South Louisiana Community College
ASDV 1220, Programming Fundamentals

Work with same partner unless your instructor reassigns you to work with another partner! Use ONE computer together with your partner. ALTERNATE the roles of Coder, Navigator in each problem.

Learning Objectives

After completion of this lab, you should be able to

1. Understand parameters, arguments of static methods.
2. Understand invocation, of static methods.
3. Understand return types types of static methods.
5. Understand overloading of static methods.
6. Understand menus

Create project Lab14

1. Create a class **StaticMethods**. DO NOT add the main method to this class. Type in the method ***printJava*** as shown below.

```
5    public class StaticMethods
6    {
7        public static void printJava()
8        {
9            System.out.println( "Java");
10       }
```

2. From the main method of class Lab14, invoke (call) the method *printJava* as shown below. Does the call of *StaticMethods.printJava();* remind you of calls such as *Math.pow(x, 0.5)* or Double.compare(x, y) ? This is how you can call static methods of another class(*StaticMethods)* , from within a different class (*Lab14*).

```
1    package lab14;
2    public class Lab14
3    {
4        public static void main(String[] args)
5        {
6            StaticMethods.printJava();
```

3. Inside class *StaticMethods,* create the static method *printMyName*() , as shown below that prints your name one time. Call this method from within a for loop inside the main method of class Lab14 to print your name 100 times, 5 times per line.

```
12       public static void printMyName()
```

4. Inside class *StaticMethods,* create the static method *printMessage* shown below that prints a message passed as parameter. Call this method from within a for loop inside the main method of class Lab14 to print "Java is fun!" 100 times, 2 times per line separated by tab.

```
16       public static void printMessage(String msg)
17       {
18           System.out.println( msg);
19       }
```

5. Inside class *StaticMethods,* create the static method *squareNumber* shown below that takes as a parameter a number of type integer and returns the number squared as integer. Call this method from the main method of class Lab14 and print the result.

```
20       public static int squareNumber( int number)
21       {
22           System.out.println( "INT was called");
23           return number * number;
24       }
```

6. Inside class *StaticMethods,* create the OVERLOADED static method *squareNumber* shown below that takes as a parameter a number of type double and returns the number squared as double. Call this method from the main method of class Lab14 and print the result. Do you see what makes Java distinguish overloaded methods at invocation? The argument(s) passed to the method of course.

```
26      public static double squareNumber( double number)
27      {
28          System.out.println( "DOUBLE was called");
29          return number * number;
30      }
```

7. Inside class *StaticMethods,* create the static method cube*Number* as shown below that takes as a parameter a number of type double and returns the number squared, as double. Call this method from the main method of class Lab14 and print the result. Please NOTE how the method cubeNumber, calls the method squareNumber. Look ta the comments inside the method.

```
31      public static double cubeNumber( double number)
32      {
33          return number * squareNumber( number );
34
35          // StaticMethods.squareNumber is also  correct as shown in comment below
36          //return number * StaticMethods.squareNumber( number );
37      }
```

8. Inside class *StaticMethods,* create the static and OVERLOADED method cube*Number* that takes as an argument an integer and returns a double. Test it from main.

9. Inside class *StaticMethods.,* create the static and OVERLOADED method cube*Number* that takes as an argument a String and returns a double. Test it from main.

10. Inside class *StaticMethods,* create the static method *menu* that displays a menu for the user to select 's' or 'c'. The method returns the character entered, ONLY when the user types 's' or 'c', otherwise prints a message and stays in a loop. Test it from main().

```
38      public static char menu()
39      {
40          String s = "";
41          do
42             {
43             System.out.println("type S\\s to square a number:" );
44             System.out.println("type C\\c to cube a number:" );
45             s = new  Scanner( System.in).next();
46
47             if (  "c".compareToIgnoreCase(s ) != 0 && "s".compareToIgnoreCase(s) != 0 )
48                    System.out.println("\tplease enter a valid character" );
49             }
50          while ("c".compareToIgnoreCase(s ) != 0 && "s".compareToIgnoreCase(s) != 0 );
51       return s.charAt(0);
52      }
```

11. Inside class *StaticMethods,* create the static method *menuInteger* that displays a menu for the user to select 1000 , 2000 or 3000 The method returns the INTEGER(int) entered, ONLY when the user types 1000, 2000 or 3000, otherwise prints a message and stays in a loop. Test it from main().

12. Create class *PrimeNumbers.* Test the code given below and understand through the comments and through the code how the first 50 prime numbers are generated and indented.

```
2   public class PrimeNumbers
3   {
4       public static void main(String[] args)
5       {
6           System.out.println("The first 50 prime numbers are \n");
7           printPrimeNumbers(50);
8       }
9       public static void printPrimeNumbers(int numberOfPrimes)
10      {
11          final int NUMBER_OF_PRIMES_PER_LINE = 10; // Display 10 per line
12          int count = 0; // Count the number of prime numbers
13          int number = 2; // A number to be tested for primeness
14
15              //> Repeatedly find prime numbers
16          while (count < numberOfPrimes)
17            {
18                    //>> Print the prime number and increase the count
19            if (isPrime(number))
20              {
21                count++; // Increase the count
22
23                if (count % NUMBER_OF_PRIMES_PER_LINE == 0)
24                  {
25                    //>> Print the number and advance to the new line
26                    System.out.printf("%-5d\n", number);
27                  }
28                else
29                  {
30                    System.out.printf("%-5d", number);
31                  }
32              }
33
34                    //>> Check if the next number is prime
35            number++;
36            }
37      }
38      public static boolean isPrime(int number)
39      {
40          for (int divisor = 2; divisor <= number / 2; divisor++)
41            {
42              if (number % divisor == 0)
43                { // If true, number is not prime
44                  return false; // number is not a prime
45                }
46            }
47          return true; // number is prime
48      }
49  }
50
```

13. Inside class *StaticMethods*, create the static method *area()* as described below. Test it from main.

***5.36** (*Geometry: area of a regular polygon*) A regular polygon is an n-sided polygon in which all sides are of the same length and all angles have the same degree (i.e., the polygon is both equilateral and equiangular). The formula for computing the area of a regular polygon is

$$Area = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}$$

Write a method that returns the area of a regular polygon using the following header:

```
public static double area(int n, double side)
```

Write a main method that prompts the user to enter the number of sides and the side of a regular polygon and displays its area. Here is a sample run:

```
Enter the number of sides: 5 Enter
Enter the side: 6.5 Enter
The area of the polygon is 72.69017017488385
```