Tshwane University of Technology

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

PPAF05D/TROF05D

Unit 6

User-defined static methods in Java

DEVELOPED: 2019-2021

Unit 6: User-defined static methods in Java

Assessment criteria

The content covered in this unit relates to the following outcomes and assessment criteria:

UNIT 6: User-defined static methods in Java **Outcome:** On completion of this Unit, the student will be able to develop Java programs implementing user-defined static methods in the main class.
Assessment Criteria
• The components of value-returning and void methods can be identified in the definitions of provided methods. • Value-returning methods with and without arguments can be written for the main class in a Java project considering best practices for object-oriented design. • Void methods with and without arguments can be written for the main class in a Java project considering best practices for object-oriented design.

6.1 Method construction

A method is a collection of statements that perform a specific task. You have created a main() method that was part of all the programs you have written so far. You have also called predefined methods from various Java classes.

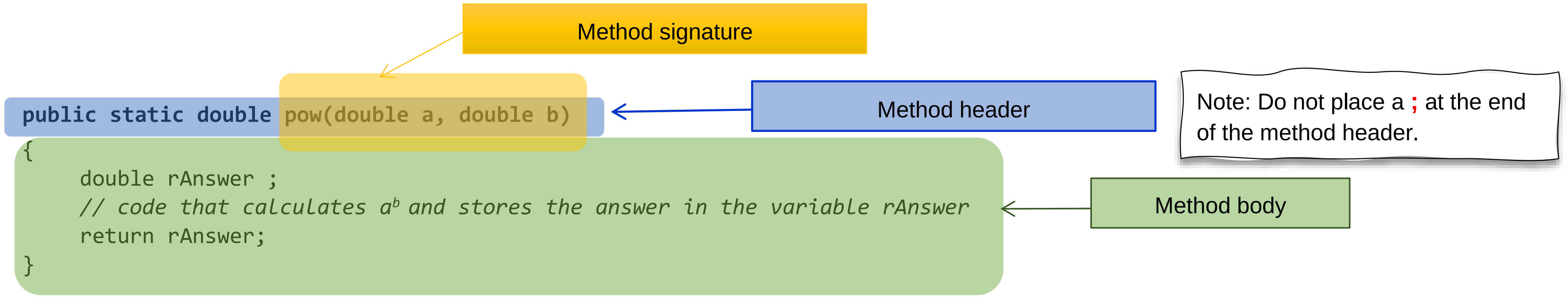
In this Unit you are going to learn to write your own methods that will be part of the same class as the main() method. These methods are called user-defined methods ("user" refers to you as the user/programmer of the Java language who will create these methods).

Let us review two different categories of methods based on the way they are called before we learn how to write user-defined methods.

1. Value-returning methods (also called typed methods) These methods return values of a specific data type when they are called: Examples are:	**2. Void methods** These methods just perform tasks: Examples are:
`rScore = keyboard.nextDouble();`	`System.out.print("Hi how are you!");`
The nextDouble() method reads a value from the keyboard, and returns it as a variable of datatype double. It has no arguments.	The print() method displays the data that is passed to it through the argument.
`iNumBoxes = (int) (Math.ceil(rNumBoxes));`	`formatter.applyPattern("#.00");`
Math.ceil() returns a double value. It has one argument of data type double.	The applyPattern() methods stores the formatting pattern as data of the class so that it can be accessed by the format() pattern when it is called. It has one argument that is a String.

Value-returning methods are called by treating it as a value in a statement. For example, in an assignment statement:
`boolean bResult = sName1.equals(sName2));`
or as part of an if-statement
`if (sName1.equals(sName2))`
	Void methods are called by specifying its name and arguments (if needed) as a statement.

To create a method in a class, the method must be *defined*. "Defined" means code must be written in the relevant class to create the method. Let us look at the Math.pow() method to explain the structure of a method's definition. The definition of a method in a class consists of the *method header*, and *method body*. In the Math class, code similar to the following is available:



```
public static double pow(double a, double b)
{
    double rAnswer ;
    // code that calculates a^b and stores the answer in the variable rAnswer
    return rAnswer;
}
```

- The method header contains important information about the way the method can be accessed and should be called.
- The method body is a collection of statements that are performed when the method is executed.

Terminologies related to the definition of a methods follows.

