INTRODUCTION TO

What is Java?

- Java is an object-Oriented programming language.
- Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).
- Java was released as a language that guarantees the “write once run anywhere” (WORA) capability.
  - This means that the software created using Java can run in any device despite hardware and software architecture.
- In 2006 sun microsystems released java as a free and open source software under the terms of the General public use license.
- Java was later acquired by Oracle and is currently maintained by Oracle.

Java Configurations

- With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms.
- They include:
  - **Java EE**(Java Enterprise Edition)- used to develop enterprise applications such as servlets, java server pages, among other online software .
  - **Java ME** (Java Micro Edition)- used for mobile application development
  - **Java SE** (Java standard Edition)- used for development of desktop applications.
- Java SE is the most used JAVA variant. JAVA SE 17.0.1 is the latest version of the SE.
- Our aim here is to develop desktop applications thus we will focus purely on the JAVA standard Edition

Features of Java

- **Object Oriented:** In Java, everything is an Object and such an object must belong to a class. Java can be easily extended since it is based on the Object model.
- **Platform Independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code.
  - This byte code is distributed over the web and interpreted by the **Java Virtual Machine (JVM)** on whichever platform it is being run on. JVM is discuss later in this lesson.
  - This makes it possible for Java to run any machine regardless of the software and hardware architecture
- **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP, Java would be easy to master.
- **Secure:** With Java's security features it is possible to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

Cont.

- **Architecture-neutral:** Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable:** Being architecture-neutral and being implementation neutral makes Java portable. Thus applications developed in Java can be moved from one machine to another easily.
- **Robust:** Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

Cont.

- **Interpreted:** Java is translated from high-level language to machine code one statement at a time.
- **High Performance:** In order to improve performance, Just-In-Time compilers interact with the Java Virtual Machine (JVM) at run time and compile suitable bytecode sequences into native machine code.
- **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment.
  - Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.
  - Also the three configurations of Java make it possible to develop software in almost any scope.

Tools Needed to program in Java

- **Java Development Kit (JDK)-**
  - is a set of tools for developing Java applications.
  - Every JDK always includes a compatible JRE, because running a Java program is part of the process of developing a Java program.
- **Java Runtime Environment (JRE)-**
  - it is a software layer that runs on top of a computer's operating system software.
  - It provides the class libraries and other resources that a specific Java program needs to run.
  - The JRE combines Java code created using the JDK with the necessary libraries required to run it on a JVM and then creates an instance of the JVM that executes the resulting program.
  - JVMs are available for multiple operating systems, and programs created with the JRE will run on all of them.
  - In this way, the JVM within the Java Runtime Environment is what enables a Java program to run in any operating system without modification.

JRE Runtime Architecture

- ClassLoader
  - The Java ClassLoader dynamically loads all classes necessary to run a Java program. Since Java classes are only loaded into memory when they're required, the JRE uses ClassLoaders to automate this process on demand.
- Bytecode verifier
  - Bytecode is a code that lies between low-level and high-level language. The bytecode is not processed by the processor. It is processed by the Java Virtual Machine (JVM)
  - The bytecode verifier ensures the format and accuracy of Java code before it passes to the interpreter. In the event that code violates system integrity or access rights, the class will be considered corrupted and won't be loaded.
- Interpreter
  - After the bytecode successfully loads, the Java interpreter creates an instance of the JVM(calls the JVM) that allows the Java program to be executed natively on the underlying machine.
  - This is where the Just-in-time compilers are used to optimize performance

JVM

- **The *Java Virtual Machine (JVM)***
- Executes live Java applications.
  - Every JRE includes a default JVM, but developers are free to choose another that meets the specific resource needs of their applications.
- **JVM can be viewed in three dimensions**
  1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm.
  2. **An implementation** Its implementation is known as JRE (Java Runtime Environment). Its implementation has been provided by Oracle.
  3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

Cont.

- Java applications are called WORA (Write Once Run Anywhere).
  - This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment.
  - This is all possible because of JVM.
- JVM performs the following activities
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment

Cont.

