

Introducción a las tecnologías de información

Prof. Mg. Rafael Mellado S.
EII147 – Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso

Ingeniería y tecnología



Necesidades humanas



Ingeniería: Desarrollo y aplicación de tecnologías



Ciencia

Sistemas de información

- Toda organización necesita información que se transmite entre sus elementos y desde y hacia el exterior del sistema.
- Una parte de esta comunicación que se realiza por medio de contactos interpersonales entre los empleados, es el sistema de información informal.
- Cuando se trata de organismos complejos, este tipo de flujo de información es insuficiente y costoso, por lo que se hace necesario un sistema de información formal u organizacional.

3

Sistemas de información

- Los sistemas de información son muy importantes en el día de hoy para cualquier organización. ¿Porqué?
 - La importancia de la información.
 - Decisiones de negocio.
 - Tecnología como herramienta para crear, encontrar, almacenar y transformar información.



4

Sistemas de información

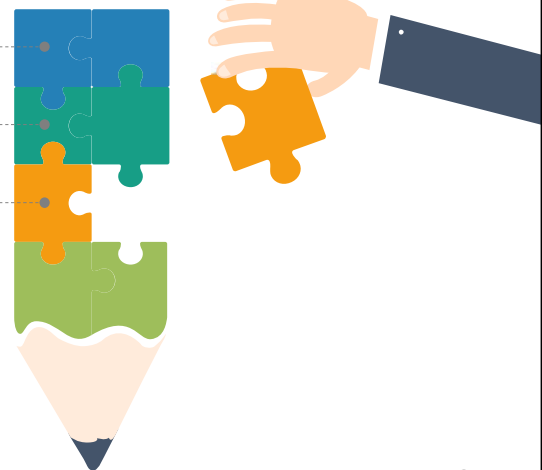
– ¿Qué es un sistema?

- Un sistema puede definirse como un conjunto de cosas que ordenadamente relacionadas entre sí contribuyen a un determinado objetivo.
- Los sistemas están limitados, natural o artificialmente.
- Todo lo que está situado fuera de los límites del sistema se denomina entorno.
- El sistema toma elementos del entorno, entradas, como materias primas para elaborar los productos que se devuelven al entorno, salidas.
- Los sistemas pueden ser naturales o artificiales, si son debidos al hombre. Un sistema de información es un sistema artificial.

5

Datos, información y conocimiento

- Datos, ejemplo: 6.5
- Información, ejemplo: Te sacaste un 6.5
- Conocimiento, ejemplo: Un 6.5 es bueno



6

Categorías de información



7

Flujo de transformación

– Componentes de un sistema:



8

Clasificación

- Clasificación General de los Sistemas de Información:
 - Transaccionales
 - Sistemas de Apoyo a la Toma de Decisiones
 - Sistemas Estratégicos (Sistemas Expertos)

9

Uso de los sistemas de información

- En todo momento, a través del uso de los sistemas de información en las organizaciones se busca cumplir con una serie de objetivos mínimos:
 - Automatización de procesos operativos.
 - Proporcionar información de apoyo al proceso de toma de decisiones.
 - Lograr ventajas competitivas a través de su implantación y uso.

10

Objetivos de este curso



- Reconocer y describir los principales recursos tecnológicos disponibles para el tratamiento y transmisión de datos.
- Obtener una visión general, tanto de los recursos y servicios disponibles, como de las nuevas tendencias en el ámbito de tecnologías de información, y observar su impacto en ámbito público y privado.
- Implementar algoritmos de complejidad simple-mediana en un lenguaje de programación de propósito general.

11

Objetivos de este curso



- Conocer los fundamentos de la programación orientada a objetos, y la aplicación de estos conceptos en el desarrollo de aplicaciones simples.
- Emplear un computador personal con aplicaciones de productividad personal.
- Practicar la comunicación escrita de informes técnicos, y su presentación oral

