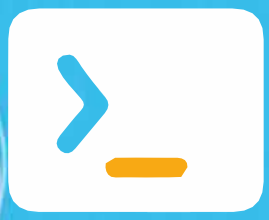


Argentina programa

Programación Orientada a Objetos con JAVA

Guía II

Fundamentos del Lenguaje

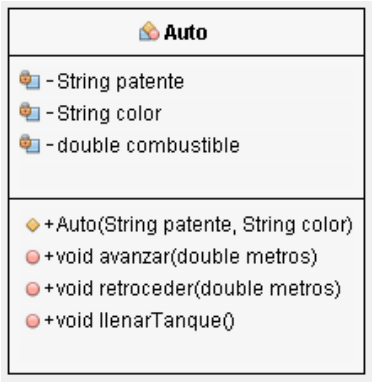
MANEJO DE LA MEMORIA RAM POR PARTE DE LA JVM

La máquina virtual Java, organiza a la memoria RAM en dos bloques, **memoria Stack y memoria Heap.**

La memoria **Stack** se usa para almacenar las variables locales (cuyo ámbito de acción está limitada solo al bloque donde se declaró, por ejemplo dentro de un método o como parámetro del mismo) y también las llamadas de métodos en Java. Las variables almacenadas en esta memoria por lo general tienen un periodo de vida corto, viven hasta que terminen la función o método en el que se están ejecutando.

Por otro lado, la memoria **Heap** es utilizada para almacenar los objetos (incluyendo sus atributos), los objetos almacenados en este espacio de memoria normalmente tienen un tiempo de duración más prolongado que los almacenados en **Stack**.

Siguiendo con el ejemplo de la guía 1, en donde vamos a suponer que tenemos en nuestro proyecto una clase de nombre Auto con la estructura que se muestra en la imagen.



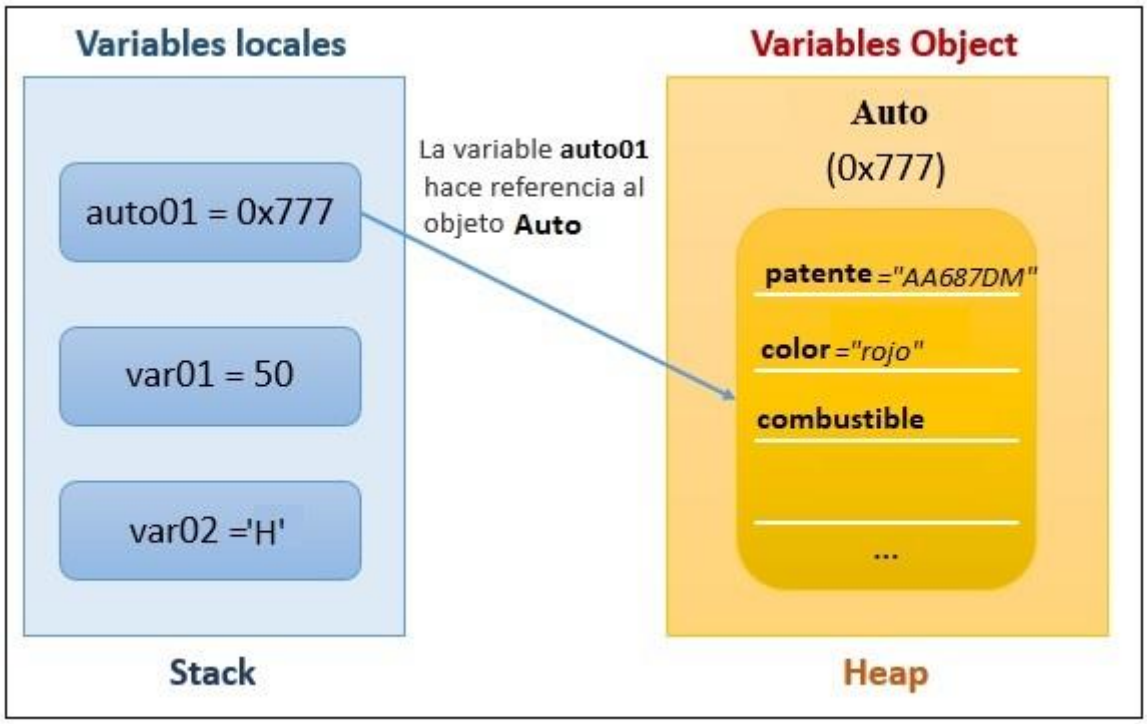
Y el siguiente fragmento de código:

```
Auto auto01 = new Auto(“AA687DM”,”rojo”);
int var01 = 50;
char var02 = ‘H’;
```

