ChronoUnit;
```

Having imported the required packages, we will start working on the main method. In Java, we call what is traditionally called as *function* in other languages as *methods*.

2.2.2 Preparing to get input from the user:

We will create an instance of the Scanner object. Please note that in this case, we have used _import java.util.*;_ in the place of *import java.util.Scanner;* as used previously.

We have two variables which have to store values. Just as in the previous case, weJust as in the previous case, we will create two strings to store the data. We will call them *dob* and *name*.

Java does not have a built in date type. So, we will be getting the date as a string of characters and use the methods in java.time.LocalDate to help Java understand the same as if it was a date.

Java does not have a built in date type. So, we will be getting the date as a string of characters and use the methods in java.time.LocalDate to help Java understand the same as if it was a date.

We will use the methods of the ChronoUnit class to compute the time interval in days.

As we create dob and name, we will also initialize them by taking an input from the user, using the inputObject.

[3]:
```
Scanner inputObject = new Scanner(System.in);
String name = inputObject.next();
String dob = inputObject.next();
```

```
JSP
25-12-2000
```

As of now, Java only has two strings of information. We need to tell Java the way to convert the text to a date time. We will use an instance of *LocalDate* to achieve the same.

2.2.3 Converting the input into an instance of LocalDate

LocalDate class helps us to work with the dates and time according to the locale of the computer that we are working on. The computer stores a lot of information which is set when the computer has its operating system installed like the timezone the computer should use for setting the time, if day light saving is present in the region where the computer will be using the computer and so on. We can use the LocalDate class to access these setting and do computation on the time accordingly.

The same is available in the java.time package. To convert the string, to an instance of LocalDate, we need to have the day, month and year as numbers.

In the following lines of code, we will extract the day, month and year.

String handling is aided by many built in methods. We will use some of these methods to convert the entered string into a format which we can use to do date-time calculation .

Now, we know that the first two characters of the string will be a number which will represent the day of the month and the characters in the third and fourth position the month and the last four characters the year.

One of the methods that Java has to handle string is parseInt(String S). So, when we pass a string value, if it can be converted to an integer, it will convert the same and give it.

We will introduce one more concept. String data is nothing but an *array* of characters, of the primitive data type *char*. So, when I give a String "JSP", it is nothing but 'J','S','P'. The array can be accessed by using the index.

We will introduce the method substring(*startIndex, endIndex*). Arrays in Java start at 0. Hence to access the first two characters, startChar will be 0 and the lastChar will be 1 in the string dob. Let us try it and see the output. Earlier, we have given dob to be 25-12-2000

[4]:
```
int intDay = Integer.parseInt(dob.substring(0,2));
System.out.print(intDay);
```

```
25
```

This is one of the ways we can take a part of a string. Let us take the month part and the date part.

[5]:
```
int intMonth = Integer.parseInt(dob.substring(3,5));
System.out.println(intMonth);
int intYear = Integer.parseInt(dob.substring(6));
System.out.println(intYear);
```

```
12
2000
```

If the second parameter for the method substring() is omitted, it will take the characters from the start index till the end of the string. This is what we have done in the code snippet above on line 3. Thus, we see how we can extract the date, month and year part.

```
The _System.out.println_ method adds a _newline_ character, which is the
equivalent of pressing the enter key after the output has been printed for the
computer.
```

Creating an instance of the LocalDate is simple. We make use of the method of and have now converted the string to an instance of LocalDate.

[6]:
```
LocalDate ldDob = LocalDate.of(intYear, intMonth, intDay);
```

It is required that a person should be born after 01-June-2022. This means that we can do two things. Find if the date of birth is greater than 01-June-2022 or find the age of the person from the said date. If it is positive, the first condition becomes valid and we print ”NOT ELIGIBLE”, otherwise, we will be printing the age in the required format.

Let us create an instance of the local date ldCutOffDate with the cut off date.

```
[7]: LocalDate ldCutOffDate = LocalDate.of(2022, 06, 01);
```

2.2.4 Computing the difference in years

We can now use the Duration class to find the difference betwwen the the two dates and store it in the variable *lngInterval*.

