# Lab 1: Introduction to JAVA.

**Objective(s):**

1. What is Java?
2. Features and applications of Java.
3. JAVA Basics.
4. Installing JDK.
5. Writing HelloWorld.java in Text Editor
6. JAVA variables & data types.
7. Input & Output
8. Java Variable Type Conversion & Type Casting

## 1: What is JAVA?

Java was developed by Sun Microsystems in 1995, but it has stood the test of time and remains highly relevant and widely used to this day. Java is a high-level, general-purpose object-oriented programming language.

History of JAVA

### Java Versions

There are many java versions that has been released. Current stable release of Java is Java SE 8.

1. JDK Alpha and Beta (1995)
2. JAVA 1.0 (23rd Jan, 1996)
3. JAVA 1.1 (19th Feb, 1997)
4. JAVA 1.2 (8th Dec, 1998)
5. JAVA 1.3 (8th May, 2000)
6. JAVA 1.4 (6th Feb, 2002)
7. JAVA 5.0 (30th Sep, 2004)
8. JAVA 6 (11th Dec, 2006)
9. JAVA 7 (28th July, 2011)
10. JAVA 8 (18th March, 2014)
11. JAVA 9 (21th Sep, 2017)

12. JAVA 10 (20th March, 2018)
13. JAVA 11 (25th Sep 2018)
14. JAVA 12 (19th March 2019)
15. JAVA 13 (17th Sep 2019)

**Java Platforms**

According to Oracle, there are four platforms of the Java programming language

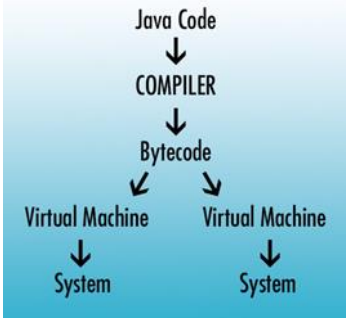
- Java Platform, Standard Edition (Java SE)
- Java Platform, Enterprise Edition (Java EE)
- Java Platform, Micro Edition (Java ME)
- JavaFX

## 2: Features and applications of JAVA

**Features of Java**

The following are Java's main features:

- **High level and general purpose**: Rather than being created to accomplish one very specific task, Java allows us to write computer-readable instructions in an open-ended environment. Because it's not really feasible, or even desirable, for every computer system to have its own specialized programming language, the vast majority of the code is written in high-level, general-purpose languages such as Java.
- **Object-oriented**: Java is also what we call an object-oriented language. While we won't get into the specifics of objects and classes until a bit later in this book, know for now that objects allow us to define modular entities within our program that make them much more human-readable and much more manageable to create large-scale software projects. A firm grasp of object-oriented concepts is absolutely essential for any modern software developer.
- **Platform-independent**: Lastly, Java was designed with the intention that it be a write once, run anywhere language. This means if you and I both have systems with Java installed and even if our systems are not normally identical--for example, I'm on a Windows machine and you're on a Mac--a Java program on my machine that I give to you will still run essentially the same on your machine without the need for it to be recompiled.



**Note:** Compiling a programming language such as JAVA is the act of taking the human-readable code that we've written and converting it into an interpreted machine-friendly code. Unfortunately, it's usually not very friendly for humans to read or write. To do this, we use a program called the compiler that takes in our code as text and converts it into machine code.

Traditionally, we would have to recompile a program for every system that it was going to run on because all systems have a different idea of what their machine code should look like. Java circumvents this issue by compiling all Java programs to the same type of interpreted code called bytecode.

A compiled Java program in bytecode can be run by any system in which Java is installed. This is because when we install Java on your system, we also install a Java virtual machine with it that's specific to that system. It is this machine's responsibility to convert the bytecode into the final instructions that head to the processor in that system.

By making it the system's responsibility to do this final conversion, Java has created a write once, run anywhere language where I can hand you a Java program and you can run it on your machine while being fairly certain that it's going to run in the same manner that it did on mine. This impressive level of cross-platform support on a language as powerful as Java has made it one of the software developing world's go-to tools for quite some time.

"Write once, run everywhere"

**JAVA Applications:**

In today's modern times, Java is used to develop desktop applications, web servers, and client-side web applications. It's the native language of the Android operating system, which operates on Android phones and tablets.

