UNIVERSIDAD NACIONAL
DE LA PATAGONIA
SAN JUAN BOSCO

Profesor: Lic. Renato Mazzanti
JTP: Lic. Gustavo Samec
Aux: Dr. Marcos Zárate

ALGORÍTMICA Y PROGRAMACIÓN II

Unidad 1

INTRODUCCIÓN

Presentación:

- Campus: https://campusvirtual.unp.edu.ar/login/index.php
- Material en la LAN:
  - Windows:
  - \\192.168.16.145\alumnoing\facultadIngenieria\2 - Algorítmica II
  - Linux:

    smb://192.168.16.145/alumnoing/facultadIngenieria/2 - algorítmica II

INTRODUCCIÓN:

- ¿Qué lenguajes de programación son los más utilizados?.
- Un entorno de desarrollo Java típico.
- Rol de Java en el desarrollo de aplicaciones distribuidas cliente/servidor para Internet y la web.
- La historia del lenguaje de diseño orientado a objetos UML
- El estándar de la industria para probar las aplicaciones Java.

INTRODUCCIÓN

- Java es un lenguaje preferido para desarrollar aplicaciones de Web
- Java Standard Edition (Java SE) https://www.oracle.com/java/technologies/java-se-glance.html Implementación de Sun
- JRE (Java Runtime Environment)
- JDK (Java Development Kit)
- JEE (Java Enterprise Edition)) Orientada hacia aplicaciones distribuidas a gran escala y aplicaciones web.
- JME (EEJava Micro Edition) Orientada hacia aplicaciones para dispositivos pequeños limitados en memoria.
- Programación Orientada a Objetos

LENGUAJES DE MÁQUINA, LENGUAJES ENSAMBLADORES Y LENGUAJES DE ALTO NIVEL

- Lenguaje de máquina
  - “lenguaje natural” del componente de computadora
  - Dependiente de la máquina
- Lenguaje assembler
  - Abreviaturas en Inglés que representan operaciónes de computadora
  - Programas traductores (assemblers) lo convienten en lenguaje de máquina
- Lenguaje de Alto nivel
  - Permiten escribir intrucciones más parecidae al lenguaje
    - Contiene operadores matemáticos generalemente
  - El compilador lo convierte a lenguaje de máquina
- Interprete
  - Ejecuta programas en lenguaje de alto nivel sin compilación

HISTORIA DE JAVA

- Java
  - Originalmente para dispositivos inteligentes electrónicos de consumo
  - Luego se usó para crear páginas web con contenidos dinámicos
  - Ahora también se usa para:
    - Desarrollo de aplicaciones empresariales a gran escala
    - Mejorar la funcionalidad del servidor web
    - Proporcionar aplicaciones para dispositivos de consumo (teléfonos celulares, etc.)
    - Programación de tiempo real

