

Tabla de contenidos

1. INTRODUCCIÓN A JAVA	8
1.1. QUÉ ES JAVA.....	8
1.1.1. <i>Introducción</i>	8
1.1.2. <i>Lenguaje orientado a objetos</i>	8
1.1.3. <i>Sintaxis basada en C/C++</i>	8
1.1.4. <i>Multiplataforma</i>	9
1.1.5. <i>Manejo automático de memoria</i>	9
1.1.6. <i>Evolución permanente</i>	9
1.2. ORGANIZACIÓN	10
1.2.1. <i>JME (Mobile / Wireless)</i>	10
1.2.2. <i>JSE (Core / Desktop)</i>	10
1.2.3. <i>JEE (Enterprise / Server)</i>	10
1.3. LA HISTORIA	11
1.3.1. <i>El comienzo</i>	11
1.3.2. <i>Aparición de Internet</i>	11
1.3.3. <i>Por qué el nombre JAVA</i>	12
2. DESARROLLO, COMPIILACION Y EJECUCION	13
2.1. ORGANIZACION	13
2.1.1. <i>El Java Development Kit (JDK)</i>	13
2.1.2. <i>El compilador</i>	13
2.1.3. <i>El Java Runtime Environment (JRE)</i>	13
2.2. LA JAVA VIRTUAL MACHINE (JVM).....	14
2.2.1. <i>Que es</i>	14
2.2.2. <i>La variable de entorno CLASSPATH</i>	14
3. SINTAXIS Y SEMÁNTICA DE JAVA	15
3.1. DEFINICIÓN DE VARIABLES.....	15
3.1.1. <i>Identificador</i>	15
3.1.2. <i>Tipos de variables</i>	15
3.1.3. <i>Declaración y definición</i>	15
3.1.4. <i>Vectores</i>	16
3.2. TIPOS DE DATO PRIMITIVOS	17
3.2.1. <i>boolean</i>	17
3.2.2. <i>char</i>	17
3.2.3. <i>byte</i>	17
3.2.4. <i>short</i>	17
3.2.5. <i>int</i>	17
3.2.6. <i>long</i>	18
3.2.7. <i>float</i>	18
3.2.8. <i>double</i>	18
3.3. OPERADORES	18
3.3.1. <i>Operadores Aritméticos</i>	18
3.3.2. <i>Operadores de Asignación</i>	19
3.3.3. <i>Operador instanceof</i>	19
3.3.4. <i>Operador condicional ?:</i>	19
3.3.5. <i>Operadores incrementales y decrementales</i>	20
3.3.6. <i>Operadores relacionales</i>	20
3.3.7. <i>Operadores lógicos</i>	20

3.3.8. Operador concatenación de caracteres	21
3.3.9. Operadores aplicables a bits.....	21
3.3.10. Clasificación	22
3.4. ESTRUCTURAS DE CONTROL DE FLUJO	23
3.4.1. Introducción.....	23
3.4.2. Bifurcacion if-else.....	23
3.4.3. Bifurcacion if-else-if-else.....	24
3.4.4. Bifurcacion switch	25
3.4.5. Bucle while.....	26
3.4.6. Bucle for.....	27
3.4.7. Bucle do-while.....	28
3.4.8. Sentencia break.....	29
3.4.9. Sentencia continue	29
3.5. COMENTARIOS	30
3.5.1. Tipos de comentarios	30
3.5.2. Comentarios de Linea.....	30
3.5.3. Comentarios de Bloque.....	30
3.5.4. Comentarios Javadoc	31
3.5.5. Caracteres especiales	31
3.6. VALORES EXTERNOS.....	31
3.6.1. Uso de NetBeans	32
3.6.2. Vistas de un proyecto	32
3.6.3. Directorios de un proyecto	32
3.6.4. Comandos útiles aplicables a un proyecto.....	33
3.6.5. El Debugger	33
4. INTRODUCCIÓN A OOP	35
4.1. QUÉ ES UNA CLASE	35
4.1.1. Definicion.....	35
4.1.2. Implementacion en Java	35
4.2. QUÉ ES UN OBJETO.....	35
4.2.1. Definicion.....	35
4.2.2. Implementacion en Java	36
4.3. QUE SON LOS ATRIBUTOS	36
4.3.1. Definicion.....	36
4.3.2. Atributos de Instancia.....	37
4.3.3. Atributos de Clase.....	37
4.4. QUE SON LOS MÉTODOS	38
4.4.1. Definicion.....	38
4.4.2. Métodos de Instancia.....	39
4.4.3. Métodos de Clase.....	39
4.5. ENCAPSULAMIENTO.....	40
4.5.1. Definición.....	40
4.5.2. Métodos de acceso	40
4.6. CONSTRUCTORES Y DESTRUCTORES	41
4.6.1. El constructor.....	41
4.6.2. El destructor.....	42
4.7. HERENCIA	42
4.7.1. Definición.....	42
4.7.2. Que es el Casting	43
4.7.3. Casteo Implicito (Widening Casting).....	43
4.7.4. Casteo Explicito (Narrowing Casting).....	44
4.7.5. Upcasting	44
4.8. POLIMORFISMO	45
4.8.1. Definición.....	45
4.8.2. Con redefinición.....	47
4.8.3. Sin redefinición	48

4.9. CLASE ABSTRACTA	49
4.9.1. Definición.....	49
4.9.2. Definicion.....	49
4.9.3. Utilizacion.....	49
4.10. INTERFAZ	50
4.10.1. Que es una interfaz.....	50
4.10.2. Definición.....	50
4.10.3. Utilización.....	50
4.11. PAQUETES	54
4.11.1. Que es un paquete.....	54
4.11.2. Definicion.....	54
4.11.3. Utilizacion.....	54
4.12. LA KEYWORD FINAL	55
4.12.1. Definición.....	55
4.12.2. Aplicable a atributos.....	55
4.12.3. Aplicable a métodos.....	55
4.12.4. Aplicable a clases	55
4.13. ACCESIBILIDAD	56
4.13.1. Acceso privado, palabra clave private.....	56
4.13.2. Acceso por defecto, "default" o "package"	56
4.13.3. Acceso protegido, palabra clave protected.....	57
4.13.4. Acceso publico, palabra clave public.....	59
4.13.5. Resumen de accesibilidad.....	59
5. LA INTERFAZ GRÁFICA	60
5.1. LA HISTORIA: AWT	60
5.1.1. Definición.....	60
5.1.2. Estructura de una aplicación AWT	60
5.2. LA ACTUALIDAD: SWING	61
5.2.1. Definición.....	61
5.2.2. Estructura de una aplicación Swing.....	61
5.3. SWING VS. AWT.....	62
5.3.1. Comparacion.....	62
5.3.2. Swing como negocio	62
5.4. COMPONENTES SWING: CONTENEDORES	62
5.4.1. Definición.....	62
5.4.2. JFrame	63
5.4.3. JDialog	63
5.4.4. JApplet.....	63
5.4.5. JPanel.....	63
5.5. ORGANIZACIÓN EN NETBEANS	63
5.5.1. Palette Window	64
5.5.2. Inspector Window	64
5.5.3. Properties Window.....	64
5.6. MDI DOCUMENTO DE MULTIPLES INTERFACES.....	64
5.6.1. Introducción a MDI	64
5.6.2. Pasos para construir una aplicación MDI.....	65
6. CONCEPTOS GENERALES	67
6.1. LA CLASE STRING	67
6.1.1. Definición.....	67
6.1.2. Inicializacion de un String.....	67
6.1.3. Metodos mas importantes.....	68
6.2. LA CLASE SYSTEM	68
6.2.1. Definición.....	68
6.2.2. Donde utilizarla	69
6.3. LOS WRAPPERS DE LOS TIPOS DE DATO PRIMITIVOS	69

6.3.1. Definición.....	69
6.3.2. La clase Integer.....	69
6.3.3. La clase Float.....	70
6.3.4. La clase Number	70
6.4. COMPARACIÓN ENTRE OBJETOS	70
6.4.1. El operador ==.....	70
6.4.2. El método equals().....	71
7. CONTENEDORES.....	72
7.1. QUE SON LOS CONTENEDORES	72
7.1.1. Definición.....	72
7.1.2. La interfaz Collection	72
7.2. LISTAS	73
7.2.1. La interfaz List.....	73
7.2.2. ArrayList	73
7.2.3. Vector	74
7.3. ITERADORES.....	75
7.3.1. Definición.....	75
7.3.2. Utilización.....	75
8. EXCEPCIONES	76
8.1. QUE ES UNA EXCEPCION	76
8.1.1. Definicion.....	76
8.1.2. Bloques try, catch y finally	76
8.2. TIPOS DE EXCEPCIONES	77
8.2.1. Unchecked Exceptions	77
8.2.2. Checked Exceptions	78
8.3. LA SENTENCIA “THROW”	80
8.3.1. Que es.....	80
8.3.2. Utilizacion.....	80
8.4. CREACIÓN DE EXCEPCIONES PROPIAS	80
8.4.1. La clase Exception como superclase.....	80
8.4.2. La keyword “throws”	81
9. STREAMS	82
9.1. DEFINICIÓN	82
9.1.1. Que es un Stream	82
9.1.2. Algoritmo de Lectura	82
9.1.3. Algoritmo de Escritura	83
9.2. TIPOS DE STREAMS	83
9.2.1. Organizacion.....	83
9.2.2. Streams orientados a Caracter.....	83
9.2.3. Streams orientados a Byte	84
9.3. QUE ES UN FILE STREAM	84
9.3.1. Definicion.....	84
9.3.2. Lectura de un Archivo de Texto.....	85
9.3.3. Escritura de un Archivo de Texto.....	86
9.3.4. Lectura y Escritura de Archivos Binarios	87
9.4. QUE SON LOS BUFFERS	88
9.4.1. Definicion.....	88
9.4.2. La clase BufferedReader.....	88
9.4.3. La clase BufferedWriter.....	89
9.4.4. La clase BufferedInputStream	90
9.4.5. La clase BufferedOutputStream.....	90
10. BASE DE DATOS.....	91
10.1. EL LENGUAJE SQL.....	91

10.1.1. Que es SQL.....	91
10.1.2. Donde se utiliza.....	91
10.2. MySQL COMO DATA BASE MANAGEMENT SYSTEM.....	92
10.2.1. Introduccion.....	92
10.2.2. Caracteristicas.....	92
10.3. QUÉ ES DDL?	93
10.3.1. La operacion CREATE	93
10.3.2. La operacion ALTER	93
10.3.3. La operacion DROP	94
10.4. QUÉ ES DML?	94
10.4.1. El comando SELECT	94
10.4.2. El comando INSERT	95
10.4.3. El comando UPDATE.....	95
10.4.4. El comando DELETE.....	96
11. JDBC: CONEXION CON BASE DE DATOS	97
11.1. INTRODUCCION	97
11.1.1. Que es JDBC.....	97
11.1.2. La necesidad de una libreria	97
11.2. CONEXIÓN CON LA BASE DE DATOS.....	98
11.2.1. La interfaz Connection	98
11.2.2. Construccion de un Administrador de Conexiones.....	98
11.3. COMO CONSULTAR DATOS.....	99
11.3.1. El metodo createStatement()	99
11.3.2. El metodo executeQuery()	99
11.3.3. Como realizar una consulta	100
11.4. COMO INSERTAR DATOS	101
11.4.1. El metodo createStatement()	101
11.4.2. El metodo excute()	101
11.4.3. Como realizar una insercion.....	101
11.5. COMO ACTUALIZAR DATOS	102
11.5.1. El metodo createStatement()	102
11.5.2. El metodo excute()	102
11.5.3. Como realizar una actualizacion	102
11.6. COMO ELIMINAR DATOS	103
11.6.1. El metodo createStatement()	103
11.6.2. El metodo excute()	103
11.6.3. Como realizar una eliminacion	103
11.7. TRANSACCIONES.....	104
11.7.1. Que es un DAO	104
11.7.2. Que es una transaccion	104
11.7.3. El metodo setAutoCommit().....	104
11.7.4. El metodo commit().....	105
11.7.5. El metodo rollback().....	105
11.7.6. Utilizacion de transacciones	106
11.7.7. Utilizacion de transacciones con manejo de excepciones	107
12. LABORATORIOS.....	108
12.1. LAB #1 - CONCEPTOS BASICOS DE JAVA	108
12.1.1. Ejercicio #1	108
12.2. LAB #2 - PROGRAMACION ORIENTADA A OBJETOS	109
12.2.1. Ejercicio #1	109
12.3. LAB #3 – CONCEPTOS GENERALES.....	110
12.3.1. Ejercicio #1.....	110
12.3.2. Ejercicio #2.....	110
12.4. LAB #4 - COLECCIONES	111
12.4.1. Ejercicio #1	111

