# Generalidades de Java

Prof. Mg. Rafael Mellado S.
COM2161 – Sistemas de Información 1
Escuela de Comercio
Pontificia Universidad Católica de Valparaíso

## Orígenes

- Java es un lenguaje de programación y una plataforma informática comercializada por Sun Microsystems.
- 1991 nace con el nombre "OAK", posteriormente cambiado por Green por problemas legales, y finalmente con la denominación actual JAVA.
- Objetivo de java: Crear un lenguaje de programación parecido a C++ en estructura y sintaxis, fuertemente orientado a objetos, pero con una máquina virtual propia.
- 1995: Se presenta la primera versión de Java.
- 1996: Es lanzado el primer JDK (JDK 1.0). El desarrollo de java a partir de entonces es imparable, se van presentando nuevos paquetes y librerías hasta la actualidad.
- Actualmente se utiliza para la creación de aplicaciones web, android y de escritorio.

46

# Java

- Java permite:
  - Construir **aplicaciones**: programas computacionales que apoyan el trabajo o resuelven problemas específicos de usuarios (funcionan stand-alone).
  - Construir **applets**: programas de menor envergadura que se ejecutan al momento de cargar una página WEB (son ejecutados por un browser).

  - Nota: Las applets al día de hoy no se usan, debido a que existían problemas de seguridad y han sido reemplazadas por otras tecnologías, por ej: HTML5.

47

# Principios

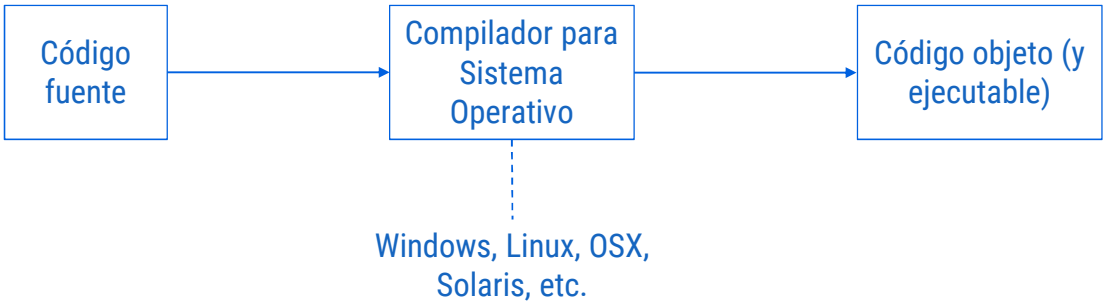

48

# Multiplataforma

- Un programa creado en Java puede ser ejecutado sobre cualquier computador sin importar su S.O (Mac, Unix, Pc)
- Para el código fuente se generan archivos .java
- Cuando se compila un programa en Java desde un archivo fuente (.java) se crean un conjunto de instrucciones que se guardan en un archivo con extensión .class
- El archivo .class es interpretado a lenguaje máquina por la Máquina Virtual de Java lo que permite ejecutar programas independiente del S.O en el que estén corriendo.

49

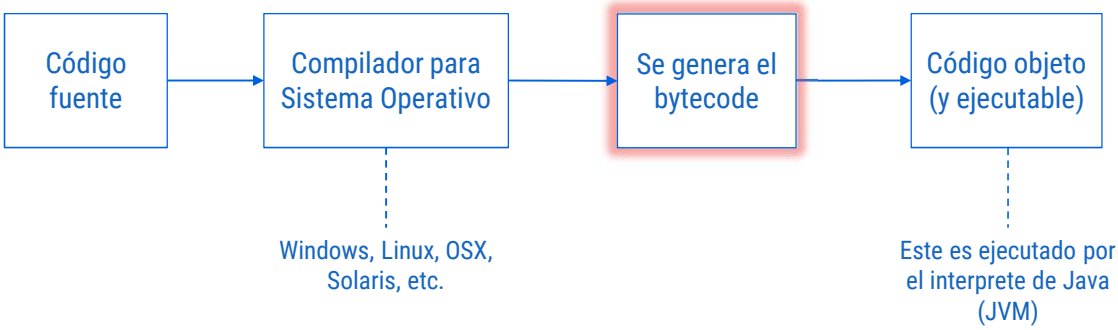
# Desarrollo genérico de aplicaciones




Nota: Se debe crear un código objeto y compilación distinta para cada uno de los sistemas operativos, con sus respectivas restricciones.

50

# Desarrollo de aplicaciones en Java




Nota: En este caso el código fuente es para todos los sistemas operativos igual, ya que es la JVM (Máquina Virtual de Java) la encargada de interpretar el bytecode.

51

# Java en distintos tamaños

JME: Java Micro Edition

JSE: Java Standard Edition


JEE: Java Enterprise Edition

52

# Elementos de aplicaciones simples

Prof. Mg. Rafael Mellado S.
COM2161 – Sistemas de Información 1
Escuela de Comercio
Pontificia Universidad Católica de Valparaíso

## Desarrollo de aplicaciones en Java


- La compilación genera como resultado el bytecode.
- La Máquina Virtual de Java se encarga de realizar el proceso de ejecución, con ello quién desarrolla no se debe preocupar del Sistema Operativo.

