Usos de Clases Java

Este capítulo describe cómo escribir declaraciones de clases en el lenguaje de programación Java.

Objetivos

- Escribir declaraciones de clase
- Definir tipos primitivos
- Declarar variables de clase utilizando tipos primitivos
- Declarar variables de clase utilizando tipos referencia
- Dar nombres a clases Java y otros identificadores de acuerdo con las reglas del lenguaje de programación Java
- Utilizar herencia correctamente
- Utilizar clases abstractas
- Utilizar la declaración import para incluir clases en un programa
- Utilizar la declaración package para agrupar clases en una jerarquía
- Entender la estructura de un programa Java
- Definir generalización y especialización y su relación con la herencia
- Definir polimorfismo y explicar cómo la herencia fomenta el polimorfismo
- Definir clases abstractas

Declaración de Clases

Para cada clase de cualquier programa que usted desee escribir, necesita definir una clase en su programa Java. Esto implica especificar la categoría de clase, su nombre, los atributos (variables en el lenguaje de programación Java) y operaciones (métodos en el lenguaje de programación Java).

Declaración de Clases - Sintaxis

[modificador_clase] class identificador_clase {bloque }

- **modificador_clase** es opcional (indicado por los corchetes) y puede ser: public, abstract o final.
- **El modificador public** se utiliza para hacer que la clase sea accesible desde todas las clases sin importar cuál sea el paquete. (Los paquetes son grupos de clases. Se explican más adelante en este capítulo en la sección "Clases y Paquetes")
- **El modificador abstract** será discutido posteriormente en este capítulo.
- Una clase declarada con el **modificador final** está completa y no puede ser subclasificada.

Declaración de Clases - Sintaxis

- Una **clase declarada sin modificador** es la declaración por defecto. Esto significa que es accesible únicamente dentro del paquete donde fue declarada.
- **class** es el texto que indica que se trata de una declaración de clase
- **identificador_clase** es el nombre de la clase y puede ser cualquier identificador legal con algunas restricciones que dependen de que el nombre ya haya sido utilizado dentro del mismo paquete. Las reglas para identificadores están en la sección "Identificadores", más adelante en este capítulo.
- **bloque** es el espacio para las variables y métodos que componen la clase. Variables y métodos son los términos específicos de la tecnología Java ("términos Java") para atributos y operaciones respectivamente. Las llaves ({ }) alrededor del bloque se utilizan para definir el comienzo y el fin de la clase. Si hay más de una variable o de un método en el bloque, se separan por punto y coma.

Declaración de Clases

Caso de Estudio

Escriba el código para una clase Order (Orden) para el programa de ingreso de órdenes de la siguiente manera:

```
public class Order { }
```

Discusión – ¿Cómo escribiría el código para las otras clases?

Variables

Variable es el término Java para los **atributos** **de una clase**. Las variables de las clases definen sus características. Por ejemplo, las variables de una clase Nube podrían ser forma, tamaño, contenido de agua. Cada variable debe consistir de un tipo y un identificador único. El tipo puede ser un tipo primitivo del lenguaje de programación Java ("tipos primitivos Java") como, por ejemplo, un número o un carácter, o puede ser un tipo referencia que refiere a otro objeto .

El identificador puede ser cualquier nombre que cumpla con las reglas de la sección "Identificadores" que aparece, más adelante, en este módulo.

Identificadores

- Los identificadores son nombres que se asignan a las clases, variables, métodos, etc. Cada clase tiene un identificador asociado. Lo mismo sucede con cada método y variable. Las siguientes reglas determinan las estructura de los identificadores:
- El primer carácter de un identificador debe ser alguno de los siguientes:
  - Una letra mayúsucla (A-Z)
  - Una letra minúscula (a-z)
  - El carácter guión bajo (_)
  - El símbolo de pesos ($)

Identificadores

- Los nombres de clases deberían comenzar con una letra mayúscula; los nombres de variables deben comenzar con una letra minúscula.
- El segundo carácter de un identificador y los que le siguen deben ser alguno de los siguientes:
  • Cualquiera de los caracteres enumerados antes.
  • Caracteres numéricos (0-9).
  • Se pueden agregar otros caracteres a la lista, incluso los caracteres con tilde. Esto está determinado por el lenguaje de programación Java.
  • Si usted utiliza dos o más palabras en el identificador, inicie cada palabra subsiguiente con mayúscula. No separe las palabras con guión bajo u otro carácter.
  • No se permite en ningún caso, utilizar una palabra clave de la tecnología Java como un identificador. Estas aparecen enumeradas en el Apéndice B, "Palabras Clave de Java"

Identificadores

Java es un lenguaje de programación case-sensitive. Esto significa que se distingue entre minúsculas y mayúsculas de cada carácter alfabético. El lenguaje de programación Java considera como diferentes la mayúscula y minúscula de una misma letra. Si usted crea una variable llamada "orden", no puede luego referirse a la misma como "Orden".

