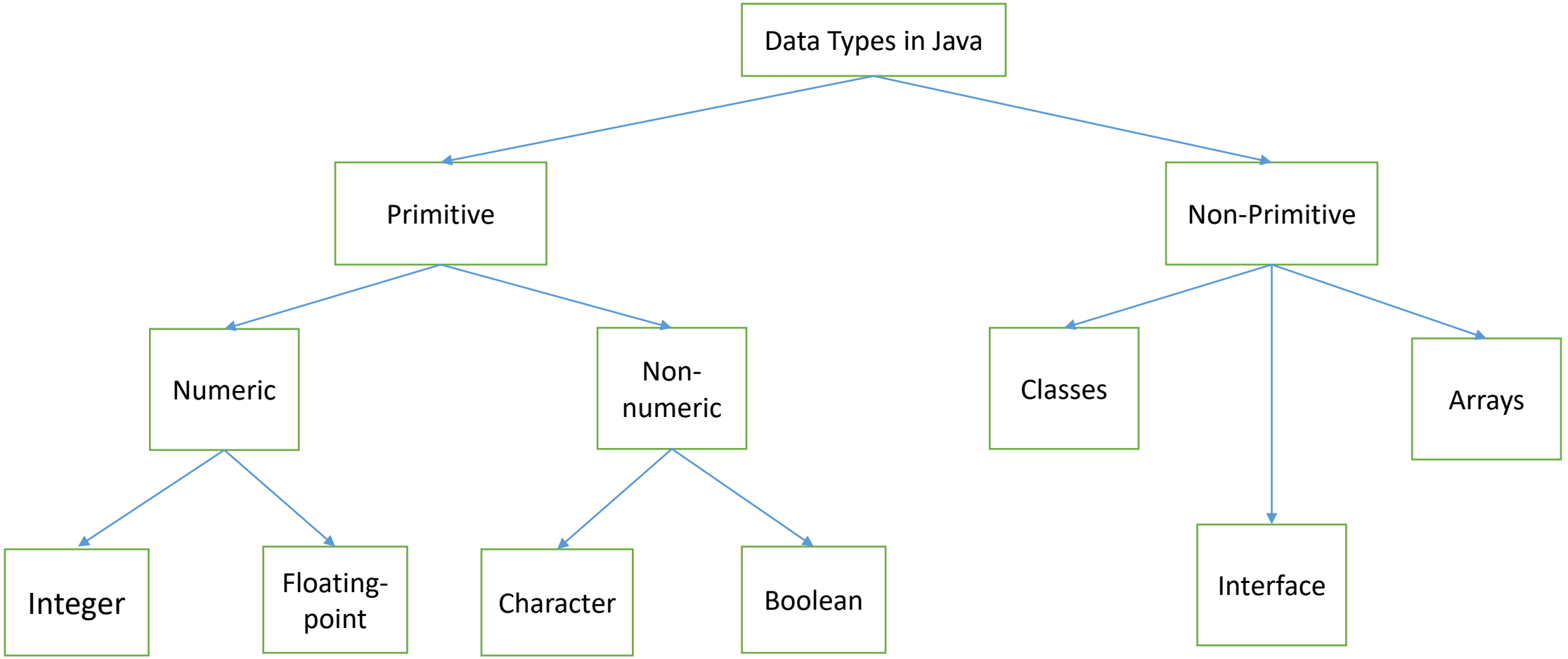
Data Types, Identifiers, variables, Java Access Modifiers, Operators

Data Types

- **Every variable in Java has a data type**
- **It specifies the size and type of values that can be stored**



Primitive Data Types

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Reference Data Types

- Reference data types are used to store references or memory addresses that point to objects stored in memory.
- These data types do not actually store the data itself
- Class objects and various types of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to any object of the declared type or any compatible type.
  - Example: Animal animal = new Animal("giraffe");

Identifier

- In programming languages, identifiers are used for identification purposes. In Java, an identifier can be a class name, method name, variable name, or label.
- **Rules for defining Java Identifiers:**
  - The only allowed characters for identifiers are all alphanumeric characters([A-Z],[a-z],[0-9]), and underscore(_) or a dollar sign ($).
  - Identifiers should not start with digits([0-9]). For example “123geeks” is a not a valid java identifier.
  - Java identifiers are case-sensitive.
  - There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.
  - Reserved Words can’t be used as an identifier. For example “int while = 20;” is an invalid

Java Modifiers:

There are two categories of modifiers:

- **Access Modifiers**: default, public, protected, private
- **Non-access Modifiers**: final, abstract

Java Variables:

We would see following type of variables in Java:

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non-static variables)

Local variable

- A variable declared inside the body of the method, constructors, or blocks is called local variable.
- You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- A local variable cannot be defined with "static" keyword.