`public`	`static`	`double`	`pow`	`(double a, double b)`
Method Modifiers			Signature of the method	
Access modifier	Non-access modifier	Return type	Name of the method	Parameter list

`public`	`static`	`void`	`exit`	`(int status)`
Method Modifiers			Signature of the method	
Access modifier	Non-access modifier	Return type	Name of the method	Parameter list

Terminology	**Explanation**
Modifier	*Modifiers* are keywords that are used to indicate how classes, methods and variables can be accessed and used.
Access modifier	The access modifier of a method specifies the *scope* or *accessibility* of the method. It indicates whether other classes can call (invoke) the method. These are the access modifiers in Java.
	• Private: Only code inside the class can call the method. • Public: The method can be accessed from within the class, outside the class, within the package and outside the package. • Protected: Visible to the package and all subclasses (subclasses are not part of the content of this module).
Non-access modifier	These are modifiers that achieve functionalities that are not related to access (scope). Examples of these type of modifiers that you have used already are *final* and *static*.
	• Final: When the *final* modifier is used in a variable declaration, it indicates that the value of the variable cannot be changed once it has been assigned. (We also refer to such a variable as being a "constant", for example: `final VAT = 0.15;.` • Static: When the modifier static is used in a method's header, it indicates that the method can be called from the class directly, without an object having to be instantiated first. Examples of static methods you have used are: Math.ceil() and System.exit().
Return type	The return type describes the type of data that will be sent back (returned) when the method is called. Void methods have no return type since they do not return any values.
Method name (identifier)	The name of the method should follow the rules for Java identifiers. According to convention, the name should start with a lowercase letter. The name should be descriptive of the task the method will perform.

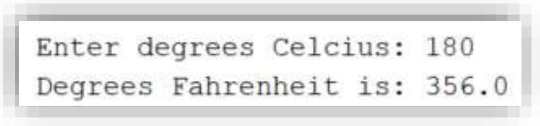
Parameter list	A parameter is a variable name with a data type that is declared in a method header. The parameter indicates the values that can be passed to a method when the method is called. It determines: • the number of values that can be passed to the method; • the data type of each value; • the variable name the value will have inside the method. This is important to note: • When values are passed to a method, it is referred to as the *arguments* of the method. Some sources refer to arguments as *actual parameters*. • When a method is defined, the values that can be passed to the method is referred to as the *parameters.* Some sources refer to *formal parameters*.
Method signature	The combination of the method's name and the parameter list.

6.2 Writing value-returning methods

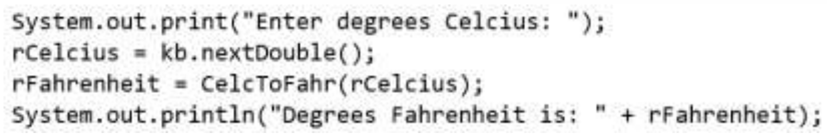
A number of examples will be provided to demonstrate the way value-returning methods can be written.

Value-returning method example 1: Convert degrees Celcius to degrees Fahrenheit.

In the following two programs the user will be asked to enter a temperature in degrees Celcius. The degrees Celcius will then be converted to degrees Fahrenheit, and displayed. In the first program, the calculation will be done in the main method. In the second program, a separate method will be written that contains the mathematical formula to convert degrees Celcius to degrees Fahrenheit. The method will then be called with the degrees Celcius as argument.



```
Enter degrees Celcius: 180
Degrees Fahrenheit is: 356.0
```



```
System.out.print("Enter degrees Celcius: ");
rCelcius = kb.nextDouble();
rFahrenheit = CelcToFahr(rCelcius);
System.out.println("Degrees Fahrenheit is: " + rFahrenheit);
```

Program without method

```java
public static void main(String[] args)
{
    Scanner kb = new Scanner(System.in);
    double rCelcius, rFahrenheit;
    System.out.print("Enter degrees Celcius: ");
    rCelcius = kb.nextDouble();
    rFahrenheit = rCelcius * (9 / 5.0) + 32;
    System.out.println("Degrees Fahrenheit is: " + rFahrenheit);
}
```

The formula to convert from degrees Celcius to degrees Fahrenheit is not complex, but in other programs we may want to remove more complex calculations from the main method to improve the readability of the program. We write a separate method where the calculations are done. In the main method, the programmer can then focus on the flow of the program.

Program with method

```java
public class CelciusToFahrenheit {
    public static double celcToFahr(double rCelc)
    {
        double rFahr;
        rFahr = rCelc * (9 / 5.0) + 32;
        return rFahr;
    } //end CelcToFahr

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        double rCelcius, rFahrenheit;
        System.out.print("Enter degrees Celcius: ");
        rCelcius = kb.nextDouble();
        rFahrenheit = celcToFahr(rCelcius);
        System.out.println("Degrees Fahrenheit is: " +
                           rFahrenheit);
    } //end main
} //end class
```

Terminologies

This is the definition of a method to convert degrees Celcius to degrees Fahrenheit:

```java
public static double celcToFahr(double rCelc)
{
    double rFahr;
    rFahr = rCelc * (9 / 5.0) + 32;
    return rFahr;
} //end CelcToFahr
```

The header of the method. It indicates that the method will have one parameter of data type double.