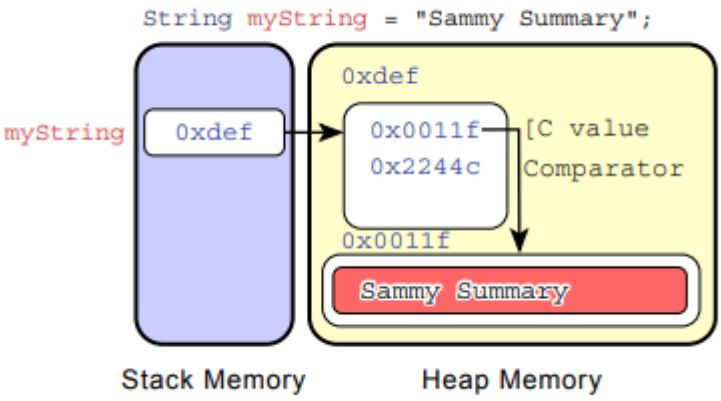
Podemos observar que la referencia del objeto que es representada por un hexadecimal por ejemplo el valor ‘0x777’, esta contiene la dirección de memoria donde está almacenado este objeto de tipo **Auto**, por lo tanto la variable local auto01 almacena únicamente esta referencia de memoria donde está almacenado, donde fue creado este objeto. Aprovechando esta misma imagen, podemos ver que variables de tipo primitivo como var01 y var02, almacenan directamente el valor donde está creada esta variable.

Si la variable local auto01 no tuviera referencia a memoria (por ejemplo igual a **null**) será eliminado por el **Garbage Collector** (recolector de basura), cuya función es eliminar los objetos que no estén siendo apuntados por ninguna variable, es decir que este objeto que se ha creado de tipo Auto como está siendo apuntado o referenciado por la variable auto01, no será eliminado. Pero si la variable auto01 se libera o es igualado a null (quitamos la referencia de memoria al objeto Auto), nuestra variable será candidata a ser borrada por el **Garbage Collector**. En este caso

la memoria **Heap** que es donde se almacenan los objetos en Java, de esta manera el **Garbage Collector** (recolector de basura) puede buscar aquellos objetos en la memoria Heap que ya no estén referenciados por ninguna otra variable y finalmente liberar el espacio en memoria que ocupaba dicho objeto.



Por lo tanto, si tenemos una variable de tipo **String**, dicho objeto se creará en la memoria **Heap.**



VALORES LITERALES PARA TODOS LOS TIPOS PRIMITIVOS:

Un literal primitivo es simplemente una representación de código fuente de los tipos de datos primitivos- en otras palabras, un entero, un número de punto flotante, un booleano, o un carácter que tú tipeas mientras escribes el código. Los siguientes son ejemplos de literales primitivos:

```
'b' // char literal
42 // int literal
false // boolean literal
2546789.343 // double literal
```

Literales enteros:

Hay tres formas de representar un número entero en Java:

Decimal (base 10), octal (base 8), y hexadecimal (base 16).