<i>12.4.2. Ejercicio #2.....</i>	111
<i>12.4.3. Ejercicio #3.....</i>	112
12.5. LAB #5 - EXCEPCIONES.....	113
<i>12.5.1. Ejercicio #1.....</i>	113
12.6. LAB #6 - STREAMS.....	114
<i>12.6.1. Ejercicio #1.....</i>	114
<i>12.6.2. Ejercicio #2.....</i>	114
<i>12.6.3. Ejercicio #3.....</i>	114
<i>12.6.4. Ejercicio #4.....</i>	114
<i>12.6.5. Ejercicio #5.....</i>	115
<i>12.6.6. Ejercicio #6.....</i>	115
<i>12.6.7. Ejercicio #7.....</i>	115
<i>12.6.8. Ejercicio #8.....</i>	116
<i>12.6.9. Ejercicio #9.....</i>	116
12.7. LAB #7 - INTERFAZ GRAFICA DE USUARIO	117
<i>12.7.1. Ejercicio #1.....</i>	117
12.8. LAB #8 – ACCESO A BASE DE DATOS	122
<i>12.8.1. Ejercicio #1.....</i>	122
13. PROYECTO INTEGRADOR.....	123
<i>13.1. FASE #1 – DETECCIÓN DE CLASES Y CONSTRUCCIÓN BASE DEL PROYECTO</i>	<i>123</i>
<i>13.1.1. Requisitos</i>	<i>123</i>
<i>13.1.2. Objetivos</i>	<i>123</i>
<i>13.1.3. Especificación</i>	<i>124</i>
<i>13.2. FASE #2 – PROFESIONALIZACIÓN DE LA ORGANIZACIÓN DEL PROYECTO</i>	<i>125</i>
<i>13.2.1. Requisitos</i>	<i>125</i>
<i>13.2.2. Objetivos</i>	<i>125</i>
<i>13.2.3. Especificación</i>	<i>126</i>
<i>13.3. FASE #3 – CONSTRUCCION DE LA INTERFAZ GRAFICA DE USUARIO</i>	<i>128</i>
<i>13.3.1. Requisitos</i>	<i>128</i>
<i>13.3.2. Objetivos</i>	<i>128</i>
<i>13.3.3. Especificación</i>	<i>129</i>
<i>13.3.4. Especificación – BONUS!</i>	<i>132</i>
<i>13.4. FASE #4 – DETERMINACIÓN DE LA NAVEGACIÓN</i>	<i>133</i>
<i>13.4.1. Requisitos</i>	<i>133</i>
<i>13.4.2. Objetivos</i>	<i>133</i>
<i>13.4.3. Especificación</i>	<i>134</i>
<i>13.4.4. Especificación – BONUS!</i>	<i>134</i>
<i>13.5. FASE #5 – VALIDACIÓN Y MANEJO DE ERRORES</i>	<i>136</i>
<i>13.5.1. Requisitos</i>	<i>136</i>
<i>13.5.2. Objetivos</i>	<i>136</i>
<i>13.5.3. Especificación</i>	<i>137</i>
<i>13.5.4. Especificación - BONUS!</i>	<i>137</i>
<i>13.6. FASE #6 – LECTURA DE RECURSOS ADICIONALES</i>	<i>138</i>
<i>13.6.1. Requisitos</i>	<i>138</i>
<i>13.6.2. Objetivos</i>	<i>138</i>
<i>13.6.3. Especificaciones</i>	<i>139</i>
<i>13.7. FASE #7 – UTILIZACIÓN DE LISTAS</i>	<i>140</i>
<i>13.7.1. Requisitos</i>	<i>140</i>
<i>13.7.2. Objetivos</i>	<i>140</i>
<i>13.7.3. Especificaciones</i>	<i>141</i>
<i>13.7.4. Especificación - BONUS!</i>	<i>141</i>
<i>13.8. FASE #8 – CONEXION CON BASE DE DATOS</i>	<i>142</i>
<i>13.8.1. Requisitos</i>	<i>142</i>
<i>13.8.2. Objetivos</i>	<i>142</i>
<i>13.8.3. Especificaciones</i>	<i>143</i>
<i>13.8.4. Especificación - BONUS!</i>	<i>144</i>

13.9. FASE #9 – INTEGRACION CON INTERFAZ GRAFICA DE USUARIO	145
13.9.1. Requisitos	145
13.9.2. Objetivos	145
13.9.3. Especificaciones	146
13.9.4. Especificación - BONUS!	147



educaciónIT

1. Introducción a Java

1.1. Qué es Java

1.1.1. Introducción

JAVA es una tecnología pensada para desarrollo de aplicaciones de gran envergadura, altamente escalables, de gran integración con otras tecnologías y sumamente robustas.

Sus principales características son presentadas a continuación:

1.1.2. Lenguaje orientado a objetos

Respeta el paradigma de orientación a objetos, permitiendo utilizar los fundamentos del mismo:

- ✓ Herencia
- ✓ Polimorfismo
- ✓ Abstracción
- ✓ Encapsulamiento

Todos estos conceptos serán presentados más adelante.

1.1.3. Sintaxis basada en C/C++

Aporta gran simplicidad ya que es una de las formas de escribir código más reconocidas y difundidas, y permite incorporar rápidamente a los programadores que conocen este lenguaje.

1.1.4. Multiplataforma

Significa que su código es portable, es decir se puede transportar por distintas plataformas. De esta manera es posible codificar una única vez una aplicación, y luego ejecutarla sobre cualquier plataforma y/o sistema operativo.

1.1.5. Manejo automático de memoria

No hay que preocuparse por liberar memoria manualmente ya que un proceso propio de la tecnología se encarga de monitorear, y por consiguiente eliminar el espacio ocupado que no está siendo utilizado. El proceso encargado de realizar este trabajo se denomina Garbage Collector.

1.1.6. Evolución permanente

La tecnología está en constante evolución debido a la gran cantidad de "consumidores" que poseen, JAVA es uno de los lenguajes más utilizados en el mundo, y SUN pretende estar a la altura de la situación ofreciendo constantemente nuevas entregas.

1.2. Organización

La tecnología esta organizada en tres grandes áreas bien definidas:

1.2.1. JME (Mobile / Wireless)

Esta área tiene como objetivo el desarrollo de aplicaciones móviles, tales como GPS, Handhelds (por ejemplo la conocida Palm), celulares y otros dispositivos móviles programables. JME significa Java Micro Edition.

1.2.2. JSE (Core / Desktop)

Esta área tiene como objetivo el desarrollo de aplicaciones de escritorio, similares a las aplicaciones tipo ventanas creadas con Visual Basic o Delphi. Incluye la funcionalidad básica del lenguaje como manejo de clases, colecciones, entrada/salida, acceso a base de datos, manejo de sockets, hilos de ejecución, etc. JSE significa Java Standard Edition.

1.2.3. JEE (Enterprise / Server)

Esta área tiene como objetivo el desarrollo de aplicaciones empresariales, de gran envergadura. Contempla ambientes web, como los ambientes manejados por servidores de aplicación. Las tecnologías principales incluidas en esta área son Servlets, JSP y EJB, entre otras. JEE significa Java Enterprise Edition.

1.3. La historia

1.3.1. El comienzo

En el año 1990 nace Java, bajo el diseño y la implementación de la empresa *Sun Microsystems*. El padre-fundador de la tecnología es el *James Gosling*, a través de una filial dentro de Sun llamada *First Person Inc.*

Gosling tuvo la visión inicial de construir una lenguaje de programación capaz de ejecutar su código sobre cualquier set de instrucciones, de distintos procesadores. Inicialmente el proyecto apuntó a la programación unificada de distintos electrodomésticos, es decir programar una sola vez y que el programa generado fuera útil para cualquier dispositivo.

El proyecto inicial de Java fue técnicamente un éxito, aunque comercialmente no tuvo el rendimiento esperado, y debió ser relegado unos años.

1.3.2. Aparición de Internet

En el año 1993, Internet da el gran salto, y se convierte de una interfaz textual a una interfaz gráfica.

Java ve una oportunidad y entra fuertemente a internet con los Applets, pequeños programitas construidos en Java – con todos sus beneficios – capaces de ejecutarse dentro de un navegador. Es aquí donde Java comienza a dar sus primeros pasos firmes como lenguaje a difundir masivamente. En el año 1995, el navegador Netscape Navigator comienza formalmente a soportar los Applets Java.

Adicionalmente, el lenguaje podía adaptarse fácilmente a las múltiples plataformas, con lo cual surge una de las primeras aplicaciones multiplataformas más conocidas: WebRunner (hoy HotJava), un navegador multiplataforma construido en Java.

1.3.3. Por qué el nombre JAVA

Inicialmente la intención fue nombrar al lenguaje de programación con el nombre de Oak, pero este ya estaba registrado. La leyenda cuenta que una visita a la cafetería le dio rápida solución al problema.

En las confiterías norteamericanas hay un café denominado Java, en el cual está inspirado el nombre del lenguaje de programación. El logotipo de Java es justamente una taza café.

educationIT

2. Desarrollo, compilacion y ejecucion

2.1. Organizacion

2.1.1. El Java Development Kit (JDK)

El Java Development Kit es el kit de desarrollo propuesto por Sun Microsystems para realizar desarrollos en JAVA. Se puede bajar de forma gratuita de la pagina <http://www.java.sun.com>

El kit incluye herramientas de desarrollo tales como un compilador, un debugger, un documentador para documentar en forma casi automatica una aplicacion, un empaquetador para crear archivos de distribucion, y otras herramientas mas.

El kit no incluye un entorno de desarrollo interactivo (o IDE) como pueden ser Netbeans, Jdeveloper o Eclipse.

2.1.2. El compilador

El compilador viene incluido como una herramienta dentro de la JDK, en el sistema operativo Windows viene presentado como *javac.exe*

El compilador transforma los archivos de codigo fuente de java, es decir los archivos de texto con extension .java, en archivo compilados, tambien denominados *bytecode*. Los archivos compilados tiene la extension .class, y son archivos binarios.

2.1.3. El Java Runtime Environment (JRE)

Java Runtime Environment es el ambiente de ejecucion de Java, y tambien esta incluido en la JDK. Tiene como componentes mas importantes a la Java Virtual

Machine y a las class libraries, que son las que contienen las clases base del lenguaje de programacion JAVA.

El JRE se distribuye tambien en forma independiente, es decir sin la JDK, ya que cuando es necesario desplegar una aplicacion hecha en JAVA en el cliente, no es necesario instalarle herramientas que son propias del proceso de desarrollo, como ser el compilador, empaquetador, documentador, y otros.

Sin una JRE instalada no es posible ejecutar una aplicacion construida en JAVA.

