Java for Beginners

Java Basic Syntax

- Object - Objects have states and behaviors. Example: A dog has states-color, name, breed as well as behaviors -wagging, barking, eating.
  - An object is an instance of a class.
- Class - A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.
- 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.
- Instance Variables - Each object has its unique set of instance variables.
  - An object's state is created by the values assigned to these instance variables.

First Java Program

```
public class MyFirstJavaProgram{

/* This is my first java program.
    * This will print 'Hello World' as the output
    */

  public static void main(String[]args){
System.out.println("Hello World");// prints Hello World
}
}
```

Basic Java Syntax:

- Java Programs:
  - Case Sensitivity - Java is case sensitive, which means that 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.
      - 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()

Basic Java Syntax:

- Java Programs:
  - 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[]) - Java program processing starts from the main() method, which is a mandatory part of every Java program.

Java Identifiers:

- Identifiers are as follows:
  - 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.
  - A keyword cannot be used as an identifier.
  - Most importantly identifiers are case sensitive.
  - Examples of legal identifiers: age, $salary, _value, __1_value
  - Examples of illegal identifiers: 123abc, -salary

Java Variables:

- A variable is a place where the program stores data temporarily.
  - As the name implies the value stored in such a location can be changed while a program is executing (compare with constants).

- Class exercise:
  - Write a program that uses two variables, var1 and var2. var1 is assigned a value directly while var2 is filled up with the result of dividing var1 by 2, i.e. var2 = var1/2. The words int refer to a particular data type, i.e. integer (whole numbers).

Java Constants:

- A variable is a place where the program stores data temporarily.
  - As the name implies the value stored in such a location can be changed while a program is executing (compare with constants).

- Class exercise:
  - Write a program that uses two variables, var1 and var2. var1 is assigned a value directly while var2 is filled up with the result of dividing var1 by 2, i.e. var2 = var1/2. The words int refer to a particular data type, i.e. integer (whole numbers).

Java Variables:

- Local variables:
  - Local variables are declared in methods, constructors, or blocks.
  - Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
  - Access modifiers cannot be used for local variables.
  - Local variables are visible only within the declared method, constructor or block.
  - There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

Java Variables:

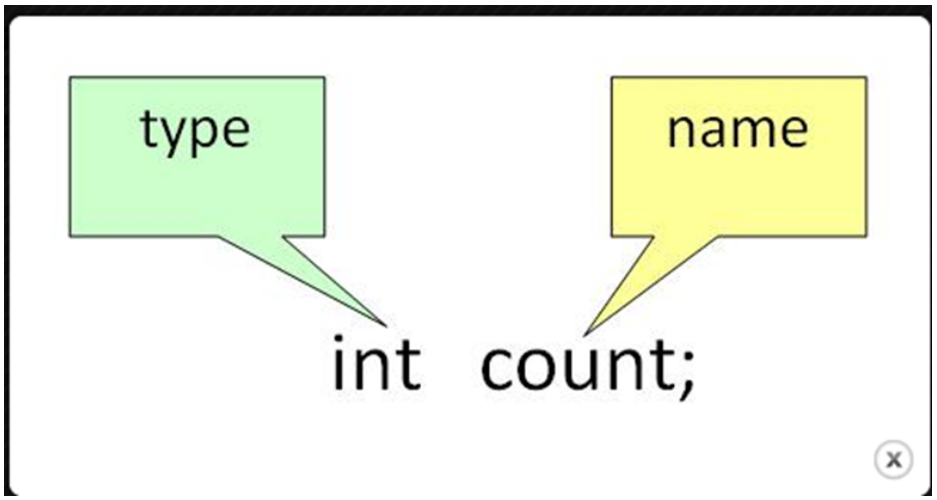
- Instance variables:
  - Instance variables are declared in a class, but outside a method, constructor or any block.
  - Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
  - Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.

Java Variables:

- Instance variables:
  - Instance variables can be declared in class level before or after use.
  - Access modifiers can be given for instance variables.
  - The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level).
  - Instance variables can be accessed directly by calling the variable name inside the class.

Java Variables and Data Types:

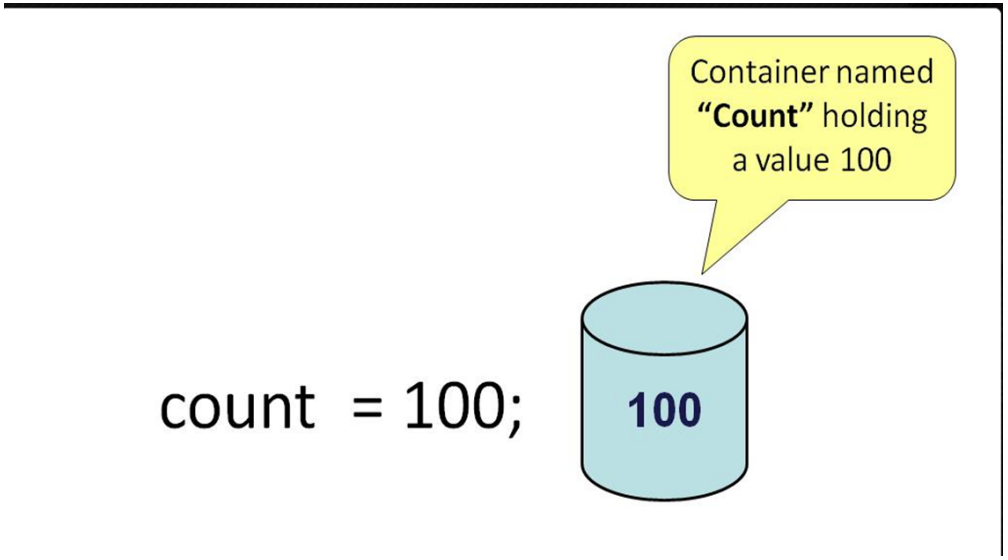
- Variable Declaration:
  - To declare a variable , you must specify the data type & give the variable a unique name.



Java Variables and Data Types:

- Variable Initialization:
  - To initialize a variable you must assign it a valid value.