54

# Tipos de datos

- Los tipos de datos utilizados por aplicaciones en Java se clasifican en:

**Primitivos**

Sirven para definir variables que guardan valores numéricos, lógicos y caracteres unitarios

**Clases e interfaces**

Sirven para definir variables que almacenan tipos de datos estructurados y las funciones asociadas

55

# Tipos de datos primitivos

- Java soporta los siguientes tipos de datos primitivos:

- **Entero** Ejemplo: 1, 5, 200
- **Punto flotante** Ejemplo: 1.8, 5.0, 200.7
- **Logico** Ejemplo: True, False
- **Caracter** Ejemplo: a, y, z

56

# Tipos de datos primitivos

- Familia de datos enteros: byte, short, int y long

| Tipo | Almacena | Rango |
|---|---|---|
| byte | 8 bits | [-128, 127] |
| short | 16 bits | [-32.768, 32767] |
| int | 32 bits | [-2.147.483.648, 2.147.483.647] |
| long | 64 bits | [-9.223.372.036.854.755.808, 9.223.372.036.854.755.807] |

57

# Tipos de datos primitivos

- Familia de datos punto flotante: float y double

| Tipo | Almacena | Rango |
|---|---|---|
| float | Punto flotante de 32 bit | 7 dígitos significativos ($10^{-46}$,$10^{38}$) |
| double | Punto flotante de 64 bit | 15 dígitos significativos ($10^{-324}$,$10^{308}$) |

58

# Tipos de datos primitivos

- Familia de datos carácter (char) y lógico (boolean)

| Tipo | Almacena | Rango |
|---|---|---|
| char | Carácter Unicode | --- |
| boolean | Lógico | False y true |

59

# Tipos de datos String

- El `String` es un tipo de dato que permite trabajar con cadenas de caracteres, por ejemplo:
  - "Hola mundo"
  - "Mario Mora"
  - "12/10/99" (como texto, en el curso siguiente se conocerán los datos tipo date.)
  - "A"
  - ""
- Es una **clase**, no un tipo **primitivo**, pero se utiliza en forma muy similar a estos últimos. Posteriormente se hablará del concepto de clase.

60

# Operadores básicos

- Operador de asignación:
  - =
- Operadores numéricos:
  - Binarios: + (suma), - (resta), * (multiplicación), / (división), % (módulo o resto de la división entera)
- Unarios:
  - ++ (autoincremento), -- (autodecremento)

61

# Operadores básicos

- Operadores relacionales:
  - == (igual a)
  - != (distinto de)
  - < (menor que), <= (menor o igual que),
  - > (mayor que), >= (mayor o igual que).
- Operadores lógicos:
  - && (AND)
  - || (OR)
  - ! (NOT)

62

## Estructuras de control

- Las estructuras de control determinan la secuencia de ejecución de las sentencias de un programa.
- Las estructuras de control se dividen en tres categorías:
  - Secuencial
  - Condicional o selectiva: `if`, `switch`
  - Iterativa o repetitiva: `while`, `do-while`, `for`.

63

## Construcción de condiciones

- Para la construcción de condiciones se usan los operadores relacionales vistos anteriormente.
- Olvidar los paréntesis al poner la condición del `if` es un error sintáctico (los paréntesis son necesarios).
- No hay que confundir el operador de comparación == con el operador de asignación =.
- Los operadores de comparación `==`, `!=`, `<=` y `>=` han de escribirse sin espacios.

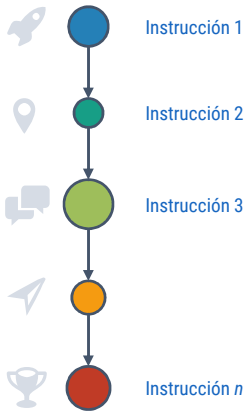
64

# Estructuras de control secuencial

- Las sentencias de un programa se ejecutan en secuencia, es decir, una después de otra, en el orden en que aparecen escritas dentro del programa.
- La estructura secuencial está formada por una sucesión de instrucciones.
- Cada una de las instrucciones están separadas por el carácter punto y coma (;).
- Las instrucciones se suelen agrupar en bloques. El bloque de sentencias se define por el carácter llave de apertura ({) para marcar el inicio del mismo, y el carácter llave de cierre (}) para marcar el final.

65

# Estructuras de control secuencial




```
public class Secuencial
{
    public static void main(String[] args)
      {
        System.out.println("Esta es la primera instrucción");
        System.out.println("Esta es la segunda instrucción");
    }
}
```

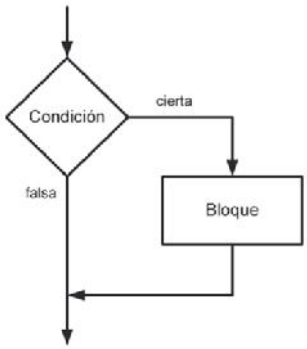
66

# Estructuras de control condicional/selectivo

- La estructura condicional determina si se ejecutan unas instrucciones u otras según se cumpla o no una determinada condición. En java la estructura condicional se implementa mediante:
  - Instrucción if.
  - Instrucción switch.