```
[8]: long lngInterval = ChronoUnit.DAYS.between(ldDob, ldCutOffDate);
     System.out.println(lngInterval + " days ...");
```

```
7828 days …
```

Now, comes the time to make the decision with the output that we get. Using the previous lines of code, play around with the date. You will get a Positive, Negative or a Zero value. If according to the logic of our program, the result in lngInterval is zero or a positive number, it means that the person is born before 01-June-2022.

Let us now use the *if* statement to branch off. if is a keyword which helps in checking logical conditions.

We introduce one more method *duration* of the java.time package. It returns a string with the time period between two points in time in Year, Month and Second. We can then extract the date part, since we are interested in that only. Ofcourse, the lngInterval could have also been used.

We will remove the System.out which we have used in the above lines as they are really used only for the purpose of debugging.

```
[9]: if(lngInterval >= 0){
         Period perInterval = Period.between(ldDob, ldCutOffDate);
         System.out.print("Hi " + name + ", you are " + perInterval.getYears() + "␣
       ↪years old");
     } else {
         System.out.print("NOT ELIGIBLE");
     }
```

```
Hi JSP, you are 21 years old
```

2.2.5 Final Code

Let us now put this code together, and give a date after 01-June-2022.

```
[10]: import java.util.*;
      import java.time.LocalDate;
      import java.time.Period;
      import java.time.temporal.ChronoUnit;


              Scanner inputObject = new Scanner(System.in);
              String name = inputObject.next();
              String dob = inputObject.next();
              int intDay = Integer.parseInt(dob.substring(0,2));
              int intMonth = Integer.parseInt(dob.substring(3,5));
              int intYear = Integer.parseInt(dob.substring(6));
              LocalDate ldDob = LocalDate.of(intYear, intMonth, intDay);
              LocalDate ldCutOffDate = LocalDate.of(2022, 06, 01);
              long lngInterval = ChronoUnit.DAYS.between(ldDob, ldCutOffDate);
              if(lngInterval >= 0){
                  Period perInterval = Period.between(ldDob, ldCutOffDate);
                  System.out.print("Hi " + name + ", you are " + perInterval.
        ↪getYears() + " years old");
              } else {
                  System.out.print("NOT ELIGIBLE");
              }
```

```
JSP
02-06-2022
```

```
NOT ELIGIBLE
```

2.2.6 Some final thoughts

When you are writing the code, java offers numerous solution based on the packages available. All the packages need not be installed or be available on your platform. For instance, joda is a third party add-on to work with date and time. However, your system might not have the same installed. Hence, better stick to your built in methods. Here we used the class Period and the class Chronounit. Familiarize yourself with all the methods which these class exposes to make the most and decide when one should be used over another. There is also Duration, which gives resolution of time to seconds.

As mentioned earlier, you will have to store your code in a class file. This is an example of the code if it lived inside a class file called *Example001*.

Observe how throughout the implementation of our class Example001, we have called many of the methods that belong to java.time package like LocalDate, Period, ChronoUnit. In each of the cases, we used specific methods like *of()*, *between()*. So also, for the Scanner object, we used the *next()* method. The Integer had the *parseInt()* method. For some of the classes like of(), between() and parseInt(), we had supplied parameters for the method to work on between the curvy brackets ().

Later, when you write your own code, you will find that you can use the same method, but simply by changing the number of parameters you can reuse the same name of the method. This makes the code extend functionality of the parent class. The ability to have more than form is called Polymorphism. Ofcourse, more of it later.

2.2.7 Final Code

```
import java.util.*;
import java.time.LocalDate;
import java.time.Period;
import java.time.temporal.ChronoUnit;
public class Example001{
    public static void main(String [] args){
        Scanner inputObject = new Scanner(System.in);
        String name = inputObject.next();
        String dob = inputObject.next();
        int intDay = Integer.parseInt(dob.substring(0,2));
        int intMonth = Integer.parseInt(dob.substring(3,5));
        int intYear = Integer.parseInt(dob.substring(6));
        LocalDate ldDob = LocalDate.of(intYear, intMonth, intDay);
        LocalDate ldCutOffDate = LocalDate.of(2022, 06, 01);
        long lngInterval = ChronoUnit.DAYS.between(ldDob, ldCutOffDate);
        if(lngInterval >= 0){
            Period perInterval = Period.between(ldDob, ldCutOffDate);
            System.out.print("Hi " + name + ", you are " + perInterval.getYears() + " years ol
        } else {
            System.out.print("NOT ELIGIBLE");
        }
    }
}
```

2.3 Additional Reading

Please visit JAVA Documentation on Oracle. All the classes which are made available under the time package is available. Clicking any of the class will lead you to all the methods available under them. You can search the internet for examples for the usage of each of the classes and methods under the time package.