¡Hola mundo!

```
class HolaMundo {
    public static void main(String[] arg) {
        System.out.println("¡Hola mundo!");
    }
}
```

Salida por consola

```
class HolaMundo {
    public static void main(String[] arg) {
        System.out.print("¡Hola mundo!");
        System.out.println("Otro mensaje");
    }
}
```

Variables

```
int intVar     = 100;
float floatVar = 100.80f;
String strVar  = "Hola";

int other;
final int pi = 3.14;
```

Entrada de datos

```
import java.util.Scanner;

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

        Scanner input = new Scanner(System.in);

        System.out.print("Dame un número entero: ");
        int number = input.nextInt();
        System.out.println("Tecleaste " + number);
    }
}
```

Java arrays

```
int[] arrayOfInt;
int[] arrayOfInt = {10, 15, 20, 30, 40};
int[] arrayOfInt = new int[5];
int value = arrayOfInt[3];
arrayOfInt[3] = 10;
int size = arrayOfInt.length;
```

Tipos de datos primitivos

byte, char, short, int y long
float y double
boolean (valores true/false) y void

Recorrido de un array

```
class Main {
    public static void main(String[] arg) {
        int[] myArray = {10, 15, 20, 30, 40};

        for (int value: myArray) {
            System.out.println(value);
        }

        for (int i = 0; i < myArray.length; i++) {
            System.out.println(myArray[i]);
        }
    }
}
```

Estructuras multidimensionales

```
int[][] myArray = {
    {10, 15, 20, 30, 40},
    {12, 14, 16, 18},
    {11, 17, 23, 29, 31}
};

System.out.println(myArray[1][3]);
```

La clase Arrays

```
import java.util.Arrays;

class Main {
    public static void main(String[] arg) {
        int[] array1 = {90, 80, 70, 60, 50, 40, 30, 20, 10};
        Arrays.sort(array1, 3, 6); // [90, 80, 70, 40, 50, 60, 30, 20, 10]
        Arrays.sort(array1);       // [10, 20, 30, 40, 50, 60, 70, 80, 90]

        int[] myArray =  {10, 15, 20, 30, 40};
        System.out.println(Arrays.binarySearch(myArray, 15)); // Muestra 1
        System.out.println(Arrays.binarySearch(myArray, 45)); // Muestra -6

        int[] shorter = Arrays.copyOf(myArray, 3)); // {10, 15, 20}
        int[] longer  = Arrays.copyOf(myArray, 7)); // {10, 15, 20, 30, 40, 0, 0}

        int[] other = Arrays.copyOfRange(myArray, 2, 5); // {20, 30, 40}

        int[] array3 = {10, 15, 20, 30, 40};
        int[] array4 = {15, 10, 30, 40, 20};
        System.out.println(Arrays.equals(array3, array4)); // Muestra false

        int[] array5 = new int[10];   // {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
        Arrays.fill(array5, 1);       // {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
        Arrays.fill(array5, 4, 8, 2); // {1, 1, 1, 1, 2, 2, 2, 2, 1, 1}

        int[] array6 = {10, 15, 20, 30, 40};
        String string1 = Arrays.toString(array6); // "[10, 15, 20, 30, 40]"
    }
}
```

Recorrido de estructuras multidimensionales

```
// Recorrido por índices
for (int row = 0; row < myArray.length; row++) {
    for (int col = 0; col < myArray[row].length; col++) {
        System.out.println(myArray[row][col]);
    }
}

// Recorrido por valores
for (int[] row: myArray) {
    for (int value: row) {
        System.out.println(value);
    }
}
```

Operadores aritméticos

```
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) );
   }
}
```

```
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
```

Operadores relacionales

```
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) );
   }
}
```

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

Operador instanceof

```
public class Test {

   public static void main(String args[]) {
      String name = "James";
      boolean result = name instanceof String;
      System.out.println( result );
   }
}
```

```
true
```

Operadores lógicos

```
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));
   }
}
```

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

Operador condicional (? :)

```
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 );
   }
}
```

```
Value of b is : 30


Value of b is : 20
```

```
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 = a + b = 30


c += a  = 40


c -= a = 30


c *= a = 300



c /= a = 1



c %= a  = 5
```

Strings

```
String string1 = "Hola mundo";
char ch1 = 'a’;
String empty1  = "";
String empty2  = new String();
int number = 10;
String numberStr = ((Integer) number).toString();
```

Secuencias de escape

```
\n Nueva línea. Coloca el cursor de la pantalla al inicio de la siguiente línea
\t Tabulador horizontal. Desplaza el cursor hasta la siguiente posición de tab
\r Retorno de carro. Coloca el cursor de la pantalla al inicio de la línea actual
\“ Imprime un carácter de doble comilla
\\ Imprime un carácter barra diagonal
```

Longitud de una string

```
String s1 = "esto es una string";
System.out.println(s1.length()); // 18
```

Acceso a carácteres y substrings

```
String s1 = "Hola mundo";
char c1 = s1.charAt(3); // ‘a

String s1 = "Hola mundo";
String s2 = s1.substring(1, 4); // "ola“

String s1 = "Hola mundo";
String s2 = s1.substring(5); // "mundo"
```

Concatenación de strings

```
String s1 = "Hola";
String s2 = "mundo";
String s3 = s1 + " " + s2; // "Hola mundo“

String s1 = "Hola";
String s2 = "mundo";
String s3 = s1.concat(" ").concat(s2); // "Hola mundo"
```