- **Integrated Development Environment:**
- Used for editing Java programs
  - Some of the most popular Java IDE include
    - NetBeans
    - Eclipse
    - Jcreator
    - Visual Studio
    - Dr. Java
- Netbeans is the most recommended IDE because it has powerful tools contained in its Abstract Window toolkit (AWT) and Swing packages that make it possible to create powerful GUI through drug and drop.
  - It also supports Java Database Connectivity (JDBC) which makes it possible to link java to MySQL database.

How to install the Necessary tools

- Download the latest versions of JDK and JRE from oracle official website.
- Download Netbeans IDE.
- Install the software in the following order
  - JDK
  - JRE
  - Netbeans
- Test the success by running a simple java program in Netbeans IDE.

Java Basic Syntax

When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each other's methods.

The syntax revolves around the following elements.

- **Object –** Objects are real world entities that have identity, states and behaviors.
  - Example: Tommy which belongs to class dog has states - color, breed as well as behavior such as wagging their tail, barking, eating.
  - An object is an instance of a class.
- **Class -** A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.
- **Methods -** A method is basically a behavior.
  - A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
  - For example: barking(), eating ().

Cont.

- **Instance Variables –**They are created when objects of a class are created. Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.
  - Example: if we take a class known as dog with the state as color, name, breed and behavior as barking.
  - Then we can create two objects namely Tommy, and Bosco. Now the instance variables can be as follows
    - Tommy.color= black
    - Tommy. Breed= German _shephard
    - Bosco.color=Brown
    - Bosco.Breed=Golden_Retriver
  - So in this case color and breed are instance variables. As you can see each object has its own values for the instance variables.

Java Syntax Rules

- **Case Sensitivity -** Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names -** For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
  - Every Java Program must have a class.
  - **Example**: *class MyFirstJavaClass*
- **Method Names -** All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.
  - **Example**: *public void myMethodName()*
  - *Public string barking()*
- *U*se Comments to enhance code readability. Java uses same comments syntax as C and C++

Syntax Rules Cont.

- **Program File Name -** Name of the program file should exactly match the class name.
  - When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).
  - **Example**: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*
- **public static void main(String args[]) –** This is the main method.
  - Java program processing starts from the main() method which is a mandatory part of every Java program.
- **Ensure all open carry braces and parenthesis are closed-** hanging braces will result to a syntax error.
- **Follow the rules of naming variables-** any variable declared must adhere to the variable naming conventions.
- **Terminate all executable statements using a semicolon**

Key Points to Note

- The name of the class is usually the name you have saved the file as.
  - For example if you create a project and save it as java101 then the name of the package in which it will be saved will be package java101 while the class name will be public class Java101
- Just like in C and C++ the code to be executed is usually written inside the braces of the main method.
- Global variables, methods, and functions can be created outside the main method.
- Any libraries and packages that are to be used are imported when needed and are placed between the package name and the class name

Writing a Simple Java Program

- Open NetBeans
- Go to files and select new project
- Select Java
- Select Java application
- Click Next
- Name the project as java101
- Select use the name as the class name
- Then the following code will be automatically generated for you by NetBeans

Simple Java Program Structure.

```
package java101;

public class Java101 {

public static void main(String[] args) {
        // TODO code application logic here

    }

}
```

Interpreting the code

```
package java101; /*this represents the package where this program will be stored*/
public class Java101 { /* this represents the class. Public means that it is accessible to all parts of the program */


public static void main(String[] args) { //main method or main function
// body of the program
 // write what you want the program to do here
}
   }
```

A simple Java program.

```
package java101;

public class Java101 {

public static void main(String[] args) {

 System.out.println("Welcome To
Java");
    }
}
```

- The following program outputs “Welcome to Java” in console.
- System.out.println- is the outputting function used in Java.
- It is similar to printf in C

Example two: A program to output I love Java

```
package java101;

public class Java101 {

public static void main(String[] args) {

 System.out.println(“I love Java");
    }
}
```