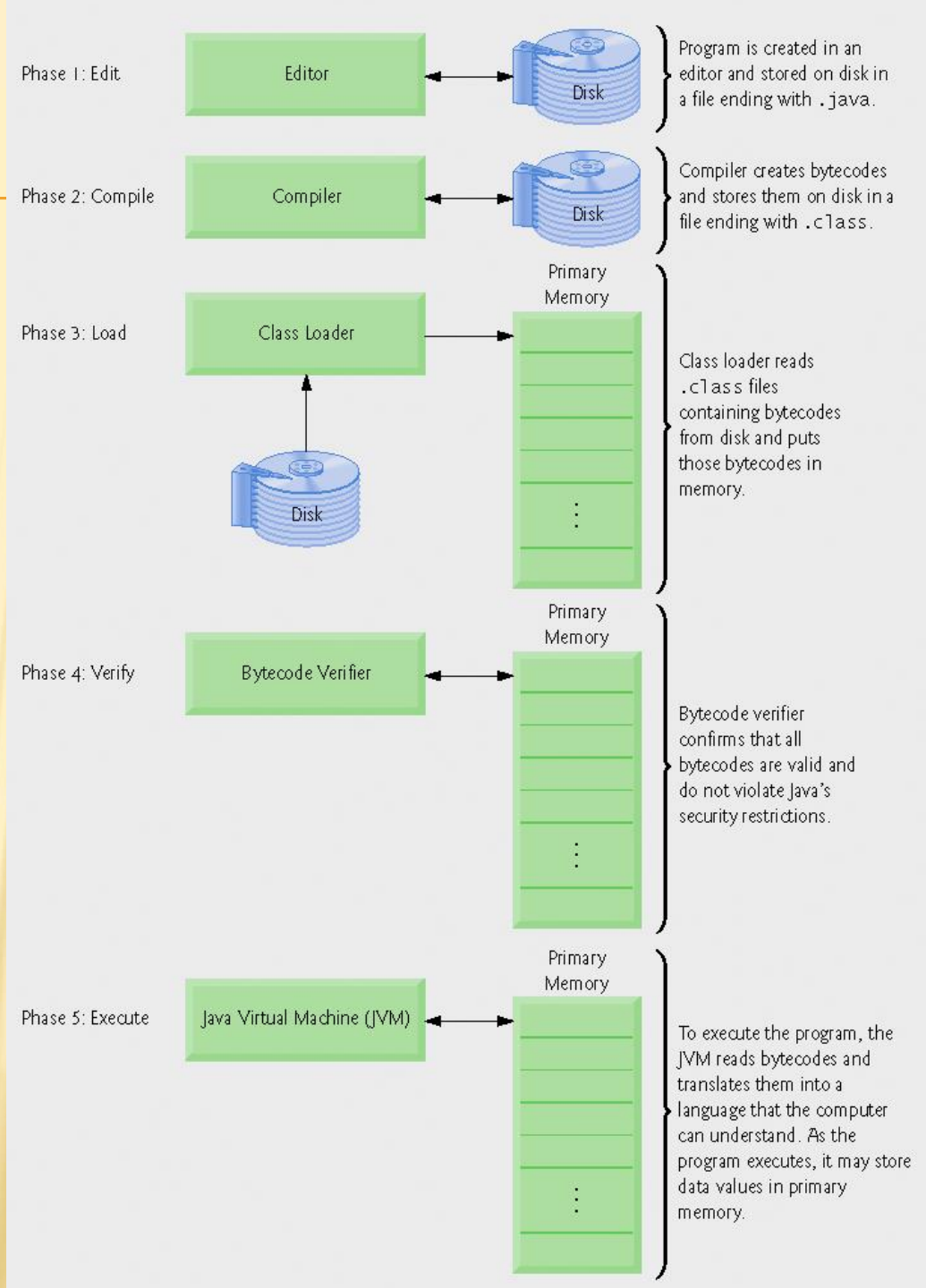
BIBLIOTECAS DE CLASES JAVA

- Los programas Java están compuestos de clases
  - Incluyen métodos que ejecutan las tareas
    - Retornan información luego de completar su tarea
- Java provee bibliotecas de clases
  - Conocidas como Java APIs (Application Programming Interfaces)
- Para utilizar Java con eficacia, debemos saber
  - Lenguaje de programación Java
  - Amplias bibliotecas de clases

TÍPICO AMBIENTE DE DESARROLLO JAVA

- Los programas Java atraviesan cinco fases
  - Edit (Edición)
    - El programador escribe programas usando un **editor**, almacena el programa en disco con extensión de nombre `.java`
  - Compile (Compilación)
    - Se usa **javac** (el compilador Java) para crear "bytecodes" desde el código fuente del programa; bytecodes son almacenados en archivos `.class`
  - Load (Carga)
    - El cargador de Clases lee los bytecodes del archivo `.class` poniéndolo en memoria
  - Verify (Verificación)
    - El verificados de Bytecode examina los bytecodes para asegurarse que ellos sean válidos y que no violen las restricciones de seguridad
  - Execute (Ejecución)
    - Java Virtual Machine (JVM) usa una combinación de interpertación y compilación "just-in-time" para traducir los bytecodes a lenguaje de máquina

TÍPICO AMBIENTE DE DESARROLLO JAVA



INTRODUCCIÓN A LAS APLICACIONES DE JAVA

OBJETIVOS

- Escribir aplicaciones sencillas en Java.
- Usar declaraciones de entrada y salida.
- Los tipos primitivos de Java.
- Conceptos básicos de memoria.
- Utilizar operadores aritméticos.
- La precedencia de los operadores aritméticos.
- Escribir declaraciones de toma de decisiones.
- Utilizar operadores relacionales y de igualdad

PRIMER PROGRAMA EN JAVA: IMPRESIÓN DE UNA LÍNEA DE TEXTO

```
1  // Fig. 2.1: Welcome1.java
2  // Text-printing program.
3
4  public class Welcome1
5  {
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.println( "Welcome to Java Programming!" );
10
11    } // end method main
12
13 } // end clazss Welcome1
```

```
Welcome to Java Programming!
```

COMPILANDO UN PROGRAMA

- Abra una ventana de símbolo del sistema, vaya al directorio donde se almacena el programa
- Tipee ***javac Welcome1.java***
- Si no hay errores de sintaxis, ***Welcome1.class*** se habrá creado
  - Contiene los bytecodes que representan la aplicación
  - Bytecodes se pasarán a la JVM

EJECUCIÓN DE UN PROGRAMA

- Tipee ***java Welcome1***
  - Lanza la ejecución de la JVM
  - JVM carga el archivo `.class` de `Welcome1`
  - La extensión `.class se omite en el comando de linea`
  - JVM llama al método `main`

```java
// Fig. 2.4: Welcome3.java
// Printing multiple lines of text with a single statement.

public class Welcome3
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.println( "Welcome\nto\nJava\nProgramming!" );

   } // end method main

} // end class Welcome3
```

```
Welcome
to
Java
Programming!
```

Escape sequence	Description
\n	Newline. Position the screen cursor at the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor at the beginning of the current line—do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
\\	Backslash. Used to print a backslash character.

\"	Double quote. Used to print a double-quote character. For example,
	`System.out.println( "\"in quotes\"" );`
	displays
	`"in quotes"`