Java has been used to write video games and is sometimes even ported to smaller devices without a traditional operating system. It remains a huge player in today's technical world.

## 3: JAVA Basics

**JDK** stand for **Java Development Kit**, is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development.

**JRE** stands for **Java Runtime Environment.** The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the *Java Virtual Machine (JVM), core classes,* and *supporting files.*

**JVM:** A **Java Virtual Machine** (JVM), an implementation of the Java Virtual Machine Specification, interprets compiled Java binary code (called bytecode) for a computer's processor (or "hardware platform") so that it can perform a Java

program's instructions. Java was designed to allow application programs to be built that could be run on any platform without having to be rewritten or recompiled by the programmer for each separate platform.

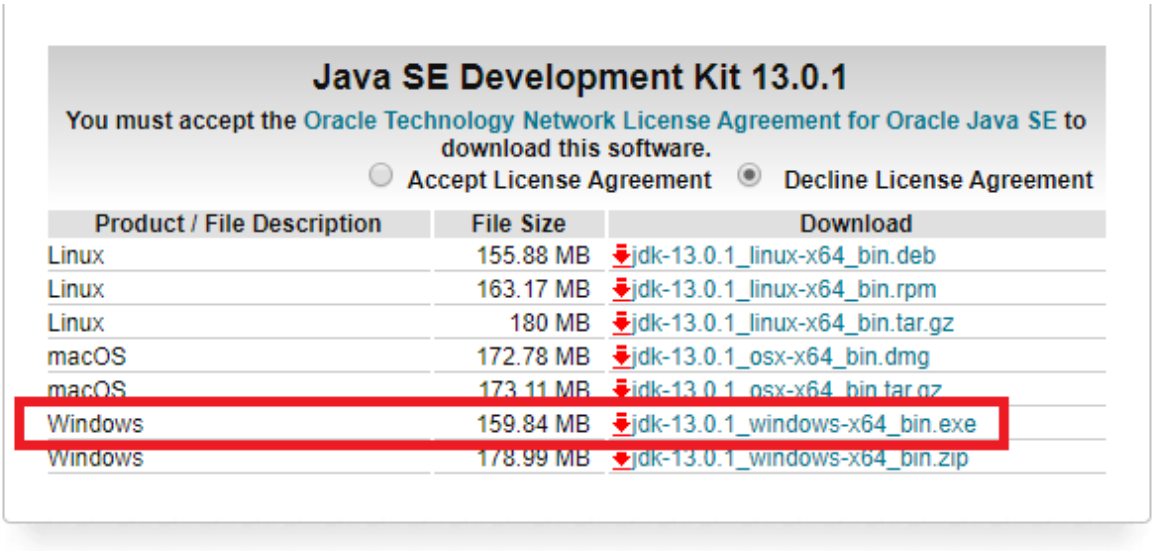
**JIT Just-in-time Compiler** is the part of the Java Virtual Machine (JVM) that is used to speed up the execution time. JIT interpret parts of the bytecode that have similar functionality at the same time, and hence reduces the amount of time needed for full interpretation.

## 4: Installing JDK

### Java Development Kit

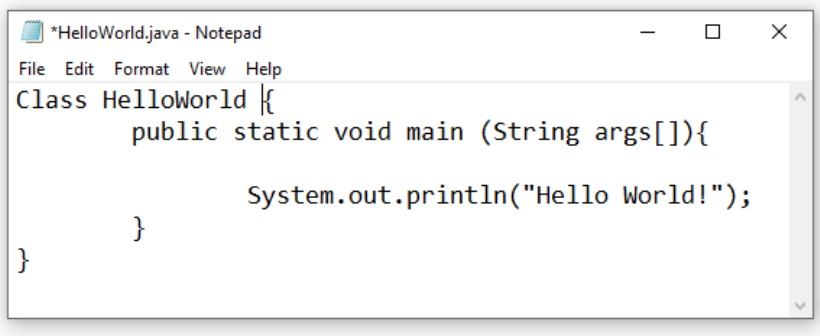
To develop Java applications on our computers, we require a JDK. Visit the link below to download the JDK setup.