A local variable where the result of the calculation will be stored.

The parameter value *rCelc* is used in the calculation and the result is stored in the local variable *rFahr*.

The keyword *return* followed by the value that must be returned by the method.

Note the following

- In programming the part of a program within which a variable can be referred to is called the variable's *scope*. The variable rFahr is declared inside the body of the method CelcToFahr. It's scope is the method - it can only be referred to from within the body of the method. The variable rFahr is also called a local variable – it is local to the method.
- The CelcToFahr() method can also be defined as follows:

`public static double celcToFahr(double rCelc)`
`{`
`    return rCelc * (9 / 5.0) + 32;`
`} //end CelcToFahr`
	The last statement to be executed in a value-returning method is always the one that returns the calculated value. You can use the return keyword with a calculation instead of first assigning a value to a variable, and then returning the variable.

- The method could also be called in the output statement. The you would not need the variable *rFahrenheit* in the main() method.

```
System.out.println("Degrees Fahrenheit is: " + celcToFahr(rCelcius));
```

- Since some programs may contain many methods, it will help if you add a comment at the end of each method's body that contains the name of the method. This will help to keep track of all the braces.

Value-returning method example 2: Convert the length and width provided by the user in centimeters to meters

Program without method

```java
package cmtometersnomethod;
import java.util.Scanner;
public class CmTometersNoMethod {
    public static void main(String[] args) {
        double rWidth, rLength;
        Scanner kb = new Scanner(System.in);

        //Input
        System.out.print("Enter the width in centimeter: ");
        rWidth = kb.nextDouble();
        System.out.print("Enter the length in centimeter: ");
        rLength = kb.nextDouble();

        //Convert width and length to meters
        rWidth = rWidth * 0.01;
        rLength = rLength * 0.01;

        System.out.println("Width in meter is " +
                           rWidth + " m");
        System.out.println("Length in meter is "+
                           rLength + " m ");
    }
```

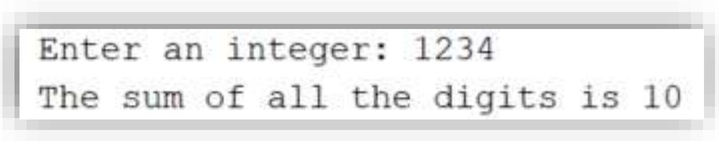
In this example the same calculation is done twice, but with different values. This is another reason to use methods. Should the programmer want to change the calculation, it only has to be done in one place (inside the method).

Program with method

```java
package cmtometersmethods;
import java.util.Scanner;
public class CmToMetersMethods {
    public static double convert (double rValue)
    {
       double rResult = rValue * 0.01;
       return rResult;
    } //end convert

    public static void main(String[] args) {
        double rWidth, rLength;
        Scanner kb = new Scanner(System.in);

        //Input
        System.out.print("Enter the width in centimeter: ");
        rWidth = kb.nextDouble();
        System.out.print("Enter the length in centimeter: ");
        rLength = kb.nextDouble();

        //Convert width and length to meters
        rWidth = convert(rWidth);
        rLength = convert(rLength);

        System.out.println("Width in meter is " +
                           rWidth + " m");
        System.out.println("Length in meter is "+
                           rLength + " m ");
    } //end main
}
```

Value-returning method example 3: Calculate the sum of the digits in an integer

```
package sumdigits;
import java.util.Scanner;
public class SumDigits {
    public static int sum(int iVal)
    {
      int iTotal = 0;
      while(iVal > 0)
      {
         iTotal=iTotal + iVal % 10;
         iVal /= 10;
      }
         return iTotal;
    } //end sum

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int iNum = kb.nextInt();
        System.out.println("The sum of all the digits is " + sum(iNum));
    }
}
```



```
Enter an integer: 1234
The sum of all the digits is 10
```

Value-returning method example 4: Generate a number with 4 random digits

In this example a method will create a 4-digit number where each digit is generated randomly. The first digit may not be a zero. The nextInt method from the Random clsas will be needed. Therefore an object based on the Random class wil be created inside the method.

```
package fourrandomdigits;
import java.util.Random;
public class FourRandomDigits {

    public static int get4DigitRandom() //The method has no parameters
    {
        Random randomize = new Random();
        double rFactor = Math.pow(10, 3);
        int iFirstDigit = randomize.nextInt(9) + 1 ; //a digit between 1 and 9
        int iOtherDigits = (int) (Math.random() * rFactor);
        return (int) (iFirstDigit * rFactor) + iOtherDigits;
    } //end get4DigitRandom

    public static void main(String[] args) {
        int iTheRandomNumber;
        iTheRandomNumber = get4DigitRandom(); //When the method is called, it has to be called with an empty argument list.
        System.out.println("The number is: " + iTheRandomNumber);
    } // end main
} //end class
```

This is an example where the calculations are more complex, and different programmers may use different algorithms to generate 4 random digits. The complex calculations are separated from the logic in the main() method.