67

# Instrucción If




- Redirige el curso de acción según la evaluación de una condición simple, sea falsa o verdadera. Si la condición es verdadera, se ejecuta el bloque de sentencias 1; de lo contrario, se ejecuta el bloque de sentencias 2.
- La sentencia condicional if puede ser simple, doble o múltiple.

68

# Instrucción If

- **Simple**: se evalúa la condición y si ésta se cumple se ejecuta una determinada acción o grupo de acciones. En caso contrario se saltan dicho grupo de acciones.

```
public class Condiciones
{
    public static void main(String[] args)
      {
            if(condicion)
            {
                  /*Se ejecutan este bloque de acciones*/
            }
    }
}
```

69

# Instrucción If

- **Doble**: Se evalúa la condición y si ésta se cumple se ejecuta una determinada instrucción o grupo de instrucciones. Si no se cumple se ejecuta otra instrucción o grupo de instrucciones.

```
public class Condiciones
{
    public static void main(String[] args)
    {
            if(condicion)
            {
                  /*Se ejecutan este bloque de acciones*/
            }
            else
            {
                  /*Se ejecutan este bloque de acciones*/
            }
    }
}
```

70

# Instrucción If

- **Múltiple**: Se obtiene anidando sentencias `if-else`. Permite construir estructuras de selección más complejas.

```
public class Condiciones
{
    public static void main(String[] args)
      {
            if(condicion)
            {
                  /*Se ejecutan este bloque de acciones*/
            }
            else
            {
                  if(condicion2)
                  {
                        /*Se ejecutan este bloque de acciones*/
                  }
                  else
                  {
                        /*Se ejecutan este bloque de acciones*/
                  }
            }
    }
}
```

71

# Instrucción switch

- Se utiliza para seleccionar una de entre múltiples alternativas.
- Primero se evalúa la expresión y salta al case cuya constante coincida con el valor de la expresión.
- Se ejecutan las instrucciones que siguen al case seleccionado hasta que se encuentra un break o hasta el final del switch. El break produce un salto a la siguiente instrucción a continuación del switch.
- Si ninguno de estos casos se cumple se ejecuta el bloque default (si existe). No es obligatorio que exista un bloque default y no tiene porqué ponerse siempre al final, aunque es lo habitual.

72

# Instrucción switch

- Los valores de cada caso del switch serán constantes
- En cada caso se finalizará con una sentencia break
- La etiqueta default indica un bloque de código a ejecutarse por defecto cuando al evaluar la expresión no coincide con ningún caso anterior.
- Ejemplo: →

73

# Instrucción switch

```
public class Secuencial
{
    public static void main(String[] args)
    {
        int variable;

        /*Aqui la variable debe sufrir cambios*/

        switch (variable)
        {
            case 1:
                System.out.println("instrucciones en caso en que valga 1");
                break;
            case 2:
                System.out.println("instrucciones en caso en que valga 2");
                break;
            case 3:
                System.out.println("instrucciones en caso en que valga 3");
                break;
            default:
                System.out.println("No tenia los valores anteriores");
        }
    }
}
```

74

# Estructuras repetitivas

- Estructuras de control repetitivas o iterativas son también conocidas como “bucles”.
- Nos permiten ejecutar secciones específicas de código con cierto grado de repetitividad.
- Algunas de ellas se pueden usar un número exacto de veces que deben repetirse las operaciones.
- Otras permiten repetir un conjunto de operaciones mientras se cumpla una condición.
- Llamaremos iteración a cada repetición de las instrucciones de un bucle.

75

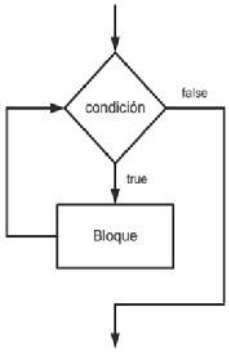
# Instrucción while

- Permite repetir la ejecución de un conjunto de sentencias **mientras** se cumpla una condición:
  - Las instrucciones se repiten mientras la condición sea verdadera (true).
  - La condición se comprueba al principio del bucle por lo que las acciones se pueden ejecutar 0 ó más veces.
  - La ejecución de un bucle `while` sigue los siguientes pasos:
    a. Se evalúa la condición.
    b. Si el resultado es **false** las instrucciones no se ejecutan y el programa sigue ejecutándose por la siguiente instrucción a continuación del cierre del while (se usan las llaves).
    c. Si el resultado es **true** se ejecutan las instrucciones y se vuelve al paso 1.

