UT1 Transicion a Java

Fecha 13/03/2024

Introduccion

- **Origen de Java (1991):**
  - Grupo de ingenieros de Sun Microsystems.
  - Diseño para electrodomésticos con limitaciones de potencia y memoria.
- **Desarrollo Independiente de CPU:**
  - Necesidad de un lenguaje independiente del tipo de CPU.
  - Creación de código "neutro" ejecutado por Java Virtual Machine (JVM).
  - Lema: "Write Once, Run Everywhere."
- **Revolución en Internet (1995-1997):**
  - Incorporación del intérprete Java en Netscape Navigator 2.0.
  - Java 1.1 en 1997 mejora significativamente el lenguaje.
- **Programación en Java:**
  - No se parte de cero; se basa en clases preexistentes.
  - Amplio conjunto de clases en el API de Java.
  - Ideal para aprender informática moderna estándar.
- **Objetivo Principal de Java:**
  - Conectar a los usuarios con la información en cualquier lugar.
  - Lenguaje completo con extensiones estándar y claras.
- **Características de Java (Según Sun Microsystems):**
  - Simple, orientado a objetos, distribuido, interpretado, robusto, seguro, arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico.
- **Complejidad y Evolución de Java:**
  - Evolución de 12 a 59 packages en distintas versiones.

- Aprender de manera iterativa, refinando en sucesivas iteraciones.

Que es Java 2?

- **Java 2 (JDK 1.2):**
  - Tercera versión importante del lenguaje Java.
  - Anteriormente conocido como Java 1.2 o JDK 1.2.
- **Cambios con respecto a Java 1.1:**
  - No hay cambios conceptuales importantes.
  - Extensiones y ampliaciones, similar a Java 1.1.
- **Ventajas de Programar en Java:**
  - Ejecución como aplicación independiente, applet o servlet.
  - Posibilidades de ejecución en diversos entornos.
- **Tipos de Ejecución en Java:**
  - Stand-alone Application: ejecución como aplicación independiente.
  - Applet: ejecución dentro de un navegador al cargar una página HTML desde un servidor web.
  - Servlet: aplicación sin interfaz gráfica que se ejecuta en un servidor de Internet.
- **Desarrollo de Arquitecturas Cliente-Servidor:**
  - Java facilita el desarrollo de arquitecturas cliente-servidor.
  - Capacidad para crear aplicaciones distribuidas.
- **Aplicaciones Distribuidas en Java:**
  - Conexión a otros ordenadores.
  - Ejecución de tareas en varios ordenadores simultáneamente.
  - Distribución de trabajo integrada en el API de Java.

El entorno de desarrollo de Java

- **Programas para Desarrollar en Java:**
  - Java(tm) Development Kit (JDK) distribuido gratuitamente por Sun Microsystems.
  - Conjunto de programas y librerías para desarrollar, compilar y ejecutar programas en Java.
  - Incluye Debugger para detener y estudiar la ejecución en puntos específicos.

- **IDEs (Integrated Development Environment):**
  - Entornos integrados para escribir, compilar y ejecutar código Java.
  - Permite desarrollo rápido, incorporando librerías con componentes predefinidos.
  - Algunos ofrecen herramientas gráficas para el Debug.
- **El Compilador de Java:**
  - Herramienta en el JDK llamada javac.exe.
  - Analiza la sintaxis de archivos fuente Java (*.java) y genera archivos compilados (*.class).
  - Muestra errores de código y su ejecución es aconsejada antes de la distribución.
- **La Java Virtual Machine (JVM):**
  - "Máquina hipotética o virtual" que ejecuta código "neutro" llamado bytecodes.
  - Convierte bytecodes (*.class) a código específico de la CPU utilizada.
  - Intérprete de Java que evita la dependencia del tipo de procesador.
- **Variables PATH y CLASSPATH:**
  - PATH: Directorios accesibles desde la ventana de comandos de MS-DOS.
  - CLASSPATH: Ruta para buscar clases, librerías y archivos compilados.
  - Fichero batch (*.bat) puede establecer variables; se ejecuta al abrir una consola de MS-DOS.
  - Cambios permanentes se realizan en Autoexec.bat (Windows 95/98) o ajustes de entorno (Windows NT).
- **Compilación y Ejecución con Variables:**
  - Compilador: javac.exe genera archivos *.class.
  - Ejecución: java.exe interpreta bytecodes (*.class) con opciones como JIT (Just-In-Time Compiler).
  - Uso de –classpath o -cp para indicar la ruta de clases y librerías.
  - Comparación de fechas para evitar recompilar clases no modificadas.