Download JDK




## 5: Writing HelloWorld.java in Text Editor

1. **Create a source file -** A source file contains code, written in the Java programming language, that you and other programmers can understand. You can use any text editor to create and edit source files. Run notepad and enter your code. Save this file with the .java extension.




2. **Compile the source file into a .class file -** Open Command Prompt and enter **javac *filename*.java.** The Java programming language *compiler* (javac) takes your source file and translates its text into instructions known as *bytecodes*. Next, enter **java *ClassName*.** The Java application *launcher tool* (java) uses the Java virtual machine to run your application.

# 6: JAVA Variables & Data Types

## Types of variabless

In Java, there are three types of variables:

1. Local Variables
2. Instance Variables
3. Static Variables

## 1. Local Variables

Local Variables are a variable that are declared inside the body of a method.

## 2. Instance Variables

Instance variables are defined without the STATIC keyword .They are defined outside a method declaration. They are Object specific and are known as instance variables.

## 3. Static Variables

Static variables are initialized only once, at the start of the program execution. These variables should be initialized first, before the initialization of any instance variables.

## Example: Types of Variables in Java

```
class Variables {
    int data = 99; //instance variable
```

```
    static int a = 1; //static variable
    void method() {
        int b = 90; //local variable
    }
}
```

## Data Types in Java

Data types classify the different values to be stored in the variable. In java, there are two types of data types:

1. Primitive Data Types
2. Non-primitive Data Types

## Primitive Data Types

Primitive Data Types are predefined and available within the Java language. Primitive values do not share state with other primitive values.

There are 8 primitive types: byte, short, int, long, char, float, double, and boolean

### Integer data types

```
byte (1 byte)
short (2 bytes)
int (4 bytes)
long (8 bytes)
```

### Floating Data Type

```
float (4 bytes)

double (8 bytes)
```

### Textual Data Type

```
char (2 bytes)
```

### Logical

```
boolean (1 bit) (true/false)
```

# Lab 7: Input & Output

**Output in Java Syntax:**

```
System.out.println("Hello World");
```

**Input in Java Syntax:**

```
// import library
import java.util.Scanner;

// Creating scanner object
Scanner obj = new Scanner(System.in);

Taking input from user
int num = obj.nextInt(); // Integer Input
double num = obj.nextDouble(); // Double Input
```

## Lab 8: Java Variable Type Conversion & Type Casting

A variable of one type can receive the value of another type. Here there are 2 cases.

**Case 1)** Variable of smaller capacity is be assigned to another variable of bigger capacity.

```
double d ;
int i = 10;
d = i;
```

This process is Automatic, and non-explicit is known as ***Conversion***

**Case 2)** Variable of larger capacity is be assigned to another variable of smaller capacity.

```
double d   = 10;
int i;
i = (int) d
```

Type Cast Operator

In such cases, you have to explicitly specify the **type cast operator. This process is known as *Type Casting*.**

In case, you do not specify a type cast operator; the compiler gives an error. Since this rule is enforced by the compiler, it makes the programmer aware that the conversion he is about to do may cause some loss in data and prevents **accidental losses.**

**Example: To Understand Type Casting**

```
class Demo {
 public static void main(String args[]) {
  byte x;
  int a = 270;
  double b = 128.128;
  System.out.println("int converted to byte");
  x = (byte) a;
  System.out.println("a and x " + a + " " + x);
  System.out.println("double converted to int");
  a = (int) b;
  System.out.println("b and a " + b + " " + a);
  System.out.println("\ndouble converted to byte");
  x = (byte)b;
  System.out.println("b and x " + b + " " + x);
 }
}
```

**Output:**

```
int converted to byte
a and x 270 14
double converted to int
b and a 128.128 128

double converted to byte
b and x 128.128 -128
```

# Lab 9: if/else, loops

## If/Else in Java Syntax

```
if(condition1)
```

```
 {
      //code to be executed if condition1 is true
 }
else if(condition2)
 {
      //code to be executed if condition2 is true
 }
else if(condition3)
 {
      //code to be executed if condition3 is true
 }
 ...
else
 {
      //code to be executed if all conditions are false
 }
```

### For Loop Syntax in Java

```
for(initialization;condition;incr/decr){
      //statement or code to be executed
}
```

## Lab Tasks:

### Exercise 1 (JAVA Environment Installation & Error Messages)

1. Set up a Java development environment. In the *main()* method of your program try to compile the following invalid Java code snippets. Record the error messages you receive. What do you think each error message indicates?

```
System.out.printn(“Hello World”)


System.out.printn(Hello World)


System.out.println”Hello World”;
```

```
println(“Hello World);
```

2. To generate one final error message, remove one of the brackets from the end of your program. Now what message do you receive?