En Windows, el comando para invocarlo es el *java.exe*

2.2. La Java Virtual Machine (JVM)

2.2.1. Que es

La Java Virtual Machine viene incluida dentro de la Java Runtime Environment, y tiene como principal objetivo la ejecucion de codigo JAVA compilado, es decir de los archivo .class

La JVM se encarga de interpretar el bytecode y convertirlo a codigo nativo en tiempo de ejecucion, lo cual hace que la ejecucion sea un poco mas lenta pero garantiza la portabilidad, es decir que el lenguaje sea multiplataforma. De esta manera el codigo compilado JAVA se puede ejecutar en cualquier plataforma (arquitectura + sistema operativo) que tenga instalada el JRE.

"Write once, run anywhere" es la politica desde el primer dia de JAVA, es decir construir la aplicacion una vez y ejecutarla en "cualquier lado".

2.2.2. La variable de entorno CLASSPATH

La variable de entorno CLASSPATH se utiliza para referenciar el directorio donde estaran ubicadas todas las clases construidas en JAVA, para que el JRE al ejecutar una clase sepa donde ubicar el resto de las clases o archivos empaquetados que contienen clases.

3. Sintaxis y Semántica de Java

3.1. Definición de variables

3.1.1. Identificador

A la hora de nombrar las variables es importante tener en cuenta que no pueden comenzar con un número ni utilizar caracteres "%" o "*" o "@" por que están reservados para otras operaciones.

Pueden comenzar con los caracteres "_" o "\$", aunque la especificación no recomienda el uso del carácter "\$" en variables.

3.1.2. Tipos de variables

Las variables pueden ser declaradas con tipo de datos primitivos (int, long, float, double, etc) o también como tipos, pertenecientes a alguna clase, como por ejemplo la clase String para trabajar con cadenas de caracteres.

3.1.3. Declaración y definición

Un variable del tipo entero puede ser declarada de la siguiente forma:

```
int var;
```

Si se desea declararla y asignarle un valor, en este caso resulta necesario definirla.

La definición se realiza de la siguiente manera:

```
int var = 200;
```

3.1.4. Vectores

Para manejar un conjunto de valores se utilizan los vectores, que agrupan una serie de valores en una única variable. Para trabajar con arreglos (vectores) es necesario determinar el tipo de información que contendrá y la longitud – o cantidad de elementos – que tendrá el propio vector.

Si, por ejemplo, resulta necesario construir un vector para almacenar números enteros, se utilizará un vector del tipo entero de, por ejemplo, 10 posiciones, que luego será necesario llenarlo con valores. Lo dicho anteriormente, se realiza de la siguiente manera:

```
int[] vector = new int[10];  
vec[0] = 150;  
vec[1] = 300;  
vec[2] = 500;
```

Si se conocen de antemano los valores que contendrá el vector, podemos armar el vector de una forma más resumida, presentada a continuación:

```
int[] vector = {150, 300, 500, 4, 5, 6};
```

Los vectores cuentan con la posibilidad de solicitarles qué longitud poseen, esto se realiza a través del atributo *length*

Es importante recordar que en JAVA – a diferencia de otros lenguajes de programación – la primera posición de un vector es la posición 0 (cero).

3.2. Tipos de dato primitivos

3.2.1. boolean

El tipo de dato boolean se utiliza para almacenar las palabras claves true o false, es decir verdadero o falso. Ocupan 1 byte en memoria.

3.2.2. char

El tipo de dato char se utiliza para almacenar un solo caracter, del tipo Unicode. Ocupan 2 bytes en memoria.

3.2.3. byte

El tipo de dato byte es un tipo de dato numerico y entero, se utiliza para almacenar numeros comprendidos entre -128 y 127. Ocupa 1 byte de memoria.

3.2.4. short

El tipo de dato short es un tipo de dato numerico y entero, se utiliza para almacenar numeros comprendidos entre -32768 y 32767. Ocupa 2 bytes de memoria.

3.2.5. int

El tipo de dato int es un tipo de dato numerico y entero, se utiliza para almacenar numeros comprendidos entre -2.147.483.648 y 2.147.483.647. Ocupa 4 bytes de memoria.

3.2.6. long

El tipo de dato long es un tipo de dato numerico y entero, se utiliza para almacenar numeros comprendidos entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807. Ocupa 8 bytes de memoria.

3.2.7. float

El tipo de dato float es un tipo de dato numerico y de punto flotante, se utiliza para almacenar numeros comprendidos entre -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38. Ocupa 4 bytes de memoria, y maneja entre 6 y 7 cifras decimales.

3.2.8. double

El tipo de dato float es un tipo de dato numerico y de punto flotante, se utiliza para almacenar numeros comprendidos de 1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308. Ocupa 8 bytes de memoria, y maneja unas 15 cifras decimales.

3. Operadores

3.3.1. Operadores Aritméticos

Los operadores aritmeticos son utilizados para realizar operaciones aritmeticas, estos son la suma (+), resta (-), multiplicación (*), división (/)

```
int suma = 100 + 500;  
int resta = 100 - 35;  
int multiplicacion = 10 * 5;  
int division = 10 / 2;
```

Tambien se encuentra disponible el resto de la división (%)

```
int resto = 10 % 3; // el resto de la division es en este caso 1
```

3.3.2. Operadores de Asignación

Los operadores de asignacion se utilizan para asignar valores a las variables. En la tabla siguiente se presentan los operadores, junto con un ejemplo de utilizacion y su expresion equivalente en una forma extendida:

Operador	Utilización	Expresión equivalente
=	op1 = op2	op1 = op2
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

3.3.3. Operador instanceof

El operador instanceof permite saber si un objeto pertenece o no a una determinada clase. Su utilizacion es la siguiente:

```
nombreObjeto instanceof nombreClase
```

Retorna true o false según el objeto pertenezca o no a la clase

3.3.4. Operador condicional ?:

El operador condicional “?:” es conocido tambien como inline-if. Se utiliza de la siguiente manera:

```
String respuesta = (numero1 > numero2) ? "SI" : "NO";
```

El operador evalúa expresion booleana “numero1 > numero2” y retorna “SI” en caso afirmativo y “NO” en otro caso.

3.3.5. Operadores incrementales y decrementales

Los operadores de incremento (++) y decremento (--) se utilizan para sumar o restar una unidad de una variable determinada.

Su forma de uso es la siguiente:

```
int numero = 100;
numero++; // numero toma el valor 101
numero++; // numero toma el valor 102
numero--; // numero toma el valor 101
```

3.3.6. Operadores relacionales

Los operadores relacionales se utilizan para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor. El resultado de estos operadores es siempre un valor booleano (true o false).

A continuación se presentan los posibles operadores junto con su utilización:

Operador	Utilización	El resultado es true
>	op1 > op2	si op1 es mayor que op2
>=	op1 >= op2	si op1 es mayor o igual que op2
<	op1 < op2	si op1 es menor que op2
<=	op1 <= op2	si op1 es menor o igual que op2
==	op1 == op2	si op1 y op2 son iguales
!=	op1 != op2	si op1 y op2 son diferentes

3.3.7. Operadores lógicos

Los operadores lógicos se utilizan para construir expresiones lógicas, combinando valores lógicos (true y/o false). En ciertos casos el segundo operando no se evalúa porque no resulta necesario.

A continuación se presentan los posibles operadores junto con su utilización:

Operador	Utilización	Resultado
&& evalúa op2	op1 && op2	true si op1 y op2 son true. Si op1 es false ya no se evalúa op2
 evalúa op2	op1 op2	true si op1 u op2 son true. Si op1 es true ya no se evalúa op2
!	! op	true si op es false y false si op es true
& 	op1 & op2 op1 op2	true si op1 y op2 son true. Siempre se evalúa op2 true si op1 u op2 son true. Siempre se evalúa op2

3.3.8. Operador concatenación de caracteres

El operador “+” se utilizan para realizar sumas, pero tambien esta definido para poder concatenar cadenas de caracteres.

Su utilizacion es la siguiente:

```
String respuesta = "Se han comprado " + variableCantidad + " unidades";
```

3.3.9. Operadores aplicables a bits

Los operadores aplicables a bits se utilizan manipular a nivel bits un valor, donde se pueden realizar operaciones logicas como tambien desplazamientos de bits a izquierda / derecha.

A continuacion se presentan los posibles operadores junto con su utilizacion:

Operador	Utilización	Resultado
>> op2	op1 >> op2	Desplaza los bits de op1 a la derecha una distancia op2
<< op2	op1 << op2	Desplaza los bits de op1 a la izquierda una distancia op2
&	op1 & op2	Operador AND a nivel de bits
	op1 op2	Operador OR a nivel de bits
^ operando es 1)	op1 ^ op2	Operador XOR a nivel de bits (1 si sólo uno de los operando es 1)
~ bit)	~ op2	Operador complemento (invierte el valor de cada bit)

3.3.10. Clasificación

Los operadores pueden ser clasificados segun la cantidad de operandos que utilicen.
Las categorias son las siguientes:

Operadores Unarios

Son aquellos operadores que necesitan un único operando para realizar la operacion, por ejemplo el operador incremento (++) o el operador negación (!)

Operadores Binarios

Son aquellos operadores que necesitan dos operandos para realizar la operacion, por ejemplo, el operador suma (+) o el operador AND (&&)

Operadores Ternarios

Son aquellos operadores que necesitan tres operandos para realizar la operacion, el unico operador ternario que posee Java es ?:

3.4. Estructuras de control de flujo

3.4.1. Introducción

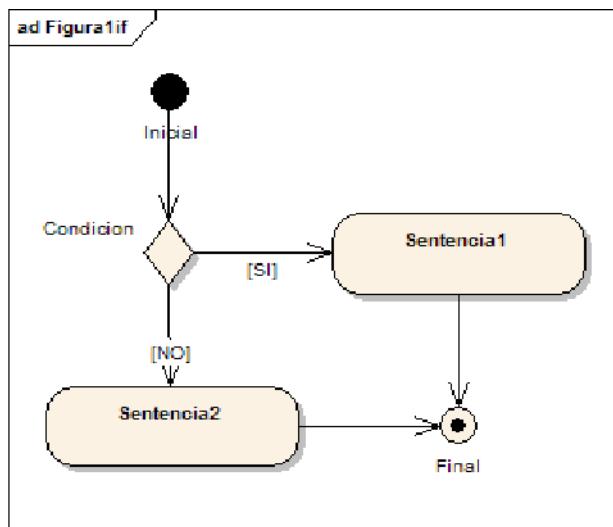
Java igual que C y C++ utiliza las estructuras de control if-else, while, do-while, y como selector de opciones la estructura switch.

Las estructuras de control se rigen en su ejecución a partir de una condición del tipo boolean, es decir, el resultado de operadores lógicos o el resultado de un método que retorne un valor de verdad, verdadero o falso.