76

# Instrucción while

- El diagrama de flujo asociado a este tipo de ciclo sería:




77

# Instrucción while

```
public class CicloWhile
{
    public static void main(String[] args)
    {
        int variable=10;

        /*Aqui la variable debe sufrir cambios*/

        while (variable>0)
        {
            System.out.println("Mostramos un mensaje");
            /*Se cambia el valor de la variable*/
            variable++;
        }
             System.out.println("Adios!");
    }
}
```

78

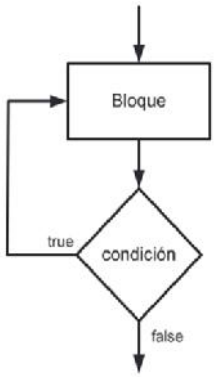
# Instrucción do-while

- Similar al `while` (las instrucciones se ejecutan mientras la condición sea cierta)
- Se realiza la comprobación de la condición después de ejecutar el bucle.
  - La condición se comprueba al final del bucle por lo que el bloque de instrucciones se ejecutarán al menos una vez.
  - La ejecución de un bucle do–while sigue los siguientes pasos:
    a. Se ejecutan las instrucciones a partir de `do{`
    b. Se evalúa la condición.
    c. Si el resultado es false el programa sigue ejecutándose por la siguiente instrucción a continuación del `while`.
    d. Si el resultado es true se vuelve al paso 1.

79

# Instrucción do-while

- El diagrama de flujo asociado a este tipo de ciclo sería:




80

# Instrucción do-while

```
public class cicloDoWhile
{
    public static void main(String[] args)
    {
        int variable=10;

        /*Aqui la variable debe sufrir cambios*/

        do
        {
            System.out.println("Mostramos un mensaje");
            /*Se cambia el valor de la variable*/
            variable--;
        }while (variable>0);
    }
}
```

81

# Instrucción for

- Logra que una instrucción o bloque de instrucciones se repitan un número determinado de veces mientras se cumpla la condición.
- Se puede definir como un “desde (inicio del ciclo), hasta (final del ciclo), paso (incremento o decremento)”.
- Un ciclo for puede ser tanto incremental como decremental.
- En resumen, un ciclo for es una estructura iterativa para ejecutar un mismo segmento de código una cantidad de veces deseada; conociendo previamente un valor de inicio, un  y un valor final para el ciclo.
- El valor inicial, el valor final y paso se separan por ;

82

# Instrucción for

```
public class CicloFor
{
    public static void main(String[] args)
    {
        for(int i=0; i<10; i++)
        {
            System.out.println("Mostramos un mensaje");
        }
    }
}
```

83

# Comentarios y documentación

- Se pueden incluir comentarios en el código usando los siguientes marcadores:

```
//  comentario hasta fin de línea
/* comentario
    de múltiples líneas
*/
/** comentario de documentación
    @author
    @exception
    @see
    @param
    @return
    @version
*/
```

84

# Identificadores

- JAVA es sensible a mayúsculas y minúsculas:
  - Es decir, JAVA considera distinto `Nombre` y `nombre`.
- Un identificador válido consiste en una combinación de de letras, dígitos y el carácter underscore. No puede comenzar por un dígito.
- Ejemplos:
  - Válidos: `edad`, `nota_Alumno`, `_peso`, `estadoSalud`
  - No válidos: `1ero`, `km/hora`, `Alumno(s)`, `peso total`

85

# Declaración de variables

- Una variable se declara según el formato:

```
tipo identificador [=valor], identificador[=valor ];
```

- Por ejemplo:

```
public class declaracionVariables
{
    public static void main(String[] args)
    {
       /*declaración de variables*/
          int a, b, c;

          /*declaración e inicialización*/
          double peso=7.05 ;
          char letra='T';
          byte edad=20, veces=3;
    }
}
```

86

# Declaración de variables

- Convención: el identificador de una variable de tipo primitivo comenzará en minúsculas.
  - Por ejemplo: `edad`, `edadAlumno`, `estadoCivil`, `esElUltimo`.
- La declaración de variables es considerada una instrucción, por ende debe finalizar con punto y coma ( `;` ).
- Toda variable utilizada **deberá ser declarada**, de lo contrario el compilador indicará un error.

87

# Métodos de utilidad

- Despliegue de datos (“escribir”):
  - System.out.print( );
  - System.out.println( );
- Por ejemplo:

```
public class Despliegue
{
    public static void main(String[] args)
    {
        System.out.println("Esto se muestra con un salto de linea");
        System.out.print("Aqui no hay salto de linea");
    }
}
```

88

# Métodos de utilidad

- Convertir de String a número:
  - Byte.parseByte( )
  - Short.parseShort( )
  - Integer.parseInt( )
  - Float.parseFloat( )
  - Double.parseDouble( )
- Por ejemplo:

```
public class Conversion
{
    public static void main(String[] args)
    {
        x = Integer.parseInt( “50” );
             y = Integer.parseInt( dato ) + 40; /* Variable dato ya declarada e inicializada */
    }
}
```