  - Example :
    - int a=2,
    - b=4,
    - c=6;
    - float pi=3.14f;
    - double do=20.22d;
    - char a='v';



Java Variables and Data Types:

- What is a Type
  - A type is what a variable represents or holds
    - Example: the number 3 in Java is an int (integer)

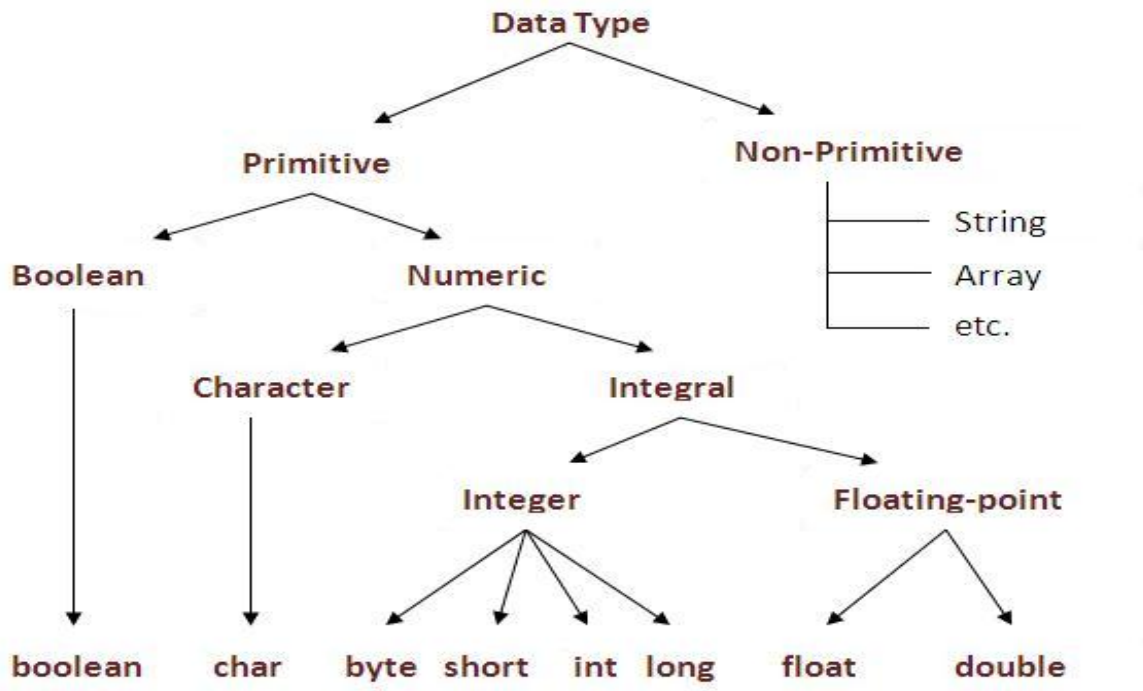
      The word “Hello” in Java is a string
    - Java is strongly typed
      - Types must be declared
      - Types cannot change after they are declared

Data Types:

Data Type	Description
int	Integer – 32bit ranging from -2,147,483,648 to 2,147,483,648
byte	8-bit integer ranging from -128 to 127
short	16-bit integer ranging from -32,768 to 32,768
long	64-bit integer from -9,223,372,036,854,775,808 to -9,223,372,036,854,775,808
float	Single-precision floating point, 32-bit
double	Double-precision floating point, 64-bit
char	Character , 16-bit unsigned ranging from 0 to 65,536 (Unicode)
boolean	Can be true or false only

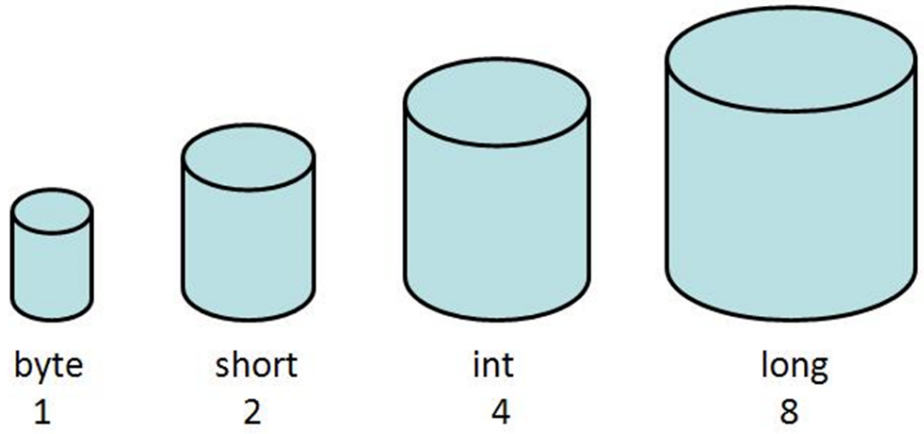
Primitive and Non-Primitive Data Types:

- There are eight primitive data types supported by Java.
  - Primitive data types are predefined by the language and named by a keyword.



Java Variables and Data Types:

- What is a Variable?
  - A variable can be thought of as a container which holds value for you during the life of your program.
  - Every variable is assigned a data type which designates the type and quantity of a value it can hold.
- Data types in java
  - The different Data Type are ---
    - Integer java data types
      - byte (1 byte)
      - short (2 bytes)
      - int (4 bytes)
      - long (8 bytes)



Data types practice:

```
public class datatypes {

        public static void main(String[] args) {
         int i = 4523;   //Can store 32 bit integer values only.
         long l = 652345;  //Can store 64 bit integer values only.
         double d1 = 56.2354;         //Can store 64 bit decimal values.
         double d2 = 12456;  //We can use it for integer values too.
         char c = 'd';   //Can store single character only.
         boolean t = true;  //Can store only boolean values like true or false.

         System.out.println("Integer Var Is --> "+i);
         System.out.println("Long Var Is --> "+l);
         System.out.println("double Var d1 Is --> "+d1);
         System.out.println("double Var d2 Is --> "+d2);
         System.out.println("char Var c Is --> "+c);
         System.out.println("boolean Var b Is --> "+t);
        }
}
```

Scope and Lifetime of Variables:

```
class ScopeExample {
    public static void main(String args[]) {
        int x; // known to all code within main
        x = 10;
        if(x == 10) { // start new scope
                int y  20; // known only to this block
          // x and y both known here.
          System.out.println("x and y: " + x + " " + y);
          x = y * 2;
        }
        // y = 100; // Error! y not known here
       // x is still known here.
       System.out.println("x is " + x);
    }
}
```

Scope and Lifetime of Variables:

- Variables exist in different contexts

Mathematical Operators:

Operator	Description	Example – given a is 15 and b is 6
+	Addition	a + b, would return 21
-	Subtraction	a - b, would return 9
*	Multiplication	a * b, would return 90
/	Division	a / b, would return 2
%	Modulus	a % b, would return 3 (the remainder)

Java Strings:

- The “special” type
- Treated like a primitive although it is an object
- Immutable
  - Changing a string results in a new string
- Java makes it easy to work with Strings

```
String salutation = “Hello World”;
String salutation = new String(“Hello World”);
```

Java Strings:

- Generally, a string is a sequence of characters.
- In java, a string is an object that represents a sequence of characters.
- String class is used to create a string object.
- There are two ways to create String object:
  - By string literal
    - Example: String salutation =“Hello java";
  - By new keyword
    - String salutation = new String(“Hello java");

Java Strings:

- Java String Example

```
public class StringExample{
    public static void main(String args[]){
        String s1="java";//creating string by java string literal

        char ch[]={'s','t','r','i','n','g','s'};
        String s2=new String(ch);//converting char array to string

        String s3=new String("example");//creating java string by new keyword

        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
    }
}
```

Java Strings:

- Java String class methods

1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	static String format(String format, Object... args)	returns formatted string
4	static String format(Locale l, String format, Object... args)	returns formatted string with given locale
5	String substring(int beginIndex)	returns substring for given begin index
6	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index
7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value
8	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string
9	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	returns a joined string

Java Strings:

- Java String class methods

19	int indexOf(int ch)	returns specified char value index
20	int indexOf(int ch, int fromIndex)	returns specified char value index starting with given index
21	int indexOf(String substring)	returns specified substring index
22	int indexOf(String substring, int fromIndex)	returns specified substring index starting with given index
23	String toLowerCase()	returns string in lowercase.
24	String toLowerCase(Locale l)	returns string in lowercase using specified locale.
25	String toUpperCase()	returns string in uppercase.
26	String toUpperCase(Locale l)	returns string in uppercase using specified locale.

Java Strings:

- String "CompareTo" Method
  - I want to check if the String that was generated by some method is equal to something that I want to verify with? How do I compare two Strings?
    - Use the method “compareTo” and specify the String that you would like to compare.
    - Use “compareToIgnoreCase” in case you don’t want the result to be case sensitive.
      - The result will have the value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument

Java Strings:

- String "CompareTo" Method

```
public class Sample_String{
    public static void main(String[] args){
        //Compare to a String
        String str_Sample = "RockStar";
        System.out.println("Compare To 'ROCKSTAR':
        "str_Sample.compareTo("rockstar"));
        //Compare to - Ignore case
        System.out.println("Compare To 'ROCKSTAR' - Case Ignored: " +
        str_Sample.compareToIgnoreCase("ROCKSTAR"));
    }
}
```

Java Strings:

- String "CompareTo" Method

```
public class Sample_String{
    public static void main(String[] args){
        //Compare to a String
        String str_Sample = "RockStar";
        System.out.println("Compare To 'ROCKSTAR':
        "str_Sample.compareTo("rockstar"));
        //Compare to - Ignore case
        System.out.println("Compare To 'ROCKSTAR' - Case Ignored: " +
        str_Sample.compareToIgnoreCase("ROCKSTAR"));
    }
}
```

Java Strings:

- String "Contain" Method
  - I partially know what the string should have contained, how do I confirm if the String contains a sequence of characters I specify?
  - Use the method “contains” and specify the characters you need to check.
  - Returns true if and only if this string contains the specified sequence of char values.

```
public class Sample_String{
    public static void main(String[] args){
        //Check if String contains a sequence
        String str_Sample = "RockStar";
        System.out.println("Contains sequence 'tar': " +
        str_Sample.contains("tar"));
    }
}
```

Java Strings:

- String "endsWith" Method
  - How do I confirm if a String ends with a particular suffix?
    - Again you answered it. Use the “endsWith” method and specify the suffix in the arguments.
  - Returns true if the character sequence represented by the argument is a suffix of the character sequence represented by this object.

```
public class Sample_String{
    public static void main(String[] args){
    //Check if ends with a particular sequence String str_Sample =
    "RockStar"; System.out.println("EndsWith character 'r': " +
    str_Sample.endsWith("r"));
    }
}
```

Java Strings:

- String "replaceAll" & "replaceFirst" Method
  - I want to modify my String at several places and replace several parts of the String?
  - Java String Replace, replaceAll and replaceFirst methods. You can specify the part of the String you want to replace and the replacement String in the arguments.

```
public class Sample_String{
    public static void main(String[] args){
        //Replace Rock with the word Duke String str_Sample = "RockStar";
        System.out.println("Replace 'Rock' with 'Duke': " +
        str_Sample.replace("Rock", "Duke"));
    }
}
```

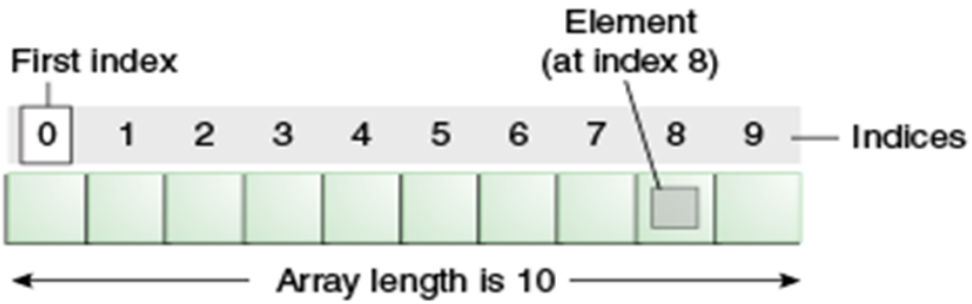
Java Strings:

- String Java "tolowercase" & Java "touppercase“
  - I want my entire String to be shown in lower case or Upper case?
  - Just use the “toLowercase()” or “ToUpperCase()” methods against the Strings that need to be converted.

```
public class Sample_String{
    public static void main(String[] args){
        //Convert to LowerCase String str_Sample = "RockStar";
        System.out.println("Convert to LowerCase: " + str_Sample.toLowerCase());
        //Convert to UpperCase
        System.out.println("Convert to UpperCase: " + str_Sample.toUpperCase());
    }
}
```

Java Arrays:

- An array is a collection of similar type of elements that have contiguous memory location
- Java array is an object the contains elements of similar data type.
- It is a data structure where we store similar elements.
- We can store only fixed set of elements in a java array.
- Array in java is index based, first element of the array is stored at 0 index.