Estructura general de un programa Java

- **Estructura de Programa Java:**
  - Clases: Grupo de datos y funciones (métodos) orientados a objetos (OOP).
  - Programa principal contiene la función main().

    - Extensión de ficheros: *.java para fuente, *.class para compilado.
- **Reglas para Nombres de Ficheros y Clases:**
    - Nombre del fichero *.java coincide con la clase public.
    - Mayúsculas y minúsculas deben coincidir.
    - Una clase puede ser public o package, no private o protected.
- **Modularidad e Independencia entre Clases:**
    - Múltiples ficheros *.class constituyen una aplicación.
    - Ejecución a través de la clase que contiene la función main().
    - Clases agrupadas en packages para modularidad.
- **Concepto de Clase:**
    - Agrupación de datos (variables) y funciones (métodos).
    - Elementos: Variables y métodos miembro.
    - Clases pueden tener variables static, comunes a la clase.
- **Herencia:**
    - Permite definir nuevas clases basadas en clases existentes.
    - Clase derivada (extends) hereda variables y métodos de la clase base.
    - Añade o redefine variables y métodos heredados.
- **Concepto de Interface:**
    - Conjunto de declaraciones de funciones.
    - Clases pueden implementar (implements) interfaces.
    - Una clase puede implementar múltiples interfaces.
    - Interface puede derivar de otras interfaces.
- **Concepto de Package:**
    - Agrupación de clases, puede incluir clases relacionadas.
    - Usuario puede crear sus propios packages.
    - Todas las clases de un package deben estar en el mismo directorio.
- **Jerarquía de Clases de Java (API):**
    - Documentación on-line del API de Java es esencial.
    - Jerarquía de clases muestra la relación de herencia.
    - Packages organizan clases; herencia crea nuevas clases.

- Toda clase en Java deriva de java.lang.Object.

Programacion Java

```
public class HolaMundo {
    public static void main(String[] args){
        System.out.println("Hola mundo");
}
```

Operadores

Operadores aritmeticos

- Suma: +
- Resta: -
- Division: /
- Multiplicacion: *
- Mod: %

Operadores aritmeticos combinados con asignacion

- Suma: +=
- Resta: -=
- Division: /=
- Multiplicacion: *=
- Mod: %=

Operador incremento y decremento

- x++;
- x - -;

```
int x = 5, y;

/* Caso 1: */
y = x++; // A "y" se le asigna x y luego se incrementa "x".
```

```
System.out.println(y); // Salida = 5.
System.out.println(x); // Salida = 6.

/* Caso 2: */
y = ++x; // Primero se incrementa "x" y luego se asigna a "y".

System.out.println(y); // Salida = 6.
System.out.println(x); // Salida = 6.
```

Condicionales

if else

```
if (condicion) {
    Instruccion1;
} else {
    Instruccion2;
}
```

switch

```
switch (dato) {
    case 1: Instrucciones1;
            break;
    case 2: Instrucciones2;
            break;
    ...
    case n: InstruccionesN;
            break;
    deafult: CasoContario;
             break;
}
```

Ciclos

Ciclo while

```
while (condition) {
    Instrucciones;
```

```
}
```

Ciclo do while

```
do {
    Instrucciones;
} while (condition);
```

La diferencia con el while es que las instrucciones se ejecutan al menos una vez antes de entrar al while.