89

# Estructura de una aplicación

```
/* Ejemplo de una aplicación */

public class IdentificadorClase <--------- Definición de la clase
{
      public static void main (String[ ]idArreglo)<------------ Método main
       {
        ...

      } // Fin método main
} // Fin clase
```

90

# Ejemplo de programa no interactivo

```
/* Programa para cálculo del Factorial de un número. */
public class Factorial1 {

  public static void main( String arg[] ) {

    int número, i=1, fact=1;
    número = 3;
    while( i <= número ) {
       fact = fact * i;
       i++;
    }
    System.out.println( "El factorial de " + número + " es " + fact );

  } // Fin método main
}
```

Definición de la clase

Método main

Declaración e identificador de variables

91

# Ejemplo de programa no interactivo

```
/* Programa para cálculo del Factorial de un número. */
public class Factorial1 {

  public static void main( String arg[] ) {

    int número, i=1, fact=1;
    número = 3;
    while( i <= número ) {
       fact = fact * i;
       i++;
    }
    System.out.println( "El factorial de " + número + " es " + fact );

  } // Fin método main
}
```

Clase

Main

92

# Valores literales

- Son valores que aparecen explícitamente en el código fuente. Por ejemplo:
  - 'S' : tipo char
  - "S", "Juan", "100" : tipo String
  - 150 : tipo int
  - 150.0 : tipo double
  - 150.0f : tipo float
  - 0150: tipo int octal
  - 0x150: tipo int hexadecimal

93

# Ejemplo de programa no interactivo

```
/* Programa para cálculo del Factorial de un número. */
public class Factorial1 {

  public static void main( String arg[] ) {

    int número, i=1, fact=1;
    número = 3;
    while( i <= número ) {
       fact = fact * i;
       i++;
    }
    System.out.println( "El factorial de " + número + " es " + fact );

  } // Fin método main
}
```

Literal

94

## Características generales de una aplicación simple

- Las aplicaciones simple:
  - Se estructuran como una clase que tiene un método main.
  - El nombre de la clase debe ser el mismo que el del archivo que la contiene.

95

## Consideraciones importantes

- Java requiere la declaración de variables antes de que sean utilizadas. El tipo de una variable no puede ser modificado. Se dice que es un lenguaje fuertemente "tipificado".
- Java distingue entre mayúsculas y minúsculas.
- Las expresiones matemáticas se evalúan con precedencia de * y / sobre + y -.
- Las instrucciones se organizan de acuerdo con las estructuras de control:
  - secuencia
  - decisión : if( cond ) ... else ...
  - Iteración: do ... while( cond ) o while( cond ) ...

96

# Consideraciones importantes

- Las instrucciones se separan mediante un punto y coma ( ; ).
- Los bloques son instrucciones agrupadas mediante paréntesis de llave ({y}). Se utilizan en decisiones e iteraciones, y no llevan punto y coma al final (después de la llave de cierre).
- Documentar los programas (comentarios explicativos).
- Adoptar convención para definición de identificadores:
  - Variables: primeraLetraEnMinúscula
  - Clases: PrimeraLetraEnMayúscula
  - Métodos: primeraLetraEnMinúscula

97

# Tipos primitivos numéricos

Prof. Mg. Rafael Mellado S.
COM2161 – Sistemas de Información 1
Escuela de Comercio
Pontificia Universidad Católica de Valparaíso

## Regla general

- Las variables almacenan datos del mismo tipo que han sido definidas.
- Existen algunas excepciones, en el cual las variables pueden recibir datos de otros tipos:
  - Promoción automática de tipos
  - Casting

# Promoción automática de tipos

- Es una de las formas de cambiar el tipo de un valor.
- También es llamada conversión por ensanchamiento.
- Una variable de un tipo de dato puede recibir un valor de otro tipo si:
  - los dos tipos son compatibles(*), y
  - el tipo de destino es "de mayor jerarquía" que el tipo de origen
- (*)Tipos compatibles: los tipos numéricos son compatibles entre sí.

100

# Promoción automática de tipos

- Ejemplo de promoción automática de tipos: dato tipo `int` es almacenado en una variable `double`.

```
public class EjemploCasting
{
    public static void main(String[] args)
    {
        int variableOrigen=4;
        double variableDestino;

        variableDestino=variableOrigen;

        System.out.println("Variable origen: "+variableOrigen);
        System.out.println("Variable destino: "+variableDestino);
    }
}
```

| x | y |
|---|---|
| 4 | --- |
| 4 | 4.0 |

101

# Casting

- Casting: permite convertir un valor de un tipo de mayor jerarquía a otro de menor jerarquía (funciona si y sólo si el tipo de menor jerarquía puede contener al valor convertido). El casting tiene prioridad sobre los operadores +,-,* y /.
  - Formato: var1 = (tipo de var1) var2

```
double j=1.0;
int i;
i = (int) j;
```

Casting

```
int j=200;
byte i;
i = (byte) j;
```

Incorrecto: el máximo valor que puede contener una variable byte es 127.
La variable *i* queda con valor incorrecto.