Here are alternative ways in which the method could be written:

```
    public static int get4DigitRandom()
    {
        Random randomize = new Random();
        int iFirstDigit = randomize.nextInt(9) + 1 ; //a digit between 1 and 9
        int iOtherDigits = (int) (Math.random() * 1000);
        return (int) (iFirstDigit * 1000) + iOtherDigits;
    } //end get4DigitRandom
    public static int get4DigitRandom()
    {
        Random randomize = new Random();
        int iFirstDigit = randomize.nextInt(9) + 1 ; //a digit between 1 and 9
        int iOtherDigits = randomize.nextInt(999);
        return (int) (iFirstDigit * 1000) + iOtherDigits;
    } //end get4DigitRandom
```

Value-returning method example 5: Generate a 'lucky code'

The rules to create this number are as follows:

- The first part of the code is a 2-digit number between 10 and 99 (both included).
- The second part of the code is a random uppercase letter of the alphabet.

```
package luckycodegenerator;
import java.util.Random;
public class LuckyCodeGenerator{
    //The definition of a method to return a random number between two limits (both limits can be included)
    public static int returnRandValBetween (int iLowNumber, int iUppNumber) //This method has two parameters
    {
        Random randomizer = new Random();
        int iRange = (iUppNumber - iLowNumber);
        return randomizer.nextInt(iRange + 1) + iLowNumber;
    } //end returnRandCharBetween

    public static void main(String[] args) {
       String sLuckyCode;
       int iFirstPart ;
       char cSecondPart;

       iFirstPart = returnRandValBetween(10, 99);
       cSecondPart = (char) returnRandValBetween((int)'A', (int)'Z');

       sLuckyCode = "" + iFirstPart + cSecondPart;
       System.out.println("The lucky code is: " + sLuckyCode);
    } //end main
}
```

Note: The parameters are separated by a comma.

Each parameter must have its own data type specified, even if they have the same data type.

You cannot declare it as

`(int iLowNumber, iUppNumber)` //not allowed

The same method is called twice from the main() method.

Note the following

- The explicit casting (int) 'A' is a way to find the ASCII value of the character 'A'.
- The method returnRandValBetween returns an integer value, therefore the explicit casting operator (char) must be used to convert the integer value returned to a character value.
- Note how integer and character values can be concatenated to form a new string.

`sLuckyCode =`	`""`	`+ iFirstPart`	`+ cSecondPart;`
A String	This is an empty string.	An integer value	A character value

This concatenation of different data types works because the values have been concatenated to an empty string. It is a Java 'trick' to have the integer value converted to a string value. Otherwise, you have to use methods such as Integer.toString() or String.valueOf(). There are other ways as well – but these methods are not part of the scope of this module.

6.3 Argument and parameter data type compatibility

When a method is defined, the parameter list indicates the number of parameters and the data type of each parameter. When a method is called, the number of arguments must match the number of parameters. The data type of each argument must be compatible with the data type of the corresponding parameter. For example: If a method header looks as follows:

```
public static int methodThatReturnsSomething (int iParam1, double rParam2, String sParam3)
```

the call to the method should have 3 arguments:

- The first one should be of a data type compatible with an integer;
- the second should be of a data type compatible with a double;
- the third one should be a String.

Activity 1: Write programs with value-returning methods

a) Complete the value-returning method for the following program. The method should round the value that is sent as an argument to the required number of decimal values. (You need to create the program. Type the provided code for the main() method, and complete the code for the RoundToDecimals() method.)

```
package roundto;
import java.util.Scanner;
public class RoundTo {

    public static double RoundToDecimals(double rTheValue, int iNum)
    {
                        Complete this code.
    } //end RoundToDecimals

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        double rFloatValue;
        int iNumDecimals;
        System.out.print("Enter floating point value: ");
        rFloatValue = kb.nextDouble();
        System.out.print("Round to how many decimal values?: ");
        iNumDecimals = kb.nextInt();
        rFloatValue = roundToDecimals(rFloatValue, iNumDecimals);
        System.out.println("The rounded value is: " + rFloatValue);
    } //end main
}// end class
```