Java Arrays:

- Advantage of Java Array
  - Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.
  - Random access: We can get any data located at any index position.

- . Disadvantage of Java Array
  - Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java
- Declaring arrays in java
  - dataType[] arr; (or)
  - dataType []arr; (or)
  - dataType arr[];

Java Arrays:

- Declaring Array Variables:
  - To use an array in a program:
    - You must declare a variable to reference the array, and
    - you must specify the type of array the variable can reference
    - Below is the syntax for declaring an array variable:

```
dataType[] arrayRefVar; // preferred way
or
dataType arrayRefVar[]; // works but not preferred way
```

Java Arrays:

- Creating Arrays:
  - You can create an array by using the new operator with the following syntax:

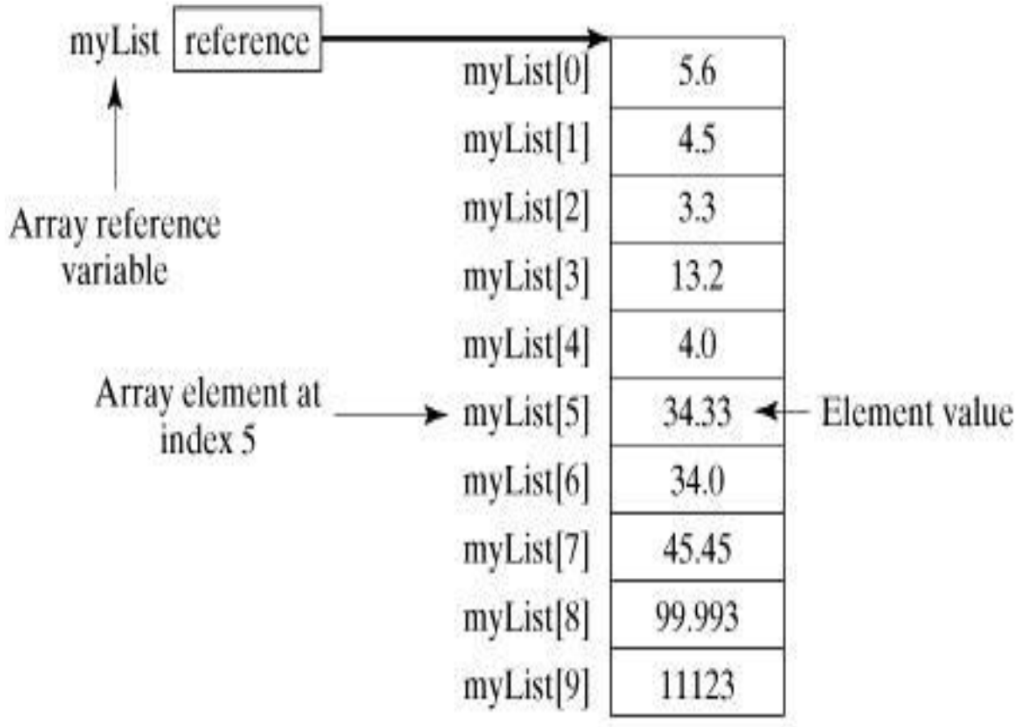
    arrayRefVar =new dataType[arraySize];

  - This statement does two things:
    1. It creates an array using new dataType[arraySize];
    2. It assigns the reference of the newly created array to the variable arrayRefVar.
  - you can also create arrays as follows:

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

Java Arrays: Exercise

- Create an array in a new project called ArrayExample
  - Array: double[] myList =new double[10];
  - Populate the myList array based on the picture below
  - Print the array element in index 5



Java Enums:

- Enums restrict a variable to have one of only a few predefined values.
  - The values in this enumerated list are called enums.
  - With the use of enums, it is possible to reduce the number of bugs in your code.
    - For example, if we consider an application for a fresh juice shop, it would be possible to restrict the glass size to small, medium and large.
      - This would make sure that it would not allow anyone to order any size other than the small, medium or large.
- Enums can be declared as their own or inside a class.
- Methods, variables, constructors can be defined inside enums as well.

Java Enums:

- Enum example:

```
Class FreshJuice{

enum FreshJuiceSize{ SMALL, MEDUIM, LARGE }
FreshJuiceSize size;
}

public class FreshJuiceTest{

public static void main(String args[]){
FreshJuice juice =new FreshJuice();
 juice.size =FreshJuice.FreshJuiceSize.MEDUIM ;
}
}
```

Java Keywords:

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

Comments in Java:

- There are 3 types of comments:
  - comments that run to the end of the line //
  - comments that mark out blocks starting /* and ending */
  - Java Doc comments starting with /** and ending */

Comments in Java:

```
public class MyFirstJavaProgram{

/* This is my first java program.
     * This will print 'Hello World' as the output
     * This is an example of multi-line comments.
     */

   public static void main(String[]args){
// This is an example of single line comment
/* This is also an example of single line comment. */
System.out.println("Hello World");
}
}
```

Statements in Java:

- A Java statement is the smallest chunk of executable Java code. We end a Java statement with ; e.g.
- assertEquals(4, 2+2);
- Java statements can span lines. This is useful to make your code more readable and line up arguments on method calls. e.g.

Control Statements :

- A group of statements executed in order is written
  - **`{ stmt1; stmt2; ...; stmtN; }`**
- The statements execute in the order 1, 2, ..., N
- Control statements alter this sequential flow of execution

Control Statements:

- Blocks
  - The block is the simplest type of structured statement
  - Its purpose is simply to group a sequence of statements into a single statement
  - The format of a block is:

```
{
    (statements )
}
```

  - An empty block consists of nothing but an empty pair of braces
  - Block statements usually occur inside other statements, where their purpose is to group together several statements into a unit

Control Statements :

TABLE A.4
Java Control Statements

Control Structure	Purpose	Syntax

`if ... else`	Used to write a decision with *conditions* that select the alternative to be executed. Executes the first (second) alternative if the *condition* is true (false).	`if (condition) {`
		`  ...`
		`} else {`
		`  ...`
		`}`