## Exercise 2 (Mathematical Expressions)

Write Java code for following mathematical expressions to compute the result.
f = 5 * 7 / 3

float f = (5 * 7) / 3.0f

int x = 5 + ‘a’

int x = 2147483647 + 1

## Exercise 3 (Java Variables)

Write a Java code in which create all of the primitives (except long and double) with different values. Concatenate them into a string and print it to the screen so it will print:
**H3110 w0r1d 2.0 true**

## Exercise 4 (Operators)

1. Write Java program to allow the user to input two integer values and then the program prints the results of adding, subtracting, multiplying, and dividing among the two values. See the example below:

```
Enter value a:30

Enter value b:10

The result of adding is 40.

The result of subtracting is 20;

The result of multiplying is 300.

The result of dividing is 3.
```

2. Change the following program to use compound assignments:

```
class ArithmeticDemo {
    public static void main (String[] args){
        int result = 1 + 2; // result is now 3
        System.out.println(result);

        result = result - 1; // result is now 2
        System.out.println(result);

        result = result * 2; // result is now 4
        System.out.println(result);

        result = result / 2; // result is now 2
        System.out.println(result);

        result = result + 8; // result is now 10
        result = result % 7; // result is now 3
        System.out.println(result);
    }
}
```

## Exercise 5 (if/else & loops)

1. Given Boolean variables A, B, and C, write if statements to check for the following conditions:
   - A and B, but not C
   - Either A, B, or C
   - All or none of A, B, and C
   - A and B
2. Using a while loop, write Java code which will multiply all the integers from 1 to a given value. Now, utilize a for loop to generate the same functionality.

# Lab 02: Structured Programming with Java

**Objective(s):**

1. Control Statements
2. Strings in JAVA
3. Arrays in JAVA
4. Java Math Class

## 1: Control Statements

Control structures are programming blocks that can change the path we take through those instructions.

**There are three kinds of control structures:**

- **Conditional Branches**, which we use for choosing between two or more paths. There are three types in Java: if/else/else if, ternary operator and switch.
- **Loops** that are used to iterate through multiple values/objects and repeatedly run specific code blocks. The basic loop types in Java are for, while and do while.
- **Branching Statements**, which are used to alter the flow of control in loops. There are two types in Java: break and continue.

### If / else/ else if:

The *if/else* statement is the most basic of control structures, but can also be considered the very basis of decision making in programming.

While *if* can be used by itself, the most common use-scenario is choosing between two paths with *if/else*:

```
if (count > 2) {
    System.out.println("Count is higher than 2");
} else {
    System.out.println("Count is lower or equal than 2");
}
```

Theoretically, we can infinitely chain or nest if/else blocks but this will hurt code readability,

and that's why it's not advised.

## Ternary Operator:

We can use a ternary operator as a shorthand expression that works like an *if/else* statement.

**Syntax:**

```
System.out.println(count > 2 ? "Count is higher than 2" : "Count is lower or equal than 2");
```

## Switch:

If we have multiple cases to choose from, we can use a switch statement.

**Syntax:**

```
int count = 3;
switch (count) {
case 0:
    System.out.println("Count is equal to 0");
    break;
case 1:
    System.out.println("Count is equal to 1");
    break;
default:
    System.out.println("Count is either negative, or higher than 1");
    break;
}
```

Three or more if/else statements can be hard to read. As one of the possible workarounds, we can use switch, as seen above.

## Loops:

We use loops when we need to repeat the same code multiple times in succession.

**Syntax:**

```
for (int i = 1; i <= 50; i++) {
    System.out.println("Hello World!");
}

int whileCounter = 1;
while (whileCounter <= 50) {
    System.out.println("Hello World!");
    whileCounter++;
}
```

## Break:

We need to use break to exit early from a loop.

```
List<String> names = getNameList();
String name = "John Doe";
int index = 0;
for ( ; index < names.size(); index++) {
    if (names[index].equals(name)) {
        break;
    }
}
```

Here, we are looking for a name in a list of names, and we want to stop looking once we've found it.