Tipos Primitivos de Java y Tipos de Referencia

Todas las variables tienen un tipo. Este puede ser un tipo primitivo de Java o un tipo Referencia. Los tipos restringen los valores que una variable puede tener. La sintaxis para la declaración de variables es:

type identificador_variable

Tipos Primitivos de Java y Tipos de Referencia

Panorama General

El lenguaje de programación Java exige que a cada variable y constante, se le asigne un tipo, utilizando *tipos primitivos o tipos referencia*. El tipo de una variable determina qué valores se pueden asignar en esa variable.

Los *tipos primitivos* comprenden tipos para almacenar números enteros (tipos int), números con decimales (tipos de punto flotante), valores lógicos verdadero/falso, etc. Estos tipos constituyen las partes simples de los datos del lenguaje de programación Java.

Los *tipos referencia* indican el tipo de las *variables referencia*, que se usan para guardar la dirección de un objeto. Cuando se crean objetos, estos se almacenan en una área separada de la memoria, pero se necesita recordar la dirección donde fueron creados.

Tipos Primitivos de Java

Los siguientes son los ocho tipos primitivos del lenguaje de programación Java:

Tipos enteros

- byte
- short
- int
- long

Tipos de punto flotante

- float
- double

Tipo de texto

- char

Tipo lógico

- boolean

Tipos Primitivos de Java y Tipos de Referencia

Sintaxis

Para los tipos primitivos y los tipos referencia, para asignar un tipo a una variable, se debe declarar la variable. Una sentencia de declaración de variable tiene la siguiente forma:

tipo identificador_variable

Ejemplo de tipo primitivo:

```
int myFirstVariable;
```

Ejemplo de tipo referencia:

```
Computer laptop;
```

Tipos Primitivos de Java y Tipos de Referencia

Sintaxis

En el ejemplo de tipo primitivo anterior, se crea una variable myFirstVariable cuyo tipo es int. Usted puede asignar un valor a la variable utilizando el operador "=" (asignación)

identificador_variable = valor

```
myFirstVariable = 27;
```

Una vez que usted ha asignado el valor, puede usar el nombre de la variable en cualquier parte que la necesite para utilizar el valor que se ha almacenado en ella.

Referencias a Objetos

Se utiliza para almacenar la dirección de un objeto.

Puede almacenar solo un tipo de referencia: una referencia a un objeto o el tipo de una subclase o superclase.

Referencias a Objetos

Definición:

Los objetos son grandes áreas de memoria que contienen tanto funcionalidades como datos. Los objetos representan ítem discretos, extraídos del problema que se intenta resolver.

Una referencia a un objeto contiene una forma de acceder a él. Comúnmente, esta es la dirección del objeto. Una referencia a objetos puede ser utilizada para referir solamente a un tipo de objetos. Una referencia a objetos que se declara para referir a un objeto Computer no podría ser usada para referir a un objeto User.

Las variables con un tipo primitivo se denominan *variables primitivas*.

Las variables que refieren a objetos se denominan *variables referencia*.