Instance variables:

- Instance variables are declared in a class, but outside a method, constructor or any block.
- It is not declared as static.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- It is called instance variable because its value is instance specific and is not shared among instances.
- Access modifiers can be given for instance variables.

Instance variables Example:

```
public class Employee {
// this instance variable is visible for any child class.
protected String name;

// salary variable is visible in Employee class only.
private double salary;

// The name variable is assigned in the constructor.
public Employee(String empName) {
name = empName;
}
// The salary variable is assigned a value.
public void setSalary(double empSal) {
salary = empSal;
}
```

```
// This method prints the employee details.
public void printEmp() {
System.out.println("name : " + name);
System.out.println("salary :" + salary);
}

public static void main(String args[]) {
Employee empOne = new Employee(“Jabir");
empOne.setSalary(1000);
empOne.printEmp();
}
}
```

Output:

```
name : jabir
salary :1000.0
```

Class/static variables:

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final and static. Constant variables never change from their initial value.
- Static variables are created when the program starts and destroyed when the program stops.
- No need to create instance for accessing static variable.

Class/static variables Example:

```
public class Employee{

// salary  variable is a private static variable
private static double salary;

// DEPARTMENT is a constant
public static final String DEPARTMENT = "Development ";

public static void main(String args[]){
            salary = 1000;
            System.out.println(DEPARTMENT+"average salary:"+salary);
    }
}
```

Output: Development average salary:1000

Note: If the variables are access from an outside class the constant should be accessed as Employee.DEPARTMENT

Java Modifiers

Access Control Modifiers:

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors.

- The four access levels are:
- Visible to the package. the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

Non Access Modifiers:

Java provides a number of non-access modifiers to achieve many other functionality.

- The static modifier for creating class methods and variables
- The final modifier for finalizing the implementations of classes, methods, and variables.
- The abstract modifier for creating abstract classes and methods.
- The synchronized and volatile modifiers, which are used for threads.

Java Access Modifiers(cont..)

Default Access Modifier - No keyword:

- Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
- A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public

Public Access Modifier - public:

- A class, method, constructor, interface etc. declared public can be accessed from any other class.
- Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe

Java Access Modifiers(cont..)

Private Access Modifier - private:

- Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.
- Private access modifier is the most restrictive access level. Class and interfaces cannot be private.
- Variables that are declared private can be accessed outside the class if public getter methods are present in the class.
- Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world

Protected Access Modifier - protected:

- Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
- The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected

Example

```
/* Printing ages. */
public class MyFirstJavaProgram {
   public static void main (String args[]) {
         int myAge, myFriendAge;                          /* declare two integer variables */
         myAge = 20;
         myFriendAge = myAge + 1;                          //one year older
         System.out.println("Hello, I am " +myAge + "years old, and my friend is " +
                  myFriendAge + " years old");

         System.out.println("Goodbye");
   } // end of main
} // end of class
```

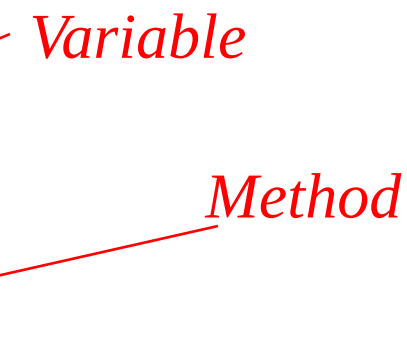
Reading from user (Input)

```
public class  MyFirstJavaProgram{
   public static void main (String args[]) {
   /* this line should appear once in your program;  basically it declares a variable in which
   knows how read input from the user */
            Scanner in=new Scanner(System.in);
   /* Then you can use num1=in.nextInt() to read an integer value from the user. */
    /* Then you can use num2=in.nextLine() to read an string value from the user. */
            System.out.print("Dear user, please enter an integer number:");
           int num1;
           String num2;
            num1=in.nextInt();
            num2=in.nextLine();
    } // end of main
} // end of class
```

An example of a class

```
class Person {
    String name;
    int age;

    void birthday ( ) {
        age++;
        System.out.println (name +
        ' is now ' + age);
    }
}
```