A loop would normally go to completion, but we've used break here to short-circuit that and exit early.

### Continue:

Simply put, continue means to skip the rest of the loop we're in:

**Syntax:**

```
List<String> names = getNameList();
String name = "John Doe";
String list = "";
for (int i = 0; i < names.size(); i++) {
    if (names[i].equals(name)) {
        continue;
    }
    list += names[i];
}
```

Here, we skip appending the duplicate names into the list.

As we've seen here, break and continue can be handy when iterating, though they can often be rewritten with return statements or other logic

## 2: Strings in JAVA

The **char** type represents only one character. To represent a string of characters, use the data type called **String**.

For example, the following code declares **message** to be a string with the value **"Welcome to Java"**.

```
String message = "Welcome to Java";
```

**String** is a predefined class in the Java library, just like the classes **System** and **Scanner**. The **String** type is not a primitive type. It is known as a *reference type*. Any Java class can be used as a reference type for a variable. The variable declared by a reference type is known as a reference

variable that references an object. Here, **message** is a reference variable that references a string object with contents **Welcome to Java**.

The `java.lang.String` class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

**Getting String Length**

You can use the `length()` method to return the number of characters in a string. For example, the following code:

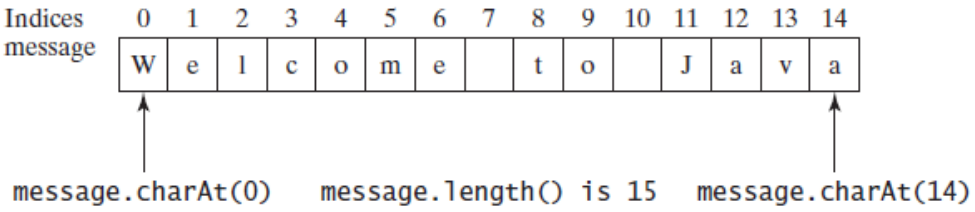
```
String message = "Welcome to Java";
System.out.println("The length of " + message + " is "+ message.length());
```

**Getting Characters from a String**

The `s.charAt(index)` method can be used to retrieve a specific character in a string **s**, where the index is between **0** and `s.length()`**–1**.

For example, `message.charAt(0)` returns the character **W**, as shown in figure.

**Note** that the index for the first character in the string is 0.




**Converting Strings**

The `toLowerCase()` method returns a new string with all lowercase letters and the `toUpperCase()` method returns a new string with all uppercase letters.

For example,

`"Welcome".toLowerCase()` returns a new string **welcome**.

`"Welcome".toUpperCase()` returns a new string **WELCOME**.

The `trim()` method returns a new string by eliminating whitespace characters from both ends of the string. The characters ' ', **\t**, **\f**, **\r**, or **\n** are known as *whitespace characters*.

## 3: Arrays in JAVA

An array stores a sequence of values that are all of the same type. We want not just to store values but also to be able to quickly access each individual value. The length of an array is established when the array is created. After creation, its length is fixed. Each item in an array is

called an *element*, and each element is accessed by its numerical *index*. The method that we use to refer to individual values in an array is to number and then *index* them—if we have *n* values, we think of them as being numbered from 0 to *n*−1.

Making an array in a Java program involves three distinct steps:

- Declare the array name.
- Create the array.
- Initialize the array values.

We refer to an array element by putting its index in square brackets after the array name.

To use an array in a program, you must declare a variable to reference the array and specify the array's *element type*.

Here is the syntax for declaring an array variable:

```
elementType[] arrayRefVar;
```

The **`elementType`** can be any data type, and all elements in the array will have the same data type.

Unlike declarations for primitive data type variables, the declaration of an array variable does not allocate any space in memory for the array. It creates only a storage location for the reference to an array. If a variable does not contain a reference to an array, the value of the variable is **null**. You cannot assign elements to an array unless it has already been created. After an array variable is declared, you can create an array by using the **new** operator and assign its reference to the variable with the following **syntax**:

```
arrayRefVar = new elementType[arraySize];
```

Java has a shorthand notation, known as the *array initializer*, which combines the declaration, creation, and initialization of an array in one statement using the following **syntax**:

```
elementType[] arrayRefVar = {value0, value1, ..., valuek};
```

**Arrays Class**

Arrays class which is in `java.util.Arrays` package, is a provision by Java that provides you a number of methods through which arrays can be manipulated. This class also lets you perform sorting and searching operations on an array.