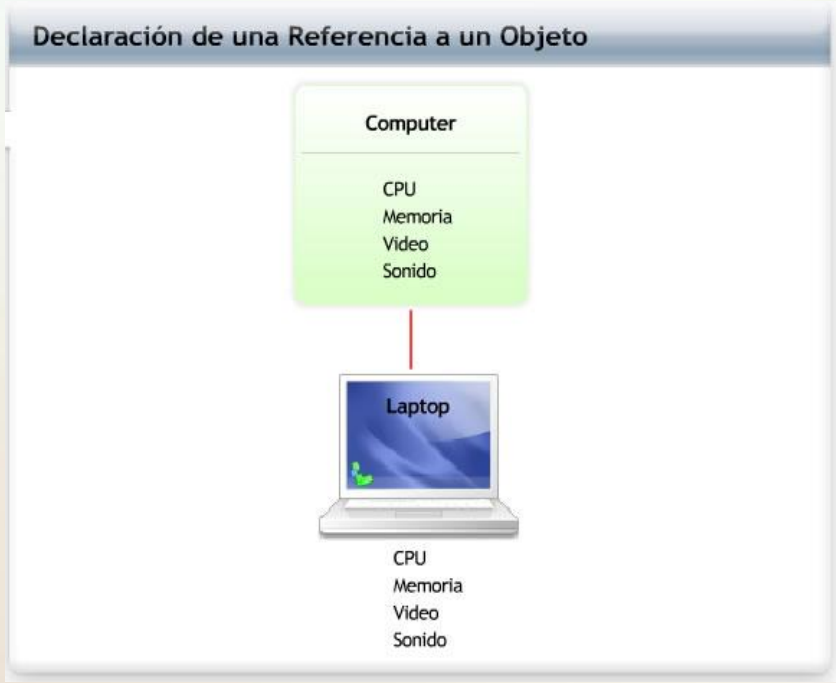
Referencias a Objetos

La declaración de una referencia a un objeto Computer utiliza este formato:

tipo identificador_variable

Computer laptop;

Usted ahora dispone de una variable laptop que puede referir a un objeto Computer.



Referencias a Objetos

Creación de Objetos

Para crear objetos de la clase Computer, utilice la palabra reservada new:

identificador_variable = new nombre_clase()

```
laptop = new Computer();
```

La variable laptop ahora hace referencia a una nueva instancia de Computer.

Referencias a Objetos

Ejemplo:

```
1 int first = 9;
2 int second = first;
3 Computer laptop = new Computer();
4 Computer anotherReferenceToLaptop = laptop;
```

Referencias a Objetos

- Uso de Referencias a Objetos Las referencias a objetos actúan de la misma manera que los ocho tipos primitivos cuando usted asigna valores a ellos.
- Las línea 1-2 crean: una variable entera de nombre first que se usa para almacenar el número 9, una variable entera de nombre second que se usa para almacenar una copia de first . Si posteriormente se cambia el contenido de first, esto no modifica automáticamente el contenido de second.
- La línea 3 crea una referencia llamada laptop y le asigna una referencia a un nuevo objeto Computer. La línea 4 crea otra referencia llamada anotherReferenceToLaptop y le asigna una referencia al mismo objeto Computer almacenado en laptop. Esto no duplica el objeto, sino que copia la forma de acceso. Ahora existirán dos referencias al mismo objeto Computer.

Referencias a Objetos - Strings

Para almacenar un solo carácter, se utiliza el tipo primitivo char. En cambio, para almacenar palabras y oraciones, se utiliza el tipo String que es una referencia a la clase String.

***Método 1: Inicialización usando la palabra clave* new.**

Usted ya sabe como crear objetos de una clase usando la palabra clave new. Para un objeto String esto podría hacerse como sigue:

identificador_variable = new String (“String_value”);

String animal = new String(“dog”);

Referencias a Objetos - Strings

***Método 2: Inicialización sin usar la palabra clave* new.**

String tiene la particularidad de ser la única clase que permite construir objetos sin utilizar la palabra clave new. La sintaxis de esta forma de inicialización es:

String variable = “String_value”;

String animal = “dog”;

Referencias a Objetos

Caso de Estudio

La declaración de la clase Person con las variables definidas, podría ser como se muestra a continuación:

```
1 public class Person
2 {
3   String firstName;
4   String lastName;
5   String initial;
6   String ID;
7   Address homeAddress;
8 }
```

Referencias a Objetos

Caso de Estudio

Discusión – ¿Cómo escribiría usted el código de la declaración de clase y de las variables de la clase Order?

Encapsulación

La encapsulación separa los aspectos externos de un objeto, accesibles a otros objetos, de los detalles internos de la implementación del objeto, que son privados y están ocultos para otros objetos.

Todas las variables en una declaración de clase deberían ser privadas y pueden ser modificadas o accedidas solamente a través de operaciones de la interfaz pública.