Un literal decimal, En el lenguaje Java, se representan tal cual, sin prefijo de ningún tipo, de la siguiente manera:

```
int edad = 34;
```

Los enteros octales usan solo los dígitos del 0 al 7. En Java, puedes representar un número entero en forma octal colocando un cero delante del número, de la siguiente manera:

```
class Octal {
 public static void main(String [] args) {
        int six = 06; // Equal to decimal 6
        int seven = 07; // Equal to decimal 7
        int eight = 010; // Equal to decimal 8
        int nine = 011; // Equal to decimal 9
        System.out.println("Octal 010 = " + eight);
 }
}
```

Los números literales hexadecimales (hex) numbers son construidos utilizando 16 símbolos diferentes:

0 1 2 3 4 5 6 7 8 9 a b c d e f

Java aceptará letras mayúsculas o minúsculas para los dígitos adicionales

Se le permite hasta 16 dígitos en un número hexadecimal, sin incluir el prefijo 0x o la extensión del sufijo opcional L, que será explicado luego.

Por ejemplo:

```
class HexTest {
 public static void main (String [] args) {
          int x = 0X0001;
          int y = 0x7fffffff;
          int z = 0xDeadCafe;
           System.out.println("x = " + x + " y = " + y + " z = " + z);
 }
}
```

Si ejecutamos **HexTest** tendremos la siguiente salida:

x = 1 y = 2147483647 z = -559035650

Los tres enteros literales (octal, decimal, y hexadecimal) son definidos como **int** por defecto, pero también pueden especificarse como **long** colocando un sufijo L o l después del número:

long jo = 110599L;

long so = 0xFFFFl; // Observe la 'l' minúscula.

Literales de punto flotante.

Los números de punto flotante se definen como un número, un símbolo decimal y más números que representan la fracción.

double d = 11301874.9881024;

En el ejemplo anterior, el número 11301874,9881024 es el valor literal. Los literales de punto flotante se definen como double (64 bits) de forma predeterminada, por lo que si desea asignar un literal de punto flotante a una variable de tipo float (32 bits), debe adjuntar el sufijo F o f al número. Si no lo hace, el compilador se informará de una posible pérdida de precisión, porque está tratando de encajar un número en un “envase” (potencialmente) menos preciso. El sufijo F le brinda una manera de decirle al compilador: "Oye, sé lo que estoy haciendo, y me arriesgaré, muchas gracias".

float f = 23.467890; // error de Compilación, posible pérdida de precisión.

float g = 49837849.029847F; // OK; tiene el sufijo "F"

Literales Booleanos

Los literales booleanos son la representación del código fuente para los valores booleanos. Un valor booleano solo se puede definir como true o false. Aunque en C (y algunos otros lenguajes) es común usar números para representar verdadero o falso, esto no funciona en Java.

```
boolean t = true; // Legal
boolean f = 0; // error de compilación!
```

Literales de Carácter

Un literal char es la representación de un carácter entre comillas simples.

```
char a = 'a';
char b = '@';
```

También puedes escribir el valor Unicode del carácter, usando la notación Unicode anteponiendo **\u** de la siguiente manera:

```
char letterN = '\u004E'; // La letra 'N'

char c = '\"'; // Una comilla doble.
char d = '\n'; // Un salto de línea.
char v=’\u0000’ //carácter vacío
```

CLASES DE UTILIDAD

Dentro del API de Java existe una gran colección de clases que son muy utilizadas en el desarrollo de aplicaciones. Las clases de utilidad son clases que definen un conjunto de métodos que realizan funciones, normalmente muy reutilizadas. Estas nos van a ayudar junto con las estructuras de control, a lograr resolver problemas de manera más sencilla.