## 4: JAVA Math Class

The Java programming language supports basic arithmetic with its arithmetic operators: +, -, *, /, and %. The `Math` class provides methods and constants for doing more advanced mathematical computation.

The `Math` is located in the `java.lang` package, and not in the `java.math` package. Thus, the fully qualified class name of the Math class is `java.lang.Math`

The methods in the Math class are all static, so you call them directly from the class, like this:

```
Math.cos(angle);
```

**Note:** Using the static import language feature, you don't have to write Math in front of every math function: `import static java.lang.Math.*;`

This allows you to invoke the Math class methods by their simple names.

For example: `cos(angle);`

**Constants**

The `Math` class includes two constants:

- `Math.E`, which is the base of natural logarithms, and
- `Math.PI`, which is the ratio of the circumference of a circle to its diameter.

**Basic Math Mehtods**

The `Math` class includes more than 40 static methods. They can be categorized as *trigonometric methods*, *exponent methods*, and *service methods*. Service methods include the rounding, min, max, absolute, and random methods.

**Trigonometric Methods**

The `Math` class contains the following methods.

| *Method* | *Description* |
|---|---|
| `sin(radians)` | Returns the trigonometric sine of an angle in radians. |
| `cos(radians)` | Returns the trigonometric cosine of an angle in radians. |
| `tan(radians)` | Returns the trigonometric tangent of an angle in radians. |
| `toRadians(degree)` | Returns the angle in radians for the angle in degree. |
| `toDegree(radians)` | Returns the angle in degrees for the angle in radians. |
| `asin(a)` | Returns the angle in radians for the inverse of sine. |
| `acos(a)` | Returns the angle in radians for the inverse of cosine. |
| `atan(a)` | Returns the angle in radians for the inverse of tangent. |

The parameter for `sin`, `cos`, and `tan` is an angle in radians. The return value for `asin`, `acos`, and `atan` is a degree in radians in the range between -pi/2 and pi/2.

- One degree is equal to pi/180 in radians,
- 90 degrees is equal to pi/2 in radians
- 30 degrees is equal to pi/6 in radians.

**Exponent Methods**

There are five methods related to exponents in the `Math` class.

| *Method* | *Description* |
|---|---|
| `exp(x)` | Returns e raised to power of x ($e^x$). |
| `log(x)` | Returns the natural logarithm of x ($\ln(x) = \log_e(x)$). |
| `log10(x)` | Returns the base 10 logarithm of x ($\log_{10}(x)$). |
| `pow(a, b)` | Returns a raised to the power of b ($a^b$). |
| `sqrt(x)` | Returns the square root of x ($\sqrt{x}$) for x >= 0. |

**The Rounding Methods**

The `Math` class contains five rounding methods

| *Method* | *Description* |
|---|---|
| `ceil(x)` | x is rounded up to its nearest integer. This integer is returned as a double value. |
| `floor(x)` | x is rounded down to its nearest integer. This integer is returned as a double value. |
| `rint(x)` | x is rounded up to its nearest integer. If x is equally close to two integers, the even one is returned as a double value. |
| `round(x)` | Returns (int)Math.floor(x + 0.5) if x is a float and returns (long)Math.floor(x + 0.5) if x is a double. |

**The Service Methods**

The `min`, `max`, and `abs` Methods

The `min` and `max` methods return the minimum and maximum numbers of two numbers (`int`, `long`, `float`, or `double`).

For example, `max(4.4, 5.0)` returns `5.0`, and `min(3, 2)` returns **2**.

The `abs` method returns the absolute value of the number (`int`, `long`, `float`, or `double`).

This method generates a random `double` value greater than or equal to 0.0 and less than 1.0 (`0 <= Math.random() < 1.0`). You can use it to write a simple expression to generate random numbers in any range.