Para poder tener acceso a esas variables, se necesita crear operaciones públicas. Las operaciones (denominadas métodos en el lenguaje de programación Java)

Encapsulación

Ejemplo:

```
1 class Person
2 {
3     private String firstName;
4     private String lastName;
5     private String initial;
6     private int ID;
7     private Address homeAddress;
8     public String getName(){return firstName;}
```

Encapsulación

Ejemplo:

```
9    public void setFirstName(String fName){}
10   public void setLastName(String lName){}
11   public void setInitial(String init){}
12   public int getID(){return ID;}
13   public void setID(Sting id){}
14   public Address getAddress(){return homeAddress;}
15    public void setAddress(Address addr){}
16   public int getID() {
17     return ID;
18   }
19
20   public void setDescription(String d) {
```

Encapsulación

```
21    description = d;
22  }
23
24  public String getDescription() {
25    return description;
26  }
27
28  public void setPrice(double p) {
29    price = p;
30  }
31
32  public double getPrice() {
33    return price;
34  }
```

Encapsulación

```
35
36   public void setQuantityInStock(int q) {
37     quantityInStock = q;
38   }
39
40   public int getQuantityInStock() {
41     return quantityInStock;
42   }
43
44 } // end of class
```

Herencia

- Ejemplo: Las clases Manager (Gerente) y Clerk (Vendedor) tienen las características de un Employee (Empleado)
- Los ítems comunes se definen en una clase y las siguientes clases se basan en ella

Herencia

Ejemplo:

Considere una compañía que necesita un sistema de personal para almacenar objetos Gerente (Manager), objetos Ingeniero (Engineer), etc. Los gerentes, ingenieros y otros empleados comparten muchas características, tales como nombre, dirección, número de seguro social, información de impuestos, etc. En lugar de crear estas características por separado en cada uno de los tipos de empleado, el sistema de personal podría hacer que cada tipo de empleado, herede características de una clase Employee.

Herencia

Definición:

Como se explicó en el Capítulo 2, "Clases", la herencia es el mecanismo mediante el cual los miembros comunes (variables y métodos) se definen en una clase y las clases subsiguientes se basan en dichos miembros o los heredan. Esta sección explica cómo implementar la herencia en el lenguaje de programación Java.

Utilización de Herencia

En los siguientes ejemplos se presentan dos declaraciones de clases, usando la sintaxis que se explicó en la sección "Declaración de Clases":

Muchas variables aparecen en ambas clases, lo cual implicaría trabajo extra si la definición de cualquiera de las variables requiriera cambios. Además de las variables mencionadas ¿qué tienen en común las dos clases? Ambas definen objetos que, vistos como parte de un sistema de personal de una compañía, representan empleados de esa compañía. Por lo tanto, heredan características comunes a todos los empleados.

La clase Employee podría ser similar a las dos clases ya vistas.

Se dice que Manager y Engineer extienden (proceso de heredar de) la clase Employee. Utilice la palabra clave extends para indicar herencia en su programa.

La clase Manager(Gerente) puede ser:

```
1 class Manager
2 {
3    int employeeNumber;
4    String name;
5    int departmentNumber;
6    int extensionNumber;
7    int salary;
8    int numberOfWorkers;
9    // etcétera
10 }
```

La clase Engineer(Ingeniero) puede ser:

```
1 class Engineer
2 {
3    int employeeNumber;
4    String name;
5    int departmentNumber;
6    int extensionNumber;
7    int salary;
8    Manager worksFor;
9   // etcétera
10 }
11
```

La clase Employee(Empleado) puede ser:

```
1 class Employee
2 {
3    int employeeNumber;
4    String name;
5    int departmentNumber;
6    int extensionNumber;
7    int salary;
8    // etcétera
9 }
```

Relación de Herencia

```
1 class Manager extends Employee
2 {
3    int numberOfWorkers;
4    // etcétera
5 }
6
7 class Engineer extends Employee
8 {
9    Manager worksFor;
10   // etcétera
11 }
```

Verificación de la Herencia

Según el paradigma OO puro, una clase puede heredar solamente de una superclase a la vez (aunque algunos lenguajes, como C++, permiten que una clase herede de múltiples padres). El lenguaje de programación Java, tal como lo establece el paradigma OO, permite solamente herencia simple.

Dado que cada clase puede heredar los miembros de una única clase, es muy importante considerar el mejor uso de la herencia y utilizarla sólo cuando es completamente válida o inevitable. La forma de verificar si un cierto vínculo de herencia propuesto es válido es la utilización de la frase **"es un/a"** ("un Gerente es un Empleado").

