CONCEPTOS RELACIONADOS CON LA PROGRAMACIÓN

Programación

"El término programación se define como un conjunto de instrucciones consecutivas y ordenadas que llevan a ejecutar una tarea específica. Dichas instrucciones se denominan "código fuente", el cual es único para cada lenguaje y está diseñado para cumplir una función o propósito específico." (Morales, 2014)

Lenguaje de Programación

Un lenguaje de programación es una "estructura que, con una cierta base sintáctica y semántica, imparte distintas instrucciones a un programa de computadora" (Perez y Merino, 2009).
Un lenguaje de programación está compuesto por:

- Símbolos
- Reglas sintácticas y Semánticas
- Estructura
- Significado
- Elementos y expresiones

IDE (Entorno de Desarrollo Integrado)

Existen diferentes programas comerciales que permiten desarrollar código Java. Los IDE's (Integrated Development Environment), tal y como su nombre indica, son entornos de desarrollo donde, en un mismo programa es posible escribir el código en un lenguaje de programación, compilarlo (o interpretarlo) y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar Debug tanto de manera textual (mediante una consola) o gráficamente. Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos bibliotecas con componentes ya desarrollados para ser utilizados durante la manipulación del programa. Estas herramientas brindan una interfaz gráfica para facilitar y agilizar el proceso de escritura de los programas.

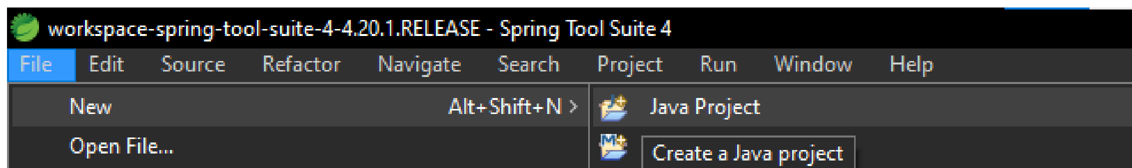
Además, agregan funcionalidades para acceso y gestión de base de datos, integración con otras herramientas para control de versiones, simulación de servidores, generación de documentación, gestión de interfaces de usuarios, modelado de artefactos, etc.

Los ejemplos usados en este apunte se han desarrollado en el IDE Sprint Tool Suite.

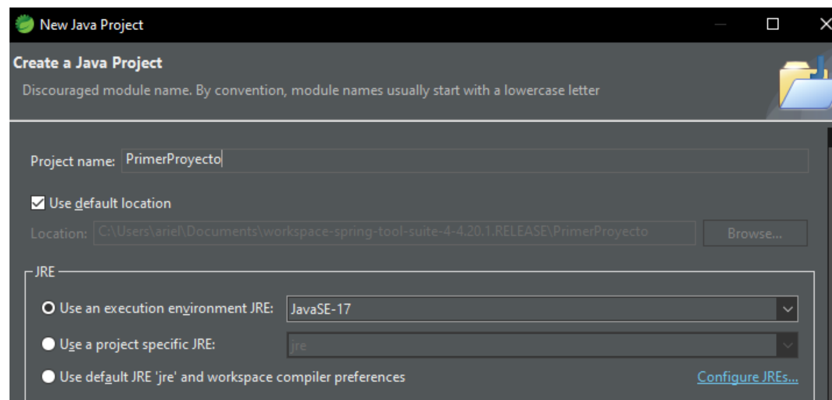
PROGRAMANDO CON JAVA

Primer proyecto en STS

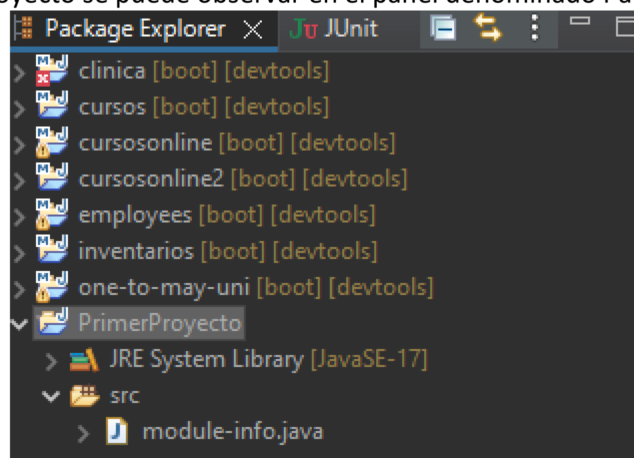
Elija el menú File -> New -> Java Project



Se abrirá una ventana, donde lo más importante es indicar el nombre del proyecto y verificar que esté seleccionado en la sección JRE el ambiente de desarrollo Java-SE correspondiente. Spring Too Suite creará una carpeta con el nombre del proyecto indicado y generará todas las subcarpetas y archivos necesarios para empezar a crear el programa.
En este caso el nombre del proyecto es PrimerProyecto y el ambiente de ejecución java es JavaSE-17



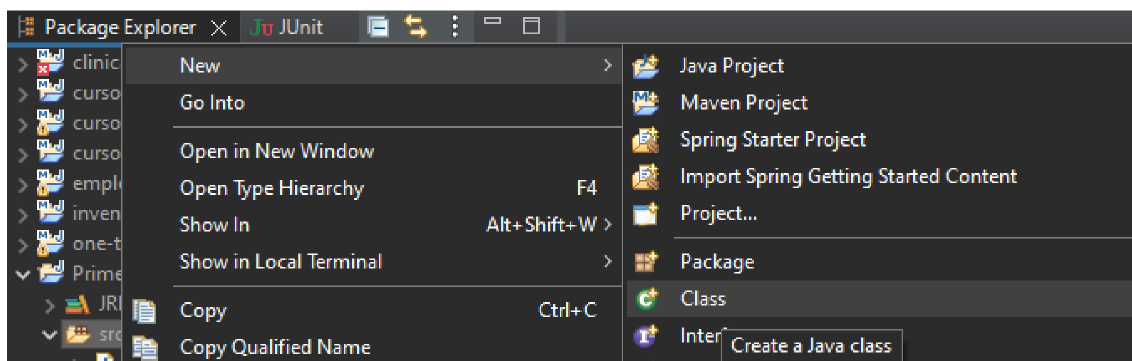
La estructura del proyecto se puede observar en el panel denominado Package Explorer



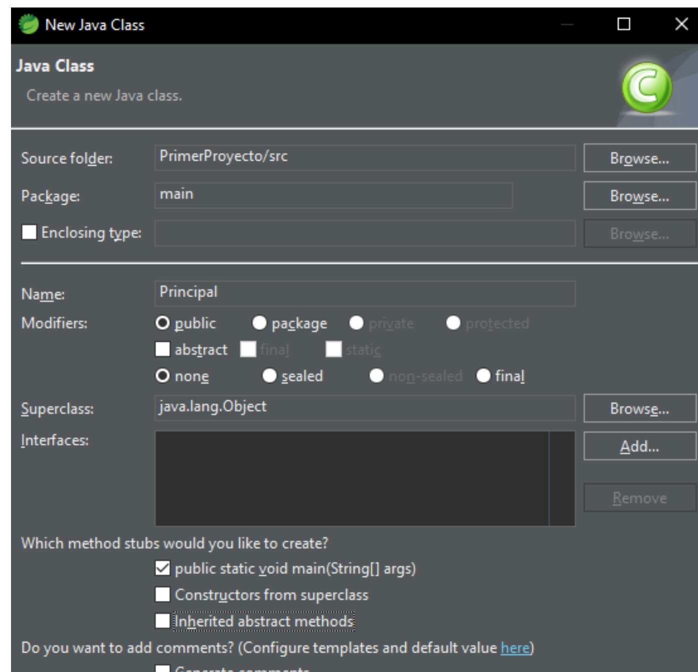
Un proyecto Java simple contiene una carpeta donde se encuentra la biblioteca de clases java, es decir los recursos de Java que nos permiten programar; y una carpeta denominada src (de source) donde se podrán crear los archivos fuente de la aplicación.