b) Complete the value-returning method for the following program. The method should calculate the factorial of the value that is sent as an argument. (You need to create the program. Type the provided code for the main() method, and complete the code for the factorial () method.)

```
package calcfactorial;
import java.util.Scanner;
```

```
public class CalcFactorial {
    public static double factorial(int iValue)
    {
            Complete this code.
    } //end factorial

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int iNumber;
        double rFact;
        System.out.println("I will calculate the factorial.");
        System.out.print("Enter a number: ");
        iNumber = keyboard.nextInt();
        rFact = factorial(iNumber);
        System.out.println("The factorial of " + iNumber + " is :" + rFact);
    } //end main

}
```

c) Complete the value-returning method for the following program. The method should ask the user how many random digits a number should contain. Tip: Refer to example 2 – the example created a value containing 4 digits that was created randomly – you can adapt that program so that the number of digits is a parameter of the method. (You need to create the program. Type the provided code for the main() method, and complete the code for the getRandNum() method.)

```
package randomnumberdigits;
import java.util.Random;
import java.util.Scanner;
public class RandomNumberDigits {
    public static int getRandNum(int iHowManyDigits)
    {
            Complete this code.

    } //end getRandNum
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        int iNumRandom, iTheRandomNumber;
        System.out.print("How many random digits do you want?: ");
        iNumRandom = kb.nextInt();
        iTheRandomNumber = getRandNum(iNumRandom);
        System.out.println("The number is: " + iTheRandomNumber);
    } //end main

}//end class
```

6.4 Writing void methods

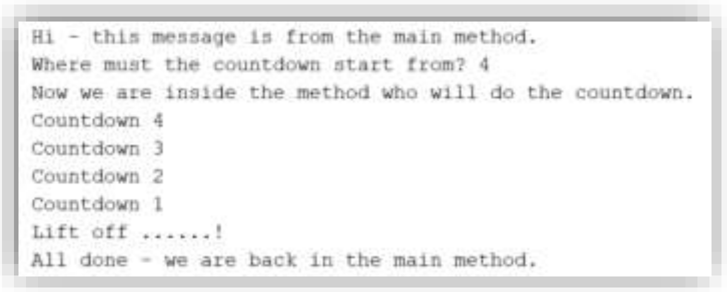
A void method performs a task, it does not return a value. A number of examples will be provided to demonstrate the way value-returning methods can be written.

Void method example 1: Test if the user is old enough to register for an event. People younger than 18 cannot register.

```
package testage;
import java.util.Scanner;
public class TestAge {
    public static void checkAge(int iAge)
    {
        if (iAge < 18) System.out.println("You can't register , you are not old enough!");
        else System.out.println("You can register!");
    } //end checkAge

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("What is your age? ");
        int iPersonAge = kb.nextInt();
        checkAge(iPersonAge);
    } //end main
} //end class
```

Void method example 2: Display count-down messages

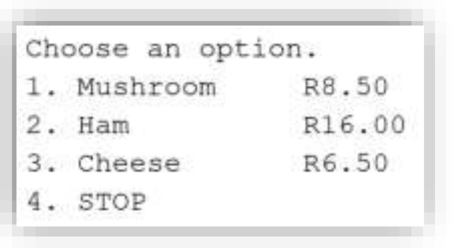
```java
package countdownmessages;
import java.util.Scanner;
public class CountDownMessages {
    public static void display(int iCount)
    {
        System.out.println("Now we are inside the method who will do the countdown.");
        while(iCount > 0)
        {
            System.out.println("Countdown " + iCount);
            iCount--;
        }
        System.out.println("Lift off ......!");
    } //end display

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        int iHowMany;
        System.out.println("Hi - this message is from the main method.");
        System.out.print("Where must the countdown start from? ");
        iHowMany = kb.nextInt();
        display(iHowMany);
        System.out.println("All done - we are back in the main method.");
    }// end main
} //end class
```



```
Hi - this message is from the main method.
Where must the countdown start from? 4
Now we are inside the method who will do the countdown.
Countdown 4
Countdown 3
Countdown 2
Countdown 1
Lift off ......!
All done - we are back in the main method.
```