Clases Contenedoras

Ejemplo:

La clase Kitchen puede utilizarse para contener a las clases Stove y Refrigerator. Es usada para contener referencias a objetos relacionados para manejarlos.

Clases Contenedoras

Ejemplo:

Considere una clase Kitchen (Cocina) que puede ofrecer los servicios de ser capaz de cocinar comidas y mantener comidas calientes. Usted, además, desarrolla una clase Stove (Horno) y una clase Refrigerator (Refrigerador). Los objetos de las tres clases están relacionados, pero la herencia no es apropiada puesto que Stove y Refrigerator no comparten miembros con Kitchen. Para permitir que Kitchen controle el funcionamiento de Stove y Refrigerator, se puede utilizar una clase contenedora (en este caso, Kitchen).

Clases Contenedoras

Definición:

Una ***clase contenedora*** se usa para contener o agrupar referencias a objetos relacionados, con el objetivo de manipularlos o crear nuevas funcionalidades.

Clases Contenedoras

```
1 class Stove
2 {
3 // lo que la clase hace
4 }
5 class Refrigerator
6 {
7 // lo que la clase hace
8 }
9 class Kitchen
10 {
11 Stove myStove;
12 Refrigerator myRefrigerator;
13 // etcétera.
14 }
```

Clases Contenedoras

Verificación de Contención

Así como la frase **"es un/a" valida la herencia**, la frase "tiene un/a" valida la contención. La sección anterior sugiere el ejemplo de la figura .

Utilice la frase de validación "La [clase contenedora] **tiene un/a** [clase contenida]." ¿Es sensato decir que "la Cocina tiene un Horno"?

Clases Contenedoras

La clase Person es una clase contenedora ya que contiene la dirección de la persona.

```
1 class Person
2     private String firstName;
3     private String lastName;
4     private String initial;
5     private int ID;
6     private Address homeAddress;
```

Clases Abstractas y Herencia

- Ejemplo: Una clase Drawing contiene métodos para una gran variedad de funcionalidades de dibujo, implementados de forma tal que sean independientes de la plataforma
- Las clases abstractas pueden definir los métodos que deberían existir. Los métodos podrían estar implementados en subclases.
- Las clases y las operaciones abstractas se marcan con la palabra abstract.
- No se puede crear una instancia de una clase abstracta.
- Las subclases deben proveer implementación para todos los métodos abstractos en la superclase.

Clases Abstractas y Herencia

- Recuerde que una clase abstracta define los atributos y operaciones que se deben implementar. No es posible obtener objetos que sean instancias de estas clases.
- Considere una clase Drawing (Dibujo). La clase contiene métodos para varias características de un dibujo, pero estas se deben implementar de una manera independiente de la plataforma. No es posible acceder al hardware de video de la máquina y permanecer aún independiente de la plataforma. La intención es que la clase dibujo defina qué métodos deben existir, pero las subclases especiales dependientes de la plataforma serán las que efectivamente implementan el comportamiento.

Clases Abstractas y Herencia

Una clase como Drawing, que declara la existencia de métodos pero no la implementación así como la implementación de métodos con comportamiento conocido, es una clase abstracta.

Las subclases de una clase abstracta deben proveer una implementación para todos los métodos abstractos de la superclase. De no hacerlo, esas subclases también serán abstractas. Los métodos que se declaran pero no se implementan (esto es, aquellos que no tienen un cuerpo o { }), también se deben marcar como abstractos.

Clases Abstractas y Herencia

```
1 public abstract class Drawing
2 {
3    public abstract void drawDot(int x, int y);
4    public void drawLine(int x1, int y1,
5        int x2, int y2)
6    {
7      // dibujar usando el método drawDot() repetidamente
8    }
9 }
```

Clases y paquetes

Un paquete es un grupo de clases relacionadas. Se pueden importar clases de un paquete y empaquetar clases en un programa.

Un paquete es un agrupamiento de clases relacionadas entre sí. Usted puede importar una clase o varias clases de un paquete para utilizar en sus programa o empaquetar sus propias clases.