12

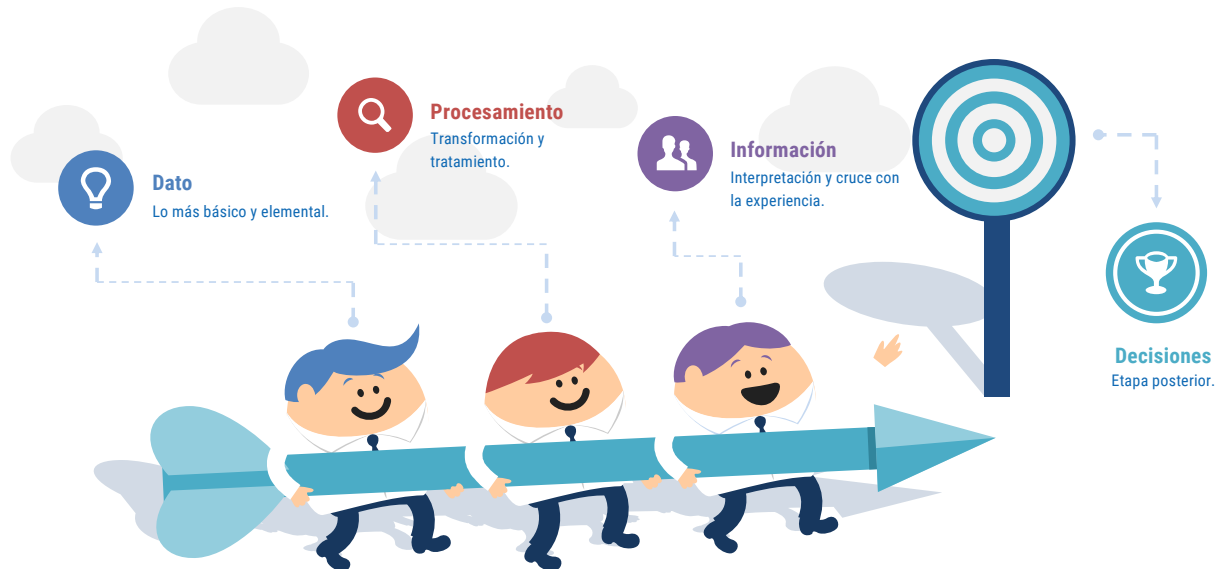
Introducción

Prof. Mg. Rafael Mellado S.
EII147 - Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso

Introducción

- Como sabemos, dentro de la resolución de problemas tenemos una serie de elementos a considerar, tales como:
 - Datos
 - Información
 - Procesamiento
 - Entre otros.

Introducción



15

Introducción

- En razón a lo anterior, es que para resolver un problema lo primero que se debe hacer es organizar las ideas y metódicamente establecer un proceso de resolución.
- Un gran apoyo para ser metódico y aprender la lógica de resolución algorítmica son los diagramas de flujo.
- Nunca olvidar que un programa codificado es el resultado de diseñar antes una correcta solución de ingeniería.

16

Introducción






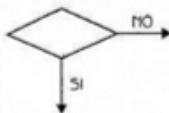

17

Diagramas de flujo

- Son la representación de los algoritmos y también utilizados para el modelamiento de procesos (algo muy relevante para el Ingeniero Civil Industrial).
- Es muy común su utilización en la programación, la economía, los procesos industriales y la psicología cognitiva.
- Esta notación utiliza símbolos con significados previamente definidos que representan los pasos correspondientes al algoritmo, y representan el flujo de ejecución mediante flechas que conectan los puntos de inicio y finalización de los determinados procesos.
- **Para efectos de este curso utilizaremos una notación simplificada!!**

18

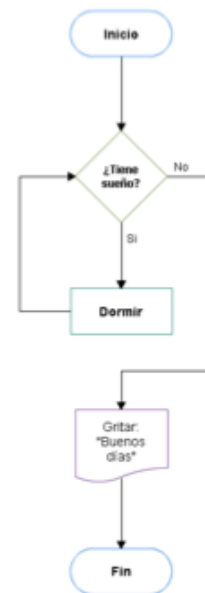
Notación

Representación del Símbolo	Explicación del Símbolo
	Símbolo utilizado para marcar el inicio y el fin del diagrama de flujo.
	Símbolo utilizado para introducir los datos de entrada. Expresa lectura.
	Símbolo utilizado para representar un proceso. En su interior se expresan asignaciones, operaciones aritméticas, cambios de valor de celdas en memoria, etc.
	Símbolo utilizado para representar una decisión. En su interior se almacena una condición, y dependiendo del resultado de la evaluación de la misma se sigue por una de las ramas o caminos alternativos. Este símbolo se utiliza en la estructura selectiva <i>si entonces</i> y <i>si entonces/sino</i> que estudiaremos en el siguiente capítulo, y en las estructuras repetitivas <i>repetir</i> y <i>mientras</i> que analizaremos en el capítulo 3.
	Símbolo utilizado para representar la impresión de un resultado. Expresa escritura.