102

# Evaluación de expresiones numéricas

- Regla general: La operación de dos números de un mismo tipo, genera un resultado también del mismo tipo.
- Se exceptúan de la regla anterior las operaciones sobre datos tipo byte o short.

```
int a=10, b=4, c;
c =  a  +  b ;
```

10 + 4 = 14 ← int

```
int a=10, b=4, c;
c =  a  /  b ;
```

10 / 4 = 2 ← int

103

# Promoción automática de byte y short

- La operación de dos datos de tipo byte o short, genera como resultado un int.

```
byte i=2, j=3, k;

k =  i  +  j ;
```

2 + 3 = 5 ← int

Error: no puede almacenarse un ***int*** en una variable ***byte***.

```
byte i=2, j=3;
int k;
k =  i  +  j ;
```

2 + 3 = 5 ← int

104

# Resultado de una operación

- El resultado de cualquier expresión es del tipo correspondiente al del operando de mayor jerarquía, en el orden:
  - double
  - float
  - long
  - int

```
int i=10, k;
double j=2.0;

k =  i  /  j ;
```

Resultado 5.0 (Double) ----►

Automáticamente es promovido a ***double*** antes de realizarse la operación.

Error: no puede almacenarse un ***double*** en una variable ***int***.

105

# Ejemplos

- Suponer variable x definida como **double**:
  - x = 10 / 4; ----------► x toma valor 2.0 double
  - x = 10.0 / 4; ----------► x toma valor 2.5 double
  - x = 10.0f / 4; ----------► x toma valor 2.5 double
  - x = (double) 10 / 4; ----------► x toma valor 2.5 double
  - x = (double) (10/4); ----------► x toma valor 2.0 double

106

# Ejemplos

- Suponer variable x definida como **float**:
  - x = 10 / 4; ----------► x toma valor 2.0 float
  - x = 10.0 / 4; ----------► Error: x no puede almacenar valor 2.5 double
  - x = 10.0f / 4; ----------► x toma valor 2.5 float
  - x = (float) 10 / 4; ----------► x toma valor 2.5 float
  - x = (double) (10/4); ----------► Error: x no puede almacenar valor 2.0 double

107

# Precisión de los cálculos

- Las operaciones aritméticas con tipos enteros (byte, short, int o long), se realizan con alta precisión.
- Las operaciones aritméticas con tipos de punto flotante (float o double), se realizan con menor precisión. Ejemplos*:
  - 0.1 sumado 10 veces = 0.9999999999999999
  - 0.01 sumado 100 veces = 1.0
  - 0.001 sumado 1000 veces = 1.0000000000000007
- (*) pruebas efectuadas con variable double



# Precisión de los cálculos

- Las operaciones aritméticas con valores float o double “fallan” en la evaluación de igualdades:

```
public class ComparacionVariables
{
    public static void main(String[] args)
    {
            if(variable1==variable2)
            {
                  System.out.println("Las variables son iguales");
            }
            else
            {
                  System.out.println("Las variables son distintas");
            }
    }
}
```

# Precisión de los cálculos

- Solución: determinar si la diferencia en valor absoluto de las variables es menor que un error predeterminado:

```
public class ComparacionVariables
{
    public static void main(String[] args)
    {
            if( Math.abs(variable1-variable2)< 0.0000001)
            {
                  System.out.println("Son iguales");
            }
            else
            {
                  System.out.println("No son iguales");
            }
    }
}
```

110

# Aplicaciones interactivas

Prof. Mg. Rafael Mellado S.
COM2161 – Sistemas de Información 1
Escuela de Comercio
Pontificia Universidad Católica de Valparaíso

## Aplicación interactiva

```
import java.io.*;
public class CalculoFactorial
{
    public static void main(String[] args) throws IOException
    {
       /*Programa ejemplo para cálculo del Factorial de un número específico*/

        int numero, i=1, factorial=1;
        BufferedReader lector = new  BufferedReader(new InputStreamReader( System.in ));
        String ingresado;

        System.out.println("Ingrese número para cálculo de factorial: ");
        ingresado = lector.readLine();

        numero = Integer.parseInt( ingresado );

        while(i<=numero)
        {
            factorial = factorial*i;
            i++;
        }

        System.out.println( "El factorial de " + numero + " es " + factorial );
    }
}
```

# Aplicación interactiva

Permite crear el objeto de la clase BufferedReader identificado como con el nombre "lector", que se encargará de leer.