Java Basic Operators

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators

Arithmetic Operators

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

Arithmetic Operators Example:

```
public class Test{
public static void main(String args[]){
int a =10;
int b =20;
int c =25;
int d =25;
System.out.println("a + b = "+(a + b));
System.out.println("a - b = "+(a - b));
System.out.println("a * b = "+(a * b));
System.out.println("b / a = "+(b / a));
System.out.println("b % a = "+(b % a));
System.out.println("c % a = "+(c % a));
System.out.println("a++ = "+(a++));
System.out.println("b--  = "+(a--));
```

```
// Check the difference in d++ and ++d
System.out.println("d++ = "+(d++));
System.out.println("++d = "+(++d));
}
}
```

Output:

```
a + b =30
a - b =-10
a * b =200
b / a =2
b % a =0
c % a =5
a++=10
b--=11
d++=25
++d  =27
```

The Relational Operators:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.

The Relational Operators Examples:

```
public class Test{
public static void main(String args[]){
int a =10;
int b =20;
System.out.println("a == b = "+(a == b));
System.out.println("a != b = "+(a != b));
System.out.println("a > b = "+(a > b));
System.out.println("a < b = "+(a < b));
System.out.println("b >= a = "+(b >= a));
System.out.println("b <= a = "+(b <= a));
}
}
```

Output:

```
a == b    =false
a != b    =true
a > b     =false
a < b     =true
b >= a    =true
b <= a    =false
```

The Logical Operators:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

The Logical Operators Example:

```
public class Test{
public static void main(String args[]){
boolean a =true;
boolean b =false;
System.out.println("a && b = "+(a&&b));
System.out.println("a || b = "+(a||b));
System.out.println("!(a && b) = "+!(a && b));
}
}
```

Output:

```
a && b      =false
a || b      =true
!(a && b)   =true
```

The Assignment Operators:

=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

The Assignment Operators Example:

```
public class Test{
public static void main(String args[]){
int a =10;
int b =20;
int c =0;
c = a + b;
System.out.println("c = a + b = "+ c );
     c += a ;
System.out.println("c += a = "+ c );
    c -= a ;
System.out.println("c -= a = "+ c );
    c *= a ;
System.out.println("c *= a = "+ c );
   a =10;
   c =15;
   c /= a ;
System.out.println("c /= a = "+ c );
a =10;
c =15;
c %= a ;
```

```
System.out.println("c %= a = "+ c );
     c <<=2;
System.out.println("c <<= 2 = "+ c );
      c >>=2;
System.out.println("c >>= 2 = "+ c );
       c >>=2;
System.out.println("c >>= a = "+ c );
       c &= a ;
System.out.println("c &= 2 = "+ c );
       c ^= a ;
System.out.println("c ^= a = "+ c );
        c |= a ;
System.out.println("c |= a = "+ c );
}
}
```

```
Output:
c = a + b  =30
c += a     =40
c -= a     =30
c *= a     =300
c /= a     =1
c %= a     =5
c <<=2     =20
c >>=2     =5
c >>=2     =1
c &= a     =0
c ^= a     =10
c |= a     =10
```

Ternary Operators

Conditional Operator (?:)

- Conditional operator is also known as the ternary operator.
- The goal of the operator is to decide which value should be assigned to the variable.

- The operator is written as:

variable x =(expression)? value iftrue: value iffalse

```
public class Test{
public static void main(String args[]){
int a , b;
a =10;
b =(a ==1)?20:30;
System.out.println("Value of b is : "+  b );
b =(a ==10)?20:30;
System.out.println("Value of b is : "+ b );
}
}
```

Output:

Value of b is:30

Value of b is:20

Methods, arguments and return values

- Java methods are like C/C++ functions. General case:

  returnType methodName (*arg1*, *arg2*, … *argN*) {

  methodBody

  }

The return keyword exits a method optionally with a value

```
int storage(String s) {return s.length() * 2;}
boolean flag() { return true; }
float naturalLogBase() { return 2.718f; }
void nothing() { return; }
void nothing2() {}
```

Example

```
public class Circle {

  public static final double PI= 3.14159;                    // A class field
                                                             // A useful constant

  // A class method: just compute a value based on the arguments
  public static double radiansToDegrees(double rads) {
    return rads * 180 / PI;
  }

  // An instance field
  public double r;                                           // The radius of the circle

  // Two methods which operate on the instance fields of an object
  public double area() {                                     // Compute the area of the circle
    return PI * r * r;
  }
  public double circumference() {                            // Compute the circumference of the

    return 2 * PI * r;
  }
}
```