Practice Exercise

- Write a Java program to output the following lines of code

  My name is Evans

  I am Learning Java

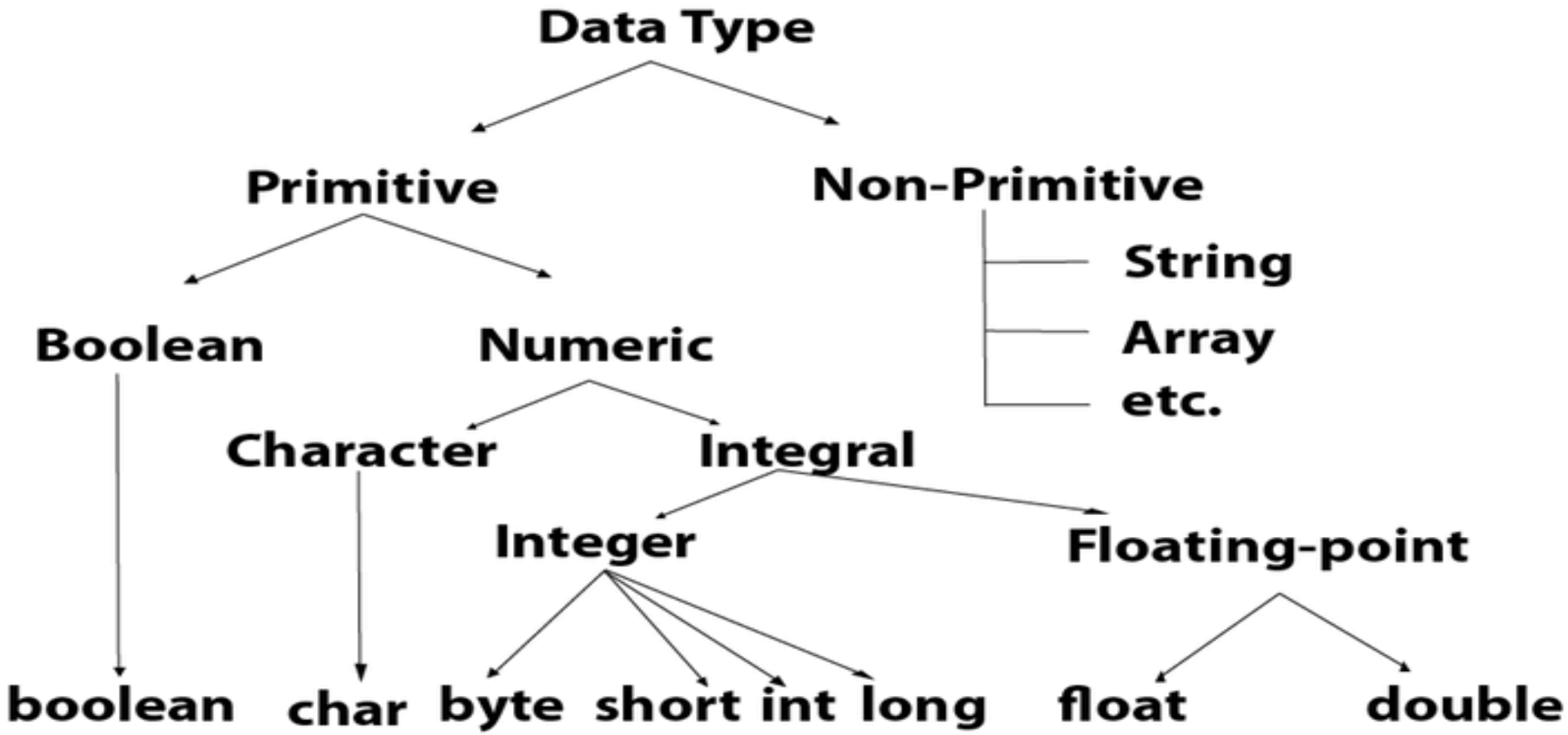
- Write a Java program to output the following in different lines. Note, try an minimize the number of lines of codes used by suing escape sequence
  - Your Name
  - Your home county
  - Your home sub-county
  - Your constituency