`switch`	Used to write a decision with scalar values (integers, characters) that select the alternative to be executed. Executes the *statements* following the *label* that is the *selector* value. Execution falls through to the next `case` if there is no `return` or `break`. Executes the statements following `default` if the *selector* value does not match any *label*.	`switch (selector) {`
		`  case label : statements; break;`
		`  case label : statements; break;`
		`  ...`
		`  default : statements;`
		`}`

`while`	Used to write a loop that specifies the repetition *condition* in the loop header. The *condition* is tested before each iteration of the loop and, if it is true, the loop body executes; otherwise, the loop is exited.	`while (condition) {`
		`  ...`
		`}`

`for`	Used to write a loop that specifies the *initialization*, repetition *condition*, and *update* steps in the loop header. The *initialization* statements execute before loop repetition begins, the *condition* is tested before each iteration of the loop and, if it is true, the loop body executes; otherwise, the loop is exited. The *update* statements execute after each iteration.	`for (initialization; condition; update) {`
		`  ...`
		`}`

Control Statements :

TABLE A.4 (continued)

Control Structure	Purpose	Syntax

`do ... while`	Used to write a loop that specifies the repetition *condition* after the loop body. The *condition* is tested after each iteration of the loop and, if it is true, the loop body is repeated; otherwise, the loop is exited. The loop body always executes at least one time.	`do {`
		`  ...`
		`while (`*condition*`) ;`

Java Controls:

- The Basic While Loop
  - A while loop is used to repeat a given statement over and over
    - but only so long as a specified condition remains true
  - A while loop has the form:

```
while (boolean-expression)
    (statement )
```

  - Here is an example of a while loop that simply prints out the numbers 1, 2, 3, 4, 5:

```
int number; // The number to be printed.
number = 1; // Start with 1.
while ( number < 6 ) { // Keep going as long as number is < 6.
    System.out.println(number);
    number = number + 1; // Go on to the next number.
}
System.out.println("Done!");
```

Java Controls:

- The do..while Statement
  - Sometimes it is more convenient to test the continuation condition at the end of a loop, instead
  - The do..while statement is very similar to the while statement, except that the word “while,” along with the condition that it tests, has been moved to the end.
  - The word “do” is added to mark the beginning of the loop
  - A do..while statement has the form

```
Do {
    (statement )
    ...
} while ( boolean-expression );
```

- Here is an example of a do.. while loop

```
boolean wantsToContinue; // True if user wants to play again.
do {
    Checkers.playGame();
    TextIO.put("Do you want to play again? ");
    wantsToContinue = TextIO.getlnBoolean();
} while (wantsToContinue == true);
```

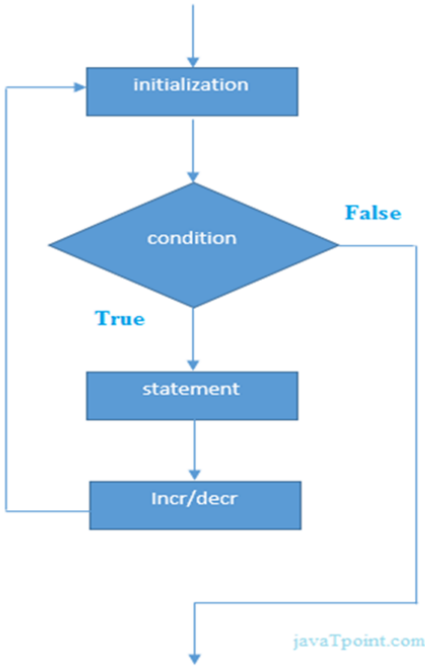
Java Controls:

- Java For Loop
  - The Java for loop is used to iterate a part of the program several times
  - If the number of iteration is fixed, it is recommended to use for loop.
  - There are three types of for loop in java.
    - Simple For Loop
    - For-each or Enhanced For Loop
    - Labeled For Loop

Java Controls:

- Java For Loop
  - Simple For Loop
    - In a simple for loop we can initialize variable, check condition and increment/decrement value.
    - Syntax:

```
For (initialization;condition;incr/decr) {
        //code to be executed
}
```



Java Controls:

- Java For Each Loop
  - For-each or Enhanced For Loop
    - The for-each loop is used to traverse array or collection in java
    - It is easier to use than the simple for loop because we don't need to increment value and use subscript notation
    - It works on elements basis not index
    - It returns element one by one in the defined variable.
    - Syntax:

```
for(Type var:array){
    //code to be executed
}
```

Example:

```
public class ForEachExample {
    public static void main(String[] args) {
        int arr[]={12,23,44,56,78};
        for(int i:arr){
            System.out.println(i);
        }
    }
}
```

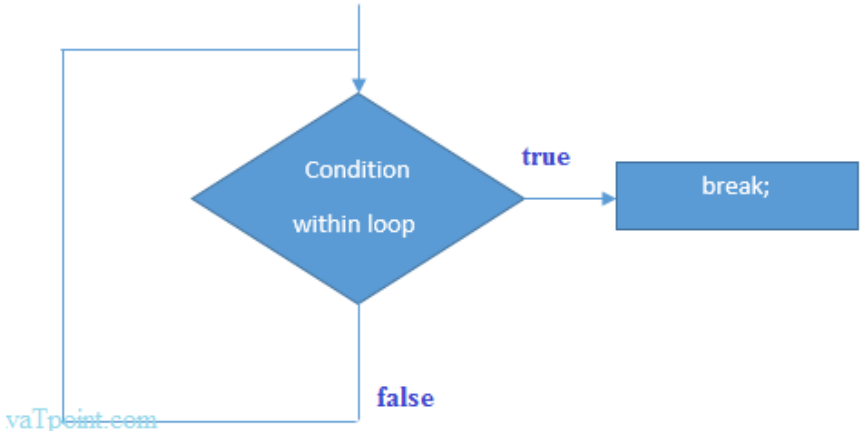
Java Controls:

- Java Break Statement
  - The Java break is used to break loop or switch statement
  - It breaks the current flow of the program at specified condition.
  - In case of inner loop, it breaks only inner loop
  - Syntax:

```
jump-statement;
break;
```

  - Example:

```
public class BreakExample {
        public static void main(String[] args) {
           for(int i=1;i<=10;i++){
              if(i==5){
                 break;
              }
              System.out.println(i);
           }
        }
}
```



Java Controls:

- Java Continue Statement
  - The Java continue statement is used to continue loop
  - It continues the current flow of the program and skips the remaining code at specified condition.
  - In case of inner loop, it continues only inner loop
  - Syntax:

```
jump-statement;
continue;
```