19

Ejemplo

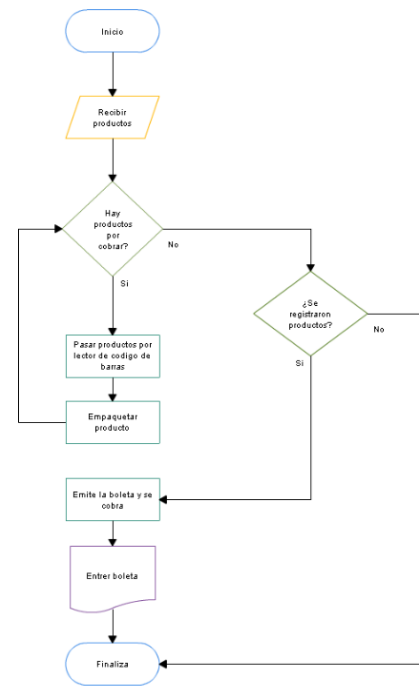
- ¿Podemos entender lo que busca representar el diagrama de flujo?



20

Ejemplo

- Ahora, imaginemos que nos piden modelar el proceso de vender productos en el supermercado. Como sería el diagrama de flujo?



21

Desafío

- Realice el diagrama de flujo para:
 - Calcular la sumatoria de todos los valores desde el 1 al 1000
 - Representar la decisión si un numero es primo o no lo es. Dicho numero es dado por el usuario.

22

Diagramas de flujo



- Quieres hacer tus propios diagramas de flujo?
 - Puedes hacerlos en papel o si quieres hay una útil herramienta gratuita online: <http://www.cacoo.com>

Generalidades de Java

Prof. Mg. Rafael Mellado S.
EI1147 – Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
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.



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.

26

Principios



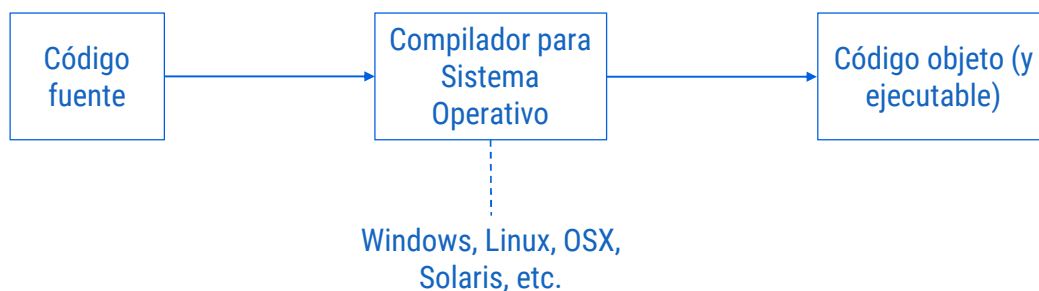
27

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.

28

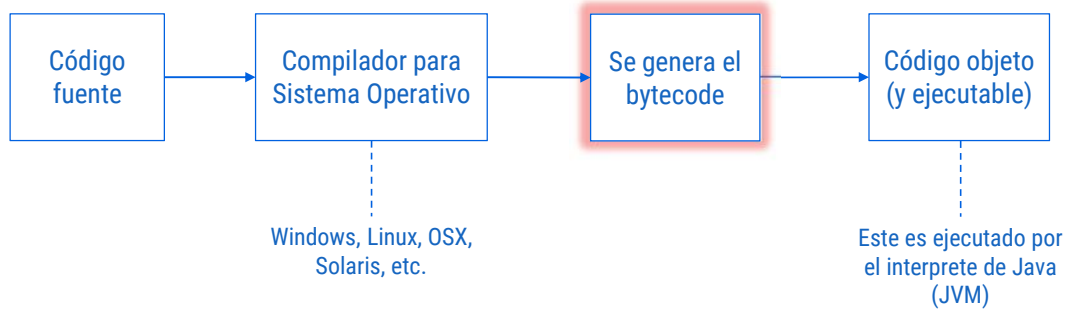
Desarrollo genérico de aplicaciones



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

29

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.