3.4.2. Bifurcación if-else

La bifurcación if-else es la más simple de las estructuras:

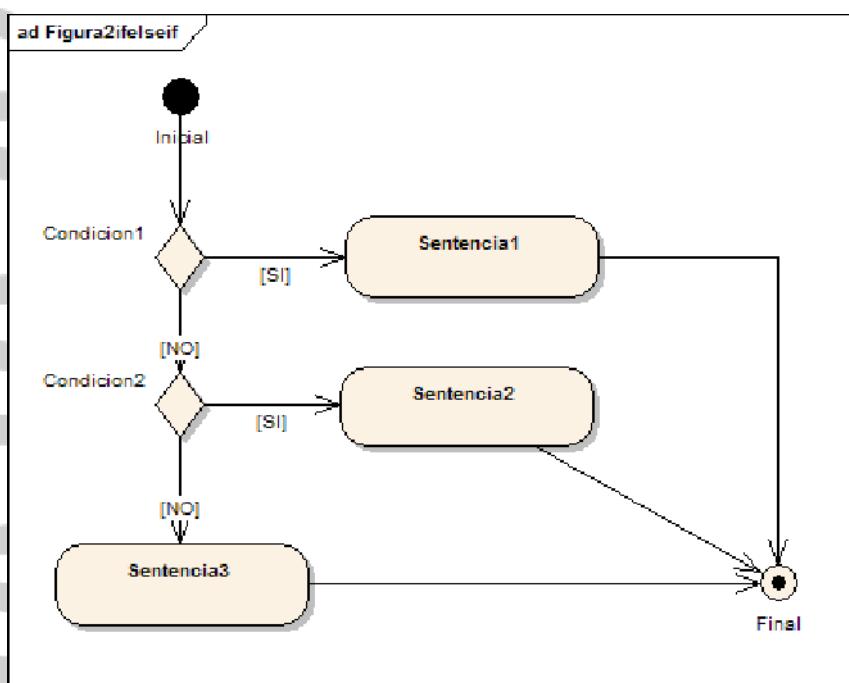
```
if (expresionBooleana) {
    sentencias1;
} else {
    sentencias2;
}
```



3.4.3. Bifurcación if-else-if-else

En su forma genérica puede escribirse como:

```
if( expresionBooleana) {
    sentencias1;
} else if( expresionBooleana2) {
    sentencias2;
} else{
    sentencias3;
}
```

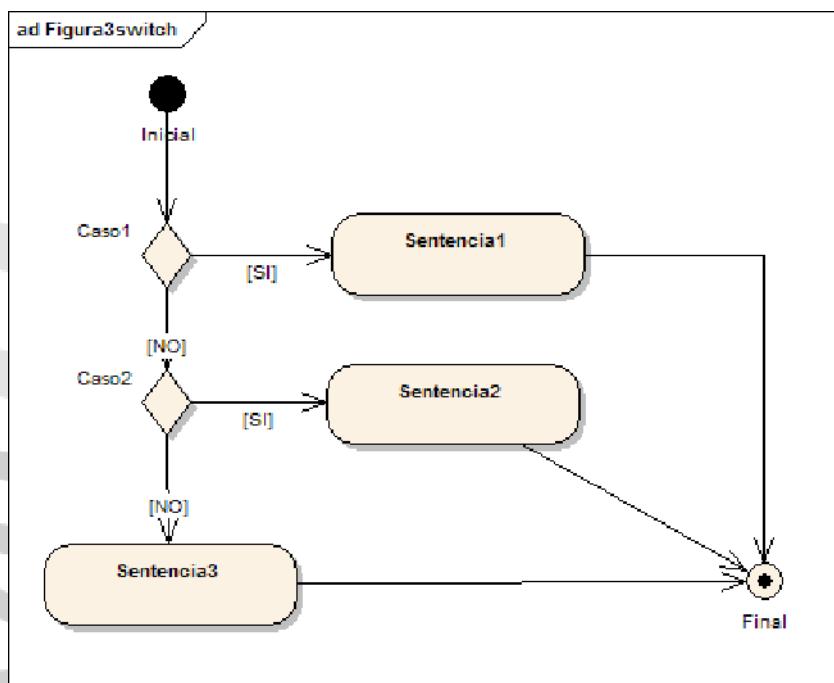


3.4.4. Bifurcacion switch

El switch es clasificado como selector de sentencias de ejecucion evaluando una expresion integral.

```
switch(expresionIntegral) {  
  
    case valor1: sentencia1; break;  
    case valor2: sentencia2; break;  
    case valor3: sentencia3; break;  
    default: sentencia4; break;  
}
```

La ejecucion se resuelve evaluando la expresion integral y se busca desde la primera expresion hacia la expresion por defecto, cual etiqueta de evaluacion de caso resuelve ser igual en valor, al valor de la expresion integral. En caso de no ser ninguna de las etiquetas igual al valor de la expresion se ejecutan las sentencias de la opcion default. En caso de no agregar la sentencia break, se ejecutara de continuo una etiqueta tras otra hasta encontrar un break o salir del alcance definido por el switch.

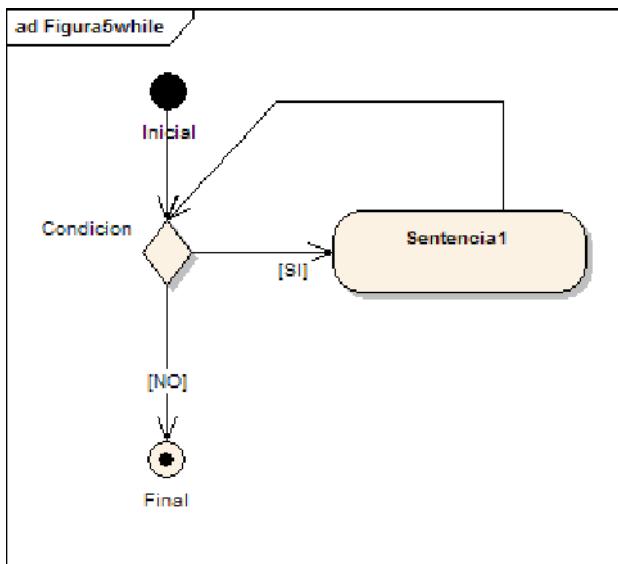


3.4.5. Bucle while

Las sentencias de bucles, son clasificadas como sentencias de iteracion. Las sentencias seran ejecutadas en la medida en que sea verdadera la expresion booleana, solo dejaran de ejecutarse cuando esta expresion sea falsa. La expresion puede nunca ser verdadera, en ese caso nunca se ejecutaran las sentencias que estan dentro del ciclo.

```

while(expresionBooleana){
    sentencias;
}
  
```

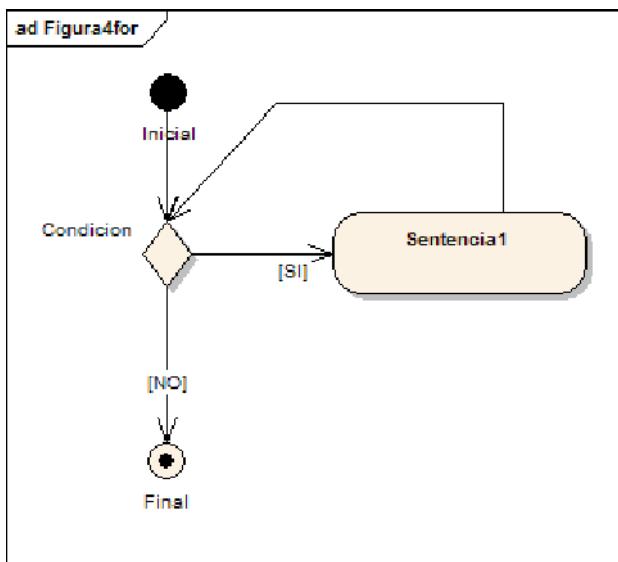


3.4.6. Bucle for

El bucle for permite en un principio una inicializacion, luego itera comenzando con una evaluacion de la expresion booleana y por ultimo realiza algun tipo de paso a proximo, o reduccion de la complejidad del algoritmo, en camino hacia convertir la expresion booleana en falso. Se ejecutan las sentencias, mientras la expresion booleana se mantenga en un estado de verdadero por cada paso.

```

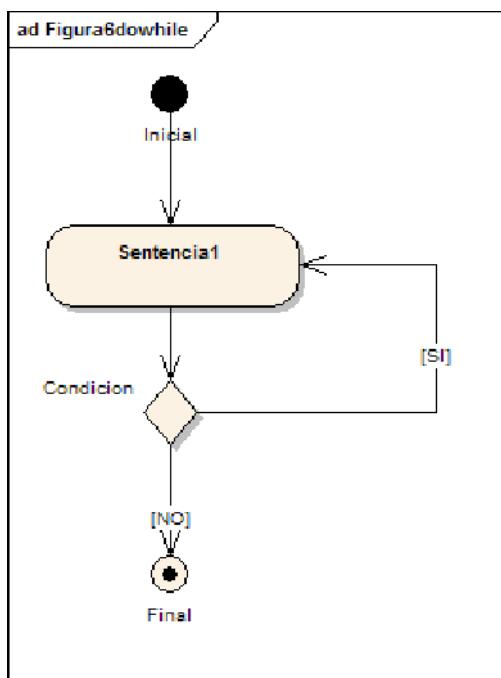
for (inicializacion; expresionBooleana; postAccion) {
    sentencias;
}
  
```



3.4.7. Bucle do-while

La única diferencia con el bucle while es que el bucle do-while ejecuta al menos una vez las sentencias, aunque desde el principio la expresión booleana sea falsa.

```
do{  
    sentencias;  
} while(expresionBooleana);
```



3.4.8. Sentencia break

La sentencia break puede encontrarse dentro del alcance de un ciclo. Generalmente se encuentra dentro de una bifurcación, tal que si una condición booleana se cumple se ejecutara el break y saldra del alcance del ciclo.

Si bien se escribe de la misma forma que el break del switch, su funcionamiento es distinto dado que no salta al siguiente paso, sino que corta la ejecución del ciclo, sin importar el estado de verdad o falsedad de la expresión booleana, y lo deja sin efecto.

3.4.9. Sentencia continue

La sentencia continue puede utilizarse en ciclos, y permite la omisión de la ejecución de las sentencias definidas justo luego de la sentencia, dado que salta la ejecución hasta la próxima iteración. Generalmente se encuentra dentro de una bifurcación, de forma tal que una expresión booleana permita saltar las sentencias siguientes, hasta la próxima iteración del ciclo.

Debe tomarse muy en cuenta que antes de la ejecución de un continue debe haberse logrado reducir la complejidad del algoritmo, ya que al saltar a la próxima

iteracion, si la expresion booleana sigue en la misma condicion y no se ha reducido la complejidad, puede generarse un ciclo infinito de ejecucion.

3.5. Comentarios

3.5.1. Tipos de comentarios

Existen tres tipos de comentarios. Los comentarios de linea, de bloque y los javadoc.

3.5.2. Comentarios de Linea

Los comentarios de linea se utilizan para comentar en una sola linea, una sentencia, explicar la definicion de una variable, o un paso a paso del codigo que se esta observando. Un ejemplo de comentario de linea es el siguiente:

```
int num = 5; // este comentario me permite decir que esta es una cota del sistema.
```

3.5.3. Comentarios de Bloque

Los comentarios de bloque permiten comentar un bloque de codigo o simplemente agregar varias lineas de texto para explicar algo que solo debe ser visto en el momento de la programacion. El modo de utilizarlo es el siguiente:

```
/*
   En este comentario se puede explicar varias cosas.

   Algunas condiciones que se deben cumplir para continuar con la
   ejecucion del codigo.

   Algun codigo alternativo en caso de estar testeando alguna version,
   o modificando el programa.

   O simplemente dejar alguna advertencia sobre el codigo que se esta
   observando.

*/
```

3.5.4. Comentarios Javadoc

Los comentarios del tipo javadoc permiten documentar de una forma mas util, mas publica nuestro programa. Pueden ir en cualquier sector de las clases, los packages, los atributos, los metodos. Una vez que esta documentacion se ha escrito explicando o definiendo una clase un paquete o los atritubos o metodos, con la ejecucion de la herramienta javadoc se obtiene un sitio con la api del programa, donde se encontraran las explicaciones que se fueron agregando en las clases. Todos estos comentarios son vinculados como en la api de java standard.