Java Datatypes

Categories of Datatypes in Java

- Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:
- **Primitive data types:** The primitive data types are the most basic datatypes in Java.
- They include Boolean, char, byte, short, int, long, float and double.
- **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.
- Assignment: Discuss each of these datatypes in details

A summary of the various categories of datatypes in Java



Primitive/ Basic data types

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Keywords in Java

- They are reserved words in java that have a specific meaning
- These keywords cannot be used as identifier names.
- Some of the common keywords I java include: Class, abstract, goto, Try, catch, int, double, if, else, case, import, private, short, switch, interface, package, throws, switch, static, synchronized, protected, implements, among many others.
- Research on the meaning and purpose of the keywords in Java for purposes of understanding their usage in a program

Java Identifiers

What are Identifiers?

- All Java components require names.
- Names used for classes, variables, arrays, and methods are called **identifiers**.
- There exists some rules that are followed when naming identifiers in Java Programming language
- The rules include:

Rules of naming Identifiers in Java

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- It must be less than 255 characters
- No spacing is allowed
- A key word cannot be used as an identifier.
- The same name cannot be repeated within the same scope
- Most importantly, identifiers are case sensitive.
  - Examples of legal identifiers: age, $salary, _value, __1_value.
  - Examples of illegal identifiers: 123abc, -salary.

Variable declaration

- A variable is a name given to a storage unit in a program
- All variables must be declared before they are used
- General syntax:

Data type variable_name= value

Data type:

- it's a property of a variable that holds the type of data to be stored in that variable
- they are used to define a variable before its' use.
- They include basic datatypes such as int, char, double, float, Boolean.

Cont.

Variable name:

- This the name that you want to give a variable.
- These names are defined by you the programmer. The name you choose should not be a keyword in Java.
- You must adhere to the rules of naming variables discussed in slide 3 above.

Example.

- If you want to write a program to multiply two numbers you can name your variables as.
  - Num1- to represent first number
  - Num2- to represent second number
- Others would just use (a, b) or (x,y) etc.

Initializing variables

To initialize a variable is to assign a value to a variable.

Syntax:

Datatype variable_name=value

e.g

int marks=76

int is the datatype, marks is the variable_name, 76 is the value

This means that marks now stores a value which is 76. so if you output marks it will give you 76. check example below

Cont.

- There are two ways of declaring variables.

1. You can first declare the variable then initialize it later

Example: check Example 1 below

2. You can declare the variable and initialize it in the same line or at the same time.

Example; check example 2 below

Example1: declare the initialize later

```
package java101;

public class Java101 {
    public static void main(String[] args) {

        double n;
        n=10;
        System.out.println(n);
    }

}
```

Example 2:Declare and initialize at the same time

```
package java101;

public class Java101 {

    public static void main(String[] args) {

        double n =10;

        System.out.println(n);
    }

}
```

Declaring multiple variables

- This refers to declaring more than one variables. You can use two approaches.

1. Declare all the variables in the same line without initializing and then initialize later.

2. Declare each variable in its own line and initialize them later

3. Declare each variable in its own line initialize them immediately.

Example:

Think of a program that is supposed to subtract discount from price and output totalcost

Ideally here we have three variables namely (discount, price, totalcost)

Example using approach 1

```
package java101;
public class Java101 {
    public static void main(String[] args) {

        double price, total_cost, discount;
        price=1000;
        discount=50;
        total_cost=price-discount;
        System.out.println(total_cost);
    }
}
```

Example using approach 2

```
package java101;
public class Java101 {
   public static void main(String[] args) {

      double price;
      double total_cost;
      double discount;
      price=1000;
      discount=50;
      total_cost=price-discount;
      System.out.println(total_cost);
   }
}
```

Example using approach 3

```
package java101;
public class Java101 {

    public static void main(String[] args) {

        double price=1000;
        double discount=50;
        double total_cost=price-discount;
        System.out.println(total_cost);
    }
}
```

Types of Variables in Java

- **Local Variables**
- A variable defined within a block or method or constructor is called a local variable.
- These variables are created when the block is entered, or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variable is declared. i.e., we can access these variables only within that block.
- Initialization of the local variable is mandatory before using it in the defined scope.