30

Java en distintos tamaños



JME: Java Micro Edition



JSE: Java Standard Edition



JEE: Java Enterprise Edition

31

Elementos de aplicaciones simples

Prof. Mg. Rafael Mellado S.
EII147 – Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
Pontificia Universidad Católica de Valparaíso

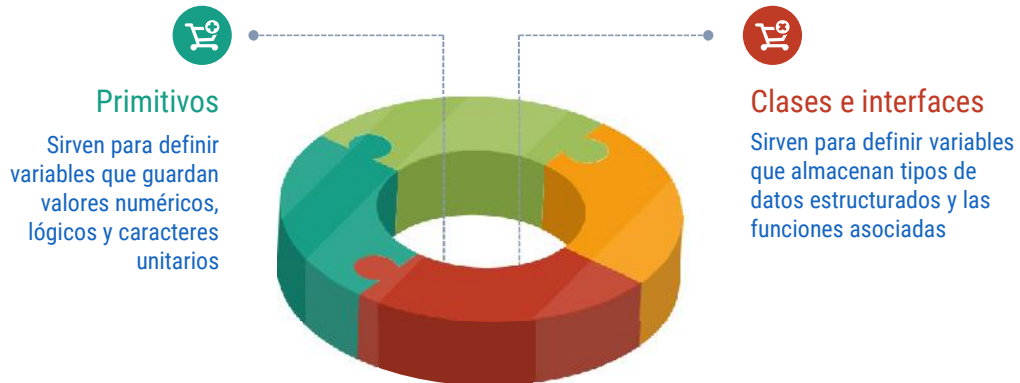
Desarrollo de aplicaciones en Java



- 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.

Tipos de datos

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



34

Tipos de datos primitivos

- Java soporta los siguientes tipos de datos primitivos:



35

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]

36

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 ($10^{-324}, 10^{308}$)

37

Tipos de datos primitivos

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

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

38

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.

39

Operadores básicos

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

40

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)

41

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`.

42

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.

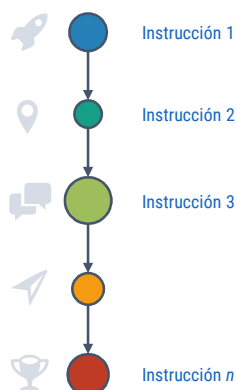
43

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.

44

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");
    }
}
```

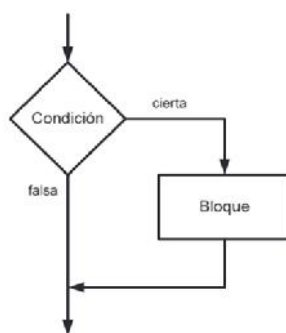
45

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.

46

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.

47

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*/
        }
    }
}
```

48

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*/
        }
    }
}
```

49

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*/
            }
        }
    }
}
```

50

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.

51

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: →

52

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");
        }
    }
}
```

53

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.

54

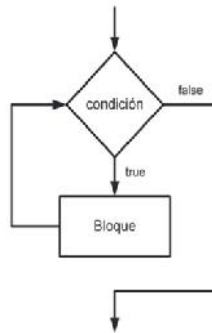
Instrucción while

- 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.

55

Instrucción while

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



56

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!");
    }
}
  
```

57

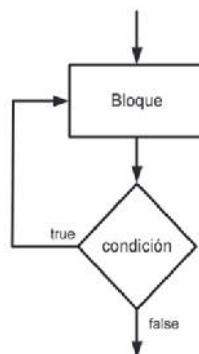
Instrucción do-while

- 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.

58

Instrucción do-while

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



59

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);
    }
}
```

60

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 ;

61

Instrucción for

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

62

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
*/
```

63

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

64

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;
    }
}
```

65

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.

66

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 línea");
        System.out.print("Aquí no hay salto de línea");
    }
}
```

67

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 */
    }
}
```

68

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
```

69

Ejemplo de programa no interactivo

```

/* Programa para cálculo del Factorial de un número. */
public class Factorial1 { ←----- Definición de la clase

    public static void main( String arg[] ) { ←----- Método main

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

    } // Fin método main
}
    
```