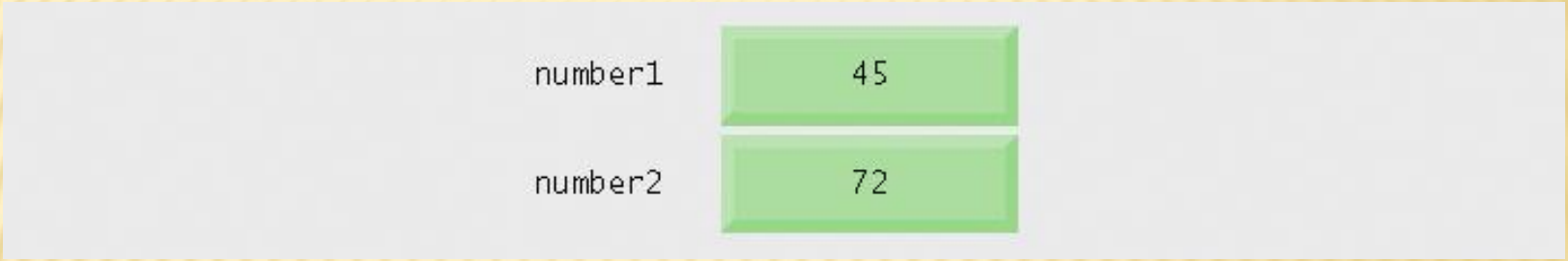
OTRA APLICACIÓN JAVA: SUMAR NÚMEROS ENTEROS

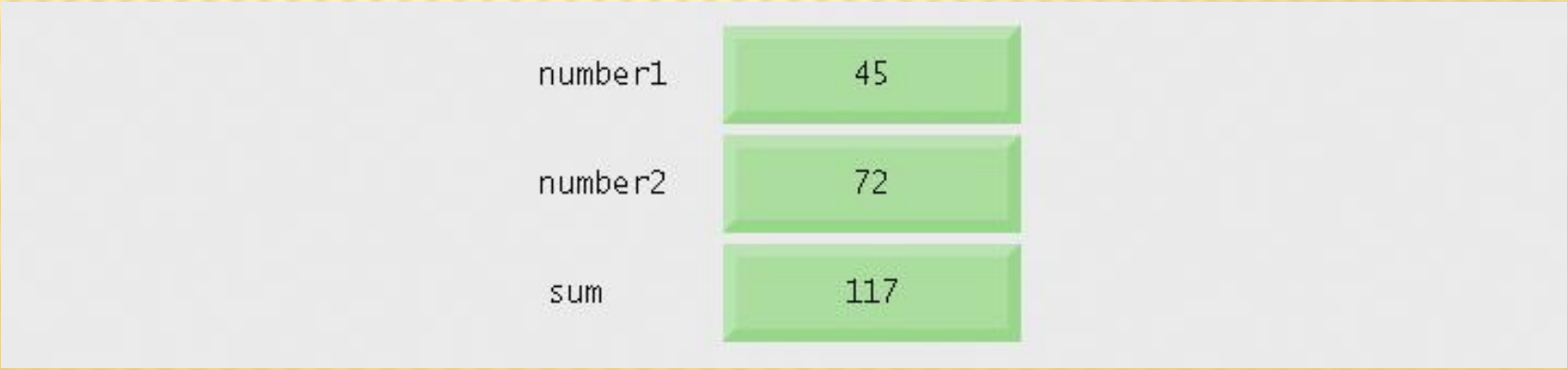
```
1  // Fig. 2.7: Addition.java
2  // Addition program that displays the sum of two numbers.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class Addition
6  {
7     // main method begins execution of Java application
8     public static void main( String args[] )
9     {
10       // create Scanner to obtain input from command window
11       Scanner input = new Scanner( System.in );
12
13       int number1; // first number to add
14       int number2; // second number to add
15       int sum; // sum of number1 and number2
16
17       System.out.print( "Enter first integer: " ); // prompt
18       number1 = input.nextInt(); // read first number from user
19
```

CONCEPTOS DE MEMORIA

- Variables
  - Cada variable tiene un nombre, un tipo, un tamaño y un valor
    - Los nombres se corresponden con la ubicación (location) en memoria
  - Cuando un valor nuevo se ubica en una variable, reemplaza (y destruye) el valor previo
  - La lectura de variables desde la memoria no las cambia

VISUALIZACIÓN DE UBICACIONES DE MEMORIA





ARITMÉTICA

- Cálculos aritméticos utilizados en la mayoría de los programas
  - Uso
    - * para multiplica
    - / para división
    - % para resto
    - +, –
  - División entera truncamiento

    7 / 5 evalúa como 1
  - Operador resto % retorna el resto

    7 % 5 evalúa como 2

Java operation	Arithmetic operator	Algebraic expression	Java expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`

ARITMÉTICA (CONT.)

- Precedencia de Operadores
  - Algunos operadores aritméticos actúan antes que otros (ej., multiplicación antes que adición)
    - Use parentesis cuando es necesario
  - Ejemplo: encu
    a, b y c
    - No use: a
    - Use: (a +

Operator(s)	Operation(s)	Order of evaluation (precedence)
*	Multiplication	Evaluated first. If there are several operators of this type, they are evaluated from left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated next. If there are several operators of this type, they are evaluated from left to right.
-	Subtraction	

TOMA DE DECISIONES: OPERADORES DE IGUALDAD Y RELACIONAL

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

```java
// Fig. 2.15: Comparison.java
// Compare integers using if statements, relational operators
// and equality operators.
import java.util.Scanner; // program uses class Scanner

public class Comparison
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to compare
      int number2; // second number to compare

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

      System.out.print( "Enter second integer: " ); // prompt
      number2 = input.nextInt(); // read second number from user

      if ( number1 == number2 )
         System.out.printf( "%d == %d\n", number1, number2 );

      if ( number1 != number2 )
         System.out.printf( "%d != %d\n", number1, number2 );

      if ( number1 < number2 )
         System.out.printf( "%d < %d\n", number1, number2 );
```

```
31
32        if ( number1 > number2 )
33           System.out.printf( "%d > %d\n", number1, number2 );
34
35        if ( number1 <= number2 )
36           System.out.printf( "%d <= %d\n", number1, number2 );
37
38        if ( number1 >= number2 )
39           System.out.printf( "%d >= %d\n", number1, number2 );
40
41     } // end method main
42
43 } // end class Comparison
```

```
Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777
```

```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

PRECEDENCIA Y ASOCIATIVIDAD DE LAS OPERACIONES DISCUTIDAS

Operators				Associativity	Type
*	/	%		left to right	multiplicative
+	-			left to right	additive
<	<=	>	>=	left to right	relational
==	!=			left to right	equality
=				right to left	assignment

INTRODUCCIÓN A CLASES Y OBJETOS

OBJETIVOS

- Qué son clases, objetos, métodos y variables de instancia.
- Cómo declarar una clase y usarla para crear un objeto.
- Cómo declarar métodos en una clase para implementar los comportamientos de la clase.
- Cómo declarar variables de instancia en una clase para implementar los atributos de la clase.
- Cómo llamar al método de un objeto para que ese método realice su tarea.
- Las diferencias entre las variables de instancia de una clase y las variables locales de un método.
- Cómo utilizar un constructor para asegurarse de que los datos de un objeto se inicializan cuando se crea el objeto.
- Las diferencias entre tipos primitivos y de referencia.

CLASES, OBJETOS, METODOS Y VARIABLES DE INSTANCIA

- La **clase** provee uno o más **métodos**
- El **método** representa la tarea en un programa
  - Describe los mecanismos que realmente realizan sus tareas
  - Oculta de su usuario las tareas complejas que realiza
  - La llamada de método indica el método que realice su tarea
- Las **clases** contienen uno o más atributos
  - Se especifica con variables de instancia
  - Indica como se usa el objeto

DECLARAR UNA CLASE CON UN MÉTODO E INSTANCIAR UN OBJETO DE UNA CLASE

- Cada declaración de clase que comience con la palabra clave *public* debe almacenarse en un archivo que tiene el mismo nombre que la clase y termina con la extensión de nombre de archivo .java.
- keyword *`public`* es un modificador de acceso
- La declaración de Clase incluye:
  - Modificador de acceso
  - Keyword `class`
  - Par de llaves

LA CLASE GRADEBOOK

- La declaración incluye:
  - Modificador de acceso
  - Keyword `class`
  - Par de llaves izquierda y derecha
- Declaración de métodos
  - Keyword *`public`* indica que el método está disponible al público
  - Keyword *`void`* indica que no retorna tipo
  - El modificador de acceso, el tipo retornado, el nombre de método, y los paréntesis comprenden el encabezado del método

CLASE GRADEBOOK