CLASE STRING

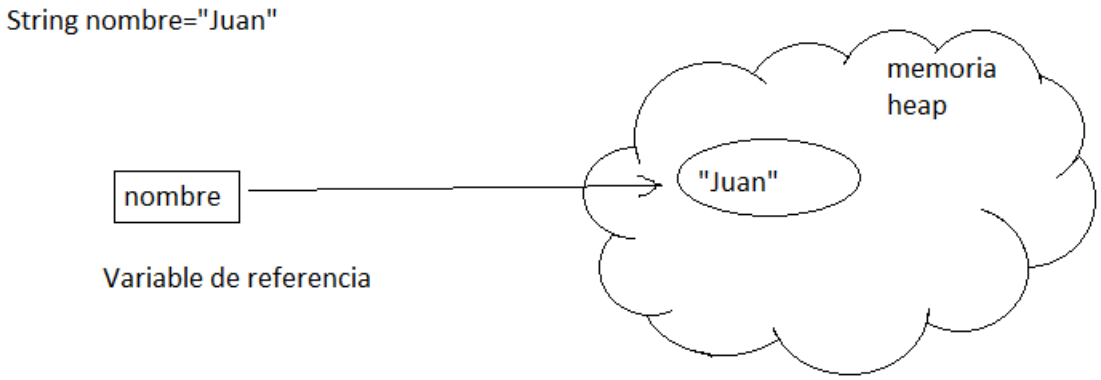
La clase String está orientada al manejo de cadenas de caracteres y pertenece al paquete java.lang del API de Java. Los objetos que son instancias de la clase String, se pueden crear a partir de cadenas constantes también llamadas literales, las cuales deben estar contenidas entre comillas dobles. Una instancia de la clase String **es inmutable**, es decir, una vez que se ha creado y se le ha asignado un valor, éste no puede modificarse (añadiendo, eliminando o cambiando caracteres).

Al ser un objeto, una instancia de la clase String no sigue las normas de manipulación de los datos de tipo primitivo con excepción del operador concatenación. El operador + realiza una concatenación cuando, al menos, un operando es un String. El otro operando puede ser de un tipo primitivo. El resultado es una nueva instancia de tipo String.

Ejmplo de porque decimos que String es inmutable:

Si tenemos las siguientes declaraciones de variables:

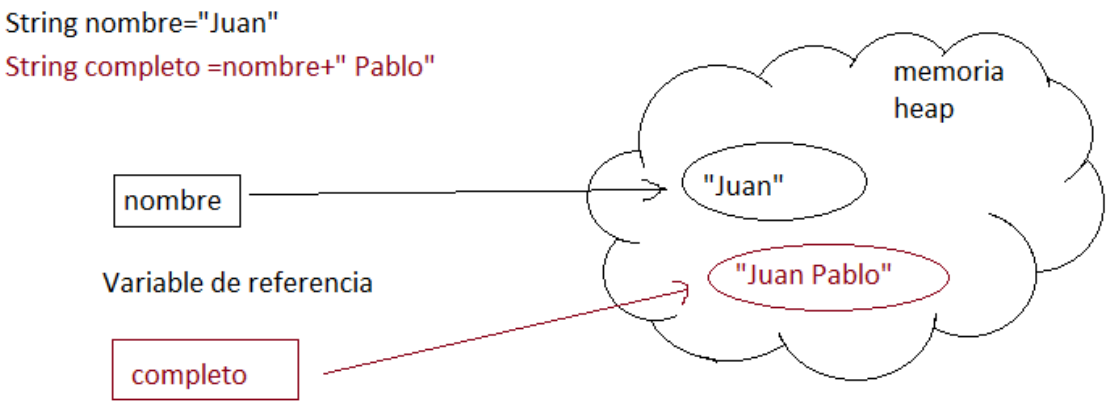
String nombre=”Juan”;



Si luego por ejemplo hacemos;

String completo=nombre+” Pablo”;

El String original “nombre” no se modificará, al utilizar el operador + para concatenar la cadena “Pablo” se creará un nuevo objeto String con la cadena completa “Juan Pablo”