Creación de el punto de ejecución: La clase con el método main

Un programa Java es un conjunto de uno o varios archivos que especifican clases. Ningún código Java puede ejecutarse fuera de la definición de una clase Java.
Para crear una clase Java en STS proceda de la siguiente manera: sobre el nombre de la carpeta src realizar botón derecho, elegir opción New -> Class



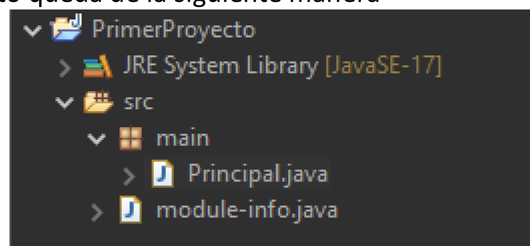
Se abrirá una ventana donde debe llenarla de la siguiente manera



Donde:

- Package: permite organizar las clases definidas, similar a una estructura de carpetas (de hecho genera carpetas)
- Name: aquí se indica el nombre de la clase
- Casilla de verificación public static void main: Esto genera una porción de código, que le permitirá a Java ejecutar el programa.

La estructura del proyecto queda de la siguiente manera



Puede observar que el paquete tiene la forma de una caja y se puede interpretar como una subcarpeta de src.

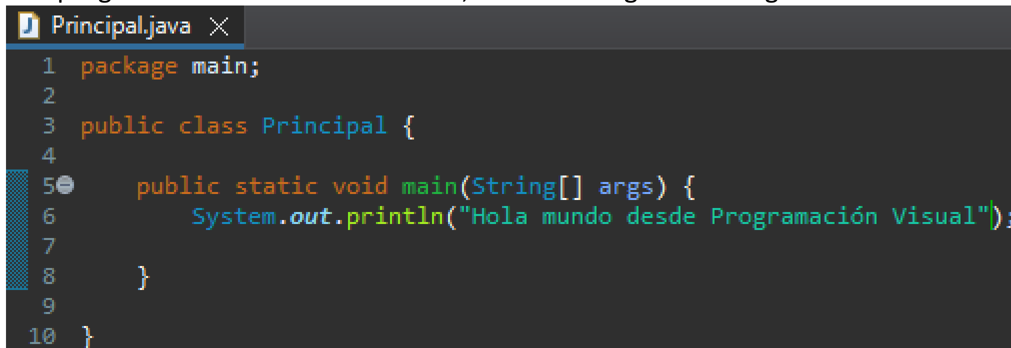
Además del paquete main, cuelga el archivo Principal.java. Toda definición de clase se aloja dentro de un archivo con extensión .java. Además, puede percatarse que el archivo tiene el mismo nombre de la clase.

Ahora veamos el interior de la clase. Si hace doble click sobre el nombre del archivo, se abrirá en el panel del Editor de Código de STS

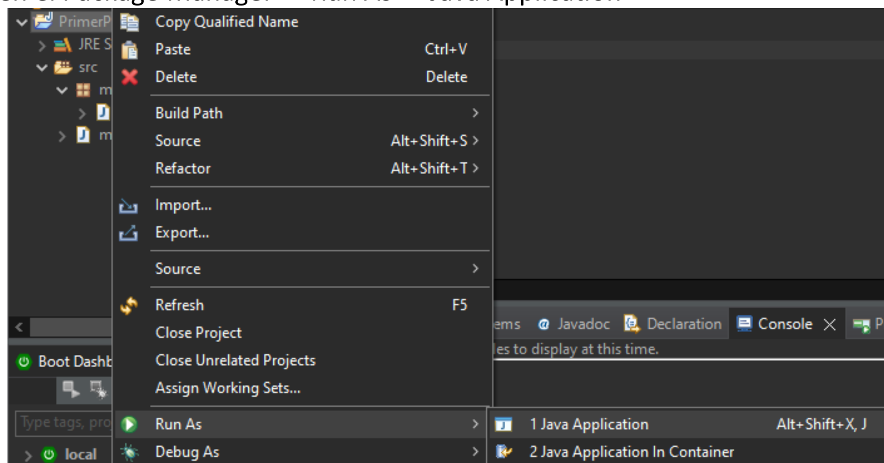
Principal.java

```
1  package main;
2
3  public class Principal {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7
8      }
9
10 }
11
```

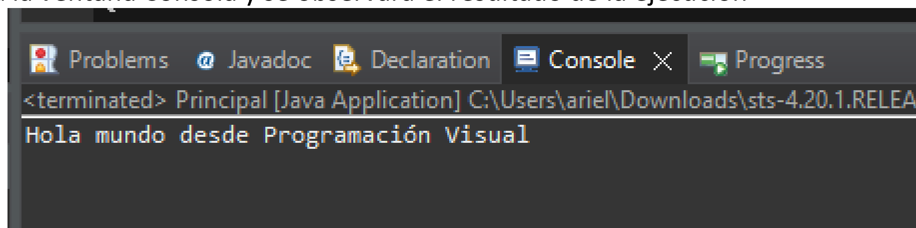
La primera línea de código de un archivo .java siempre es la sentencia package que indica el nombre del paquete sobre el cual se aloja la clase. Observe además que la sentencia finaliza en ;. Todas las sentencias Java finalizan en ;.
En la línea 3 se define la clase mediante public class Principal{. La palabra reservada class es la que permite indicar que se está definiendo una clase. La palabra public es obligatoria si el archivo posee una única definición de clase. En ese caso, además, es obligatorio que la clase se llame igual que el archivo.
En este caso, en lugar de finalizar con ; finaliza con {. Esto se debe a que la definición de una clase, debe incluir {}. Estas llaves indican donde empieza y donde finaliza la definición de una clase. En nuestro ejemplo la clase inicia en la línea 3 y finaliza en la línea 10.
Finalmente entre la línea 5 y 8 se ha definido el método main(). Un método, es un espacio donde se puede escribir el código del cuerpo de un algoritmo. No se puede codificar el cuerpo de un algoritmo fuera de un método.
El método main() es especial. Java busca una clase que tenga este método para iniciar la ejecución del código que está dentro de ese método. Es decir una aplicación java sin una clase que contenga el método main() nunca se ejecutará.
Vamos a programar el famoso Hola Mundo, observe la siguiente imagen



Para ejecutar este código proceda de la siguiente manera: botón derecho sobre el nombre del proyecto en el Package Manager -> Run As -> Java Application



Se abrirá la ventana Consola y se observará el resultado de la ejecución

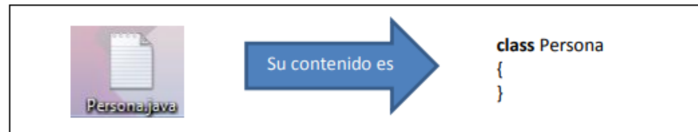


Resumen de esta sección:

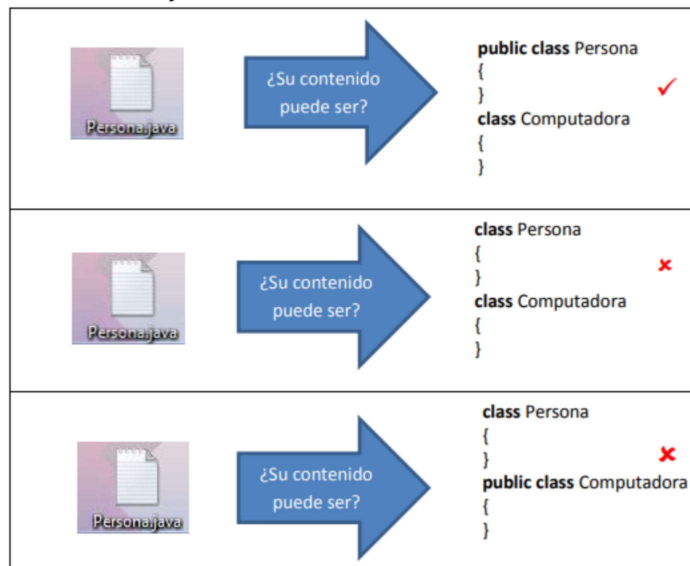
- Las clases son contenedores especiales, tienen la capacidad de indicar qué representan, con que datos trabajan y que acciones pueden hacer. Por lo tanto, se definen usando sustantivos que permitan abarcar de manera coherente las capacidades mencionadas. Por ejemplo, suponga los siguientes sustantivos: Bicicleta, Puesto de Préstamos y Persona. En Java quedarían declaradas o definidas de la siguiente manera:

`public class Bicicleta { }`	`public class PuestoPrestamo { }`	`public class Persona { }`

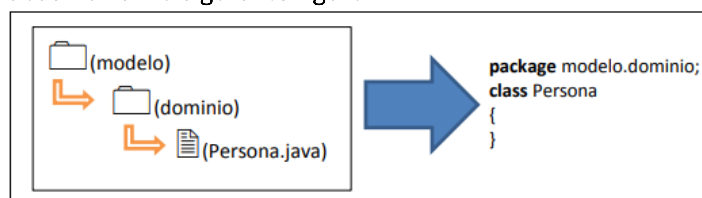
- Las llaves indican el inicio y fin del contenedor definido para la clase, es decir los componentes que se definan entre estas llaves pertenecen a la clase.
- Las clases en Java son definidas en archivos de texto. Estos archivos tienen extensión .java. El nombre del archivo debe coincidir con el nombre de la clase. Por ejemplo, para la clase Persona se creará un archivo denominado Persona.java, es decir



- Java permite que se definan más de una clase en un único archivo con extensión .java. Cuando se presenta esta situación el nombre del archivo debe coincidir con el nombre de la **única** clase que se haya definido como `public`. Esto es, solo puede existir una clase pública en un archivo .java.



- Un paquete puede contener a uno o varios paquetes (denominados sub-paquetes). Para indicar la jerarquía de paquetes a la que pertenece un archivo .java se establece un separador entre los nombres de los paquetes de la jerarquía. Este separador es el ".". La jerarquía de paquetes físicamente es una jerarquía de directorios. Esta representación se puede observar en la siguiente figura



Los comentarios en Java

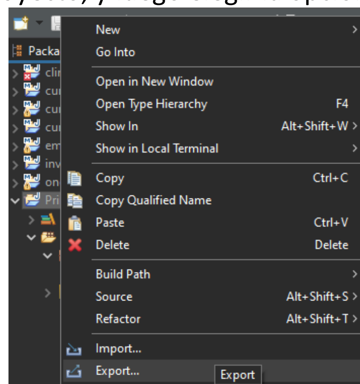
En Java se pueden definir tres tipos diferentes de comentarios:

- Comentario de línea //: indican al compilador del lenguaje que lo que se escriba a continuación de estas sobre la misma línea de código es un comentario.
- Comentario multilínea /* … */: todo lo que se escriba entre estos símbolos será considerado por el compilador de Java como comentario. Se utilizan cuando se desea escribir comentarios extensos.
- Comentario de documentación /**… */: Este tipo de comentario es similar al anterior en el sentido que permite abarcar más de una línea para realizar los comentarios. Se denomina de comentario porque permiten generar la documentación del código desarrollado. Más específicamente permite documentar la definición de las clases, métodos y las propiedades.

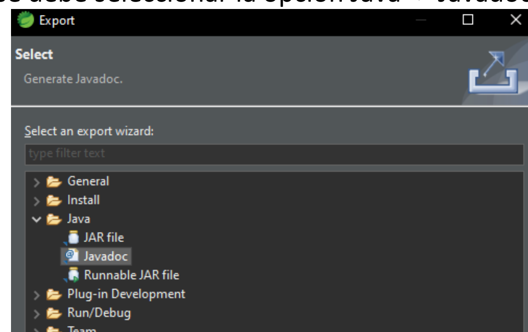
En el siguiente ejemplo se colocan comentarios

```
1  package main;
2
3  /**
4   * @author Ariel Vega
5   * @version 1.0
6   * Contiene el punto de arranque de la aplicación
7   */
8  public class Principal {
9
10     /**
11      * Es el método que usa Java como punto de partida de ejecución
12      * @param args parámetros que puede recibir el método
13      */
14     public static void main(String[] args) {
15         // genera un mensaje por consola
16         System.out.println("Hola mundo desde Programación Visual");
17
18     }
19
20 }
```

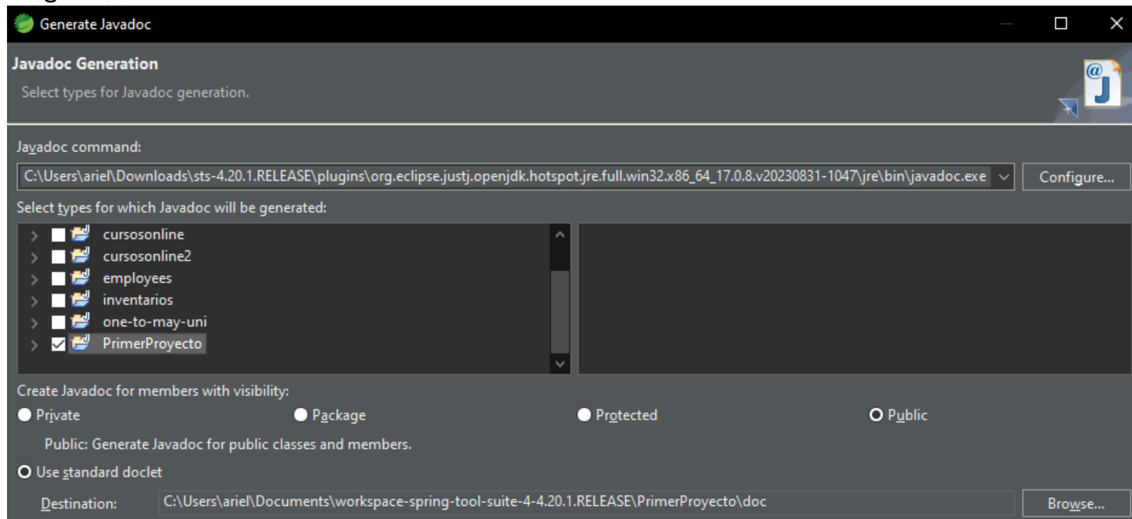
Para generar la documentación de código, se realiza de la siguiente manera desde el STS: botón derecho sobre el nombre del proyecto, y luego elegir la opción Export…



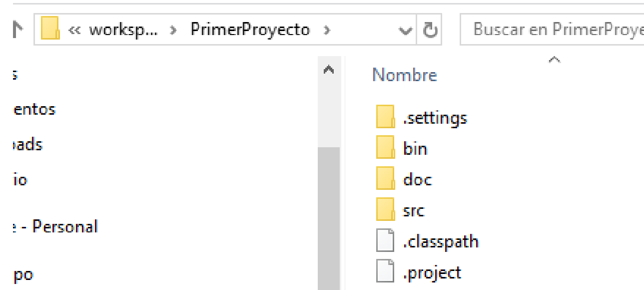
Se abrirá una ventana y se debe seleccionar la opción Java -> Javadoc



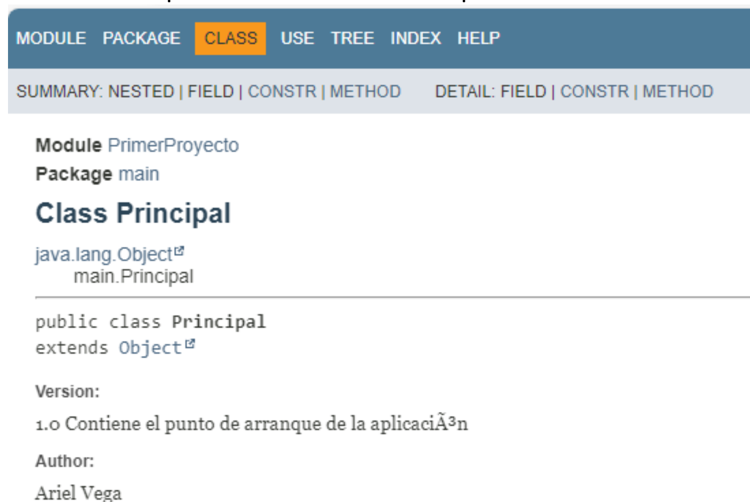
En la siguiente pantalla se debe indicará el proyecto del que se generará la documentación y luego el destino de la documentación.