Comparación de strings

```
String s1 = "esto es una string";
String s2 = "esto es otra string";
String s3 = "esto es una string";
System.out.println(s1.equals(s2)); // false
System.out.println(s1.equals(s3)); // true

System.out.println("Hola".equalsIgnoreCase("hola")); // true

System.out.println(s1.compareTo(s2)); // 6
System.out.println(s2.compareTo(s1)); // -6

También existe la versión .compareToIgnoreCase()
```

Contención

```
String s1 = "Hola mundo";
System.out.println(s1.contains("la mu")); // true
System.out.println(s1.contains("casa"));  // false
```

Localización

```
String s1 = "Hola, bienvenidos a mi mundo";
int pos = s1.indexOf("bienvenidos"); // 6

String s1 = "Hola, bienvenidos a mi mundo";
int pos = s1.indexOf("bienvenidos", 9); // -1

String s1 = "La araña con maña teje la telaraña";
int pos = s1.lastIndexOf("araña"); // 29

String s1 = "La araña con maña teje la telaraña";
int pos = s1.lastIndexOf("araña", 9); // 3
```

Otras operaciones útiles

Comienzo	public boolean startsWith(String chars)
Fin	public boolean endsWith(String chars)
Truncar	public String trim()
Conv. a Minúsculas	public String toLowerCase()
Conv. a Mayúsculas	public String toUpperCase()
Dividir	public String[] split(String regex)
	public String[] split(String regex, int limit)

Clases y objetos

```
class MyClass {}
MyClass myObject = new MyClass();
```

Atributos de datos de objeto, inicialización

```
class MyClass {
    int attribute1;
    int attribute2;

    public MyClass(int value1, int value2) {
        this.attribute1 = value1;
        this.attribute2 = value2;
    }
}

class Main {
    public static void main(String[] arg) {
        MyClass myObject = new MyClass(10, 15);
    }
}
```

Encapsulamiento de datos

```
class Person {
    private String name;

    Person(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String newName) {
        this.name = newName;
    }
}

class Main {
    public static void main(String[] arg) {
        Person aPerson = new Person("Pepe");
        System.out.println(aPerson.getName());
        aPerson.setName("Juan");
        System.out.println(aPerson.getName());
    }
}
```

Visibilidad

- public.- visible para todo el mundo
- protected.- visible para la clase, sus clases derivadas (subclases) y las clases definidas en el mismo package.
- private.-visible solo para la clase

```
public class MyClass {
    private int attribute1;
    private int attribute2;

    MyClass(int value1, int value2) {
        this.attribute1 = value1;
        this.attribute2 = value2;
    }

    private int auxMethod(int param) {
        return (this.attribute1 + this.attribute2) / param;
    }

    public int myMethod(int param) {
        return auxMethod(param);
    }
}
```

Herencia

```
class One {} // Superclase

class Two extends One {} // Subclase
```

Inicialización de subclases

```
class One { // Superclase
    int attr1;

    public One(int value) {
        this.attr1 = value;
    }
}

class Two extends One { // Subclase
    int attr2;

    public Two(int value1, int value2) {
        super(value1);
        this.attr2 = value2;
    }
}
```

Atributos de clase

```
class MyClass {
    private static int clsAttr = 0;
    private int objAttr1, objAttr2;

    MyClass(int value1, int value2) {
        this.objAttr1 = value1;
        this.objAttr2 = value2;
    }

    public static void setClsAttr(int value) {
        clsAttr = value;
    }

    public static int getClsAttr() {
        return clsAttr;
    }
}

class Main {
    public static void main(String[] arg) {
        MyClass.setClsAttr(3);
        System.out.println(MyClass.getClsAttr());
    }
}
```

Sustitución de métodos (overriding)

```
class ClassOne extends BaseClass { // Subclase
    @Override
    public String message() {
        return super.message() + " and then a message from ClassOne";
    }
}
```

Métodos heredados de Object

```
protected Object clone()
boolean equals(Object obj)
Class<?> getClass()
String toString()
```

Clases abastractas

```
public abstract class Shape {
    public abstract double area();
}
```

Interfaces

```
public interface IShape {
    public double area();
}

public class Square implements IShape {
    double side;

    Square(double side) {
        this.side = side;
    }

    @Override
    public double area() {
        return side * side;
    }
}

public class Square extends Paralelogram implements IShape, Cloneable{...}
```

La interfaz Cloneable

```
public class CloneableClass implements Cloneable {
    int[] data;
    int sum;

    CloneableClass() {
        this.data = new int[]{1, 2, 3, 4, 5};
        this.sum = 15;
    }

    @Override
    public CloneableClass clone()  throws CloneNotSupportedException {
        CloneableClass newObject = (CloneableClass) super.clone();

        if (data != null) {
            newObject.data = this.data.clone();
        }

        newObject.sum = this.sum; // No necesita clone por ser primitivo
        return newObject;
    }
}
```

La interfaz Comparable<T>

```
public class Pareja implements Comparable<Pareja>{
    int a, b;

    public Pareja(int a, int b){
        this.a= a;
        this.b= b;
    }

    public int compareTo(Pareja o){
        if(a > o.a) return 1;
        if(a < o.a) return -1;
        if(b > o.b) return 1;
        if(b < o.b) return -1;
        return 0;
    }
}
```