70

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

71

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

72

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; ----- Literal
        número = 3;
        while( i <= número ) {
            fact = fact * i;
            i++;
        }
        System.out.println( "El factorial de " + número + " es " + fact );
    } // Fin método main
}

```

73

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.

74

Traspaso de datos por parámetro

- El método main puede recibir parámetros desde la línea de comandos del intérprete. Estos se almacenan en las posiciones del arreglo de String indicado en la declaración del método.
- Un arreglo es una colección de datos de un mismo tipo, referenciada por un nombre. Cada dato ocupa una posición dentro de la colección. La cantidad total de posiciones es fija, y son accesibles por medio de un subíndice, que se inicia en cero (0).
- El concepto de arreglo será visto posteriormente en el curso.

75

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) ...`

76

Consideraciones importantes

- Las instrucciones se separan mediante un punto y coma (;).
- Los bloques son instrucciones agrupadas mediante paréntesis de llave ({ }). 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`

77

Tipos primitivos numéricos

Prof. Mg. Rafael Mellado S.
EII147 – Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
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.

80

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

81

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.

82

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

83

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

84

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;
```

Resultado 5.0 (Double)

```
k = i / j ;
```

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

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

85

Ejemplos

– Suponer variable x definida como **double**:

- `x = 10 / 4;` \dashrightarrow x toma valor 2.0 double
- `x = 10.0 / 4;` \dashrightarrow x toma valor 2.5 double
- `x = 10.0f / 4;` \dashrightarrow x toma valor 2.5 double
- `x = (double) 10 / 4;` \dashrightarrow x toma valor 2.5 double
- `x = (double) (10/4);` \dashrightarrow x toma valor 2.0 double

86

Ejemplos

– Suponer variable x definida como **float**:

- `x = 10 / 4;` \dashrightarrow x toma valor 2.0 float
- `x = 10.0 / 4;` \dashrightarrow Error: x no puede almacenar valor 2.5 double
- `x = 10.0f / 4;` \dashrightarrow x toma valor 2.5 float
- `x = (float) 10 / 4;` \dashrightarrow x toma valor 2.5 float
- `x = (double) (10/4);` \dashrightarrow Error: x no puede almacenar valor 2.0 double

87

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

88

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");
        }
    }
}
```

89

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");
        }
    }
}
```

90

Aplicaciones interactivas

Prof. Mg. Rafael Mellado S.
EII147 – Introducción a las tecnologías de información
Escuela de Ingeniería Industrial
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 );
    }
}
```

92

Aplicación interactiva

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

```
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 );
    }
}
```

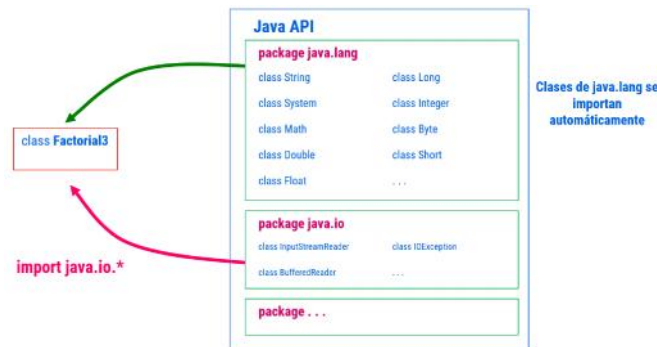
Debe explicitarse qué se hará con eventuales errores de I/O

Indica a Java dónde encontrar las clases requeridas

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

Importación desde la API de Java

- Otras clases deben importarse explícitamente, indicando el nombre del package que las contiene



Componentes de la API de Java

- Algunos de los packages de la Java API son:
 - `java.applet` : construcción de applets.
 - `java.beans` : desarrollo y reutilización de componentes.
 - `java.awt` : componentes visuales.
 - `java.io` : entrada y salida de datos.
 - `java.lang` : clases de uso común.
 - `java.math` : soporte para tipos numéricos con SQL.

95

Componentes de la API de Java

- Algunos de los packages de la Java API son:
 - `java.net` : componentes para trabajos en red.
 - `java.rmi` : invocación remota de métodos.
 - `java.sql` : soporte para JDBC.
 - `java.security` : firma de applets y codificación de datos.
 - `java.text` : manejo de distintos tipos de textos.
 - `java.util` : clases varias de utilidad.

96

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);
    }
}
```

97

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.

Argumento o parámetro actual
↓

número = **Integer.parseInt(ingresado);**

↑ ↑

Clase *Integer* Método de la clase *Integer* (los métodos llevan paréntesis al final)

98

Método readLine

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

"Método del objeto lector" (este método no
recibe parámetros, pero de todas formas lleva
paréntesis)

↓

`ingresado= lector.readLine();`

↑

"Objeto de la clase `BufferedReader`",
también llamado "instancia de la clase
`BufferedReader`"

99

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

100

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