```
1  // Fig. 3.1: GradeBook.java
2  // Class declaration with one method.
3
4  public class GradeBook
5  {
6     // display a welcome message to the GradeBook user
7     public void displayMessage()
8     {
9        System.out.println( "Welcome to the Grade Book!" );
10    } // end method displayMessage
11
12 } // end class GradeBook
```

CLASE GRADEBOOKTEST

- Java es extensible
  - Los programadores pueden crear nuevas clases
- Expresión de creación de instancia de clase
  - Keyword *new*
  - Luego el nombre de la clase a crear y paréntesis
- Llamando a un método
  - Nombre del Objeto, luego el punto separador (**.**)
  - El nombre de método y paréntesis

```java
// Fig. 3.2: GradeBookTest.java
// Create a GradeBook object and call its displayMessage method.

public class GradeBookTest
{
   // main method begins program execution
   public static void main( String args[] )
   {
      // create a GradeBook object and assign it to myGradeBook
      GradeBook myGradeBook = new GradeBook();

      // call myGradeBook's displayMessage method
      myGradeBook.displayMessage();
   } // end main

} // end class GradeBookTest
```

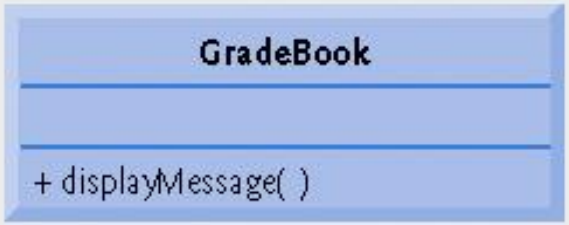
```
Welcome to the Grade Book!
```

COMPILACIÓN DE UNA APLICACIÓN CON VARIAS CLASES

- Compilando múltiples clases
  - Listar todos los archivos `.java` en el comando de compilación separados por espacios
  - Compilar con `*.java` para compilar todos los archivos `.java` en ese directorio (carpeta)

DIAGRAMA DE CLASE UML PARA LA CLASE GRADEBOOK

- Diagramas de clase UML (Unified Modeling Languaje)
  - El compartimiento superior contiene el nombre de la clase
  - Compartimiento medio contiene atributos de clase o variables de instancia
  - El compartimento inferior contiene operaciones o métodos de la clase



DECLARACIÓN DE UN MÉTODO CON UN PARÁMETRO

```
1  // Fig. 3.4: GradeBook.java
2  // Class declaration with a method that has a parameter.
3
4  public class GradeBook
5  {
6     // display a welcome message to the GradeBook user
7     public void displayMessage( String courseName )
8     {
9        System.out.printf( "Welcome to the grade book for\n%s!\n",
10          courseName );
11    } // end method displayMessage
12
13 } // end class GradeBook
```

```java
// Fig. 3.5: GradeBookTest.java
// Create GradeBook object and pass a String to
// its displayMessage method.
import java.util.Scanner; // program uses Scanner

public class GradeBookTest
{
   // main method begins program execution
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      // create a GradeBook object and assign it to myGradeBook
      GradeBook myGradeBook = new GradeBook();

      // prompt for and input course name
      System.ou
      String na
      System.ou

      // call m
      // and pa
      myGradeBook.displayMessage( nameOfCourse );
   } // end main

} // end class GradeBookTest
```


```
Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

NOTAS SOBRE LA DECLARACIÓN `IMPORT`

- `java.lang` es implícitamente importado dentro de cada programa
- Paquete predeterminado (Default package)
  - Contiene clases compiladas en el mismo directorio
  - Importado implícitamente dentro del código fuente de otros archivos en el directorio
- Las importaciones son innecesarias si se usan nombres totalmente calificados