## Lab Tasks:

### Exercises

1. **Game:** How many fingers am I holding up? Write a program that prompts user to guess the number of fingers that the system is holding up. The program should then tell the user whether the guesses number was correct or not. (**Hint:** Use math functions)

2. **Wind-chill Temperature**

   How cold is it outside? The temperature alone is not enough to provide the answer. Other factors including wind speed, relative humidity, and sunshine play important roles in determining coldness outside.

   In 2001, the National Weather Service (NWS) implemented the new wind-chill temperature to measure the coldness using temperature and wind speed. The formula is

   $$t_{wc} = 35.74 + 0.6215t_a - 35.75v^{0.16} + 0.4275t_av^{0.16}$$

   where $t_a$ is the outside temperature measured in degrees Fahrenheit and v is the speed measured in miles per hour. $t_{wc}$ is the wind-chill temperature. The formula cannot be used for wind speeds below 2 mph or temperatures below -58 $^oF$ or above 41$^oF$.

   Write a program that prompts the user to enter a temperature between -58 $^oF$ and 41$^oF$ and a wind speed greater than or equal to 2 and displays the wind-chill temperature. **(Hint:**Use Math.pow(a, b) to compute $v^{0.16}$.**)**

3. **Generate vehicle plate numbers**
   Assume a vehicle plate number consists of three uppercase letters followed by four digits. Write a program to generate a plate number.

4. **Beautifying the sentences**

   Write an application that takes a sentence from user and checks if the sentence stars from a capital letter and ends with a full stop. If it doesn't, the program should add it.

5. Given the following array, display its data graphically by plotting each numeric value as a bar of asterisks (*) as shown in the diagram.

   ```
   int[] array = {10, 19, 5, 1, 7, 14, 0, 7, 5};
   ```

```
Element  Value   Histogram
0        10      **********
1        19      ******************
2        5       *****
3        1       *
4        7       *******
5        14      **************
6        0
7        7       *******
8        5       *****
```

6. Write a Java program(with switch) that allows the user to choose the correct answer of a question.

   See the example below:

   **What is the correct way to declare a variable to store an integer value in Java?**

   a. int 1x=10;

   b. int x=10;

   c. float x=10.0f;

   d. string x="10";

   Enter your choice: c

7. Write a Java program to test if a given string contains the specified sequence of char values.
8. **Random Sentences**

   Write an application that uses random-number generation to create sentences. Use four arrays of strings called article, noun, verb and preposition. Create a sentence by selecting a word at random from each array in the following order: article, noun, verb, preposition, article and noun. As each word is picked, concatenate it to the previous words in the sentence.

   The words should be separated by spaces. When the final sentence is output, it should start with a capital letter and end with a period. The application should generate and display 20 sentences.

   The article array should contain the articles "the", "a", "one", "some" and "any"; the noun array should contain the nouns "boy", "girl", "dog", "town" and "car"; the verb array should contain the verbs "drove", "jumped", "ran", "walked" and "skipped"; the preposition array should contain the prepositions "to", "from", "over", "under" and "on".

9. **Game: heads or tails**

   Write a program that lets the user guess whether the flip of a coin results in heads or tails. The program randomly generates an integer 0 or 1, which represents head or tail. The program prompts the user to enter a guess and reports whether the guess is correct or incorrect.

Sukkur IBA University
Merit - Quality - Excellence

**CSC-150 – Object Oriented Programming**
**Semester II (Spring 2020)**
**Course Instructor(s): Dr. Sher Muhammad, Muhammad Atif, Khalid Hussain**

## Lab 12: Graphics With Swing

1. Design and code a Swing GUI to translate text that is input in English into Pig Latin. You can assume that the sentence contains no punctuation. The rules for Pig Latin are as follows:

    a. For words that begin with consonants, move the leading consonant to the end of the word and add “ay.” Thus, “ball” becomes “allbay”; “button” becomes “uttonbay”; and so forth.