Types of Variables in Java

- **Instance Variables**
- Instance variables are non-static variables and are declared in a class outside any method, constructor, or block.
- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers, for instance, variables. If we do not specify any access specifier, then the default access specifier will be used.
- Initialization of Instance Variable is not Mandatory. Its default value is 0
- Instance Variable can be accessed only by creating objects.

Types of Variables in Java

- **Static Variables**
- Static variables are also known as Class variables.
- These variables are declared similarly as instance variables. The difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Initialization of Static Variable is not Mandatory. Its default value is 0
- If we access the static variable like the Instance variable (through an object), the compiler will show the warning message, which won't halt the program. The compiler will replace the object name with the class name automatically.
- If we access the static variable without the class name, the compiler will automatically append the class name.

Practice Exercise

- Write a Java program that adds two numbers and prints their sum.
- Create a java program that calculates the area of a circle given that pie is 3.142 and the diameter is 14 cm.
- Write a Java program to find the total marks for a student using the following formula Total_marks=(cat1+cat2+Assignment)/3 +Exam_marks.

Input and output in Java

- So far we have been dealing with programs that do not require the user to input anything from the keyboard.
- We have also seen that System.out.println() is used to output a line in Java.
- So how do we input in java.
- We need to use the Scanner package which is found within the java.util package.
- So you start by importing the package as follows
  - import java.util.Scanner
- Then in the main method you create an instance of this package.

Cont.

- The instance is created as follows:
  - Scanner sc=new Scanner (System.in)
  - So sc here now represents an instance of Scanner whose role is to accept inputs as defined by System.in
- This format of importing packages before using them is very common among powerful programming languages such as python, R, and Java.
- Python is especially prone to importing third party libraries.
- This is so because majority of the powerful libraries are stored as third party libraries.

Example in a program

```
package java101;

import java.util.Scanner;

public class Java101 {
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter a number");
        double x= sc.nextDouble();
        System.out.println(x);
    }
}
```

- In this program we have imported the scanner package.
- We have also created an instance of it in the main method. In this case the instance is sc.
- We have then assigned that instance to variable x.
- So whenever x is called the value of sc (what the user inputted is considered)
- So when the user runs the program, they will get a message “Enter a number”
- Then they type one number e.g 10 then the program will output the number the user inputted.

Example 2

- Think of a program that prompts the user to enter two numbers then calculates the sum of the two numbers, then outputs the sum of the two numbers.
- Check next slide for the solution

Example 2: Solution

```
package java101;
import java.util.Scanner;
public class Java101 {
    public static void main(String[] args) {
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter first Number");
        double x= sc.nextDouble();
        System.out.println("Enter second Number");
        Scanner sc2=new Scanner (System.in);
        double y= sc2.nextDouble();
        double sum=(x+y);
        System.out.println("The sum of the two numbers your entered is:"+sum);
    }
}
```

Example 2: Cont.

- So example 2 is an advancement of example 1 since this one now deals with two numbers.
- Here we have created two instances of Scanner namely sc for the first number and Sc2 for the second number.
- We have also assigned these instances of Scanner to variables x and y to represent the first and second numbers respectively.
- So the number of Scanner instances you create is directly determined by the number of values you wish to input from the keyboard.

Practice Exercise

- Write a Java program that prompts the user to enter two numbers then finds the average of the two numbers and outputs the value of the average.
- Create a java program that calculates the area of a circle. The program should allow the user to input the values of pie and radius from the keyboard.
- Write a Java program to find the total marks for a student using the following formula Total_marks=(cat1+cat2+Assignment)/3 +Exam_marks. The program should allow the user to input the values from the keyboard.