  - Example:

```
public class ContinueExample {
        public static void main(String[] args) {
            for(int i=1;i<=10;i++){
                if(i==5){
                    continue;
                }
                System.out.println(i);
            }
        }
}
```

Java Controls:

- Java If-else Statement
  - The Java if statement is used to test the condition.
  - It returns true or false.
  - There are various types of if statement in java.
    - if statement
    - if-else statement

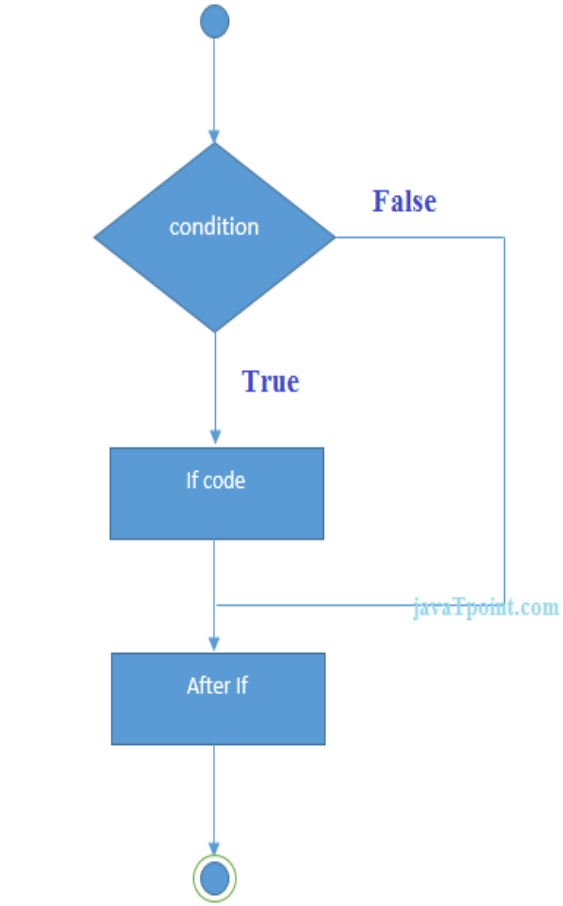
Java Controls:

- Java If-else Statement
  - if statement
    - The if statement tests the condition.
    - It executes the if statement if condition is tr
  - Syntax:

```
if(condition){
   //code to be executed
}
```

  - Example:

```
public class IfExample {
      public static void main(String[] args) {
         int age=20;
         if(age>18){
            System.out.print("Age is greater than 18");
         }
      }
}
```

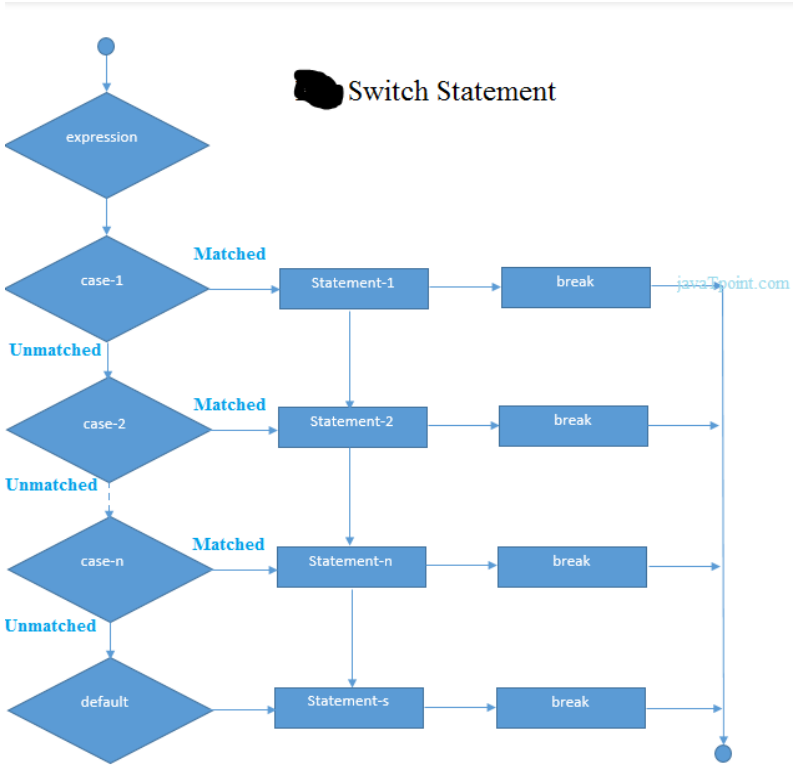


Java Controls:

- Java Switch Statement
  - if statement
  - The Java switch statement is executes one statement from multiple conditions.
  - It is like if-else-if ladder statement
  - Syntax:

```
switch(expression){
        case value1:
             //code to be executed;
             break;  //optional
        case value2:
              //code to be executed;
              break;  //optional
              ......

             default:
             code to be executed if all cases are not matched;
}
```



Methods:

- A Java method defines a group of statements as performing a particular operation
- **`static`** indicates a static or class method
- A method that is not **`static`** is an instance method
- All method arguments are call-by-value
  - Primitive type: value is passed to the method
  - Method may modify local copy but will not affect caller's value
  - Object reference: address of object is passed
  - Change to reference variable does not affect caller
  - But operations can affect the object, visible to caller

Packages in Java:

- Java allows us to group our Classes into packages. Each class has to be uniquely named
- within a package. We can have multiple classes with the same name, provided they are all in different packages.

```
package
com.javafortesters.chap007basicsofjavarevisited.examples;
```

- To add a class to a package you write a package declaration statement like the above, very
- often the first line in the class, and certainly before the code that declares the class.

Java Classes in Java:

- Java allows us to group our Classes into packages. Each class has to be uniquely named
- within a package. We can have multiple classes with the same name, provided they are all in different packages.

```
package
com.javafortesters.chap007basicsofjavarevisited.examples;
```

- To add a class to a package you write a package declaration statement like the above, very
- often the first line in the class, and certainly before the code that declares the class.