Luego de seleccionar las demás opciones, se producirá la ejecución. En la ventana de consola se mostrará el proceso de construcción. En este caso el destino es la misma ubicación del proyecto. Se genera una carpeta denominada doc



En ella hay páginas web con la documentación de las clases generadas, por ejemplo a continuación se observa una porción de la clase Principal



Los tipos de Datos en Java

En Java se disponen básicamente de dos tipos diferentes de datos:

A. Los tipos primitivos
B. Los tipos referencia

Los tipos de Datos Primitivos

Son definiciones dentro del lenguaje que no son objetos. Todos los tipos numéricos están firmados (esto significa que a pesar de no ser objetos representan el mismo tipo y alcance en cualquier plataforma que se ejecute). Para lograr esto, al definir una variable con un tipo primitivo, el lenguaje almacena en memoria un espacio de memoria, este espacio de memoria permite representar el rango de valores que se le puede asignar a la variable según el tipo de datos primitivo usado. Esto significa que al declarar una variable primitiva se asume que consumirá memoria, aunque no se la use desde el principio de ejecución del programa ya que se le asigna un espacio en memoria al momento de declararse. Los primitivos poseen valores por defecto y este valor depende del tipo de dato.

Los tipos referencia

En Java, los objetos se crean (instancian) a partir de clases y en ese momento se almacenan en espacios de memoria. Java gestiona la dirección en memoria de estos objetos. Para acceder a ellos Java asigna a cada objeto creado una “referencia”.

Esta referencia, conceptualmente hablando es similar a la dirección de una casa. Para ubicar una casa recurrimos a la dirección. La referencia de un objeto permite acceder al objeto en memoria. Los elementos que permiten generar instancias (objetos) son las clases de la biblioteca de Java, las clases creadas por los programadores, las interfaces, los arreglos y las colecciones. En definitiva, los tipos referencias son los objetos que se crean a partir de las entidades mencionadas en la oración precedente. Más detalles sobre el funcionamiento en memoria de las referencias se verán más adelante.

Nombre	Declaración	Rango	Descripción
Booleano	boolean	true - false	Define una bandera que puede tomar dos posibles valores: true o false.
Byte	byte	[-128 .. 127]	Representación del número de menor rango con signo.
Entero pequeño	short	[-32,768 .. 32,767]	Representación de un entero cuyo rango es pequeño.
Entero	int	[-2^{31} .. $2^{31}-1$]	Representación de un entero estándar. Este tipo puede representarse sin signo usando su clase Integer a partir de la Java SE 8.
Entero largo	long	[-2^{63} .. $2^{63}-1$]	Representación de un entero de rango ampliado. Este tipo puede representarse sin signo usando su clase Long a partir de la Java SE 8.
Real	float	[$\pm 3{,}4\cdot 10^{-38}$ .. $\pm 3{,}4\cdot 10^{38}$]	Representación de un real estándar. Recordar que al ser real, la precisión del dato contenido varía en función del tamaño del número: la precisión se amplía con números más próximos a 0 y disminuye cuanto más se aleja del mismo.
Real largo	double	[$\pm 1{,}7\cdot 10^{-308}$ .. $\pm 1{,}7\cdot 10^{308}$]	Representación de un real de mayor precisión. Double tiene el mismo efecto con la precisión que float.
Carácter	char	['\u0000' .. '\uffff'] o [0 .. 65.535]	Carácter o símbolo. Para componer una cadena es preciso usar la clase String, no se puede hacer como tipo primitivo.

Las variables dentro de los métodos

En los métodos se pueden definir variables. Java es un lenguaje de tipado estático (también se dice que es fuertemente tipado). Por lo cual todas las variables tendrán un tipo de dato (ya sea un **tipo de dato primitivo** o una **referencia**) y un nombre como identificador. El tipo de dato se asignará a la hora de definir la variable.

Para definir una variable se debe seguir la siguiente nomenclatura

```
tipo_variable identificador;
```

Observe un ejemplo de variables numéricas primitivas en el método main()

```java
public static void main(String[] args) {
    // genera un mensaje por consola
    System.out.println("Hola mundo desde Programación Visual");
    // los tipos de datos primitivos numéricos
    byte numeroByte = 127;
    short numeroShort = 255;
    int numeroInt = 1024;
    long numeroLong = 12000;
    float numeroFloat = 100.34f;
    double numeroDouble = 4000.23;

    System.out.println("Numero byte: "+numeroByte);
    System.out.println("Numero short: "+numeroShort);
    System.out.println("Numero Int: "+numeroInt);
    System.out.println("Numero Long: "+numeroLong);
    System.out.println("Numero Float: "+numeroFloat);
    System.out.println("Numero Double: "+numeroDouble);

}
```

Que genera la siguiente salida

```
Hola mundo desde Programación Visual
Numero byte: 127
Numero short: 255
Numero Int: 1024
Numero Long: 12000
Numero Float: 100.34
Numero Double: 4000.23
```

Ahora se genera un ejemplo donde se muestra los valores del primitivo boolean y de un String que es un objeto.

```java
public static void main(String[] args) {
    // genera un mensaje por consola
    System.out.println("Hola mundo desde Programación Visual");
    // el tipo de datos primitivo boolean

    boolean isPositive = false;
    boolean isNegative = true;

    // el tipo de datos String

    String nombre = "Juan Marcelo";

    System.out.println("isPositive: "+isPositive);
    System.out.println("isNegative: "+isNegative);
    System.out.println("Nombre: "+nombre);
}
```

Que genera una salida por consola

```
Problems  @ Javadoc  Declaration  Console
<terminated> Principal [Java Application] C:\Users\ariel\Downl
Hola mundo desde Programación Visual
isPositive: false
isNegative: true
Nombre: Juan Marcelo
```

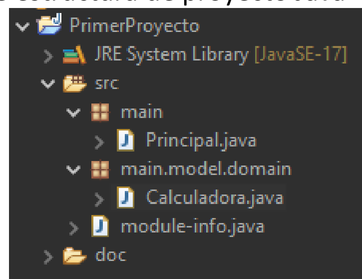
Ahora se genera un ejemplo donde se muestra los valores del primitivo boolean y de un String que es un objeto.

La creación de objetos

En Java, las instancias de las clases; es decir los objetos, se manejan a través de referencias. Para crear estas referencias se utiliza el operador `new` en combinación con el ***constructor*** de la clase involucrada.

Un constructor es una porción de código que tiene por objetivo inicializar un objeto cuando se crea. Tiene el mismo nombre que su clase y es sintácticamente similar a un método. Sin embargo, los constructores no tienen un tipo de devolución explícito (esto quiere decir que el constructor no devuelve ningún valor). Dentro del cuerpo de un constructor se coloca el código que se desea que se ejecute al momento de crear el objeto.

Observe el siguiente ejemplo de estructura de proyecto Java



Como puede observar tenemos dos clases, ubicados en paquetes diferentes. Ahora observe la clase Calculadora

```java
package main.model.domain;

public class Calculadora {

    public Calculadora() {
        System.out.println("Se ha creado el objeto");
    }

    public void prenderCalculadora() {
        System.out.println("Se ha prendido la computadota");
    }

}
```

Si tiene que distinguir si se ha definido un constructor, puede notar que la respuesta es afirmativa porque el constructor tiene el mismo nombre de la clase, además no tiene tipo de retorno. En cambio prenderCalculadora() es un método. Fíjese que es similar a la forma en que se ha definido el método main() en la clase Principal.