Java Classes and Objects

Classes in Java

- A class is a blueprint from which individual objects are created.
- Here we are going to see how we can create our own classes in Java and define parameters within those classes.
- Previously we have seen that the parameters defined within a class defines what the objects will be made of.
- In that case the state and behavior of objects expected to emanate from a class are what we define as parameters within a class.
- Take an example of a class known as Dog.
  - In this class we can have objects such as MyDog.
  - MyDog can have state such as color, name, age
  - It can also have behavior such as eating, barking, sleeping etc
  - The class would be created as shown below

Java Code to Create A Class Known as Dog

```
public class Dog{
        String breed;
        int age;
        String color;
        void barking(){
        }
        void hungry(){
        }
        void sleeping(){
        }
}
```

- All states are declared just like ordinary variables in java.
- Behavior is declared as a method in java.
- A method in Java follows the following syntax.

  Datatype MethodName () {

  }
- Inside the braces one can put what the method is expected to return or output.
- So in this case we have methods such as

  Void barking () { }

Example two

- Lets create a class called car with the following features.
- State- color, brand, Model
- Behavior- Move, stop, hoot.

```
Public Class car {
        String color;
        String Brand;
        String Model;
        Void Move (){     }
        Void Stop() {     }
        Void hoot () {    }
}
```

Cont.

- A class can have any of the three types of variables. i.e Local variable, Instance variable, and class variables.
- A class can have any number of methods to access the value of various kinds of methods. In the example 1, barking(), hungry() and sleeping() are methods.
- In order to make it possible for objects to be created from classes, Java depends heavily on constructors.
- Lets look at methods and constructors more deeply.

Methods in Java

- A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
- They are also referred to as functions.
- It is used to achieve the **reusability** of code. We write a method once and use it many times.
- We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code.
- The method is executed only when we call or invoke it.
- In classes and objects, methods are used to implement behaviors.

Creating A method

- A method must be declared within a class. It is defined with the name of the method, followed by parentheses **().**
- A method in Java follows the following syntax.

```
Public class ClassName{
        Modifier Datatype methodName () {
        }}
```

- Example think of a class known as cat with method sound.

```
public class Cat {
    static void sound(){
        System.out.println("Meow");
    }
```

Cont.

- Modifiers include the following:
  - Static
  - Public
  - Private
- If you omit the modifier (i.e fail to specify the modifier it takes public modifier by default)
- For example *sound(){}* is the same as *public sound (){}*
- Note that it is advisable to use public only when you want to create an instance (object of a class) where such is not the case always use static.

Cont.

- This is so because static allows you to output the value of the method without necessarily having to create an object of a class unlike public and private in which you first have to create the object of a class.
- Note that when you use static, private or public together with the void datatype you don’t need to have a return systems in the body of the method to specify what the output will be. You can just use Syestem.out.println()
- On the other hand if you use static, public, or private modifiers with any other datatype you must have a return statement in the body. Check the two examples in the next page.

Cont. Example to compare how we output from void datatype and other datatypes

Any other datatype

```
static String bark(){
    return ("woo! Wooo!");
    }
```

Two

```
public String bark(){
    return ("woo! Wooo!");
    }
```

Void datatype

```
static void bark(){
    System.out.println(“Woo! Wooo!);
    }
```

Two

```
public  void bark(){
    System.out.println(“Woo! Wooo!);
    }
```

Calling a method

- You call a method by referring to the method name in the main method. Lets call the method we have created in the previous page.

```
public class Cat {
    static void sound(){
        System.out.println("Meow");
    }
    public static void main (String []args){
    sound();
    } }
```

  The output of this program will be “Meow”

- As you may have noticed methods in Java are called in the same way as functions in other programming languages.

- Lets now use the String datatype in the same example rather than void

```
public class Cat {
    static String sound(){
        return ("Meow");
    }
    public static void main (String []args){
    System.out.println( sound() );    } }
```

The output of this program will be “Meow”

Creating More methods: A method to find the maximum number between two numbers.

```
public class Java102 {
    static double findMax(double a, double
b){
    double x;
    if (a>b){
     x=a;    }
    else if (b>a){
     x=b;
    }
    else { x=0;}
    return x;
    }

 public static void main (String []args) {
    double height=10;
    double width=20;
    double x=findMax(height,width);
    System.out.println(x);
    }


}
```

Cont.