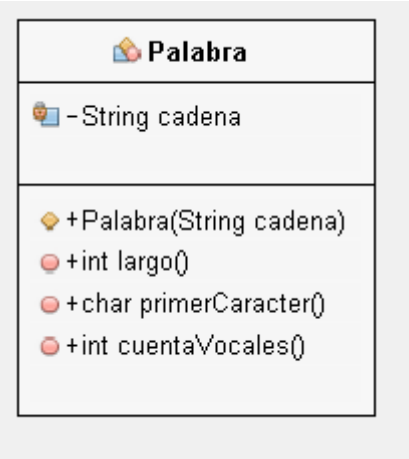
Método	Descripción.
charAt(int index)	Retorna el carácter especificado en la posición index
equals(String str)	Sirve para comparar si dos cadenas son iguales. Devuelve true si son iguales y false si no.
equalsIgnoreCase(String str)	Sirve para comparar si dos cadenas son iguales, ignorando la grafía de la palabra. Devuelve true si son iguales y false si no.
compareTo(String otraCadena)	Compara dos cadenas de caracteres alfabéticamente. Retorna 0 si son iguales, entero negativo si la primera es menor o entero positivo si la primera es mayor.
concat(String str)	Concatena la cadena del parámetro al final de la primera cadena.
contains(CharSequence s)	Retorna true si la cadena contiene la secuencia tipo char del parámetro.
endsWith(String suffix)	Retorna verdadero si la cadena es igual al objeto del parámetro

indexOf(String str)	Retorna el índice de la primera ocurrencia de la cadena del parámetro
isEmpty()	Retorna verdadero si la longitud de la cadena es 0
length()	Retorna la longitud de la cadena
replace(char oldChar, char newChar)	Retorna una nueva cadena reemplazando los caracteres del primer parámetro con el carácter del segundo parámetro
split(String regex)	Retorna un arreglo de cadenas separadas por la cadena del parámetro
startsWith(String prefix)	Retorna verdadero si el comienzo de la cadena es igual al prefijo del parámetro
substring(int beginIndex)	Retorna la sub cadena desde el carácter del parámetro
substring(int beginIndex, int endIndex)	Retorna la sub cadena desde el carácter del primer parámetro hasta el carácter del segundo parámetro
toCharArray()	Retorna el conjunto de caracteres de la cadena
toLowerCase()	Retorna la cadena en minúsculas
toUpperCase()	Retorna la cadena en mayúsculas

Veamos algunos de los métodos de la clase String en un ejemplo:

Supongamos que nos piden crear una clase de nombre Palabra que contendrá como atributo una cadena de caracteres (String) un constructor que nos permitirá inicializar dicha cadena y una serie de métodos:

a) largo: Este método retornará la longitud de la cadena.
b) primerCaracter: Este método retornará el primer carácter de la cadena.
c) cuentaVocales: Este método retornará la cantidad de vocales guardada en la cadena.



Como te habrás dado cuenta, por una cuestión de simplicidad a esta clase no le agregamos los métodos getter y setter, por lo tanto una vez creada una Palabra, no podremos modificarla o ver el valor contenido en su atributo oculto "cadena", pero para nuestros fines, nos sirve.

Analizaremos uno a uno cada método:

a) largo: Este método se comportará como una función, ya que retornará la longitud de cadena; y como cadena es un String, podemos usar su método length para saber su largo!!!.

```
public int largo(){

    return this.cadena.length();
}
```

Eso fue fácil!!!, veamos el otro método:

b) primerCaracter: Recordemos que String guarda la cadena de caracteres como un arreglo de tipo char, si bien veremos más adelante como usar arreglos en Java, espero que algo recuerdes de PSEINT!!!
   Por ejemplo si en cadena guardamos la palabra "ULP", en la posición 0, se encontrará el carácter 'U' y en la posición 2, el carácter 'P'.
   Ahora veamos…, que método tiene String para que me retorne un carácter dada una posición (índice)… Exacto!!! Podemos utilizar el método charAt.

```
public char primerCaracter(){
  if(this.cadena!=null){
        return this.cadena.charAt(0);

}else {

        return '\u0000';
}
}
```

En este caso nos aseguramos que si la cadena es null retorne carácter vació '\u0000'