3.5.5. Caracteres especiales

La representacion de caracteres especiales como el salto de linea o la tabulacion, se logran a partir de la barra (\). Los caracteres especiales mas utilizados son:

- ✓ \n → Nueva linea
- ✓ \t → Tabulador
- ✓ \' → Comilla simple
- ✓ \" → Comilla doble

3.6. Valores externos

Desde la linea de comandos se pueden pasar parametros al programa, estos llegaran en forma de arreglo de cadena de caracteres al metodo main. Estos parametros pueden ser utilizados para generar configuraciones de corrida o de instalacion del programa.

En el NetBeans, se puede configurar en el proyecto configurado como principal, desde las propiedades del proyecto, running project, arguments.

Este mismo metodo desde la linea de comandos seria:

```
java -classpath aplicacion.jar ar.com.educacionIT.Programa 1000 2000 3000
```

Y el programa quedaría de la siguiente forma:

```
public class Programa{  
    public Programa(){  
    }  
    public static void main(String[] args){  
        for(int i = 0; i<args.length; i++){  
            System.out.println("Argumento Indice:" + i + " Valor:" + args[i]);  
        }  
    }  
}
```

En el resultado de la salida de este programa, imprime:

```
Argumento Indice: 0 Valor:1000  
Argumento Indice: 1 Valor:2000  
Argumento Indice: 2 Valor:3000
```

3.6.1. Uso de NetBeans

3.6.2. Vistas de un proyecto

Las vistas son distintas formas de ver un proyecto. Project View es una vista lógica que representa al proyecto como paquetes, clases, métodos y atributos. FilesView es una vista física de los archivos y directorios que conforman al proyecto.

3.6.3. Directorios de un proyecto

El NetBeans genera la siguiente estructura de directorios para manejar el desarrollo de un proyecto:

```
\src, contiene el código fuente, los archivos Clase.java  
\build, se genera en la compilación del proyecto y contiene el código compilado, los Clase.class
```

\ dist, es el directorio donde se alojan los archivos para distribuir, tales como archivos miprograma.jar, o miaplicacionweb.war de forma tal que sea simple llevarlas a un ambiente productivo.

\ nbproject, contiene archivos propios de la administracion del proyecto de NetBeans.

3.6.4. Comandos útiles aplicables a un proyecto

Al seleccionar la vista de Project View, se puede hacer un click derecho sobre el proyecto y apareceran los comando mas importantes, entre ellos encontraremos:

Build Project, compila el proyecto y genera el archivo aplicacion.jar

Clean Project, elimina todos los archivos compilados de la version anterior del proyecto, dejando solo el codigo fuente.

Run Project, ejecuta el proyecto desde la clase principal preconfigurada, de no haber una preconfigurada mostrara las opciones de las clases con metodos main definidos.

Debug Project, se utiliza para debuguear el proyecto, hacer una corrida paso a paso viendo el contenido de las variables.

Set Main Project, configura el proyecto como el proyecto principal, entre varios proyectos.

Close Project, cierra el proyecto.

Properties, permite visualizar las propiedades del proyecto.

3.6.5. El Debugger

Herramienta que se utiliza para examinar el programa en busca de errores y depurarlos, en un paso a paso por las lineas del codigo fuente de toda la aplicacion.

Las partes mas importantes son:

Utilizacion de Breakpoints, se coloca un breakpoint en la linea de codigo donde se quiere detener la ejecucion del programa para evaluar el estado de las variables.

Utilizacion de Watches, se utiliza para visualizar el valor de variables que uno desee y expresiones del codigo en ejecucion.

Visualizacion de Local Variables, se utiliza para visualizar valores de las variables locales al codigo en ejecucion.

educationIT

4. Introducción a OOP

4.1. Qué es una clase

4.1.1. Definicion

Una clase es una agrupación de reglas de negocio, o representación de un concepto de la vida real. Es una plantilla, con reglas, para armar un objeto. Esta formada por atributos, que definen un estado particular, y métodos, que definen su uso o comportamiento.

4.1.2. Implementacion en Java

Ejemplos de cómo codificar diferentes clases en Java:

```
public class Persona{  
}  
  
public class Puerta{  
}  
  
public class Auto{  
}
```

4.2. Qué es un objeto

4.2.1. Definicion

Un objeto es una instancia de una clase. Como ejemplo, una instancia de la clase Persona, podría ser una persona llamada Mario. Puede tenerse varias instancias de una misma clase, pero una única clase de Persona, una persona Mario, y una persona María, con ambas instancias de la clase Persona.

4.2.2. Implementacion en Java

Ejemplos de cómo codificar diferentes objetos en Java:

```
Persona mario = new Persona();
Persona maria = new Persona();
```

4.3. Que son los Atributos

4.3.1. Definicion

Los atributos determinan el estado que tiene un objeto. Estos pueden ser tipos de datos primitivos u objetos de cualquier clase.

Ejemplo:

La clase Persona tiene como atributos la altura, el color de ojos y la edad.

Ejemplo en Java:

Definicion

```
public class Persona{
    public int altura;
    public String colorDeOjos;
    public int edad;
}
```

Utilización:

```
Persona mario = new Persona();
mario.altura = 183;
mario.colorDeOjos = "marrones";
mario.edad = 40;
```

4.3.2. Atributos de Instancia

Los atributos de instancia son aplicables a un solo objeto. Determinan el estado en el que se encuentra un objeto.

Ejemplo:

El atributo altura en la clase Persona, debido a que cada persona tendrá su propia altura.

Ejemplo en Java:

```
public class Persona{  
    public int altura;  
}
```

Luego:

```
Persona mario = new Persona();  
mario.altura = 184;  
Persona maria = new Persona();  
maria.altura = 152;
```

4.3.3. Atributos de Clase

Un atributo de clase es un atributo común a todos los objetos instanciados de la clase. Al estar definido en la clase no hace falta instanciar la clase para utilizarlo. Las constantes se suelen definir como atributos de clase.

Ejemplo:

El atributo cantidadDeOjos en la clase Persona, debido a que todas las instancias de la clase persona tendrán igual cantidad de ojos.

Ejemplo en Java:

```
public class Persona{  
    public static int cantidadDeOjos = 2;  
}
```

Por lo tanto, en su uso:

```
Persona mario = new Persona();
Persona maria = new Persona();
mario.cantidadDeOjos // es igual a 2
maria.cantidadDeOjos // es igual a 2
Persona.cantidadDeOjos // es igual a 2
```

y este 2 es siempre el mismo, no se instancia un cantidadDeOjos para cada objeto, siempre es el mismo 2 de cantidadDeOjos.

Si modiflico esta cantidad de ojos, se modifica para todas las instancias del objeto:

```
Persona.cantidadDeOjos = 3;
```

entonces

```
mario.cantidadDeOjos // es igual a 3
maria.cantidadDeOjos // es igual a 3
```

4.4. Que son los Métodos

4.4.1. Definicion

Los metodos determinan el comportamiento y la responsabilidad que tendran las clases. Se definen metodos que representen como se van a utilizar las clases. Este como se van a utilizar las clases significa que debemos representar el uso que se les da a las clases en la logica de negocios de la vida real, es decir, a una puerta se le pedira que se abra y se cierre, por lo tanto se le agregaran metodos abrir() y cerrar().

4.4.2. Métodos de Instancia

Los métodos de instancia, como su nombre lo indica, son aplicables a una instancia de la clase en particular. Es decir, que un método de instancia trabaja sobre el estado actual de la instancia, y para cada instancia tendrá un resultado distinto, por ejemplo, el hecho de comer permitirá a la instancia mario estar satisfecho, mientras que la instancia maria estará insatisfecha hasta que también se le aplique el método comer.

Ejemplo en Java:

```
public class Persona{  
    public void comer(int cantidadDeRaciones){  
        // Esto es un comentario. Aquí va la definición del método.  
    }  
}
```

4.4.3. Métodos de Clase

Los métodos de clase son un comportamiento común a todas las instancias que pertenecen a la misma clase. Al ser un método de clase, no hace falta instanciar un objeto de la clase para utilizarlo. Estos métodos no hablan del estado actual de la clase, sino solo de un comportamiento genérico de la clase, de un procedimiento que solo utiliza los parámetros de entrada o las variables estáticas. Son para un uso particular y es bien visible cuando un método debe ser estático.

Ejemplo en Java:

```
public class Persona{  
    public static int obtenerAlturaMaxima(){  
        // Esto es un comentario. Aquí va la definición del método.  
    }  
}
```

4.5. Encapsulamiento

4.5.1. Definición

El encapsulamiento habla del modo de ocultar como ha sido implementado el estado, los atributos, de un objeto. Se accede a este estado a travez de los metodos publicos, es decir su interfaz publica. Una buena practica es hacer las validaciones correspondientes a los posibles estados del objeto, en estos metodos, de modo tal de mantener al objeto en un estado consistente.

Tambien se lo llama "information hiding". De la misma forma podemos respetar el encapsulamiento si se tiene, en la clase Auto, un atributo privado velocidad, que sea privado, luego el unico modo de modificar la velocidad es a travez del metodo acelerar() y frenar(), es decir que esta encapsulada la velocidad, y solo se la modifica por los metodos acelerar() y frenar(), no se puede cambiar la velocidad de ningun otra forma.

Ejemplo en Java:

```
public class Persona{  
    // Atributos  
    private int altura;  
  
    // Metodos  
    public int getAltura(){  
        return this.altura;  
    }  
  
    public void setAltura(int unaAltura){  
        this.altura = unaAltura;  
    }  
}
```

4.5.2. Métodos de acceso

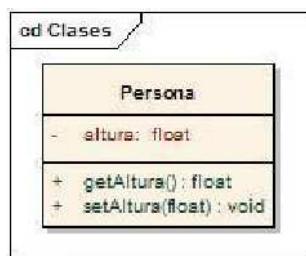
Los metodos de acceso son el medio de acceder a los atributos privados del objeto. Son metodos publicos del objeto.

El "getter"

El metodo para acceder a los atributos en forma de solo lectura se los denomina "getters". Son los metodos que retornan el valor de los atributos. El NetBeans, como la mayoria de los entornos de desarrollo, permite generarlos de forma automatica.

El "setter"

El metodo para acceder a los atributos en forma de escritura se los denomina "setters". Son los metodos que establecen el valor de los atributos. Tambien se los genera de forma automatica en los entornos de desarrollo.



4.6. Constructores y Destructores

4.6.1. El constructor

Los constructores son metodos pertenecientes a la clase. Se utilizan para construir o instanciar una clase. Pueden haber varios constructores, de acuerdo a las necesidades del usuario.

Ejemplo en Java:

```

public class Persona{
    // Atributos
    private int altura;
    // Constructores
    public Persona(){
    }

    public Persona(int unaAltura){
        this.altura = unaAltura;
    }
}

```

4.6.2. El destructor

El destructor se utiliza para destruir una instancia de una clase y liberar memoria. En Java no hay destructores, ya que la liberación de memoria es llevada a cabo por el Garbage Collector cuando las instancias de los objetos quedan desreferenciadas. El método `dispose()` de cada objeto se llama previo a ser "recolektado".

4.7. Herencia

4.7.1. Definición

La herencia se produce a partir de dos clases relacionadas, es decir una subclase que hereda los atributos y los métodos de la superclase. La jerarquía de clases, o árbol de herencia se lleva a cabo a partir de esta herencia entre objetos. Se puede entender como la relación "es un" entre dos clases. Si la subclase C2 hereda de la superclase C1, se dice que C2 es un C1.