VARIABLES DE INSTANCIA, MÉTODOS *SET* Y MÉTODOS *GET*

- Variables declaradas en el cuerpo del método
  - Se llaman variables locales
  - Solo se puede usar dentro de ese método
- Variables declaradas en una declaración de clase
  - Se llaman campos o variables de instancia
  - Cada objeto de la clase tiene una instancia separada de la variable

```java
// Fig. 3.7: GradeBook.java
// GradeBook class that contains a courseName instance variable
// and methods to set and get its value.

public class GradeBook
{
   private String courseName; // course name for this GradeBook

   // method to set the course name
   public void setCourseName( String name )
   {
      courseName = name; // store the course name
   } // end method setCourseName

   // method to retrieve the course name
   public String getCourseName()
   {
      return courseName;
   } // end method getCourseName

   // display a welcome message to the GradeBook user
   public void displayMessage()
   {
      // this statement calls getCourseName to get the
      // name of the course this GradeBook represents
      System.out.printf( "Welcome to the grade book for\n%s!\n",
         getCourseName() );
   } // end method displayMessage

} // end class GradeBook
```

MODIFICADORES DE ACCESO PUBLIC Y PRIVATE

- private keyword
  - Se utiliza para la mayoría de variables de instancia
  - Las variables privadas y los métodos sólo son accesibles a los métodos de la clase en la que se declaran
  - La declaración de variables de instancia *private* se conoce como la forma de ocultar los datos
- Tipo de Retorno
  - Indica el ítem retornado por el método
  - Se declara en el encabezado del método

GRADEBOOKTEST CLASE QUE TESTEA LA CLASE GRADEBOOK

- Valor inicial predeterminado
  - Se proveen para todos los campos no inicializados
  - Iguales a `null` para `Strings`

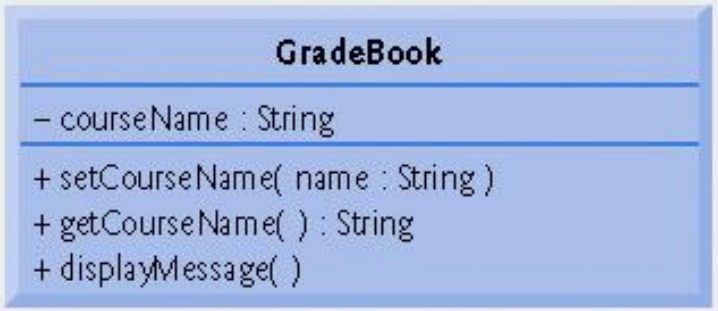
MÉTODOS *SET* Y *GET* METHODS

- `Variables de instancia` *`private`*
  - No se pueden acceder directamente por los clientes del objeto
  - Utilice los métodos *set* para modificar el valor
  - Utilizar métodos *get* para recuperar el valor

```java
// Fig. 3.8: GradeBookTest.java
// Create and manipulate a GradeBook object.
import java.util.Scanner; // program uses Scanner

public class GradeBookTest
{
   // main method begins program execution
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      // create a GradeBook object and assign it to myGradeBook
      GradeBook myGradeBook = new GradeBook();

      // display initial value of courseName
      System.out.printf( "Initial course name is: %s\n\n",
         myGradeBook.getCourseName() );

```

```
20        // prompt for and read course name
21        System.out.println( "Please enter the course name:" );
22        String theName = input.nextLine(); // read a line of text
23        myGradeBook.setCourseName( theName ); // set the course name
24        System.out.println(); // outputs a blank line
25
26        // display welcome message after specifying course name
27        myGradeBook.displayMessage();
28     } // end main
29
30 } // end class GradeBookTest
```

```
Initial course name is: null

Please enter the course name:
CS101 Introduction to Java Programming

Welcome to the grade book for
CS101 Introduction to Java Programming!
```

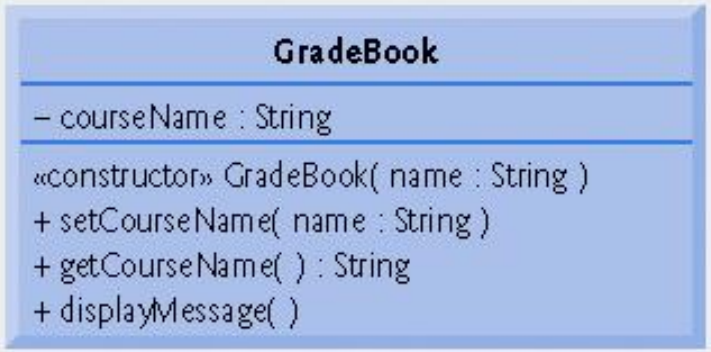


TIPOS PRIMITIVOS VS. TIPO DE REFERENCIA

- Tipos en Java
  - Primitivo
    - `boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`
  - Referencias (a veces llamados tipos no-primitivos)
    - Objetos
    - Valor default `null`
    - Se usan para invocar métodos de objetos

```java
// Fig. 3.10: GradeBook.java
// GradeBook class with a constructor to initialize the course name.

public class GradeBook
{
   private String courseName; // course name for this GradeBook

   // constructor initializes courseName with String supplied as argument
   public GradeBook( String name )
   {
      courseName = name; // initializes courseName
   } // end constructor

   // method to set the course name
   public void setCourseName( String name )
   {
      courseName = name; // store the course name
   } // end method setCourseName

   // method to retrieve the course name
   public String getCourseName()
   {
      return courseName;
   } // end method getCourseName
```

```
25
26    // display a welcome message to the GradeBook user
27    public void displayMessage()
28    {
29       // this statement calls getCourseName to get the
30       // name of the course this GradeBook represents
31       System.out.printf( "Welcome to the grade book for\n%s!\n",
32          getCourseName() );
33    } // end method displayMessage
34
35 } // end class GradeBook
```