Ahora se complica más!!!

c) cuentaVocales: Este método también será una función que retornará la cantidad de vocales que tiene la cadena, pueden estar en mayúsculas o minúsculas. Ojo!!!
Sabiendo que String maneja a la cadena de caracteres como un arreglo, podríamos recorrer posición por posición la cadena e ir preguntando si allí hay alguna de las vocales "aAeEiIoOuU"

```
public int cuentaVocales() {

int contador=0;
String palabra=this.cadena.toUpperCase();
                for(int i=0;i<palabra.length();i++){
                        if(palabra.charAt(i)=='A'||palabra.charAt(i)=='E'||
palabra.charAt(i)=='I' || palabra.charAt(i)=='O' || palabra.charAt(i)=='U'){
                                contador++;
                            }

}
return contador;
}
```

A Continuación código completo:

C:/Users/Usuario/Documents/NetBeansProjects/EjemploString/src/ejemplostring/Palabra.java

```
 1
 2 package ejemplostring;
 3
 4 public class Palabra {
 5     private String cadena;
 6
 7     public Palabra(String cadena) {
 8         this.cadena = cadena;
 9     }
10
11
12 public int largo(){
13
14       return this.cadena.length();
15 }
16 public char primerCaracter(){
17    if(this.cadena!=null){
18 return this.cadena.charAt(0);
19
20 }else {
21
22   return '\u0000';
23 }
24 }
25
26 public int cuentaVocales() {
27
28 int contador=0;
29 String palabra=this.cadena.toUpperCase();
30 for(int i=0;i<palabra.length();i++){
31     if(palabra.charAt(i)=='A'||palabra.charAt(i)=='E'||
32             palabra.charAt(i)=='I' || palabra.charAt(i)=='O'
33             || palabra.charAt(i)=='U'){
34 contador++;
35         }
36
37 }
38 return contador;
39 }
40
41 }
```

Luego, desde el método main de una clase Test, podemos probarlo de la siguiente forma:

C:/Users/Usuario/Documents/NetBeansProjects/EjemploString/src/ejemplostring/Test.java

```
1
2 package ejemplostring;
3
4
5 public class Test {
6
7
8     public static void main(String[] args) {
9        Palabra pal=new Palabra("Hola");
10        System.out.println("Largo: "+pal.largo());
11        System.out.println("Primer Caracter:  "+pal.primerCaracter());
12        System.out.println("Cantidad de vocales: "+pal.cuentaVocales());
13     }
14
15 }
```

CLASE MATH

En ocasiones nos vemos en la necesidad de incluir cálculos, operaciones, matemáticas, estadísticas, etc en nuestro programas Java.

Es cierto que muchos cálculos se pueden hacer simplemente utilizando los operadores aritméticos que java pone a nuestra disposición, pero existe una opción mucho más sencilla de utilizar, sobre todo para cálculos complicados. Esta opción es la clase Math del paquete java.lang.

La clase Math nos ofrece numerosos y valiosos métodos y constantes estáticos, que podemos utilizar tan sólo anteponiendo el nombre de la clase.

Método	Descripción.
abs(double a)	Devuelve el valor absoluto de un valor double introducido como parámetro.
abs(int a)	Devuelve el valor absoluto de un valor Entero introducido como parámetro.
abs(long a)	Devuelve el valor absoluto de un valor long introducido como parámetro.
max(double a, double b)	Devuelve el mayor de dos valores double
max(int a, int b)	Devuelve el mayor de dos valores Enteros.
max(long a, long b)	Devuelve el mayor de dos valores long.
min(double a, double b)	Devuelve el menor de dos valores double.
min(int a, int b)	Devuelve el menor de dos valores enteros.

min(long a, long b)	Devuelve el menor de dos valores long.
pow(double a, double b)	Devuelve el valor del primer argumento elevado a la potencia del segundo argumento.
random()	Devuelve un double con un signo positivo, mayor o igual que 0.0 y menor que 1.0.
round(double a)	Devuelve el long redondeado más cercano al double introducido.
sqrt(double a)	Devuelve la raíz cuadrada positiva correctamente redondeada de un double.
floor(double a)	Devuelve el entero más cercano por debajo.