Void method example 3: Display a menu

In Unit 5, Activity 15 you wrote a program where a menu had to be displayed. The println() method to display the menu appeared twice in the program. Should you decide to change the layout, or the words displayed in the menu, you will have to change code in two places. This is not efficient programming, and it may lead to errors. A void method can be written to display the menu.



```
Choose an option.
1. Mushroom     R8.50
2. Ham          R16.00
3. Cheese       R6.50
4. STOP
```

```
package pizzatoppings;
import java.util.Scanner;
import java.text.DecimalFormat;
public class PizzaToppings {

    //The definition of a void method to display a menu
    public static void displayMenu(String sOption1, String sOption2, String sOption3,
                                   double rCost1, double rCost2, double rCost3)
    {
        DecimalFormat formatter = new DecimalFormat ("R#,###.00");
        System.out.println("\nChoose an option. \n1. " + sOption1 + "\t" + formatter.format(rCost1) +
                           "\n2. " + sOption2 + "\t\t" + formatter.format(rCost2) +
                           "\n3. " + sOption3 + "\t" + formatter.format(rCost3) +
                           "\n4. STOP ");
    }

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        DecimalFormat formatter = new DecimalFormat ("R#,###.00");
        double rTotal = 0;  //The total amount the customer should pay
        int iOption;        //The menu option the user chooses
        int iCountMush = 0;
        int iCountHam = 0;
        int iCountCheese = 0;
        final double MUSHROOM = 8.50, HAM = 16.00, CHEESE = 6.50;
        final String sMenuOption1 = "Mushroom", sMenuOption2 = "Ham", sMenuOption3 = "Cheese" ;
        displayMenu(sMenuOption1, sMenuOption2, sMenuOption3, MUSHROOM, HAM, CHEESE);
        iOption = keyboard.nextInt();
        while (iOption != 4)
        {
            switch (iOption)
            {
                case 1: rTotal += MUSHROOM; iCountMush++; break;
                case 2: rTotal += HAM; iCountHam++; break;
                case 3: rTotal += CHEESE; iCountCheese++; break;
                default: System.out.println(iOption + " is not a valid option.");
            } //end switch
            System.out.println("Your total so far is: " + formatter.format(rTotal));
            displayMenu(sMenuOption1, sMenuOption2, sMenuOption3, MUSHROOM, HAM, CHEESE);
            iOption = keyboard.nextInt();
        } //end while

        System.out.println("You ordered " + iCountMush + " " + sMenuOption1 + " topping(s)");
        System.out.println("You ordered " + iCountHam + " " + sMenuOption2 + " topping(s)");
        System.out.println("You ordered " + iCountCheese + " " + sMenuOption3 + " topping(s)");
        System.out.println("You should pay " + formatter.format(rTotal));
    } //end main method
}
```

The displayMenu() method is called twice in the program.

Void method example 4: Display the name of a season

This example contains one value-returning methods, and one void method.

```
package displayseasonname;
import java.util.Scanner;
public class DisplaySeasonName {

    //The definition of a void method to display the name of the season for a given month number
    public static void displaySeason(int iMonth)
    {
        switch (iMonth)
        {
            case 12: case 1: case 2: System.out.println("Summer"); break;
            case 3: case 4: case 5: System.out.println("Autumn"); break;
            case 6: case 7: case 8: System.out.println("Winter"); break;
            case 9: case 10: case 11: System.out.println("Spring");
        } //end switch
    }  //end displaySeason

     //The definition of a value-returning method to return a valid month number.
    public static int getValidMonth()
    {
       Scanner kb = new Scanner(System.in);
       int iMonthNum;
       do
       {
           System.out.print("Enter month number <value between 1 and 12> : ");
           iMonthNum = kb.nextInt();
       }
       while (iMonthNum < 1 || iMonthNum > 12);
       return iMonthNum;
    } //end getValidMonth
```

```
      public static void main(String[] args) {
         int iMonth = getValidMonth();
         displaySeason(iMonth);
     } // end main
   } // end class
```

Reasons for creating user-defined static methods

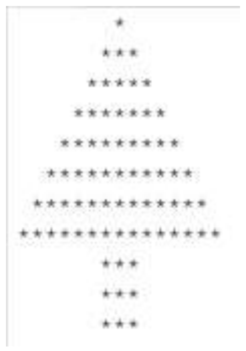
The reasons why it is useful to create user-defined static methods in the 'main' class are:

- When you have code that will be called more than once in a program, it is more efficient to write a method, and call the method more than once. If you have to make changes to the code, you only need to change it once – inside the body of the method. We had two examples where methods were called more than once in a program – Value-returning method example 3: Generate a 'lucky code', and Void method example 1: Display a menu.
- When you have code that is 'complex' you may want to rather enter the code in one or more methods. This will make the main() method easier to read and the programmer can then focus on the flow of the program, rather than the detail. For example, look at this main() method:

```
    public static void main(String[] args) {
        int iMonth = getValidMonth();
        displaySeason(iMonth);
    } // end main
```