Ciclo for

```
for (inicializacion; condicion; aumento o decremento) {
}
```

Arreglos

- Estructura de datos que nos permite almacenar un conjunto de datos de un mismo tipo.
- El tamano de los arrays se declara en un primer momento y no puede cambiar.
- Array unidimensional.
- Sintaxis: Tipo_de_variable[] Nombre_del_array = new Tipo_de_variable[dimension];

```
int[] numeros = new int[3];    // Se reserva en memoria los 3 espacios con
/* Dar valores: */
numeros[0] = 5;
numeros[1] = 7;
numeros[2] = 9;

/* Tambien se puede definir asi: */
int[] numeros = {5,7,9};
```

ForEach

```
/* Utilizando un for: */
String[] nombres = {"Ale", "Maria", "Luisa", "Juan"}
for (i=0; i<nombres.lenght; i++) {
    System.out.println(nombres[i]);
}
```

```
/* Utilizando un for each: */
for(String i:nombres) {
    System.out.println(i);
}
```

Ordenamientos

Metodo burbuja

- Revisa cada elemento de la lista que va a ser ordenada con el siguiente, intercambiandolos de posicion si estan en el orden equivocado. Es necesario revisar varias veces las listas hasta que no se necesiten mas intercambios.

```
public class MetodoBurbuja {
    public static void main(String[] args) {
            Scanner entrada = new Scanner(System.in);
        int arreglo[];
        int nElementos;
        int aux;

        nElementos = entrada.NextInt;

        arreglo = new int[nElementos]; // Le asignamos el numero de elemer

        for(int i = 0; i < nElementos; i++) {
            System.out.println((i+1) + ". Digite un numero");
            arreglo[i] = entrada.NextInt();
        }

        /* Metodo burbuja: */
        for(int i = 0; i < (nElementos-1); i++) { // nElementos-1, no es r

                    for (int j = 0; j < (nElementos-1); j++) {
                        if (arreglo[j] > arreglo[j + 1]) { //Si numeroActu
                            aux = arreglo[j];
                            arreglo[j] = arreglo[i];
                            arreglo[j + 1] = aux;
                        }
                      }
        }
```

```
        /*Fin del Metodo burbuja. */
    }
}
```

Ordenamiento por insercion

Busquedas

Busqueda secuencial

Matrices

- Array bidimensional.
- Necesita dos indices para acceder a sus elementos.
- La siguiente figura representa una matriz M de 3 filas y 5 columnas

	0	1	2	3	4
0					
1					
2					

- Cada fila es un array

Array	0	1	2	3	4
referencia	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]

Array	0	1	2	3	4
referencia	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]

Array	0	1	2	3	4
referencia	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

```
/* Llenado de una matriz manualmente: */
int matriz[][] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

for (int i = 0; i < 3; i++) { // Numero de filas.
    for (int j = 0; j < 5; i++) { // Numero de columnas.
        System.out.print(matriz[i][j]);
    }
    System.out.print(" ");
```

```
}

/* Llenado de una matriz x: */
int matriz[][];
int nFilas;
int nColumnas;

Scanner entrada= new Scanner(System.in);

nFilas = System.out.println("Digite el numero de filas: ");
nColumnas = System.out.println("Digite el numero de columnas: ");

matriz = new int [nFilas][nColumnas];

System.out.println("Digite la matriz: ");

for (i = 0; i < nFilas, i++) {
    for (j = 0; j < nColumnas, j++) {
          System.out.println("Matriz [" + i + "] [" + j + "]: ");
    }
}

for (int i = 0; i < nFilas; i++) { // Numero de filas.
    for (int j = 0; j < nColumnas; i++) { // Numero de columnas.
        System.out.print(matriz[i][j]);
    }
    System.out.print(" ");
}
```

POO

CodeConventions

File Names

File suffixes

- Java source: .java
- Java bytecode: .class

Common file names

- GNUmakefile: makefiles (para construir nuestro software).
- README: summarizes.

File Organization

- Dividir secciones con lineas en blanco.
- Comentarios para identificar las secciones.

Java source files

- Una clase o interfaz PUBLICA por archivo fuente.
- Si una clase o interfaz PRIVADA esta asociada a una publica pueden estar dentro del mismo archivo.
- Orden del archivo:
  - Comentarios iniciales (programadores, fecha, derechos de autor y aviso copyright). Ej:

```
/*
* Nombre de la clase
*
* Información de la versión
*
* aviso de copyright
* /
```

  - Declaraciones de importacion de paquetes.
  - Declaraciones de la clase o interfaz.

Declaration	Notas
class or interface statement (op)	
Class / Interface implementation comment	Informacion que no sea apropiada para la documentacion
Class (static) variables	1. Public, 2. Protected, 3. Private.
Instance variables	Idem.
Constructors	
Methods	Agruparse por funcionalidad no por accesiblidad.