MÉTODO RANDOM() DE LA CLASE MATH

El método **random** podemos utilizarlo para generar números al azar. El rango o margen con el que trabaja el método random oscila entre 0.0 y 1.0 (Este último no incluido)

Por lo tanto, para generar un número entero entre 0 y 9, hay que escribir la siguiente sentencia:

```
int numero = (int) (Math.random() * 10);
```

CLASE SCANNER

En Java hay muchas maneras para hacer que nuestro programa permita el ingreso de datos por teclado, en el curso vamos a usar la clase Scanner.

Scanner es una clase en el paquete **java.util** utilizada para obtener la entrada de los tipos primitivos como int, double etc. y también String. Es la forma más fácil de leer datos en un programa Java.

Ejemplo definición clase Scanner:

Scanner leer = new Scanner(System.in);

• Para poder instanciar un objeto de tipo Scanner, vamos a tener que importar explícitamente la clase ya que se encuentra en un paquete del api de java que no se auto-importa, en este caso el paquete "java.util". Esta sentencia de importación, va debajo de la sentencia **package**. La sentencia se ve así: **import java.util.Scanner;**

• Para crear un objeto de clase Scanner, normalmente pasamos por parámetro a su constructor el objeto predefinido **System.in**, que representa el flujo de entrada estándar.

• Para utilizar los métodos del objeto Scanner, vamos a utilizar el nombre que le hemos asignado en este caso "leer" y después del nombre ponemos un punto (.), de esa manera podremos llamar a las funciones del Scanner.

• Para leer valores numéricos de un determinado tipo de datos, la función que se utilizará es **nextTipo().** Por ejemplo, para leer un valor de tipo int (entero), podemos usar **nextInt()**, para leer un valor de tipo double(real), usamos **nextDouble()** y etc. Para leer un String (cadenas), usamos **nextLine()**.

En la siguiente definición estamos diciendo que cuando se ejecute esa línea de código, se guardará en la variable entera "numero" el entero que se lea desde teclado:

int numero = leer.nextInt();

El mismo ejemplo, sólo que la variable está declarada una línea más arriba:

int numero;

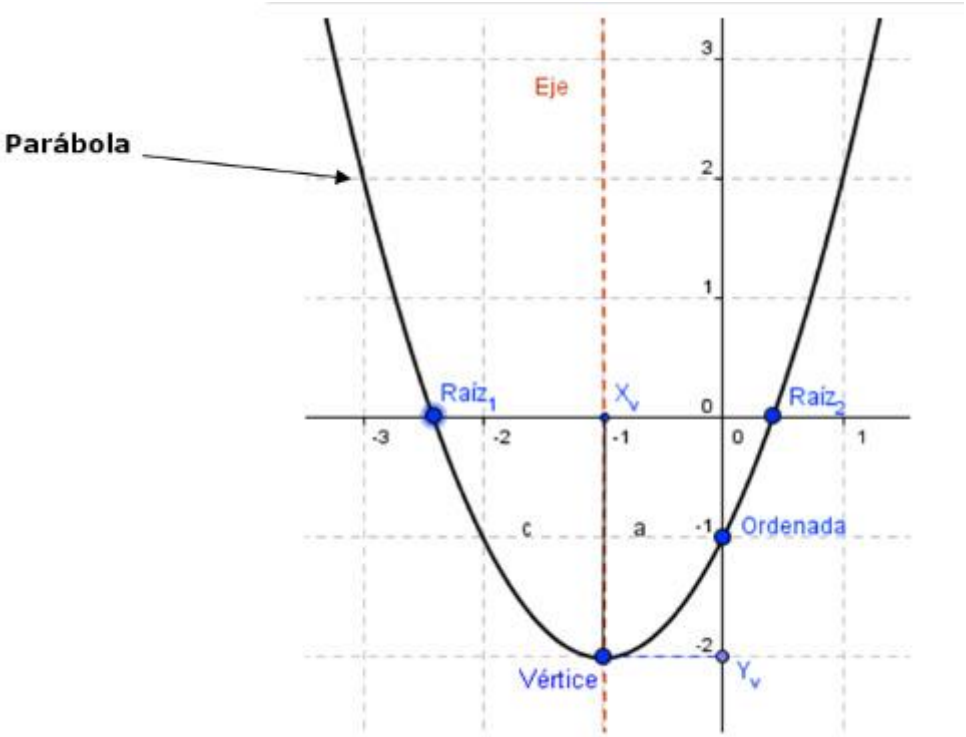
numero = leer.nextInt();