Activity 2: Write programs with void methods

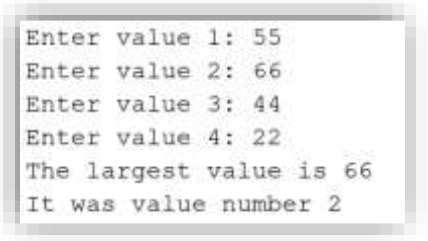
a) Create a Java program that will draw a tree as displayed in the screenshot alongside. A framework of the program is provided. You need to complete the code as indicated. (The tree has 8 rows of stars. The longest row has 15 stars.)



```
       *
      ***
     *****
    *******
   *********
  ***********
 *************
***************
      ***
      ***
      ***
```

```
package drawtree;
public class DrawTree {

    public static void drawBlanks(int iNum)
    {
        Complete the code to draw the required number of spaces.

    } //drawBlanks
    public static void drawStars(int iNum)
    {
            Complete the code to draw the required number of stars.
    } //drawStars

    public static void goNextLine()
    {
        System.out.println();
    } //goNextLine

    public static void main(String[] args) {
        int iCountLines, iCountBlanks = 7, iCountStars = 1;
        //draw the branches
        for (iCountLines = 1; iCountLines <= 8; iCountLines++)
        {
            drawBlanks(iCountBlanks);
            drawStars(iCountStars);
            goNextLine();
            // decrease iCountBlanks by 1
            // increase iCountStars by 2
        }
         //draw the stem

              Complete the code to draw the stem of the tree.

    } //end main

} //end class
```

b) Complete the following program. The purpose of the program is to read 4 integer values, and then display the value as well as the position (index) of the largest number.



```
Enter value 1: 55
Enter value 2: 66
Enter value 3: 44
Enter value 4: 22
The largest value is 66
It was value number 2
```

```
package biggestof4values;
import java.util.Scanner;
public class BiggestOf4Values {

    public static void displayLargest(int i1, int i2, int i3, int i4)
    {
```

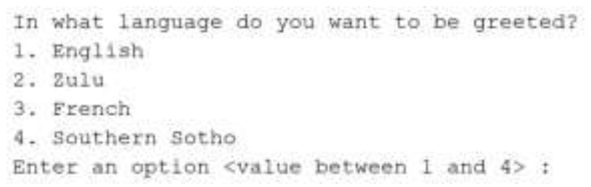
Complete the code to determine the largest of the 4 values as well as the position of the value in the list.

```
        System.out.println("The largest value is " + iLargest);
        System.out.println("It was value number " + iLargestIndex);
    } // end displayLargest

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        int iVal1, iVal2, iVal3, iVal4;

        System.out.print("Enter value 1: ");
        iVal1 = kb.nextInt();
        System.out.print("Enter value 2: ");
        iVal2 = kb.nextInt();
        System.out.print("Enter value 3: ");
        iVal3 = kb.nextInt();
        System.out.print("Enter value 4: ");
        iVal4 = kb.nextInt();
        displayLargest(iVal1, iVal2, iVal3, iVal4);
    } //end main
}
```

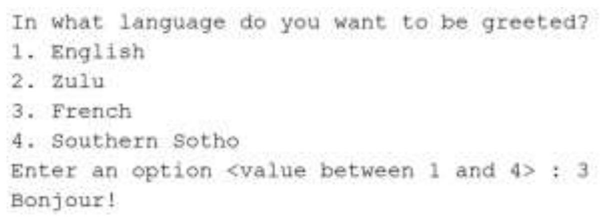
Activity 3: Write programs with value-returning and void methods

a) Create a Java program that will do the following.

- Use a value-returning method getMenuOption to display the menue, and read a menu option until the option entered is a valid value. See Void method example 2: Display the name of a season for an example of such a method.
- Use a void method that receives the menu option (1 – 4), and displays a greeting in the chosen language.



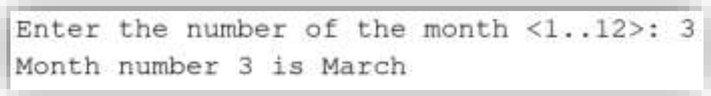
```
In what language do you want to be greeted?
1. English
2. Zulu
3. French
4. Southern Sotho
Enter an option <value between 1 and 4> :
```



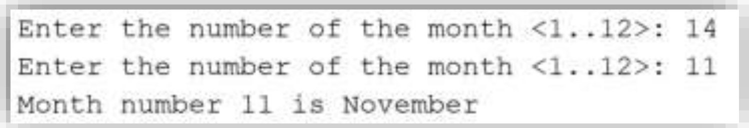
```
In what language do you want to be greeted?
1. English
2. Zulu
3. French
4. Southern Sotho
Enter an option <value between 1 and 4> : 3
Bonjour!
```

b) Create a Java prorgam that will read a month number (1 – 12), and then display the name of the corresponding month.