Debe explicitarse qué se hará con eventuales errores de I/O

Indica a Java dónde encontrar las clases requeridas

El objeto "lector" se encarga de leer datos del usuario. Para leer invoca el método readLine( ) .

```
import java.io.*;
public class CalculoFactorial
{
   public static void main(String[] args) throws IOException
   {
      /*Programa ejemplo para cálculo del Factorial de un número específico*/

      int numero, i=1, factorial=1;
      BufferedReader lector = new  BufferedReader(new InputStreamReader( System.in ));
      String ingresado;

      System.out.println("Ingrese número para cálculo de factorial: ");
      ingresado = lector.readLine();

      numero = Integer.parseInt( ingresado );

      while(i<=numero)
      {
          factorial = factorial*i;
          i++;
      }

      System.out.println( "El factorial de " + numero + " es " + factorial );
   }
}
```



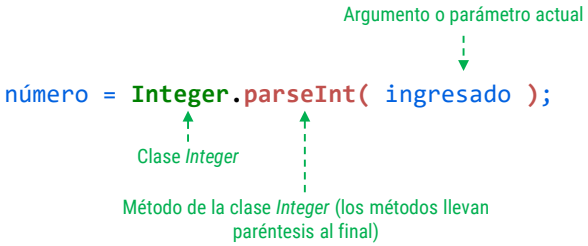
# Objeto lector

– Puede utilizarse las veces que sea necesario:

```
import java.io.*;
public class UsoLector
{
    public static void main(String[] args) throws IOException
    {
        String dato;
        int base, exponente;

        BufferedReader objetoLector=new BufferedReader (new InputStreamReader(System.in) );

        System.out.println("Ingrese base :");
        dato = objetoLector.readLine();
        base = Integer.parseInt(dato);

        System.out.println("Ingrese exponente :");
        dato = objetoLector.readLine();
        exponente = Integer.parseInt(dato);
    }
}
```

# Método parseInt

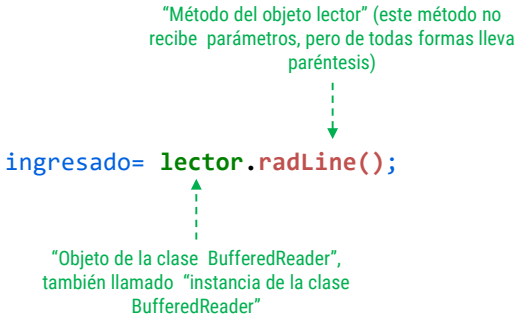
- El método **parseInt** de la clase Integer retorna el equivalente a int de un String recibido como parámetro. Este valor es asignado a la variable número.




115

# Método readLine

- El método `readLine()` del objeto lector retorna un **String** ingresado por el usuario. Este valor es asignado a la variable ingresado.




116

# Paso de parámetros

- El valor retornado por un método puede usarse como parámetro actual de otro método:

```
import java.io.*;
public class UsoLector
{
    public static void main(String[] args) throws IOException
    {
        int base, exponente;

        BufferedReader objetoLector=new BufferedReader (new InputStreamReader(System.in) );

        System.out.println("Ingrese base :");
        base = Integer.parseInt(objetoLector.readLine());

        System.out.println("Ingrese exponente :");
        exponente = Integer.parseInt(objetoLector.readLine());
    }
}
```

usuario.readLine() retorna un String que es recibido como parámetro por el método parseInt de la clase Integer

117

# Ejercicios

- Implemente una aplicación que dado un número por el usuario, se le indique si es positivo, negativo o cero.
- Desarrolle un programa que dado un número (entero) por el usuario muestre por pantalla todos los divisores de dicho número, por ejemplo, si se ingresa el número 2, sus divisores son el 1 y el 2.
- Desarrollar una aplicación que dada una cantidad de números (n, con ciclos iterativos de petición y que terminan cuando el usuario ingrese 0) guarde el impar mayor y el par menor y luego los muestre por pantalla

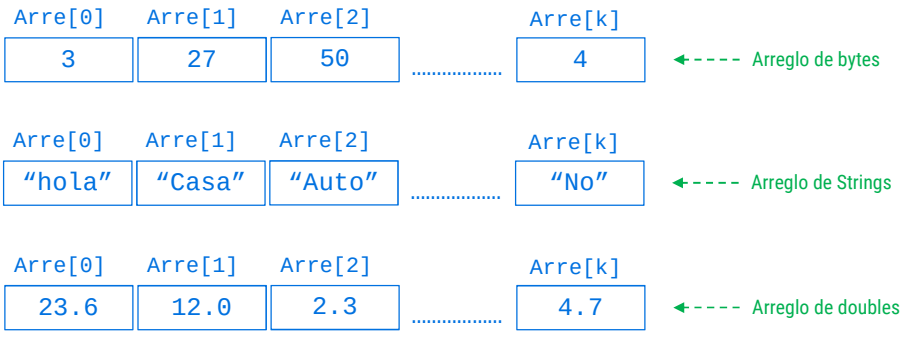
118

# Arreglos en Java

Prof. Mg. Rafael Mellado S.
COM2161 – Sistemas de Información 1
Escuela de Comercio
Pontificia Universidad Católica de Valparaíso

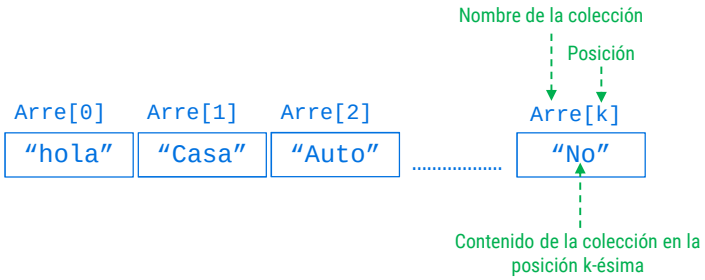
## Arreglos en Java

- Los arreglos son colecciones ordenadas de datos del mismo tipo.
- Ejemplos:



# Arreglos en Java

- Cada arreglo se reconoce por un identificador y cada dato se almacena en una posición **indexada**.
- Un arreglo de largo N, tiene posiciones indexadas mediante enteros desde 0 a N-1.