Java Classes in Java:

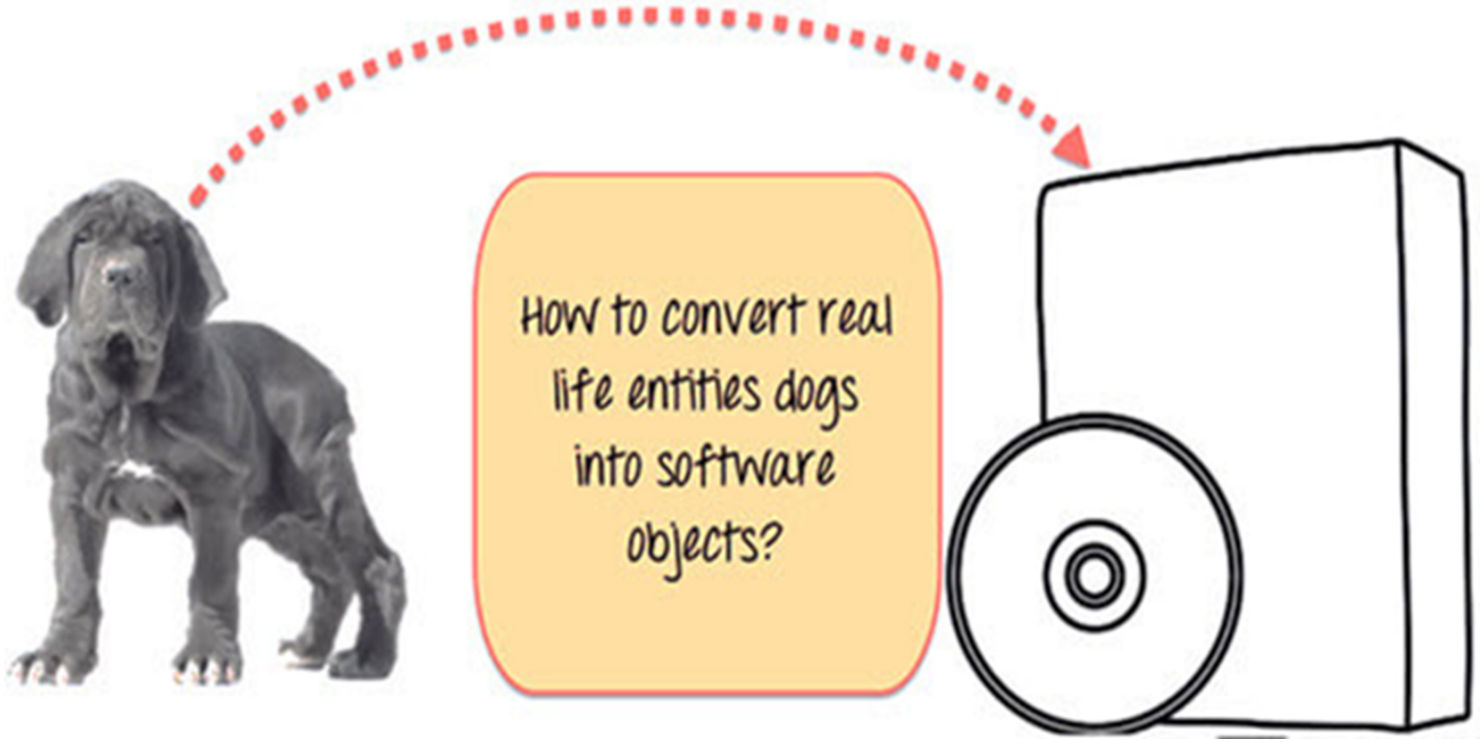
- A class is a blueprint for constructing objects.
- A class can contain any of the following variable types
  - Local variables: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
  - Instance variables: Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
  - Class variables: Class variables are variables declared within a class, outside any method, with the static keyword.
- A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Objects and Classes:

- Object-oriented programming is modeled on the observation that in the physical world, objects are made up of many kinds of smaller objects.
- Another important feature of Object-Oriented programming is the use of classes.
  - A class is a template used to create an object. Every object created from the same class has similar features
  - When you write a program in an object-oriented language, you don’t define individual objects; instead, you define classes used to create those objects.

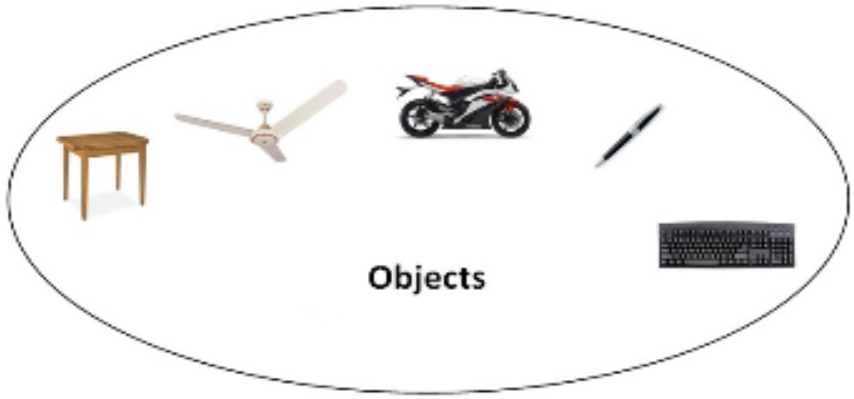
Object-Oriented Programming:

- Java is n Object-Oriented programming language



Object-Oriented Programming:

- Object means a real word entity such as pen, chair, table etc.
- Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects
- It simplifies the software development and maintenance by providing some concepts:
  - Object
  - Class
  - Inheritance
  - Polymorphism
  - Abstraction
  - Encapsulation



- Any entity that has state and behavior is known as an object

Attributes and Behavior:

- A Java class consists of two distinct types of information: attributes and behavior
- Attributes of a Class of Objects
  - Attributes are the data that differentiate one object from another. They can be used to determine the appearance, state, and other qualities of objects that belong to that class
- Behavior of a Class of Objects
  - Behavior refers to the things that a class of objects can do—both to themselves and to other objects.
  - Behavior can be used to change an object's attributes, receive information from other objects, and send messages to other objects, asking them to perform tasks.

Attributes and Behavior:

- Attributes of a Class of Objects
  - Below is the picture of three different breeds of dogs

  - List the differences between these dogs on a piece of paper.