Ahora observemos la clase Principal

```java
1  package main;
2
3  import main.model.domain.Calculadora;
4
5  /**
6   * @author Ariel Vega
7   * @version 1.0
8   * Contiene el punto de arranque de la aplicación
9   */
10 public class Principal {
11
12     /**
13      * Es el método que usa Java como punto de partida de ejecución
14      * @param args parámetros que puede recibir el método
15      */
16     public static void main(String[] args) {
17         // se crea una instancia de un objeto Calculadora
18         Calculadora unaCalculadora = new Calculadora();
19     }
20
21 }
```

En la línea 3, hay una sentencia import. Esta sentencia es necesaria cuando se requiere una clase que no está en el mismo paquete. Como se dijo anteriormente, Principal esta en el paquete main, mientras que Calculadora está en el paquete model.domain. Dado que en la línea 18 se está creando un objeto de la clase Calculadora, Java necesita saber donde se encuentra esa clase. Esto se logra con la sentencia import.

Si ejecutamos esta aplicación obtendremos lo siguiente:

```
Problems  @ Javadoc  Declaration  Console
<terminated> Principal [Java Application] C:\Users\ariel\Downl
Se ha creado el objeto
```

Observe que en la consola se muestra el mensaje “Se ha creado el objeto” que es código que se escribió dentro del constructor de la clase Calculadora.

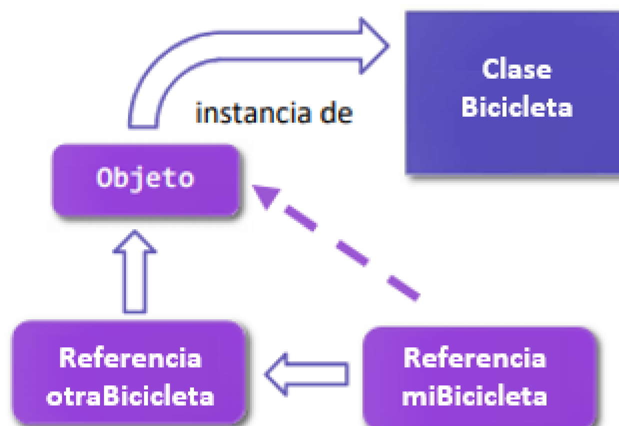
Todas las clases tienen constructores, ya sea que el programador le defina uno o no, porque Java proporciona automáticamente un constructor predeterminado. Una clase puede tener más de un constructor. Más adelante se profundizará en estos aspectos referidos a los denominados ***constructores sobrecargados***.

Una particularidad interesante del lenguaje es que permite tener varias referencias al mismo objeto. Preste atención al siguiente fragmento de código:

```
1  package ar.edu.unju.fi.appbici;
2
3  import ar.edu.unju.fi.appbici.model.domain.Bicicleta;
4
5  public class Principal {
6      public static void main(String[] args) {
7          System.out.println("Hola mundo con JAVA");
8          //creación del objeto unaBicicleta
9          Bicicleta unaBicicleta = new Bicicleta();
10         Bicicleta otraBicicleta = new Bicicleta();
11         Bicicleta miBicileta = otraBicicleta;
12     }
13 }
```

Tanto ***unaBicicleta*** como ***otraBicicleta*** son referencias a dos espacios de memoria diferentes debido a que para cada una de ellas se utilizó el constructor por defecto.

En cambio, en la línea 11 se declara la referencia ***miBicicleta*** a la que se le asigna la referencia ***otraBicicleta***. En este caso no se crea un nuevo objeto en memoria, sino que tanto como ***otraBicicleta*** hacen referencia al mismo objeto en memoria, tal como se observa en la siguiente imagen



Como *otraBicicleta* y *miBicicleta*, hacen referencia a la misma instancia, los cambios sobre el objeto se pueden realizar a través de cualquiera de ellas. Esto se tornará más evidente cuando se profundice en los conceptos de atributos y métodos, momento en el que se volverá sobre este ejemplo para ver el efecto que tiene referenciar un mismo objeto con varias variables.

Los métodos

Como se mencionara anteriormente, los métodos son las secciones de una clase destinadas a la codificación de los algoritmos. Por lo tanto, el objetivo de un método es ejecutar un algoritmo o porción de código.

La estructura general de método Java es la siguiente:

```
[especificadores] tipoDevuelto nombreMetodo([lista parámetros]) [throws listaExcepciones]
{
   // instrucciones
  [return valor;]
}
```

Donde:

- Especificaciones: opcional. Permite indicar el modificador de acceso del método, el cual podrá optar entre los valores [`public |private |protected |` (si no se coloca nada=`default)`]. Normalmente se utilizará el valor `public`, el cual indicará que el método estará disponible para ser utilizado por cualquier entidad que lo requiera.
- tipoDevuelto: este valor es obligatorio. Permite indicar si el método devolverá un valor o no. En Java si se desea que el método devuelva un valor se debe indicar el tipo de dato que ha de devolver (primitivo o tipo referencia). En caso de que el método no deba devolver un valor se debe explicitar mediante la palabra reservada `void`.
- nombreMetodo: es el nombre que se le da al método. Refleja explícitamente "**que**" hará el algoritmo codificado dentro del método.
- Lista de parámetros (opcional): después del nombre del método y siempre entre paréntesis puede aparecer una lista de parámetros separados por comas. Estos parámetros representan los datos de entrada que recibe el método. Un método puede no recibir parámetros. Se debe especificar para cada parámetro su tipo y asignarle un nombre. Los paréntesis son obligatorios, aunque el método no requiera parámetros.
- throws listaExcepciones (opcional): indica las excepciones que puede generar y manipular el método. Sobre las excepciones se dedicará un capítulo especial más adelante.
- Instrucciones: todo aquello que se encuentra delimitado por las llaves de inicio { y fin } del método se denomina cuerpo del método, y se utiliza para escribir las sentencias o instrucciones que ejecuta el método.
- `return`: se utiliza para devolver un valor. La palabra clave `return` va seguida de una expresión que será evaluada para saber el valor de retorno. Esta expresión puede ser compleja o puede ser simplemente el nombre de un objeto, una variable de tipo primitivo o una constante. El tipo del valor de retorno debe coincidir con tipoDevuelto (que se ha indicado en la declaración del método). Si el método no devuelve nada (tipoDevuelto = `void`) no se requiere la instrucción `return`. Un método puede devolver un tipo primitivo o un tipo referencia.

Un método tiene un único punto de inicio, representado por la llave de inicio {. La ejecución de un método termina cuando se llega a la llave final } o cuando se ejecuta la instrucción `return`.

La instrucción `return` puede aparecer en cualquier lugar dentro del método, no tiene que estar necesariamente al final.

Suponga que se tiene las siguientes definiciones de métodos

```
1  package main.model.domain;
2
3  public class Calculadora {
4
5      public Calculadora() {
6          System.out.println("Se ha creado el objeto");
7      }
8
9      // método sin tipo de retorno y sin parámetros
10     public void prenderCalculadora() {
11         System.out.println("Se ha prendido la computadota");
12     }
13
14     // método con tipo de retorno y dos parámetros
15     public int sumar(int numA, int numB) {
16         return numA+numB;
17     }
18
19 }
```

En la clase Calculadora tenemos un constructor y dos métodos. El ámbito de la clase inicia en la línea 3 (por la llave {) y finaliza en la línea 19 (por la llave }). En la línea 5 se define el constructor. En la línea 15 se define el segundo método. Observe que en esa definición:

- Especificaciones: el modificador de acceso es `public`
- tipoDevuelto: el tipo de retorno es `int`. Es decir, debe devolver un valor numérico entero, representado por el tipo primitivo `int`.
- nombreMetodo: el método tiene por nombre ***sumar***. El nombre indica unívocamente cual es la función de ese método (sumar dos números). Observe también, que hay una relación entre el nombre de la clase y el nombre del método. Entre las acciones que puede realizar una calculadora se halla aquella que posibilita sumar números.
- Lista de parámetros: observe que en este caso el método sumar recibe dos parámetros: *numA* y *numB*, ambos de tipo `int`. Los valores de estos parámetros son los que sumará la calculadora.
- `return`: observe que se ha usado la palabra reservada `return`. A la derecha de esta se ha realizado la suma de los parámetros. Por tanto, `return` hará que el método devuelva el resultado de esa suma. Observe además que al sumar dos números enteros se obtiene otro valor de tipo entero. Este valor es el que se devuelve y debe obligatoriamente coincidir con el tipo de datos indicado por el modificador de acceso (tipoDevuelto)

También observe el ámbito del método: siempre se define dentro del ámbito de una clase (sino Java lo interpretará como un error [no pueden definirse métodos fuera de una clase]) y un método tiene su propio ámbito delimitado por las llaves del método (por tanto el método inicia en la línea 15 con la llave { y finaliza en la línea 17 con la llave }).