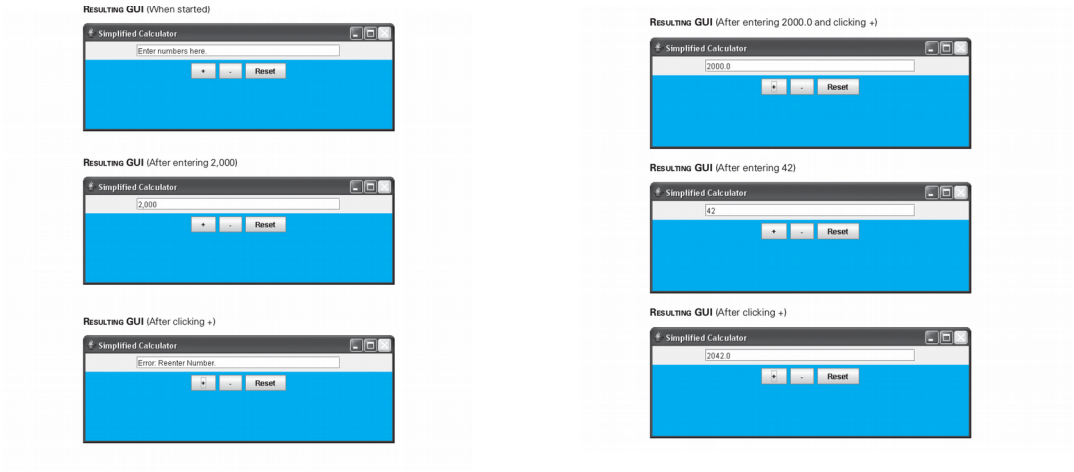
    b. For words that begin with vowels, add “way” to the end of the word. Thus, “all” becomes “allway”; “one” becomes “oneway”; and so forth.

2. Design and code a Swing GUI for a two-player tic-tac-toe (noughts and crosses) game on a 3 * 3 game board. The JFrame should use a BorderLayout with a JLabel in the NORTH region to display messages (e.g., who won the game), and a JPanel in the CENTER region to display the game board. For the game board in the JPanel , use a GridLayout manager with a 3 * 3 layout of JButton ’s in each cell to display the game board. The button labels should initially be blank. When a player clicks on an empty button, an appropriate “X” or “O” should be placed in the label field of the button. If there is a winner (three in a row), then the program should display the winner in the JLabel located at the top of the window. If all nine cells have been filled without a winner, the program should indicate that there is a tie.

3. Design and code a Swing GUI calculator. You can use Picture below as a starting point, but your calculator will be more sophisticated. Your calculator will have two text fields that the user cannot change: One labeled "Result" will contain the result of performing the operation, and the other labeled "Operand" will be for the user to enter a number to be added, subtracted, and so forth from the result. The user enters the number for the "Operand" text field by clicking buttons labeled with the digits 0 through 9 and a decimal point, just as in a real calculator. Allow the operations of addition, subtraction, multiplication, and division. Use a GridLayout manager to produce a button pad that looks similar to the keyboard on a real calculator.

When the user clicks a button for an operation, the following occurs: the operation is performed, the "Result" text field is updated, and the "Operand" text field is cleared. Include a button labeled "Reset" that resets the "Result" to 0.0 . Also include a button labeled "Clear" that resets the "Operand" text field so it is blank.

Hint: Define an exception class named DivisonByZeroException . Have your code throw and catch a DivisonByZeroException if the user attempts to “divide by zero.” Your code will catch the DivisonByZeroException and output a suitable message to the "Operand" text field. The user may then enter a new substitute number in the "Operand" text field. Because values of type double are, in effect, approximate values, it makes no sense to test for equality with 0.0 . Consider an operand to be “equal to zero” if it is in the range -1.0e-10 to +1.0e-10 .




4. (The Swing part of this project is pretty easy, but to do this programming project you need to know how to convert numbers from one base to another.) Write a program that converts integers from base ten (ordinary decimal) notation to base two notation. Use Swing to perform input and output via a window interface. The user enters a base ten numeral in one text field and clicks a button with "Convert" written on it; the equivalent base two numeral then appears in another text field.
Be sure to label the two text fields. Include a "Clear" button that clears both text fields when clicked. ( Hint: Include a private method that converts the string for a base ten numeral to the string for the equivalent base two numeral.)

5. Here we will a game known as Trivia (Kon banayga Crorepati) use a layout of your choice with the appropriate text fields, labels, and buttons to implement your design. The game should ask only one question at a time with the time limit of 60 seconds to answer and output the correct answer if the player answers a question incorrectly or time limit exceeds (if time limit ends answer will be considered wrong). When all questions have been answered, show the final score and exit the program.