INICIALIZACIÓN DE OBJETOS CON CONSTRUCTORES

- Constructores
  - Inicializan un objeto de una clase
  - Java requiere un constructor para cada clase
  - Java proveerá un constructor por default, sin argumentos si no se provee uno
  - Se llama cuando la keyword **new** es seguida por el nombre de una clases seguido por paréntesis

```java
// Fig. 3.11: GradeBookTest.java
// GradeBook constructor used to specify the course name at the
// time each GradeBook object is created.

public class GradeBookTest
{
   // main method begins program execution
   public static void main( String args[] )
   {
      // create GradeBook object
      GradeBook gradeBook1 = new GradeBook(
         "CS101 Introduction to Java Programming" );
      GradeBook gradeBook2 = new GradeBook(
         "CS102 Data Structures in Java" );

      // display initial value of courseName for each GradeBook
      System.out.printf( "gradeBook1 course name is: %s\n",
         gradeBook1.getCourseName() );
      System.out.printf( "gradeBook2 course name is: %s\n",
         gradeBook2.getCourseName() );
   } // end main

} // end class GradeBookTest
```

```
gradeBook1 course name is: CS101 Introduction to Java Programming
gradeBook2 course name is: CS102 Data Structures in Java
```

REQUISITOS DE PRECISIÓN Y MEMORIA DE NÚMERO DE PUNTOS FLOTANTES

- float
  - Números floating-point single-precision
  - Siete dígitos significativos
- double
  - Números floating-point double-precision
  - Quince dígitos significativos

```java
// Fig. 3.13: Account.java
// Account class with a constructor to
// initialize instance variable balance.

public class Account
{
   private double balance; // instance variable that stores the balance

   // constructor
   public Account( double initialBalance )
   {
      // validate that initialBalance is greater than 0.0;
      // if it is not, balance is initialized to the default value 0.0
      if ( initialBalance > 0.0 )
         balance = initialBalance;
   } // end Account constructor

   // credit (add) an amount to the account
   public void credit( double amount )
   {
      balance = balance + amount; // add amount to balance
   } // end method credit

   // return the account balance
   public double getBalance()
   {
      return balance; // gives the value of balance to the calling method
   } // end method getBalance

} // end class Account
```

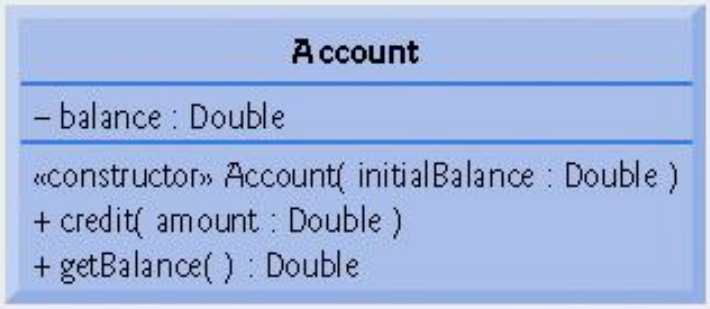
ACCOUNTTEST LA CLASE PARA USAR LA CLASE ACCOUNT

- Especificación de formato %f
  - Se usa para la salida de números floating-point
  - Coloque un punto decimal y un número entre el signo de porcentaje y el f para especificar una precisión

```java
// Fig. 3.14: AccountTest.java
// Create and manipulate an Account object.
import java.util.Scanner;

public class AccountTest
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      Account account1 = new Account( 50.00 ); // create Account object
      Account account2 = new Account( -7.53 ); // create Account object

      // display initial balance of each object
      System.out.printf( "account1 balance: $%.2f\n",
         account1.getBalance() );
      System.out.printf( "account2 balance: $%.2f\n\n",
         account2.getBalance() );

```

```java
// Fig. 3.14: AccountTest.java
// Create and manipulate an Account object.
import java.util.Scanner;

public class AccountTest
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      Account account1 = new Account( 50.00 ); // create Account object
      Account account2 = new Account( -7.53 ); // create Account object

      // display initial balance of each object
      System.out.printf( "account1 balance: $%.2f\n",
         account1.getBalance() );
      System.out.printf( "account2 balance: $%.2f\n\n",
         account2.getBalance() );

```

```
41        // display balances
42        System.out.printf( "account1 balance: $%.2f\n",
43
44
45
46     } /
47
48 } // end class AccountTest
```




```
account1 balance: $50.00
account2 balance: $0.00

Enter deposit amount for account1: 25.53

adding 25.53 to account1 balance

account1 balance: $75.53
account2 balance: $0.00

Enter deposit amount for account2: 123.45

adding 123.45 to account2 balance

account1 balance: $75.53
account2 balance: $123.45
```

MOSTRANDO TEXTO EN UN DIALOG BOX

```
1  // Fig. 3.17: Dialog1.java
2  // Printing multiple lines in dialog box.
3  import javax.swing.JOptionPane; // import class JOptionPane
4
5  public class Dialog1
6  {
7     public static void main( String args[] )
8     {
9        // display a dialog with the message
10       JOptionPane.showMessageDialog( null, "Welcome\nto\nJava" );
11    } // end main
12 } // end class Dialog1
```

VISUALIZACIÓN DE TEXTO EN UN CUADRO DE DIÁLOGO

- Package `javax.swing`
  - Contiene clases para ayudar a crear interfaces gráficas de usuario (GUIs)
  - Contiene la clase `JOptionPane`
    - Declara métodos `static`, `showMessageDialog` para `visualizar un mensaje de diálogo`

INTRODUCCIÓN DE TEXTO EN UN CUADRO DE DIÁLOGO

- Diálogo de entrada
  - Permite a los usuarios ingresar información
  - Se crea usando el método `showInputDialog` de la clase `JOptionPane`

```java
// Fig. 3.18: NameDialog.java
// Basic input with a dialog box.
import javax.swing.JOptionPane;

public class NameDialog
{
   public static void main( String args[] )
   {
      // prompt user to enter name
      String name =
         JOptionPane.showInputDialog( "What is your name?" );

      // create the message
      String message =
         String.format( "Welcome, %s, to Java Programming!", name );

      // display the message to welcome the user by name
      JOptionPane.showMessageDialog( null, message );
   } // end main
} // end class NameDialog
```

INSTRUCCIONES DE CONTROL

OBJETIVOS

- Utilizar técnicas básicas de resolución de problemas.
- Desarrollar algoritmos a través del proceso de refinamiento paso a paso.
- Utilizar las sentencias de selección *if* y *if ... else* para elegir entre acciones alternativas.
- Utilizar la sentencia repetición *while* para ejecutar sentencias en un programa repetidamente.
- Utilizar la repetición controlada y la repetición controlada por centinela.
- Utilizar los operadores de asignación, incremento y decremento.
- Los tipos de datos primitivos.