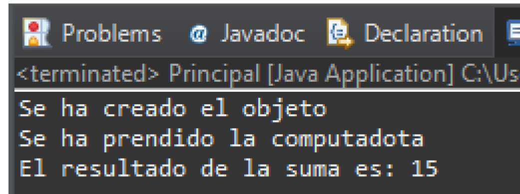
Para poder usar estos métodos se procede como se observa en la siguiente imagen

```
public static void main(String[] args) {
    // se crea una instancia de un objeto Calculadora
    Calculadora unaCalculadora = new Calculadora();

    // se invoca el método prenderCalculadora
    unaCalculadora.prenderCalculadora();

    int numero = 10;
    int resultadoSuma = unaCalculadora.sumar(5, numero);
    System.out.println("El resultado de la suma es: "+resultadoSuma);
}
```

Si ejecutamos este código obtendremos



Para invocar un método se coloca el nombre del objeto, seguido de un punto (.) y a continuación se indica el nombre del método. Si el método requiere argumentos, se los indica en ese comento (como es el caso del método sumar que requiere dos argumentos). En ese caso, los argumentos pueden ser valores literales (ejemplo 5 para el primer argumento) o variables, e incluso el resultado de invocar otros métodos.

NOMENCLATURA

Nomenclatura para la definición de nombres de clases

1) Los nombres de las clases deben ser sustantivos en Singular.
2) La primera letra del nombre de la Clase debe ser escrita en mayúscula.
3) El resto de la palabra debe ser escrita en minúscula.
4) En caso de conformar el nombre de la clase con varias palabras se indica la distinción de cada una de ellas con letra en mayúscula.

Nomenclatura para la definición de nombres de métodos

1) Los nombres de los métodos deben ser verbos en infinitivo (terminación ar, er o ir).
2) La primera letra del nombre del método debe ser escrita en minúscula.
3) El resto de la palabra debe ser escrita en minúscula.
4) En caso de conformar el nombre del método con varias palabras se indica la distinción de cada una de ellas con su primera letra en mayúscula.

Nomenclatura de variables en métodos

Cuando vayamos a dar un nombre a una variable deberemos de tener en cuenta una serie de normas. Es decir, no se admite como buena práctica de programación asignar cualquier nombre a una variable:

- Los identificadores son secuencias de texto unicode, sensibles a mayúsculas cuyo primer carácter solo puede ser una letra, número, símbolo $ o subrayado _. En la práctica el símbolo $ no es utilizado por convención.
- Es recomendable que los nombres de los identificadores sean legibles y no acrónimos para evitar que no se puedan interpretar o que den lugar a una ambigüedad de interpretación al leerlos. Esto facilitará su documentación y auto documentación. Además, estos identificadores nunca podrán coincidir con las palabras reservadas.
- Algunas reglas no escritas, pero que se han asumido por convención son:
  1. Los identificadores siempre se escriben en minúsculas. (ej. nombre). Y si son dos o más palabras, el inicio de cada siguiente palabra se escriba en mayúsculas (ej. nombrePersona)
  2. Si el identificador implica que sea una constante, dicho nombre se suele escribir en mayúsculas (ej. LETRA). Y si la constante está compuesta de dos palabras, estas se separan con un subrayado (ej. LETRA_PI).

FLUJO DE EJECUCIÓN DE METODOS

Ejemplo: Cree una clase denominada Rectángulo con un método ***calcularArea()*** que recibe como parámetros la base y la altura. Además:

$$base = numeroA + numeroB; \quad altura = numeroA - numeroB$$

```
1  package demoJSE;
2
3  public class Calculadora {
4
5      // metodo que permite sumar dos números
6      public int sumar(int numero1, int numero2) {
7          return numero1+numero2;
8      }
9
10     // método que permite restar dos números
11     public int restar(int numero1, int numero2) {
12         return numero1-numero2;
13     }
14
15 }
```

Definición de la clase Calculadora en el paquete demoJSE
Posee dos métodos *sumar*() y *restar*()
Ambos métodos poseen dos parámetros de tipo entero.
Ambos parámetros devuelven como resultado una variable de tipo entero que almacena los resultados.

```
1  package demoJSE;
2
3  public class Rectangulo {
4
5      public int calcularArea(int base, int altura) {
6          return base * altura;
7      }
8
9  }
```

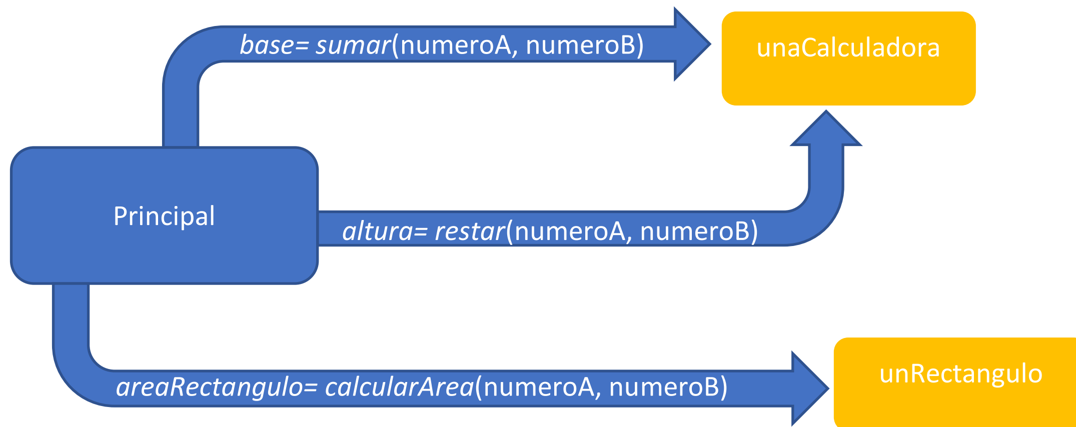
Definición de la clase Rectángulo en el paquete demoJSE.
Posee un método denominado calcularArea().
El método recibe dos parámetros de tipo entero: base y altura.
El método devuelve el producto (operador *) entre base y altura (definición del área de un rectángulo).

El programa arranca por el método main() de la clase Principal

```
3  public class Principal {
4
5      public static void main(String[] args) {
6          // creamos e inicializamos dos variables de tipo entero
7          int numeroA = 5, numeroB=3;
8
9          // creamos un objeto de tipo Calculadora
10         Calculadora unaCalculadora = new Calculadora();
11
12         // se utilizan los servicios de la Calculadora para
13         // obtener la base y la altura del rectangulo
14         int base = unaCalculadora.sumar(numeroA, numeroB);
15         int altura = unaCalculadora.restar(numeroA, numeroB);
16
17         // creamos un objeto de tipo Rectangulo
18         Rectangulo unRectangulo = new Rectangulo();
19
20         int areaRectangulo = unRectangulo.calcularArea(base, altura);
21
22         // se muestra el resultado por consola
23         System.out.println(areaRectangulo);
24     }
25
26 }
```

En este ejemplo se crean dos variables enteras con sus respectivos valores de inicialización.
Luego se crea un objeto de tipo `Calculadora` (línea 10).
Observe que tanto la base como la altura se obtienen usando los métodos *sumar*() y *restar*() del objeto `unaCalculadora` (requisito del problema).
El resultado de cada uno de estos métodos se guarda en las variables `base` y `altura` respectivamente.
En la línea 18 se crea un nuevo objeto de tipo `Rectángulo`.
Se utiliza este objeto para calcular el área del rectángulo invocando el método *calcularArea*() con los argumentos representados por las variables `base` y `altura`.
Finalmente se muestra el resultado por consola.