Indentation

- Los bloques de código y las declaraciones deben tener una sangría de 4 espacios.

Line lenght

- Se recomienda un máximo de 80 caracteres por línea.
- En documentacion 70.

Wrapping lines

- Cada nivel sucesivo de código debe tener una sangría adicional de 4 espacios.
- Si la expresion no cabe en una sola linea:
  - Dividir despues de una coma.
  - Frenar antes de un operador.
  - Preferir dividir en nivel superior a nivel inferior.
  - Alinear la segunda linea con el comienxo de la linea anterior.
  - Si el codigo es confuso simplemente usar sangria de 8 espacios.

Comments

- El exceso de comentarios refleja una mala calidad del codigo.
- Evitar ser redundante.
- No deben usarse caracteres especiales ni encerrarlos en cuadrados.

Implementation comment formats

- Delimitados por /* ... */ y //.
- Comentar codigo sobre implementaciones.
- Formatos:
  - Block
    - Descripciones de archivos, métodos, estructuras de datos y algoritmos.
    - Usar al principio de cada archivo y antes de cada método (en su defecto dentro de un metodo, deben tener tab al mismo nivel que el codigo que describen).
    - Debe ir precedido por una linea en blanco.

```
/*
 * Here is a block comment.
```

```
 */
```

- Single-line
  - Al nivel del codigo que sigue.

```
if (condition) {
    /* Maneja la condicion. */ ...
}
```

- Trailing
  - Comentarios muy cortos.
  - Separados de las declaraciones.
  - Estar alineados.

```
public class HolaMundo {
    public static void main(String[] args){
        System.out.println("Hola mundo");            /* Comment */
}
```

- End-of-line
  - No usar en varias lineas consecutivas en comentarios de texto pero si para secciones de codigo.

```
public class HolaMundo {
    public static void main(String[] args){
        System.out.println("Hola mundo");            // Comment.
}
```

Documentation comments

- Delimitados por /** ... */.

```
/**
* La clase de ejemplo proporciona...
* /
clase Ejemplo {...
```

- Describir la especificacion del codigo libre de implementaciones (para ser leido por programadores que no necesariamente tienen acceso al codigo fuente).

- Describen clases, interfaces, constructores, métodos y campos de Java.
- Para proporcionar info sobre una clase, interfaz, variable o metodo que no es apropiado para la documentacion utilizar un bloque de implementacion o un comentario en una linea.
- Los comentarios de documentacion NO deben colocarse dentro de un bloque de definicion de metodo porque java asocia los comentarios con la primera declaracion luego del comentario.

Declarations

Number per line

- Una declaracion por linea ya que fomenta los comentarios.

```
int number;         // Numero.
int sum;            // Suma.
```

Placement

- Declaraciones solo al inicio del bloque (bloque = codigo entre llaves).
- No esperar para declarar variables hasta su primer uso (excepcion: indices de loops for).
- No repetir declaraciones de variables en bloques internos.

Initialization

- Inicializar las variables donde estan declaradas (excepcion: si el valor depende de algun calculo).

Class and Intreface declarations

- Condiciones:
  - No hay espacio entre el nombre del metodo y el parentesis de los parametros.
  - La llave abierta va en la misma linea que la declaracion.
  - La llave de cierre va en una nueva linea identada a la altura del codigo que abre.
  - Los metodos estan separados por una linea en blanco.

Statements

Simple Statements

- Una linea debe tener como maximo una declaracion.

```
argv++;
argc--;
```

- No usar el operador comma para agrupar varias declaraciones a menos qie sea obvio.