Clases y paquetes

Algunos paquetes de Bibliotecas de Clases Java son:

- El paquete **java.lang** contiene varias clases que forman el núcleo del lenguaje, tales como String, Math, Integer y Thread.
- **java.awt** contiene clases que controlan el AWT. Este paquete se utiliza para construir y controlar la interfaz gráfica de la aplicación.
- **java.applet** contiene clases que ofrecen el comportamiento específico de los applets.
- **java.net** contiene clases para ejecutar operaciones relacionadas con redes y tratamiento de sockets y URLs.
- **java.io** contiene clases que tratan con E/S de archivos.
- **java.util** contiene clases de utilidades para tareas tales como generación de números randómicos, definición de propiedades del sistema, procesamiento de fechas y funciones de calendario.

Agrupamiento de Clase en Paquetes

- El lenguaje de programación Java ofrece el mecanismo package como una forma de agrupar clases relacionadas entre sí. Usted puede indicar que las clases que están en un archivo fuente pertenecen a un paquete en particular, utilizando la declaración **package**.
- Empaquetar sus clases puede ser útil cuando se distribuye una aplicación como vendedor. Los paquetes permiten ocultar las clases y métodos sin hacerlos privados. Otros desarrolladores podrán acceder los ítem del paquete sólo si las clases y métodos pertenecen al paquete. Ellos tendrán acceso a las clases y métodos del paquete con sólo utilizar la declaración **import** que se mostró en la sección anterior.
- Utilice la siguiente sintaxis para la declaración de un paquete, que agrupará todas las clases del archivo fuente en el paquete nombrado:

  package nombre_paquete

Estructura del código

Java es un lenguaje de programación de formato libre, por lo que se puede ubicar el código dentro de su programa de cualquier manera que resulte apropiada o lógica. Esta sección explica cuáles son los pocos requisitos para la estructura de un archivo fuente de tecnología Java ("archivo fuente Java").

Estructura del código

Un archivo fuente de lenguaje de programación Java puede contener tres elementos en el siguiente orden:

- Una declaración de paquete (opcional).
- Cualquier cantidad de declaraciones import .
- Definiciones de clases e interfaces.

Estos ítem deben aparecer en ese orden. Esto es, toda declaración import debe preceder a toda definición de clase y, si se utiliza una declaración package, debe preceder tanto a las clases como a los import.

Estructura del código

El archivo fuente puede tener cualquier nombre con la extensión .java.

El nombre del archivo fuente debe coincidir con el nombre de la clase pública en el archivo fuente. Por ejemplo, suponga que un archivo fuente tiene el nombre Car.java. Este debe contener una única declaración de clase pública como la que sigue:

```
public class Car
{
// class statement
}
```

Usted sólo puede tener una clase pública en un archivo fuente, pero tantas clases no públicas como desee.

Estructura del código

El archivo fuente Java siempre se nombra como:

nombre_clase_pública.java.

Si no tiene una clase pública, entonces se puede usar cualquier nombre que no sea un nombre de clase en el archivo fuente. Cuando se compila, el resultado será uno o más archivos llamados nombre_clase.class. Para cada una de las clases en el archivo fuente, se generará un archivo .class durante la compilación. La razón para incluir más de una clase en un archivo fuente tiene que ver con la importación: si se importa un paquete dentro de un archivo fuente, las clases de ese paquete pueden ser usadas por todas las clases del archivo fuente.

Resumen del Capítulo

Discusión – Las siguientes oraciones definen palabras clave o nombres de clases en el lenguaje de programación Java. Indique cuál es la palabra que mejor se ajusta a cada definición.

- Define un bloque que contiene todos los atributos y operaciones.
- Modificador de acceso que asigna acceso a todo el mundo.
- Modificador que restringe el uso de las subclases.
- Tipo de datos primitivo que puede tener valores true o false.
- Tipo de datos primitivo que contiene información de texto.

Resumen del Capítulo

- Tipo de datos para números enteros (el que se usa por omisión)
- Tipos de datos primitivo que se utiliza más frecuentemente para números decimales.
- Palabra clave que reserva espacio e inicializa variables miembros.
- Clase de java.lang que contiene información de texto.
- Palabra clave que permite declarar subclases.
- Palabra clave que define una clase que no puede ser creada.
- Palabra clave que define un código como parte de una biblioteca.
- Palabra clave que carga código desde una biblioteca.

.Bibliografía .

- Migración a la Programación OO
  con tecnología Java
  SL-210 – Guía del estudiante