El funcionamiento anterior se puede resumir mediante esquema



La figura es un ejemplo típico de la manera en que se organiza la programación orientada a objetos: Un programa principal desde el cual se instancian objetos que "colaboran" para alcanzar el objetivo de la aplicación. Esta colaboración consiste en que los objetos solicitarán la ejecución de los servicios de otros objetos y de esa manera cada uno contribuirá a la consecución de la finalidad del programa.

Ámbito de las variables en los métodos

Toda variable tiene un ámbito. Esto es, la parte del código en la que una variable se puede utilizar. De hecho, las variables tienen un ciclo de vida:

1. En la declaración se reserva el espacio necesario para que se puedan comenzar a utilizar (digamos que se avisa de su futura existencia).
2. Se le asigna su primer valor (se inicializa la variable).
3. Se la utiliza en diversas sentencias.
4. Cuando finaliza el bloque en la que fue declarada, la variable muere. Es decir, se libera el espacio que ocupa en memoria.

Una vez que la variable ha sido eliminada, no se puede utilizar. Dicho de otro modo, no se puede utilizar una variable más allá del bloque en el que ha sido definida. El ámbito de las variables está determinado por el bloque donde fueron declaradas y alcanza a todos los bloques que estén anidados dentro de este.

En los métodos el máximo nivel de ámbito para las variables es el cuerpo del método el cual se logra cuando las variables son definidas en la lista de parámetros o cuando son definidas en el cuerpo del método.

Si las variables son definidas dentro de un bloque delimitado por estructuras de control (que se verán más adelante) o por otro conjunto de sub bloques de código (delimitados por { y }) su ámbito es ese bloque de código.

La siguiente figura representa un ejemplo típico donde se puede divisar el ámbito de diferentes variables definidas dentro de un método:

```
public void calcular() {
    int a = 1;   ==> Variable local del método
    {
        System.out.println(a + ", " + numero1);
        int b = 2;   ==> Variable de bloque
        System.out.println(a + ", " + b);
        {
            int c = 3;   ==> Variable de bloque
            System.out.println(a + ", " + b + ", " + c);
        } // Fin del ámbito de c. Final del bloque donde se ha declarado
        System.out.println(a + ", " + b + ", " + c); // esta línea provoca un
                                                      // error de compilación.
                                                      // c esta fuera de su ámbito
                                                      // y por tanto, no declarada
    } //Fin del ámbito de b. Final del bloque donde se ha declarado
} // Fin del ámbito de a. Final del método calcular
```

Nota importante: Si bien Java sigue la modalidad de lenguaje fuertemente tipado, a partir de la versión 8 introdujo el concepto de variables lambda y en Java 9 y 10 donde se usa la sentencia ***var*** y se infiere el tipo, como parte de las nuevas posibilidades de la programación funcional.

Los operadores

Operadores aritméticos

OPERADOR	DESCRIPCIÓN
+	Suma
–	Resta
*	Multiplicación
/	División
%	Resto de una división entre enteros (en otros lenguajes denominado mod)

Cabe destacar que el operador % es de uso exclusivo entre enteros. Por ejemplo:7%3 devuelve 1 ya que el resto de dividir 7 entre 3 es 1. El valor obtenido se denomina módulo (en otros lenguajes en vez del símbolo % se usa la palabra clave mod). Este operador a veces se denomina "operador módulo". Las operaciones con operadores siguen un orden de prelación o de precedencia que determinan el orden con el que se ejecutan. Si existen expresiones con varios operadores del mismo nivel, la operación se ejecuta de izquierda a derecha. Para evitar resultados no deseados, o en casos donde pueda existir duda del orden de ejecución, se recomienda el uso de paréntesis para dejar claro cuál es el orden de ejecución de las operaciones. Por ejemplo, la expresión

$$3 * 2 / 7 + 2$$

Podría generar cierta incertidumbre respecto del orden en el que se ejecutarán las operaciones. Al agregar paréntesis, como, por ejemplo:

$$3 * ((2/7) + 2)$$

No quedan dudas del orden en que se ejecutarán las expresiones. ¿La primera expresión es equivalente a la segunda?

Operadores lógicos y relacionales

En Java se disponen de los operadores relacionales habitualmente presentes en los lenguajes de programación, tales como "es igual", "es distinto", menor, menor o igual, mayor, mayor o igual. También posee los operadores lógicos and (y), or (o) y not (no).

La sintaxis se basa en símbolos como se verá a continuación, y cabe destacar que hay que prestar atención a no confundir = = con = porque implican distintas cosas.

OPERADOR	DESCRIPCIÓN
==	Es igual
!=	Es distinto
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual
&&	Operador and (y)
||	Operador or (o)
!	Operador not (no)

A continuación, se observa un ejemplo donde se utiliza operadores relacionales:

```
boolean mayorDeEdad, menorDeEdad;
int edad = 21;
mayorDeEdad = edad >= 18; //mayorDeEdad será true menorDeEdad = !mayorDeEdad;
//menorDeEdad será false
```

El operador && (AND) sirve para evaluar dos expresiones de modo que, si ambas son ciertas, el resultado será true sino el resultado será false. Ejemplo:

```
boolean carnetConducir=true;
int edad=20;
boolean puedeConducir= (edad>=18) && carnetConducir; //Si la edad es de al
menos 18 años y carnetConducir es //true, puedeConducir es true
```

El operador || (OR) sirve también para evaluar dos expresiones. El resultado será true si al menos una de las expresiones es true. Ejemplo:

```
boolean nieva =true, llueve=false, graniza=false;
malTiempo= nieva || llueve || graniza;
```

Los operadores && y || se llaman operadores en cortocircuito porque si no se cumple la condición de un término no se evalúa el resto de la operación. Por ejemplo:

(a == b && c != d && h >= k)

tiene tres evaluaciones: la primera comprueba si la variable a es igual a b. Si no se cumple esta condición, el resultado de la expresión es falso y no se evalúan las otras dos condiciones posteriores; debido a que el operador && realiza el cortocircuito porque que la única posibilidad de que la operación sea true es que ambos términos sean true. En un caso como:

(a < b || c != d || h <= k)

se evalúa si a es menor que b. Si se cumple esta condición el resultado de la expresión es verdadero y no se evalúan las otras dos condiciones posteriores.

Operadores de asignación

Permiten asignar valores a una variable. El operador fundamental es "=". Por ejemplo: x = 5;

Sin embargo, Java dispone de las siguientes expresiones de asignación más complejas:

Nombre	Operador	Ejemplo	Equivalencia
Operadores aritméticos			
Suma y asignación	+=	a += b	a = a+b
Resta y asignación	-=	a -= b	a = a-b
Multiplicación y asignación	*=	a *= b	a = a*b
División y asignación	/=	a /= b	a = a/b
Resto de la división y asignación	%=	a %= b	a = a%b
Operadores a nivel de bits			
AND binario y asignación	&=	a &= b	a = a&b
OR binario y asignación	|=	a |= b	a = a|b
XOR binario y asignación	^=	a ^= b	a = a^b
Desplazamiento de bits hacia la izquierda en b posiciones y asignación	<<=	a <<= b	a = a<<b
Desplazamiento de bits hacia la derecha en b posiciones y asignación	>>=	a >>= b	a = a>>b

También se pueden concatenar asignaciones. Por ejemplo:

```
x1 = x2 = x3 = 5; // todas valen 5
```

Otros operadores de asignación son "++" (incremento) y "--"(decremento), que incrementan o decrementan en una unidad el valor de la variable. Hay dos formas de utilizar el incremento y el decremento. Se puede usar por ejemplo x++ o ++x, la diferencia estriba en el modo en el que se comporta la asignación en cuanto al orden en que es evaluada. Así, si Ud. escribe el siguiente código generará diferentes asignaciones de valores:

```
int x=5, y=5, z;
z=x++; // z vale 5, x vale 6
z=++y; // z vale 6, y vale 6
```

En el caso del incremento sufijo (x++ en el ejemplo) primero se asigna el valor de x a z, y luego se incrementa en uno x. En el caso del incremento prefijo (++y en el ejemplo) primero se incrementa en uno y, y luego se asigna este valor a z.