PSEUDOCÓDIGO DEL ALGORITMO QUE USA UNA REPETICIÓN CONTROLADA POR UN CONTADOR PARA RESOLVER EL PROBLEMA DE LA CLASS-AVERAGE

```
1   Inicializar total en cero
2   Inicializar el Contador de notas en uno
3
4   Mientras el contador de notas sea menor o igual a diez
5       Solicitar al usuario que ingrese la siguiente nota
6       Ingresar la siguiente nota
7       Sumar la nota al total
8       Agregar uno al contador de notas
9
10  Indicar el promedio de la clase como el total de notas dividido diez
11  Imprimir el promedio de la clase
```

```java
// Fig. 4.6: GradeBook.java
// GradeBook class that solves class-average problem using
// counter-controlled repetition.
import java.util.Scanner; // program uses class Scanner

public class GradeBook
{
   private String courseName; // name of course this GradeBook represents

   // constructor initializes courseName
   public GradeBook( String name )
   {
      courseName = name; // initializes courseName
   } // end constructor

   // method to set the course name
   public void setCourseName( String name )
   {
      courseName = name; // store the course name
   } // end method setCourseName

   // method to retrieve the course name
   public String getCourseName()
   {
      return courseName;
   } // end method getCourseName

```

```
28    // display a welcome message to the GradeBook user
29    public void displayMessage()
30    {
31       // getCourseName gets the name of the course
32       System.out.printf( "Welcome to the grade book for\n%s!\n\n",
33          getCourseName() );
34    } // end method displayMessage
35
36    // determine class average based on 10 grades entered by user
37    public void determineClassAverage()
38    {
39       // create Scanner to obtain input from command window
40       Scanner input = new Scanner( System.in );
41
42       int total; // sum of grades entered by user
43       int gradeCounter; // number of the grade to be entered next
44       int grade; // grade value entered by user
45       int average; // average of grades
46
47       // initialization phase
48       total = 0; // initialize total
49       gradeCounter = 1; // initialize loop counter
50
```

```
51        // processing phase
52        while ( gradeCounter <= 10 ) // loop 10 times
53        {
54           System.out.print( "Enter grade: " ); // prompt
55           grade = input.nextInt(); // input next grade
56           total = total + grade; // add grade to total
57           gradeCounter = gradeCounter + 1; // increment counter by 1
58        } // end while
59
60        // termination phase
61        average = total / 10; // integer division yields integer result
62
63        // display total and average of grades
64        System.out.printf( "\nTotal of all 10 grades is %d\n", total );
65        System.out.printf( "Class average is %d\n", average );
66     } // end method determineClassAverage
67
68 } // end class GradeBook
```

FORMULACIÓN DE ALGORITMOS: SENTINEL-CONTROLLED REPETITION

```
1  // Fig. 4.9: GradeBook.java
2  // GradeBook class that solves class-average program using
3  // sentinel-controlled repetition.
4  import java.util.Scanner; // program uses class Scanner
5
6  public class GradeBook
7  {
8     private String courseName; // name of course this GradeBook represents
9
10    // constructor initializes courseName
11    public GradeBook( String name )
12    {
13       courseName = name; // initializes courseName
14    } // end constructor
15
16    // method to set the course name
17    public void setCourseName( String name )
18    {
19       courseName = name; // store the course name
20    } // end method setCourseName
21
22    // method to retrieve the course name
23    public String getCourseName()
24    {
25       return courseName;
26    } // end method getCourseName
27
```

```
28    // display a welcome message to the GradeBook user
29    public void displayMessage()
30    {
31       // getCourseName gets the name of the course
32       System.out.printf( "Welcome to the grade book for\n%s!\n\n",
33          getCourseName() );
34    } // end method displayMessage
35
36    // determine the average of an arbitrary number of grades
37    public void determineClassAverage()
38    {
39       // create Scanner to obtain input from command window
40       Scanner input = new Scanner( System.in );
41
42       int total; // sum of grades
43       int gradeCounter; // number of grades entered
44       int grade; // grade value
45       double average; // number with decimal point for average
46
47       // initialization phase
48       total = 0; // initialize total
49       gradeCounter = 0; // initialize loop counter
50
51       // processing phase
52       // prompt for input and read grade from user
53       System.out.print( "Enter grade or -1 to quit: " );
54       grade = input.nextInt();
55
```

```
56        // loop until sentinel value read from user
57        while ( grade != -1 )
58        {
59           total = total + grade; // add grade to total
60           gradeCounter = gradeCounter + 1; // increment counter
61
62           // prompt for input and read next grade from user
63           System.out.print( "Enter grade or -1 to quit: " );
64           grade = input.nextInt();
65        } // end while
66
67        // termination phase
68        // if user entered at least one grade...
69        if ( gradeCounter != 0 )
70        {
71           // calculate average of all grades entered
72           average = (double) total / gradeCounter;
73
74           // display total and average (with two digits of precision)
75           System.out.printf( "\nTotal of the %d grades entered is %d\n",
76              gradeCounter, total );
77           System.out.printf( "Class average is %.2f\n", average );
78        } // end if
79        else // no grades were entered, so output appropriate message
80           System.out.println( "No grades were entered" );
81     } // end method determineClassAverage
82
83  } // end class GradeBook
```