Compound Statements

- Statements que agrupan listas de statements encerrada en “{ }”.
- Las declaraciones adjuntas deben tener una sangría de un nivel más que la declaración compuesta.
- La llave de apertura debe estar al final de la línea que comienza la declaración compuesta; la llave de cierre debe comenzar una línea y tener sangría hasta el comienzo de la declaración compuesta.
- Las llaves se utilizan alrededor de todas las declaraciones, incluso las únicas, cuando forman parte de una estructura de control, como un if-else o un for. Esto hace que sea más fácil agregar declaraciones sin introducir errores accidentalmente debido a que se olvida agregar llaves.

return Statements

```
return;

return myDisk.size();

return (size ? size : defaultSize);
```

if, if-else, else Statements

```
if (condition) {
    statements;
}

if (condition) {
    statements;
} else {
    statements;
}

if (condition) {
    statements;
```

```
} else if (condition) {
    statements;
} else if (condition) {
    statements;
}

if (condition) //AVOID! THIS OMITS THE BRACES {}!
    statement;
```

for Statements

```
for (initialization; condition; update) {
    statements;
}
```

while Statements

```
while (condition) {
statements;
}
```

do-while Statements

```
do {
statements;
} while (condition);
```

switch Statements

```
switch (condition) {
case ABC:
    statements;
    /* falls through */
case DEF:
    statements;
    break;
case XYZ:
    statements;
    break;
```

```
default:
    statements;
    break;
}
```

try-catch Statements

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
}
```

White Space

Blank lines

- Mejoran la legibilidad.
- Utilizar dos lineas en blanco cuando:
  - Entre secciones de un archivo fuente.
  - Entre definiciones de clases o interfaz.
- Utilizar una linea en blanco cuando:
  - Entre metodos.
  - Entre variables locales de un metodo y su primera declaracion.
  - Antes de un comentario de bloque o uno de una sola linea.
  - Entre secciones logicas dentro de un metodo para mejorar su legibilidad.

Blank spaces

- Utilizar un espacio en blanco cuando:
  - Entre una palabra clave seguida de un parentesis (NO entre el nombre de un método y su paréntesis de apertura).
  - Luego de la coma en las listas de argumentos.
  - Para separar operadores binarios.
  - NUNCA deben separar operadores unarios como incremento (“++”) y decremento (“--”) de sus operandos.

- Las expreciones de un for luego del “;” debe estar separada por un espacio.
- Casts deben estar seguidos de un espacio.

Naming Conventions

- Programas mas comprensibles al ser faciles de leer.
- Informacion sobre la funcion del identificador.

Identificador	Regla	Ejemplos
Clases	Sustantivos en mayusculas y minusculas con la inicial de cada palabra en mayuscula. Evite acronimos y abreviaturas.	class Raster; class ImageSprite;
Interfaces	Idem	interface RasterDelegate; interface Storing;

Metodos	Verbos en mayusculas y minusculas con la primer letra de cada palabra interna en mayuscula.	run();
		runFast();
		getBackground();

Variables	Palabras internas comienzan con mayusculas.	int i; char *cp; float myWidth;

Constantes	Todas en mayusculas con palabras separadas por guiones.	int MIN_WIDTH = 4;
		int MAX_WIDTH = 999;
		int GET_THE_CPU = 1;

Programming Practices

Proporcionar acceso a variables de instancia y clases

- No hacer ninguna variable o clase publica sin una buena razon. Ej: Una clase que es una estructura de datos sin ningun comportamiento especifico.

Referencia a variables y metodos de clase

- No usar un objeto para acceder a una variable o metodo de clase. Usar el nombre de la clase.

Constantes

- Las constantes numericas explicitas no deber codearse explicitamente (solo -1, 0 y 1) que pueden aparecer en un for como contador.

Asignaciones de variables

- No asignar a varias variables el mismo valor en una sola declaracion.
- No usar el operador de asignacion en un lugar que se puede confundir con el operador de igualdad.
- No usar asignaciones intergadas.

Practicas diversas

- Parentesis
  - Utilizar parentesis generosamente en expresiones que involucran operadores mixtos
- Valores devueltos
  - Hacer que la estructura del programa matchee con la intencion.

```
/* Ejemplo 1: */
if (booleanExpression) {
 return TRUE;
} else {
 return FALSE;   // En lugar de hacer esto.
}

return booleanExpression;   // Hacer esto.

/* Ejemplo 2: */
if (condition) {
 return x;
}
return y;   // En lugar de hacer esto.

return (condition ? x : y);   // Hacer esto.
```

- Expresiones antes de ? en el operador condicional
  - Poner parentesis en caso de haber un operador binario antes del operador “?”.
- Comentarios especiales
  - Usar XXX para comentarios de bugs pero que funcionan.
  - Usar FIXME para comentarios de bugs que no funcionan.