Operador ternario

Este operador (conocido como if de una línea) permite devolver un valor u otro según el valor de la expresión analizada. Su sintaxis es la siguiente:

```
Expresionlogica ? valorSiVerdadero: valorSiFalso;
```

Por ejemplo:

```
float precioDeLista = aplicaDescuento == true ? 123 : 234;
```

Si la variable `aplicaDescuento` tiene valor `true`, la variable `precioDeLista` será 123, caso contrario será 234. Es importante destacar que la asignación anterior es equivalente a utilizar condicionales como veremos en las siguientes secciones.

Las constantes

Una constante es una "variable" de solo lectura. Dicho de otro modo, más correcto, es un valor que no puede variar (por lo tanto, no es una variable en sí). La forma de declarar constantes es similar a la utilizada para definir variables, sólo que hay que anteponer la palabra reservada `final` (que es la que indica que estamos declarando una constante). Como no se podrá variar su valor, es requerido inicializar la constante al momento de definirla. Observe el siguiente ejemplo de definición de una constante:

```
final double PI = 3.141591;
```

Una buena práctica comúnmente aceptada consiste en declarar los nombres de las constantes en mayúscula a fin de distinguirlas en el código del programa. Si el nombre de la constante está formado por varias palabras se las separa usando el carácter "_".

ESTRUCTURAS DE CONTROL

Estructuras condicionales

Está conformada esencialmente por dos tipos de estructuras: bifurcación condicional y selección múltiple.

Bifurcación condicional: if-else, if-else-if

Su sintaxis es

```
if (condicion)
{
   instruccion1();
   instruccion2();
   // etc
} else
{
   instruccion1();
   instruccion2();
   // etc
}
```

Es necesario que **condición** sea una variable o expresión booleana. Si sólo existe una instrucción en el bloque, las llaves no son necesarias. No es necesario que exista un bloque *else*, como en el siguiente ejemplo:

```
if (condificion)
{
  bloqueDeInstrucciones();
}

// continua la ejecución del programa
```

Es posible anidar estas estructuras de control, por ejemplo:

```
if (condificion)
{
  if(condicion2)
  {
    bloqueDeInstrucciones();
  }
  else
  {
    bloqueDeInstrucciones();
  }
}
else
{
  if(condicion2)
  {
    bloqueDeInstrucciones();
  }
  else
  {
    bloqueDeInstrucciones();
  }
}
```

Cada cláusula *else* corresponde al último *if* inmediato anterior que se haya ejecutado, es por eso por lo que se debe tener especial consideración de encerrar correctamente entre llaves los bloques para determinar exactamente a qué cláusula corresponde. En este caso, es de especial utilidad "indentar" el código utilizando espacios o tabulaciones como se muestra en los esquemas identificados hasta ahora.

Un ejemplo en particular del uso de esta estructura puede ser la siguiente:

```
final int totalMaterias = 4;
int materiasAprobadas = 2;

if (materiasAprobadas == totalMaterias)
{
  otorgarCertificado();
}
else
{
  continuarCapacitacion();
}
```

Se define una constante denominada `totalMaterias` (¿cumple la nomenclatura recomendada?). Luego se define una variable `materiasAprobadas` y se inicializa con el valor 2. Observe que en esta ocasión se invocará un método denominado *continuarCapacitacion*(). ¿Por qué? `materiasAprobadas` es igual a 2 mientras que el total de materias es igual a 4 (`totalMaterias`=4). Por tanto, cuando se compara si `materiasAprobadas` es igual a `totalMaterias`, el resultado que se obtendrá es false. Por lo tanto, ingresará al bloque definido por el *else*.

Selección múltiple: switch

Su sintaxis es la siguiente:

```
switch (expresion)
{
      case valor1:
          instrucciones();
          break;
      case valor2:
          instrucciones();
          break;
      default:
          instrucciones();
}
```

Cuando se encuentra coincidencia con un `case` se ejecutan las instrucciones a él asociadas hasta encontrar el primer `break`. Si no se encuentra ninguna coincidencia se ejecutan las instrucciones del bloque `default`, el cual es opcional. El `break` no es exigido en la sintaxis y si no se pone, el código se ejecuta atravesando todos los cases hasta que encuentra uno.

Estructuras basadas en bucles

Un bucle se utiliza para repetir la ejecución de un conjunto de sentencias. Se denomina también lazo o loop. El código incluido entre las llaves {} (opcionales si el proceso repetitivo consta de una sola línea), se ejecutará mientras se cumpla unas determinadas condiciones. Hay que prestar especial atención a los bucles infinitos, hecho que ocurre cuando la condición de finalización del bucle (expresión booleana) no se llega a cumplir nunca.

Bucle while

El bucle **while** se utiliza para repetir un conjunto de sentencias siempre que se cumpla una determinada condición. La condición se comprueba al comienzo del bucle, por lo que se podría dar el caso de que dicho bucle no se ejecutase nunca. La sintaxis es la siguiente

```
while (expresion) {

    sentencias

}
```

Ejemplo: Visualice mediante una estructura **while** los 10 primeros números naturales.

```
1  package demoJSE;
2
3  public class EjemploWhile {
4
5      public static void main(String[] args) {
6          int i=1;
7
8          while(i < 11) {
9              System.out.println(i);
10             i++;
11         }
12
13     }
14
15 }
```

Observe que se inicializa la variable `i` en 1. Mientras i sea menor que el número 11, se visualiza su valor en consola y luego se incrementa en uno su valor.

Bucle do-while

El bucle **do-while** funciona de la misma manera que el bucle **while**, con la salvedad de que **expresion** se evalúa al final de la iteración. Las sentencias que encierran el bucle **do-while**, por tanto, se ejecutan como mínimo una vez. La sintaxis es la siguiente:

```
do {

    sentencias

} while (expresion)
```

Este código es el equivalente **do-while** al ejemplo anterior que muestra los valores del 1 al 10.

```
1  package demoJSE;
2
3  public class EjemploDoWhile {
4
5      public static void main(String[] args) {
6          int i=1;
7
8          do {
9              System.out.println(i);
10             i++;
11         }
12         while(i < 11);
13     }
14
15 }
```

Bucle for

Se suele utilizar cuando se conoce previamente el número exacto de iteraciones (repeticiones) que se van a realizar. La sintaxis es la siguiente

```
for (expresion1 ; expresion2 ; expresion3) {

    sentencias

}
```

Al inicio se ejecuta **expresion1**, normalmente se usa para inicializar una variable. El bucle se repite mientras se cumple **expresion2** y en cada iteración del bucle se ejecuta **expresion3**, que suele ser el incremento o decremento de una variable. Con un ejemplo se verá mucho más claro:

```
1  package demoJSE;
2
3  public class EjemploFor {
4
5      public static void main(String[] args) {
6
7          for(int i=1;i<11;i++) {
8              System.out.println(i);
9          }
10
11     }
12
13 }
```

En este ejemplo, **`int i = 1`** se ejecuta solo una vez, antes que cualquier otra cosa dentro del bucle; como puede observar, esta expresión se utiliza para inicializar la variable `i` en 1. Mientras se cumpla la condición `i < 11` el contenido del bucle, o sea, `System`.out.*println*(i); se va a ejecutar. En cada iteración del bucle, `i++` incrementa la variable i en 1. El resultado del ejemplo es la impresión en pantalla de los números 1 al 10. Si sigue mentalmente el flujo del programa,

podría experimentar inicializando la variable `i` con otros valores, cambiando la condición con > o <= y observar lo que sucede. Pruebe también cambiar el incremento de la variable i, por ejemplo, con `i = i + 2` y reflexione los resultados.