Veamos en un ejemplo los métodos de estas dos últimas clases de utilidad: Math y Scanner.

Supongamos que nos piden crear una clase de nombre **Calculo** con un método estático de nombre ***calcularRices()*.** Este método recibe los coeficientes a, b y c de una ecuación de segundo grado y mostrará por consola sus raíces y si no las tiene también lo informará.

Luego desde el método main de una clase Prueba, solicitaremos a usuario ingrese por teclado los valores de los 3 coeficientes e invocaremos al método ***calcularRaices*** pasando dichos valores.

Recordemos…



Raíces (raíz$_1$ yraíz$_2$): las raíces o ceros de la función cuadrática son aquellos valores de **x** para los cuales la expresión vale 0. Gráficamente, las raíces corresponden a las abscisas de los puntos donde la parábola corta al eje **x**.

Podemos determinar las raíces de una función cuadrática igualando a cero la función **f(x)** = **0**, y así obtendremos la siguiente ecuación cuadrática: $\mathbf{ax^2 + bx + c = 0}$

Para calcular las raíces se utiliza la siguiente fórmula: $$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
 1
 2 package ejemplomath;
 3
 4 public class Calculo {
 5
 6     public static void calcularRaices(int a,int b, int c){
 7         double raiz1=0,raiz2=0;
 8         double discriminante=Math.pow(b, 2)-4*a*c;
 9
10         if(discriminante < 0){
11
12             System.out.println("No hay raices ");
13         }else if(discriminante==0){
14
15             raiz1=-b/2*a;
16             System.out.println("Sólo hay una raiz "+raiz1);
17
18         }else{
19
20             raiz1=(-b+Math.sqrt(discriminante))/(2*a);
21             raiz2=(-b-Math.sqrt(discriminante))/(2*a);
22             System.out.println("Una raiz es "+raiz1+" y la otra"+raiz2);
23         }
24
25     }
26
27 }
```

C:/Users/Usuario/Documents/NetBeansProjects/EjemploMath/src/ejemplomath/Prueba.java

```
 1
 2 package ejemplomath;
 3
 4 import java.util.Scanner;
 5
 6
 7 public class Prueba {
 8
 9     public static void main(String[] args) {
10
11         Scanner leer=new Scanner(System.in);
12         int a, b,c;
13
14         System.out.println("Ingrese coeficiente a");
15         a=leer.nextInt();
16
17         System.out.println("Ingrese coeficiente b");
18         b=leer.nextInt();
19
20         System.out.println("Ingrese coeficiente c");
21         c=leer.nextInt();
22
23         Calculo.calcularRaices(a, b, c);
24     }
25
26 }
```

1.1 of 1 2022.11.23 18:57:37

Ahora pondremos en práctica estos conocimientos con una serie de ejercicios, vamos!!!

BIBLIOGRAFIA:

Christopher Alexander, “A Pattern Language”, 1978

Erich Gamma, “Design Patterns: Elements of Reusable OO Software”

Martin Fowler, “Analysis Patterns: Reusable Object Models”, Addison Wesley, 1997