121

# Creación de arreglos

- Se debe definir una variable que identifique al arreglo, indicando la naturaleza de los datos que se almacenarán:

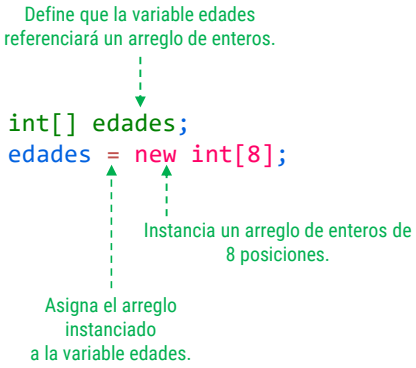
```
tipo[ ]  variable;
tipo variable[ ];
```

- Luego se debe instanciar el arreglo indicando el largo que tendrá, y asignarlo a la variable:

```
variable = new tipo[ entero ];
```

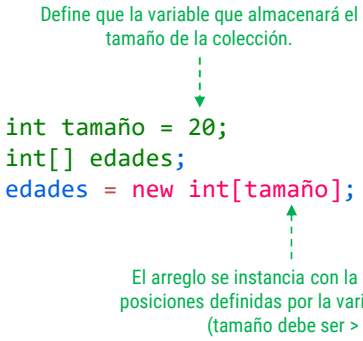
122

# Creación de arreglos




123

# Creación de arreglos




124

# Creación de arreglos

- Forma abreviada de creación de arreglos:
  - La definición de variable, instanciación del arreglo y su asignación a la variable puede realizarse en una sola instrucción.

```
tipo[ ]  variable = new tipo[ entero ];
```

  - Ejemplo:

```
double[] nota = new double[4];
```

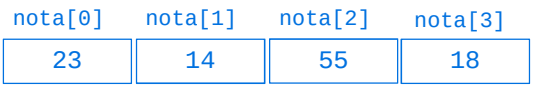
125

# Creación de arreglos

- Asignación y creación literal de arreglos:
  - También es posible instanciar arreglos escribiéndolos como literales en el código fuente:

```
int[] nota = { 23, 14, 55, 18 };
```

Instancia un arreglo de enteros de tamaño 4




126

# Recorrido de arreglos

- Se puede utilizar la propiedad **length** del arreglo para controlar procesos iterativos sobre el mismo:

```
...
int i;
long[] números;
números = new long[ 20 ];
...
i=0;
while( i < números.length ) {
   System.out.println( números[ i ] );
   i++;
}
...
```

127

# Recorrido de arreglos

- Cuando se recorre arreglos se presenta un error típico:
  - Tratar de acceder una posición inexistente del arreglo, por ejemplo, la posición 10 de un arreglo de largo 10.
  - Cuando lo anterior ocurre, se genera en tiempo de ejecución una excepción denominada: **ArrayIndexOutOfBoundsException**.
  - Se debe recordar que los arreglos de tamaño n son recorridos desde la posición **0** a la posición **n-1**.

128

# Consideraciones

- Una vez instanciado un arreglo, no puede modificarse su largo.
- **length** es una propiedad o atributo del arreglo que contiene el largo del mismo.

```
double[] nota;
nota = new double[4];
System.out.println( “El largo es ” + nota.length );
```

Las propiedades o atributos se consultan sin paréntesis al final, a diferencia de los métodos.

129

# Consideraciones

- Sobre el arreglo de parámetros declarado al inicio del método main es necesario destacar:
  - Es instanciado por Java al momento de ejecutarse la aplicación.
  - El arreglo se instancia con un largo igual a la cantidad de parámetros traspasados en la línea de comandos.
  - El arreglo de parámetros debe ser declarado como arreglo de Strings.
  - En este curso no se hará uso de dicho arreglo.

```
public static void main(String arg[ ]) throws   IOException
```

130

# Consideraciones

- Los arreglos se instancian. La instanciación ocurre de tres formas:
  - cuando se utiliza el operador **new**.
  - cuando el arreglo es declarado literalmente.
  - al ejecutar la aplicación, en el caso del arreglo de parámetros del método main.
- Los arreglos son referenciados desde una variable.
- Todas las posiciones del arreglo son del mismo tipo.
- El atributo **length** permite acceder al largo del arreglo.
- Y tratar de acceder una posición inexistente del arreglo genera una excepción **ArrayIndexOutOfBoundsException**.

131