CASTING Y PROMOCIÓN

- Operador unario cast
  - Crea una copia temporaria de su operando con un tipo de dato diferente
    - ejemplo: `(double)` crearemos una copia temporaria floating-point de su operando
  - Conversión explícita
- Promoción
  - Convierte un valor (ej. `int`) a otro tipo de dato (ej. `double`) para ejecutar un cálculo
  - Conversión implícita

```java
// Fig. 4.10: GradeBookTest.java
// Create GradeBook object and invoke its determineClassAverage method.

public class GradeBookTest
{
   public static void main( String args[] )
   {
      // create GradeBook object myGradeBook and
      // pass course name to constructor
      GradeBook myGradeBook = new GradeBook(
         "CS101 Introduction to Java Programming" );

      myGradeBook.displayMessage(); // display welcome message
      myGradeBook.determineClassAverage(); // find average of grades
   } // end main

} // end class GradeBookTest
```

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of the 3 grades entered is 257
Class average is 85.67
```

FORMULACIÓN DE ALGORITMOS: CONTROL ANIDADO

```
1  Initialize passes to zero
2  Initialize failures to zero
3  Initialize student counter to one
4
5  While student counter is less than or equal to 10
6     Prompt the user to enter the next exam result
7     Input the next exam result
8
9     If the student passed
10       Add one to passes
11    Else
12       Add one to failures
13
14    Add one to student counter
15
16 Print the number of passes
17 Print the number of failures
18
19 If more than eight students passed
20    Print "Raise tuition"
```

```
1  // Fig. 4.12: Analysis.java
2  // Analysis of examination results.
3  import java.util.Scanner; // class uses class Scanner
4
5  public class Analysis
6  {
7     public void processExamResults
8     {
9        // create Scanner to obtain input from command window
10       Scanner input = new Scanner( System.in );
11
12       // initializing variables in declarations
13       int passes = 0; // number of passes
14       int failures = 0; // number of failures
15       int studentCounter = 1; // student counter
16       int result; // one exam result (obtains value from user)
17
18       // process 10 students using counter-controlled loop
19       while ( studentCounter <= 10 )
20       {
21          // prompt user for input and obtain value from user
22          System.out.print( "Enter result (1 = pass, 2 = fail): " );
23          result = input.nextInt();
24
```

```
25          // if...else nested in while
26          if ( result == 1 )           // if result 1,
27             passes = passes + 1;      // increment passes;
28          else                         // else result is not 1, so
29             failures = failures + 1; // increment failures
30
31          // increment studentCounter so loop eventually terminates
32          studentCounter = studentCounter + 1;
33       } // end while
34
35       // termination phase; prepare and display results
36       System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );
37
38       // determine whether more than 8 students passed
39       if ( passes > 8 )
40          System.out.println( "Raise Tuition" );
41    } // end method processExamResults
42
43 } // end class Analysis
```

```java
// Fig. 4.13: AnalysisTest.java
// Test program for class Analysis.

public class AnalysisTest
{
   public static void main( String args[] )
   {
      Analysis application = new Analysis(); // create Analysis object
      application.processExamResults(); // call method to process results
   } // end main

} // end class AnalysisTest
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Raise Tuition
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 6
Failed: 4
```

OPERADORES DE ASIGNACIÓN COMPUESTOS

- Formato de declaración de asignación:
  variable = *variable operador expresión*;
  donde *operador* es +, –, *, / o % y puede ser escrita:
  variable operador= *expresión*;
- ejemplo: `c = c + 3;` se puede escribir como `c += 3;`
  - Adiciona `3` al valor de la `c` y almacena el resultado en la variable `c`

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`C = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

OPERADORES DE INCREMENTO Y DECREMENTO

- Operador de incremento unario (++) adiciona uno a su operando
- Operador de decremento unario (--) substrae uno a su
- Operador de incremento (y decremento) **postfijo**
  - Utiliza el valor actual de su operando en la expresión en la que aparece la operación, luego cambia el valor del operando

Operator	Called	Sample expression	Explanation
++	prefix increment	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	postfix increment	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	prefix decrement	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	postfix decrement	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

```java
// Fig. 4.16: Increment.java
// Prefix increment and postfix increment operators.

public class Increment
{
   public static void main( String args[] )
   {
      int c;

      // demonstrate postfix increment operator
      c = 5; // assign 5 to c
      System.out.println( c );   // print 5
      System.out.println( c++ ); // print 5 then postincrement
      System.out.println( c );   // print 6

      System.out.println(); // skip a line

      // demonstrate prefix increment operator
      c = 5; // assign 5 to c
      System.out.println( c );   // print 5
      System.out.println( ++c ); // preincrement then print 6
```

Operators	Associativity	Type
++ --	right to left	unary postfix
++ -- + - (*type*)	right to left	unary prefix
* / %	left to right	Multiplicative
+ -	left to right	Additive
< <= > >=	left to right	Relational
== !=	left to right	Equality
?:	right to left	Conditional
= += -= *= /= %=	right to left	assignment

TIPOS PRIMITIVOS

- Java es un lenguaje fuertemente tipado
  - Todas las variables tienen un tipo
- Los tipos primitivos en Java son portables en todas las plataformas que soportan Java

BIBLIOGRAFÍA

- DEITEL, PAUL J. Y HARVEY M. DEITEL
- **CÓMO PROGRAMAR EN JAVA.** Séptima edición
- PEARS ON EDUCACION, México 2008
- ISBN: 978-970-26-1190-5