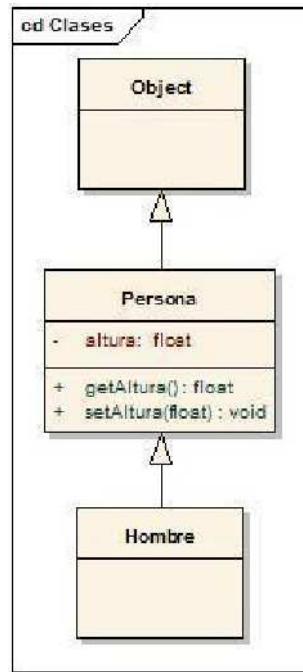
Ejemplo:

La clase Hombre hereda de la clase Persona, entonces podemos decir que Hombre "es una" Persona.

Ejemplo en Java:

```
public class Hombre extends Persona{  
}
```

Todos las clases en Java heredan de la clase Object.



4.7.2. Que es el Casting

El casting es un procedimiento para transformar una variable primitiva de un tipo a otro, o transformar un objeto de una clase a otra clase siempre y cuando haya una relación de herencia entre ambas.

Existen distintos tipos de castings de acuerdo a si se utilizan tipos de datos o clases.

4.7.3. Casteo Implicito (Widening Casting)

El *Casteo Implicito* radica en que no se necesita escribir código para que se lleve a cabo. Ocurre cuando se realiza una conversión ancha - *widening casting* - es decir, cuando se coloca un valor pequeño en un contenedor grande.

Por ejemplo:

```

// Define una variable del tipo int con el valor 100
int numero = 100;

// Define una variable del tipo long a partir de un int
long numero2 = numero;

```

4.7.4. Casteo Explicito (Narrowing Casting)

El *Casteo Explicito* se produce cuando se realiza una conversión estrecha - *narrowing casting* - es decir, cuando se coloca un valor grande en un contenedor pequeño. Son susceptibles de pérdida de datos y deben realizarse a través de código fuente, de forma explícita.

Por ejemplo:

```
// Define una variable del tipo int con el valor 250
int numero = 250;

// Define una variable del tipo short y castea la variable numero
short s = (short) numero;
```

4.7.5. Upcasting

El *upcasting* se produce a nivel objetos. Suponiendo que existe una clase Empleado y clase Ejecutivo, la cual es una subclase de ésta. Un Ejecutivo entonces *ES UN* Empleado, y se puede escribir de la siguiente forma:

```
// Instancia un ejecutivo en una variable de tipo Empleado
Empleado e1 = new Ejecutivo("Máximo Dueño", 2000);
```

A esta operación en donde un objeto de una clase derivada se asigna a una referencia cuyo tipo es alguna de las superclases se la denomina *upcasting*.

Cuando se realiza este tipo de operaciones, hay que tener cuidado porque para la referencia e1 no existen los atributos y métodos que se definieron en la clase Ejecutivo, aunque la referencia apunte a un objeto de este tipo.

Por el contrario, si ahora existe una variable de tipo Empleado y se desea acceder a los métodos de la clase derivada - suponiendo que contiene un método *ejecutarPlanificacion()* en la clase Ejecutivo - teniendo una referencia de una clase base, como en el ejemplo anterior, hay que *convertir explícitamente* la referencia de un tipo a otro. Esto se hace con el operador de cast de la siguiente forma:

```
// Instancia un ejecutivo en una variable de tipo Empleado
Empleado emp = new Ejecutivo("Máximo Dueño", 2000);

// Se convierte (o castea) la referencia de tipo
Ejecutivo ej = (Ejecutivo) emp;

// Se aumenta el sueldo del ejecutivo
ej.ejecutarPlanificacion();
```

La expresión de la segunda línea convierte la referencia de tipo Empleado asignándola a una referencia de tipo Ejecutivo. Para el compilador es correcto porque Ejecutivo es una clase derivada de Empleado. En tiempo de ejecución la JVM convertirá la referencia si efectivamente emp apunta a un objeto de la clase Ejecutivo, caso contrario lanzara una excepción.

Por el contrario, si se intenta realizar lo siguiente:

```
Empleado emp = new Empleado("Javier Todudas", 2000);
Ejecutivo ej = (Ejecutivo)emp;
```

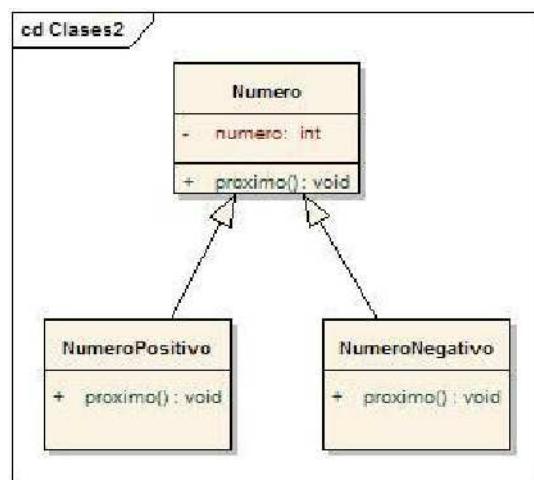
No dará problemas de compilación, pero al ejecutar se producirá un error ya que la referencia *emp* apunta a un objeto de clase Empleado y no a un objeto de clase Ejecutivo.

4.8. Polimorfismo

4.8.1. Definición

El polimorfismo se produce cuando un método adopta más de una forma. Un método puede modificar su comportamiento de acuerdo a su necesidad.

Como ejemplo se puede utilizar lo siguiente:



```

public class Numero{
    private int numero;
    public void proximo(){
    }
    public String toString(){
        return new String(numero);
    }
}

public class NumeroPositivo{
    public NumeroPositivo(){
        numero = 0;
    }
    public void proximo(){
        numero = numero + 1;
    }
}

public class NumeroNegativo{
    public NumeroNegativo(){
        numero = -1;
    }
    public void proximo(){
        numero = numero - 1;
    }
}

public class Programa{
    public static void main(String[] args){
        Numero positivo = new NumeroPositivo();
        Numero negativo = new NumeroNegativo();
        System.out.println("numero positivo inicial:"+positivo);
        System.out.println("numero negativo inicial:"+negativo);
        positivo.proximo();
        negativo.proximo();
        System.out.println("numero positivo proximo:"+positivo);
        System.out.println("numero negativo proximo:"+negativo);
        positivo.proximo();
        negativo.proximo();
        System.out.println("numero positivo proximo:"+positivo);
        System.out.println("numero negativo proximo:"+negativo);
    }
}

```

El resultado de este programa es el siguiente:

```

numero positivo inicial:0
numero negativo inicial:-1
numero positivo proximo:1

```

```

numero negativo proximo: -2
numero positivo proximo: 2
numero negativo proximo: -3

```

La conclusion que debemos observar en este ejemplo es que se tienen dos instancias de Numero, una instancia de NumeroPositivo y una de NumeroNegativo, cada cual tiene un comportamiento distinto. Sin embargo a la hora de declararlo se los declaro en su forma generica de Numero y se tiene el metodo proximo() asociado.

El polimorfismo ha permitido que cada uno se comporte de manera distinta, y sin embargo parece llamarse siempre al mismo metodo. Es decir, es distinto el proximo negativo del proximo positivo, y eso se refleja en el resultado.

Otra cosa que puede verse en este ejemplo es que al imprimir los objetos positivo y negativo se llama automaticamente al metodo `toString()`, es el modo en el que java imprime sus objetos en pantalla. Y el aprovechamiento del mecanismo de herencia, en la programación del metodo `toString()` que retorna el valor de numero como un String, es decir, algo que puede imprimirse fácilmente. Al llamarlo en un NumeroPositivo, como este no lo tiene programado, se ejecuta el que esta definido en la clase padre Numero. Al llamarlo en un NumeroNegativo tambien se llama al de la clase padre.

4.8.2. Con redefinición

Se llama polimorfismo con redefinicion cuando en una clase sobreescrivimos un metodo, definido en la superclase.

Ejemplo en Java:

```

public class Persona{
    public void cantar(){
        // Esto es un comentario. Aqui va la definicion.
    }
}

public class Cantante extend Persona{

    public void cantar(){
        // Esto es un comentario. Aqui va la redefinicion.
    }
}

```

}

En este caso se puede ver que todas las personas pueden cantar(), sin embargo no es lo mismo cantar() en la clase Cantante, pues el cantante logra mayor escala, mayor volumen de voz, y entona de una forma muy preparada. Un cantante logra llegar mas lejos con su canto que una persona comun. Al redefinir el metodo en la clase Cantante se logra que se ejecute el metodo cantar() de cantante en lugar del metodo cantar() de persona. Por lo tanto la especializacion que se logra con este mecanismo permite cambiar el comportamiento de una familia de clases.

4.8.3. Sin redefinición

Se llama polimorfismo sin redefinicion cuando un metodo tiene distinto comportamiento de acuerdo a la clase a la que pertenece.

Ejemplo en Java:

```
public class Persona{  
    public void beber(Agua unPocoDeAgua){  
        // Esto es un comentario. Aca va una definicion.  
    }  
  
    public class Animal{  
        public void beber(Agua unPocoDeAgua){  
            // Esto es un comentario. Aca va otra definicion.  
        }  
    }  
}
```

En este caso ambas clases logran beber(), es decir que ambas clases tienen definido el metodo beber, solo que uno de los metodos beber lo hace como humano, y el otro lo hace como un animal, el comportamiento es bien distinto, aunque el nombre sea el mismo.

4.9. Clase Abstracta

4.9.1. Definición

Una clase abstracta representa un concepto abstracto que no debe ser instanciado. Es simplemente una clase que no se puede instanciar. Sirve como base para que otras clases hereden de ella sus atributos y métodos, pero no tiene sentido por sí sola como instancia. Los métodos abstractos no pueden estar definidos en la clase.

Pero deberán ser obligatoriamente definidos en las subclases.

4.9.2. Definición

Ejemplo en Java:

```
public abstract class Persona{  
  
    // Atributos  
    private String nombre;  
    private Date fechaDeNacimiento;  
  
    // Métodos  
    public abstract void estudiar();  
    public String getNombre(){  
        return this.nombre;  
    }  
}
```

4.9.3. Utilización

```
public class Hombre extends Persona{  
  
    // Método que se implementa obligatoriamente,  
    // por estar definido como abstracto en la clase padre  
    public void estudiar(){  
        // Aquí va el código de estudiar. Leer, entender, reflexionar.  
    }  
}
```

4.10. Interfaz

4.10.1. Que es una interfaz

Es una declaracion de comportamiento, es decir un conjunto de metodos sin su implementacion. Define un protocolo de comportamiento. Es un contrato que publica una clase para ser utilizada por otras clases. Puede heredar de otras interfaces.

4.10.2. Definición

Ejemplo en Java, definición:

```
public interface Imprimible{  
    public void imprimir();  
    public void imprimir(Tamano t, Estilos s);  
}
```

4.10.3. Utilización

Ejemplo en Java, utilizacion

```
public class Documento implements Imprimible{  
    public void imprimir(){  
        // Esto es un comentario. Aqui va la definicion.  
    }  
    public void imprimir(Tamano t, Estilos s){  
        // Esto es un comentario. Aqui va la definicion.  
    }  
}
```

Un ejemplo de utilizacion de una interfaz, puede ser el siguiente, se tiene un Aeropuerto en el que se puede dar permiso de aterrizar.

```
public class Aeropuerto{  
    public void darPermisoDeAterrizar(Volador v){  
        // Aquí va el código de despejar pista y permitir aterrizar al Volador  
        v.  
        // En el programa principal se puede iterar sobre una lista de  
        Voladores e ir dando  
        // permiso de aterrizaje a cada volador, sin especificar que tipo  
        específico  
        // de volador se trata  
        v.atterizar();  
  
        if( v instanceof AvionPrivado ){  
            AvionPrivado ap = (AvionPrivado)v;  
            System.out.println("Aterrizo Licencia: "+ap.getLicencia());  
        }  
  
        if( v instanceof AvionDePasajeros ){  
            AvionDePasajeros ap = (AvionDePasajeros)v;  
            System.out.println("Aterrizo Aerolinea: "+ap.getAerolinea());  
        }  
  
        if( v instanceof Superman ){  
            Superman ap = (Superman)v;  
            System.out.println("Aterrizo Nombre: "+ap.getNombre());  
        }  
    }  
}
```

Luego tenemos distintos TransportesAereos, un AvionDeLinea, y un AvionPrivado, y luego tenemos a Superman.