- Use a value-returning method to read a number until it is valid.
- Use a void method to display the correct name of the month.



```
Enter the number of the month <1..12>: 3
Month number 3 is March
```



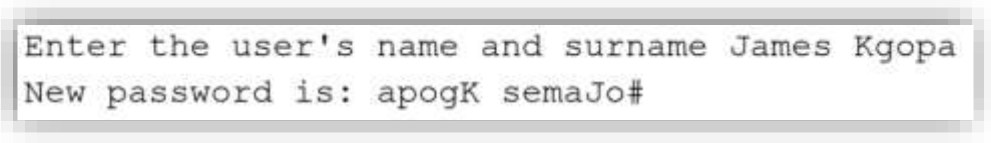
```
Enter the number of the month <1..12>: 14
Enter the number of the month <1..12>: 11
Month number 11 is November
```

c) Write a Java program to create a password for a user according to the following rules:

- Read the name and surname. Use a value-returning method to reverse the order of the letters in the string. (For example the name and surname "James Kgopa" should become " apogK semaJ". (See Unit 5 Activity 22.)
- The password should be the reversed name and surname, plus the third character of the reversed string, plus a random special character between ASCII values 33 and 47 – both included. Use a value-returning method to generate this special character.

This is the code for the main() method.

```
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        String sSentence;
        System.out.print("Enter the user's name and surname ");
        sSentence = keyboard.nextLine();
        createPassword(sSentence);
    } //end main
```



```
Enter the user's name and surname James Kgopa
New password is: apogK semaJo#
```

Integrated example

The following program displays a menu where the user can choose which shape's area to calculate. The main() method calls user-defined methods to read a valid menu option, and to read the dimension sof the different shapes, calculate and display the areas.

```
Calculate areas
1.        Area of a triangle
2.        Area of a circle
3.        Area of a rectangle
4.        Quit
Enter your choice: <value between 1 and 4>
```

```
package calcareas;
import java.util.Scanner;
public class CalcAreas {


    public static void calcTriangle()
    {
        Scanner kb = new Scanner(System.in);
        double rBase, rHeight, rArea;
        System.out.print("Enter base of triangle: ");
        rBase = kb.nextDouble();
        System.out.print("Enter height of triangle: ");
        rHeight = kb.nextDouble();
        rArea = 0.5 * rBase * rHeight;
        System.out.println("The area of the triangle is : " + rArea);
    } //end calcTriangle

    public static void calcCircle()
    {
        Scanner kb = new Scanner(System.in);
        double rRadius, rArea;
        System.out.print("Enter radius of circle: ");
        rRadius = kb.nextDouble();
        rArea = Math.PI * Math.pow(rRadius, 2);
        System.out.println("The area of the circle is : " + rArea);
    } //end calcCircle
    public static void calcRectangle()
    {
        Scanner kb = new Scanner(System.in);
        double rLenght, rWidth, rArea;
        System.out.print("Enter length of rectangle: ");
        rLenght = kb.nextDouble();
        System.out.print("Enter width of rectangle: ");
        rWidth = kb.nextDouble();
        rArea = rLenght * rWidth;
        System.out.println("The area of the rectangle is : " + rArea);
    } //end calcRectangle

    public static int readMenuOption()
    {
        Scanner kb = new Scanner(System.in);
        int iLocal;
        System.out.println("\nCalculate areas\n1.\tArea of a triangle \n2.\tArea of a circle"+
                           "\n3.\tArea of a rectangle \n4.\tQuit");
        System.out.print("Enter your choice: <value between 1 and 4> ");
        iLocal = kb.nextInt();
        return iLocal;
    } //end getMenuOption

    public static boolean isValidMenuOption(int iMenuOption)
    {
        return (iMenuOption >= 1 && iMenuOption <= 4);
    } //end isValidMenuOption
    public static int getValidMenuOption()
    {
       int iLocal;
       do
       {
           iLocal = readMenuOption();
       }
       while (!isValidMenuOption(iLocal));
       return iLocal;
    } // end getValidMenuOption

    public static void main(String[] args) {
        int iMenuOption;

        iMenuOption = getValidMenuOption();
        while (iMenuOption != 4)
        {
            switch (iMenuOption)
            {
                case 1: calcTriangle(); break;
                case 2: calcCircle(); break;
                case 3: calcRectangle();
            } //end switch

            iMenuOption = getValidMenuOption();
        } //end while
    }//end main
}
```