Attributes and Behavior:

- Attributes of a Class of Objects
  - Below is the picture of three different breeds of dogsSome of the differences you might have listed out may be
    - breed,
    - age,
    - size,
    - color, etc.
    - If you think for a minute, these differences are also some common characteristics shared by these dogs.
    - These characteristics (breed, age, size, color) can form the data members for your object.
  - In a class, attributes are defined by variables—places to store information in a computer program

Attributes and Behavior:

- Behavior of a Class of Objects
  - Behavior refers to the things that a class of objects can do—both to themselves and to other objects.
  - Behavior can be used to change an object's attributes, receive information from other objects, and send messages to other objects, asking them to perform tasks.

- In your piece of paper
  - List the common behaviors of these dogs

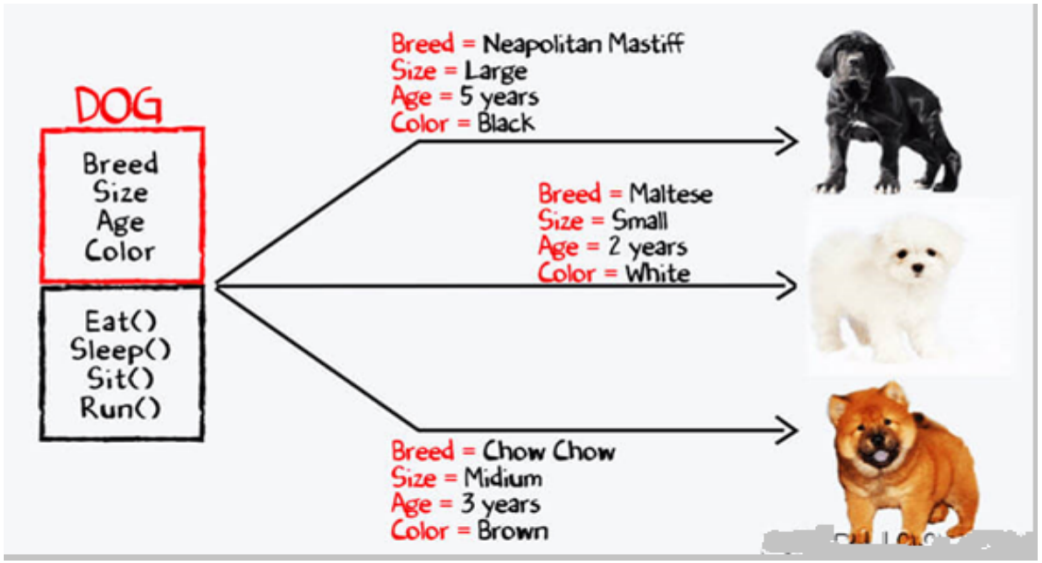
Attributes and Behavior:

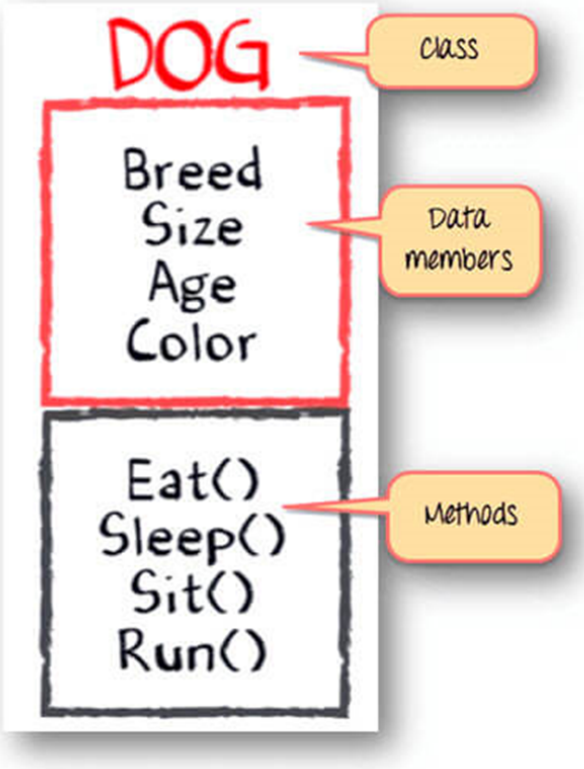
- Behavior of a Class of Objects

- Hopefully you have listed common behaviors as:
  - eat, sleep, sit, run, poop, bark, etc.

Attributes and Behavior:

- Behavior of a Class of Objects
  - So we have this:
    - Class - Dogs
    - Data members or objects- size, age, color, breed, etc.
    - Methods- eat, sleep, sit, run





Attributes and Behavior:

- Behavior of a Class of Objects
  - Behavior for a class of objects is implemented using methods.
  - Methods are groups of related statements in a class that perform a specific task.
    - They are used to accomplish specific tasks on their own objects and on other objects and are comparable to functions and subroutines in other programming languages.
    - A well designed method performs only one task and one task alone.
  - Objects communicate with each other using methods.
  - A class or object can call methods in another class or object for many reasons, including the following:
    - To report a change to another object
    - To tell the other object to change something about itself
    - To ask another object to do something

Attributes and Behavior:

- Behavior of a Class of Objects
  - Behavior for a class of objects is implemented using methods.
  - Methods are groups of related statements in a class that perform a specific task.
    - They are used to accomplish specific tasks on their own objects and on other objects and are comparable to functions and subroutines in other programming languages.
    - A well designed method performs only one task and one task alone.
  - Objects communicate with each other using methods.
  - A class or object can call methods in another class or object for many reasons, including the following:
    - To report a change to another object
    - To tell the other object to change something about itself
    - To ask another object to do something

Handling exceptions using try-catch block:

- Exceptions can arise due to many reasons:
  - Network connection issues
  - Hardware failure
  - Software failure
  - Invalid data entered by user,
  - DB server issues, etc.
- There are two types of exceptions in java:
  - Checked Exception: These exceptions are checked during compile time and needs the try-catch block so they can be caught. If the compiler doesn’t find a try-catch block an error will be thrown at compile time.
  - Unchecked Exceptions: These exceptions are not checked during compile time. Generally, unchecked exceptions occur due to error in code during run time. Example of unchecked exception Is int i = 4/0;.

public

Handling exceptions using try-catch block:

```
public class Handle_exce {

 public static void main(String[] args) {
  try{
      int i=5/0; //Exception will be thrown.
      System.out.println("Value Of i Is "+i);//This statement will be not
executed.
  }catch (Exception e)//Exception will be caught
 {
     System.out.println("Inside catch."+e);//print the exception.
  }
```