- The methods that you create in Java are highly influenced by the nature of the task you want to perform.
- In the same way you create functions in other programming languages.
- Practice Exercise
  - Create a class known as dog with method bark.
  - Create a class known as student with a method that gives pass if a student has an average marks of 50% above.

SHORT BREAK

What is a constructor

- A constructor in Java is a **special method** that is used to initialize objects.
- It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.
- The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:
- All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero.
- However, once you define your own constructor, the default constructor is no longer used.

```
Class ClassName{
    ClassName(){
    }
}
```

- So here we can see that the constructor shares the same name as a class.
- Java allows two types of constructors namely −
  - No argument Constructors
  - Parameterized Constructors

No Arguments Constructors

- The no argument constructors of Java does not accept any parameters instead, using these constructors the instance variables of a method will be initialized with fixed values for all objects.
- This is more of a static constructor since the value defined cannot change.
- The following is an example:

```
Public class MyClass {
 Int num;
MyClass() {
  num = 100;
}
}
```

- *In this example every object that is created from this class will have its value as 100*

Parameterized Constructors

- Most often, you will need a constructor that accepts one or more parameters.
- Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.
- Thus this one is more dynamic. Check example in the next page

```
class MyClass {
   int x;

MyClass(int i ) {
      x = i;
   }
}
```

- In this constructor the value of i is not fixed since a user can pass any value of x.
- Thus if you have two objects one you assign it the value of int x to be 20 and the other to be 50 this constructor will accept that.

Objects

- An object is created from a class. In Java, the ***new*** keyword is used to create new objects.
- There are three steps when creating an object from a class:
- **Declaration:** A variable declaration with a variable name and an object type.
- **Instantiation:** The 'new' keyword is used to instantiate or create the object.
- **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example 1

```
public class Puppy{
    public Puppy(String name){
    // This constructor has one parameter, name.
    System.out.println("Passed Name is :" + name );
    }

    public static void main(String []args){
    // Following statement would create an object myPuppy
    Puppy myPuppy = new Puppy( "tommy" );
    }
}
```

The output of this code is: Passed Name is :tommy

Example 2:

```
public class Dog {
    public Dog(String Name, String color){
    System.out.println("The Dog's Name is:"+Name);
    System.out.println("The Dog's Color is:" +color);  }

 public static void main(String[] args){
 Dog myDog= new Dog("Bosco","Brown");  }
}
```

The *output here is: The Dog's Name is: Bosco*

The Dogs color is: Brown

Example Three: Where you don’t declare states inside the constructor

```
public class Puppy1 {
String color;
int age;
   public Puppy1(){
      }
public static void main(String[] args) {
       Puppy1 mypuppy= new Puppy1();
       mypuppy.color="white";
      mypuppy.age=10;
      System.out.println("The color is:"+mypuppy.color);
      System.out.println("the age is:"+ mypuppy.age);   }    }
```

Accessing methods in a class

- Once you have created a method in a class to represent behavior you can easily access it in an object.
- There is need to ensure that you use the right datatype when creating the method.
- Void datatype means that nothing is returned thus the method is empty. However if you use any other datatype it must have a return value.
- To access the method you just create an object of a class then call the method using the following syntax.
  - Object.methodName();
  - See example below

Example 3: Accessing Methods

```
public class Dog {
String color;
int age;
  String bark(){
  return("Woo! Woo!");
  }
  public Dog(){
  }
    public static void main(String[] args) {
     Dog Bosco= new Dog(      );
     Bosco.color="Brown";
     Bosco.age=5;
     System.out.println("Bosco barks:"+Bosco.bark());
   } }
```

Deciphering the program

- The program creates a class called Dog, it defines two states namely name and age.
- It also has one method known as bark() which returns (“Woo!Woo!”)
- An object of class Dog is created whose name is Bosco. Then Bosco implements the states as shown in blue color.
- Bosco also implements the methods as shown in red color

The computer programmer is a creator of universes for which he alone is responsible. Universes of virtually unlimited complexity can be created in the form of computer programs.

(Joseph Weizenbaum)