Entonces lo que se hace es utilizar la interfaz para definir un mismo comportamiento a clases bien distintas, porque no heredan de la misma clase generica:

```
public interface Volador{  
    public void aterrizar(); // Obligamos a cada Volador a programar su modo  
    de aterrizar.  
}  
  
public abstract class TransporteAereo{  
}  
  
public class AvionPrivado extends TransporteAereo implements Volador{  
    public void aterrizar(){  
        // Aqui va el codigo de modo que tiene de aterrizar un avion privado  
    }  
  
    private String licencia;  
  
    public String getLicencia(){  
        return licencia;  
    }  
  
    public void setLicencia(String lic){  
        this.licencia = lic;  
    }  
  
}  
  
public class AvionDePasajeros extends TransporteAereo implements Volador{  
    public void aterrizar(){  
        // Aqui va el codigo de modo que tiene de aterrizar un avion de  
        pasajeros  
    }  
  
    private String aerolinea;  
  
    public String getAerolinea(){  
        return aerolinea;  
    }  
  
    public void setLicencia(String ae){  
        this.aerolinea = ae;  
    }  
  
}  
  
public class Superman extends Superheroe implements Volador{  
    public void aterrizar(){  
        // Aqui va el codigo de modo que tiene de aterrizar de Superman  
    }  
  
    public String getNombre(){  
        return "Kent, Clark";  
    }  
  
}
```

```
public class Programa{  
    public Programa(){  
    }  
  
    public static void main(String[] args){  
        // Una inicialización que puede existir en cualquier lado del programa  
        AvionPrivado v1 = new AvionPrivado();  
  
        v1.setLicencia("L123456F");  
  
        AvionDePasajeros v2 = new AvionDePasajeros();  
  
        v2.setAerolinea("EducacionIT Lan");  
  
        Superman v3 = new Superman();  
  
        Aeropuerto aeropuerto1 = new Aeropuerto();  
  
        aeropuerto1.darPermisoDeAterrizar(v1);  
        aeropuerto1.darPermisoDeAterrizar(v2);  
        aeropuerto1.darPermisoDeAterrizar(v3);  
    }  
}
```

En este caso se ve como se puede llamar al método aterrizar() de los objetos Voladores, porque la interfaz Volador tiene un método aterrizar() y al implementar una interfaz se está diciendo que se tendrá ese comportamiento, es decir que se implementan sus métodos.

4.11. Paquetes

4.11.1. Que es un paquete

Un paquete es un medio de organizacion de clases, para agrupar clases e interfaces. Se suelen crear a partir de necesidades funcionales. Pueden contener clases, interfases y mas paquetes.

4.11.2. Definicion

A continuación se presenta un ejemplo en Java de cómo definir un paquete:

```
package mi_paquete.mi_subpaquete;  
public class Persona{  
}
```

4.11.3. Utilizacion

A continuación se presenta un ejemplo en Java de cómo utilizar un paquete desde una clase ubicada en otro paquete

```
import mi_paquete.mi_subpaquete.*;
```

La asociacion directa es que una clase o una interface esta definida en un archivo. Todo archivo debe contener una clase publica al menos, y esta debe tener el mismo nombre que el archivo.

Estas clases e interfaces, estos archivos deben vivir dentro de paquetes, que son equivalentes a los directorios, es decir que la clase Persona, que es publica y esta en el archivo Persona.java, debe existir dentro del paquete entidades, es decir el archivo debe vivir dentro del directorio entidades.

Ejemplo:

```
/entidades/Persona.java
```

4.12. La keyword final

4.12.1. Definición

Es una palabra reservada de Java. Tiene una semantica distinta segun donde se la utilice.

4.12.2. Aplicable a atributos

Aplicada en atributos representa a un valor constante.

Ejemplo:

```
public final int valorCuota = 100;
```

Este valor no puede ser modificado en ningun momento de la programacion. Es un valor constante que no se modifica.

4.12.3. Aplicable a métodos

Aplicada en metodos representa la no modificacion de un metodo en el caso de querer sobreescribirlo en una subclase.

Ejemplo:

```
public final void moverPieza(){
    // Esto es un comentario. Aqui va la ultima definicion.
}
```

En caso de heredar de esta clase, no se podra sobreescribir el metodo moverPieza(), quedara con su comportamiento original y no se puede modificar.

4.12.4. Aplicable a clases

Aplicada en clases significa que esa clase no se puede extender o no se puede hacer una subclase, es decir que no se puede generar una clase que herede de ella.

Cierra el arbol de herencias y queda como una hoja de este arbol.

Ejemplo:

```
public final class Hombre{  
}
```

Es decir, que no se puede heredar de la clase Hombre, esta clase deja cerrado el arbol de herencia. Esta tecnica puede ser utilizada cuando se entrega un libreria o parte de un programa y se quiere que, quien utilice estas clases, no pueda heredar de estas clases para modificar su comportamiento. Es una medida de seguridad al compartir codigo o publicar programas.

4.13. Accesibilidad

4.13.1. Acceso privado, palabra clave private

Al definir un atributo, metodo o clase como privado, se le esta dando acceso solo desde la clase donde fue definido. Por ejemplo, en el caso de velocidad de un Auto

```
public class Auto{  
  
    private int velocidad;  
  
    public void acelerar(){  
        this.velocidad += 10;  
    }  
}
```

Por lo tanto el unico modo de acceder a modificar la velocidad es a travez del metodo acelerar. Esta propiedad no esta accesible desde ningun otro alcance sino el de la clase.

Para poder acceder al atributo velocidad, solo podran hacerlo los metodos de la misma clase.

4.13.2. Acceso por defecto, "default" o "package"

En el caso del acceso por defecto o "default" o "package", que es el acceso cuando no se declara la accesibilidad a un atributo, metodo o clase, el acceso es dentro del

mismo paquete o directorio. Por ejemplo si tengo dos clases Auto y Camion, y auto tiene un conductor sin definir su accesibilidad, es decir

```
package vehiculos;

public class Auto extends Vehiculo{
    Conductor getConductor(){
        // Aquí se retorna el conductor.
    }
}
```

Luego en la clase Camion que está en el mismo package:

```
package vehiculos;

public class Camion extends Vehiculo{
    public void comunicar(Vehiculo v){
        v.getConductor().conversar(); // Se tiene acceso a getConductor por
        // estar ambas clases
        // en el mismo package.
    }
}
```

Para poder acceder al método getConductor(), podrán hacerlo únicamente las clases ubicadas en el mismo package o directorio.

4.13.3. Acceso protegido, palabra clave protected

Al definir un atributo o método como protegido estamos dejándolo disponible para las clases que heredan de esta clase, ejemplo:

```
public class Vehiculo{
    protected int cantidadRuedas = 4;
}
```

Luego en la clase Auto, tendremos acceso a cantidad de Ruedas

```
public class Auto extends Vehiculo{
    public int getCantidadDeRuedas(){
        return this.cantidadRuedas;
    }
}
```

Para poder acceder al atributo cantidadRuedas, solo podran hacerlo las clases que heredan de Vehiculo, es decir la familia de clases.



4.13.4. Acceso publico, palabra clave public

Definiendo un atributo o metodo o clase como publico permite que se tenga acceso a estos desde cualquier alcance, desde todos lados.

Acceder a los atributos publicos, todos podran hacerlo.

4.13.5. Resumen de accesibilidad

Accesibilidad	Acceso en la misma clase	Acceso en el mismo package	Acceso en el arbol de herencia	Acceso desde todas las clases
private	Si	No	No	No
”default” o “package”	Si	Si	No	No
protected	Si	Si	Si	No
public	Si	Si	Si	Si

5. La interfaz gráfica

5.1. La Historia: AWT

5.1.1. Definición

Fue definido como un conjunto de paquetes inicialmente, para la construcción de interfaces gráficas. Está basado en contenedores gráficos más componentes gráficos, más modelo de eventos. Los componentes de AWT están implementados en código nativo, es decir que están restringidos a las características de la plataforma.

Por ejemplo si la plataforma no permite imágenes en los botones, entonces no será posible construir un botón con una imagen en lugar del texto. La mayoría de los componentes de AWT son subclases de Component, excepto los contenedores de mayor nivel como Frame, Dialog y Applet.

5.1.2. Estructura de una aplicación AWT

Los componentes de AWT están dentro del paquete java.awt. Todo componente se utiliza dentro de un java.awt.Frame. La pantalla a construir será una subclase de java.awt.Frame. La pantalla estará compuesta por:

Los atributos, que son componentes gráficos. Los métodos, que manejan los eventos que ocurren sobre los componentes gráficos. El método initComponents(), método llamado por el constructor donde se inicializan todos los componentes gráficos.

5.2. La actualidad: Swing

5.2.1. Definición

Swing, Nombre del proyecto dedicado a la construccion de componentes graficos. Si bien ha quedado inmortalizado como Swing, su denominacion es JFC (Java Fundation Classes) ya que incluye otros paquetes adicionales. Las JFC incluyen un grupo de caracteristicas para la creacion de interfaces graficas. Los componentes Swing GUI es una de las caracteristicas de las JFC.

Esta basado en contenedores graficos mas componentes graficos mas un modelo de eventos. Es el conjunto de paquetes utilizados para la construccion de interfaces graficas. Los componentes Swing estan implementados sin codigo nativo, es decir que estan no restringidos a las caracteristicas de la plataforma.

Por ejemplo, si la plataforma no permite imagenes en los botones, con Swing es posible agregarlos. La mayoria de los componentes de Swing son subclases de JComponent, excepto los contenedores de mayor nivel como JFrame, JDialog y JApplet. Swing utiliza algunas clases de AWT.

5.2.2. Estructura de una aplicación Swing

Los componentes de Swing estan dentro de paquetes como javax.swing. Todo componente grafico se utiliza dentro de un javax.swing.JFrame. La pantalla a construir sera una subclase de javax.swing.JFrame.

A diferencia de los componentes de AWT, los componentes Swing comienzan con la letra J. Por ejemplo, un marco en AWT se llama Frame y en Swing se llama JFrame, o un boton en AWT es Button, y en Swing es JButton. La pantalla estara compuesta por:

Los atributos, son componentes graficos. Los metodos, se manejan como los eventos que ocurren sobre un componente grafico. El metodo initComponents, metodo que es llamado por el constructor, donde se inicializan todos los componentes graficos.

5.3. Swing vs. AWT

5.3.1. Comparacion

Ambos representan distintas maneras de armar una interfaz grafica. AWT es el primer framework de Java para consgtrucion grafica y luego nace Swing. Los componentes AWT estan implementados con codigo nativo mientras que los componentes Swing no.

Por esta razon los componentes Swing pueden tener mas funcionalidad ya que no estan restringidos a las caracteristicas comunes de cada plataforma. Como regla, en Swing los componentes deben ser anadidos a objetos JPanel y no directamente a JFrame.

5.3.2. Swing como negocio

Existen empresas dedicadas a construir componentes Swing, que se venden por separado a la JDK. En Swing los componentes graficos pueden poseer mas funcionalidad ya que no son dependientes de la plataforma, como por ejemplo, los componentes Swing no tienen por que ser necesariamente rectangulares, se pueden armar botones redondos. Los botones pueden mostrar tanto texto como imagenes. Su visualizacion es la misma en todas las plataformas.

5.4. Componentes Swing: Contenedores

5.4.1. Definición

Son contenedores graficos que tienen la funcion de organizar a los distintos componentes graficos. Estos contenedores proporcionan la infraestructura que necesitan los componentes Swing para el manejo de eventos y su dibujo.

Existen contenedores de nivel general, como JFrame, JDialog y JApplet. Los contenedores de nivel general tienen un papel de contenedores, que contendran a

todos los componentes graficos, a excepcion del componente JMenuBar. Estos en general no son visibles.

Existen contenedores de nivel intermedio, como JPanel, JTabbedPane y JScrollPane. Estos son de caracter mas visible, y son utilizados, o contenidos, por los contenedores de nivel general.

5.4.2. JFrame

Es la clase utilizada para la construccion de una ventana. Representa a una ventana primaria. Es la ventana de mayor nivel dentro de una aplicacion. Sirve como contenedor para los componentes graficos.

Tip para NetBeans, para visualizar el arbol de contenido de componentes de una JFrame, oprimir las teclas ctrl+shift+F1 y ver la salida Standard, output window.

5.4.3. JDialog

Es la clase utilizada para la contruccion de una ventana secundaria. Es una ventana dependiente de una ventana primaria. Permite dialogar con el usuario.

5.4.4. JApplet

El JApplet es la clase utilizada para la contruccion de Applets que corren en el navegador. Representa el area de una aplicacion dentro de una ventana de un navegador.

5.4.5. JPanel

JPanel es un contenedor de nivel intermedio. JPanel tiene como proposito simplificar el posicionamiento de componentes.

5.5. Organización en Netbeans

5.5.1. Palette Window

Contiene los componentes graficos para agregar a los formularios. Puede visualizar los componentes como iconos, o como iconos con nombre. Posee tanto los componentes de AWT como los de Swing.

5.5.2. Inspector Window

Visualiza la jerarquia de componentes del formulario seleccionado. Los componentes visibles por el usuario son visualizados dentro del formulario. Los componentes no visibles por el usuario son visualizados dentro del item, other components.

5.5.3. Properties Window

Visualiza las opciones a setear del componente seleccionado. Visualiza las propiedades del componente seleccionado.

5.6. MDI Documento de Multiples Interfaces

5.6.1. Introducción a MDI

El hecho de necesitar agrupar funcionalidades distintas en una unica aplicacion genera la necesidad de ejecutar una unica aplicacion y tener acceso a resolver varias funcionalidades.

Un ejemplo de esto es la aplicacion de una clinica donde se necesita resolver en una misma aplicacion el concepto de turnos para los pacientes, agendas de los medicos, y datos personales e historias clinicas de los pacientes.

Todo esto puede ser agrupado en un mismo sistema, en una misma aplicacion, al poder ejecutar varias interfaces o varias ventanas dentro de una ventana principal. El modo de acceder a estas multiples ventanas de a una puede lograrse a travez de un menu de seleccion de la funcionalidad que se requiere utilizar.

En nuestro ejemplo conceptual podemos tener un Menu Pacientes, y un submenu Datos Personales, de modo tal de resolver los temas de los datos personales del paciente, luego un submenu Historia Clinica, donde aparecen la lista de consultas y se puede seleccionar que se resolvio como diagnostico en cada consulta, luego un submenu de Turnos del paciente, donde se consoliden los datos de asignacion de pacientes a medicos desde la perspectiva de un paciente.

Como otro menu podemos tener el Menu Medico, el cual puede tener un submenu Datos Personales del Medico, por ejemplo datos de contactos y horarios en los que se lo puede encontrar, luego un submenu Agenda Semanal, donde se puede acceder a la informacion de agenda diaria del medico, que pacientes ya tiene asignados en que horarios y en que momento queda libre para una nueva asignacion.

Logicamente se espera que todas estas ventanas internas, ventanas que se disparan a partir de los submenues vivan dentro de la aplicacion de la ventana principal, de modo de poder abrir varias ventanas internas y poder pasar de una a otra sin cerrarlas.

5.6.2. Pasos para construir una aplicación MDI

Todo esto podemos lograrlo, en este caso con el NetBeans, a traves de los siguientes elementos:

Paso 1: Generamos un JFrame nuevo, desde la Menu Opciones File -> Nueva Clase Tipo JFrame, para este caso se llamara AplicacionJFrame.

Paso 2: Dentro de AplicacionJFrame se agrega un JMenuBar, que encontramos como elemento en la solapa derecha, llamada Swing.

Paso 3: Dentro de AplicacionJFrame se agrega un JDesktopPane, que encontramos como elemento en la solapa derecha, llamada Swing.

Paso 4: Creamos una clase PacientesDatosPersonalesVentana del tipo JInternalFrame, desde Menu Opciones File -> Nueva Clase Tipo JInternalFrame.

Paso 5: Creamos la opcion de menu PacientesMenu, desde la solapa izquierda de Inspector, click derecho sobre JMenuBar, opcion Add JMenu.

Paso 6: Creamos la opcion de submenu PacientesDatosPersonalesSubmenu, desde la solapa izquierda de Inspector, click derecho sobre el nuevo JMenu, y Add JMenuItem.

Paso 7: Damos lugar a la instanciacion de la Ventana Interna PacientesDatosPersonalesSubmenu, desde click derecho sobre el JMenuItem recien creado, opcion events, opcion actionPerformed.

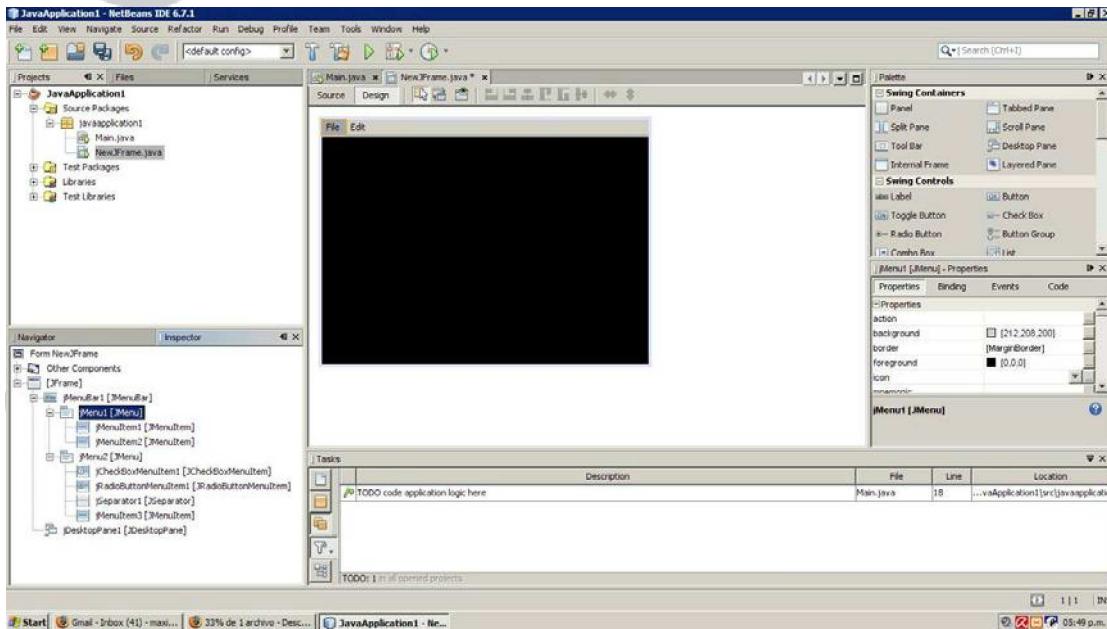
Paso 8: Dentro del metodo agregamos lo siguiente:

```
PacientesDatosPersonalesVentana pacientesDatosPersonalesVentana = new
PacientesDatosPersonalesVentana();

jDesktopPane.add(pacientesDatosPersonalesVentana);
```

Al correr el programa se tendra el menu y submenu y al momento de seleccionar el submenu se disparara la ventana interna. De esta forma se puede continuar sumando opciones del menu e items de submenu en cada menu.

Y las ventanas internas se programan independientes unas de las otras como si fuesen aplicaciones por separado, pero luego funcionaran todas juntas compartiendo recursos como acceso a la base de datos y otros recursos.



6. Conceptos Generales

6.1. La clase String

6.1.1. Definición

La clase String es utilizada para manejar cadenas de caracteres. El operador + se utiliza para concatenar Strings.

Entre los métodos más conocidos están charAt(), equals(), length(), replaceAll().

6.1.2. Inicialización de un String

Las formas de declarar e inicializar un objeto de la clase String son las siguientes:

```
String cadena = "Soy una cadena de caracteres";
String cadena = new String("Soy una cadena de caracteres");
char[] caracteres = {'h','o','l','a'};
String cadena = new String(caracteres);
```

Todos los objetos en Java heredan de la clase Object, y la clase Object tiene un método `toString()` que nos retorna la representación del objeto en un String.

Este método puede ser sobrescrito de modo tal que la información que se retorne en el String resultado sea propia de la instancia de la clase.

Ejemplo:

```
public class Auto{
    private String patente;
    private String marca;
    private String modelo;
```

```

    // Aqui van los Constructores

    // Aqui van los setters y los getters

    // Este es el metodo que retorna al objeto Auto como un String

    public String toString(){
        return "[ Patente:" +this.patente+ "] Marca:" +this.marca+
        Modelo:" +this.modelo;
    }
}

```

6.1.3. Metodos mas importantes

La clase String tambien tiene los siguientes metodos:

- ✓ `compareTo(String)` que compara un String con otro por igualdad.
- ✓ `concat(String)` que concatena un String con otro.
- ✓ `contains()` que evalua si una cadena de caracteres esta contenida en el String.
- ✓ `startsWith(String prefijo)` que evalua si el String comienza con el prefijo.
- ✓ `endsWith(String sufijo)` que evalua si el String termina con el sufijo.
- ✓ `valueOf()` que convierte cualquier tipo basico, boolean, long, double, int, en un String.
- ✓ `split(String regularExpression)` que separa en partes segun la expresion regular.
- ✓ `subString(int comienzo, int fin)` que retorna un bloque desde comienzo y hasta fin.
- ✓ `matches(String regularExpression)` que evalua si el String contiene la expresion regular.

6.2. La clase System

6.2.1. Definición

La clase System representa al sistema donde se esta ejecutando el programa Java. Por lo tanto se la puede utilizar para interactuar con el entorno en el que corre, y utilizar las propiedades del entorno, sistema operativo, usuario y demas

6.2.2. Donde utilizarla

Puede accederse a la salida estandard del proceso, a la salida de error del proceso y a la entrada estandard del proceso mediante, System.out, System.err, System.in respectivamente, de modo de poder interactuar con el Sistema Operativo. Tambien permite obtener el tiempo en milisegundos llamando al sistema operativo para lograr este objetivo.

Otra cosa muy util de la Clase System es que permite cargar librerias externas en una linea de ejecucion con el metodo load(), mediante esta operacion se logra cargar librerias en tiempo de corrida del programa y sumar asi funcionalidades a nuestros programas.

Posee atributos y metodos de uso general, y son todos estaticos, es decir atritutos y metodos de clase. No es una clase instanciable.

Entre los metodos mas conocidos estan:

- ✓ exit(): termina la ejecución de la Java Virtual Machine
- ✓ gc(): invoca al Garbage Collector
- ✓ getProperties(): trae todas las propiedades del sistema
- ✓ getProperty(): trae una propiedad en particular del sistema

6.3. Los Wrappers de los tipos de dato primitivos

6.3.1. Definición

Es la represenacion de los valores primitivos como objetos, son utilizados para envolver los tipos de datos primitivos. En general tienen dos constructores:

- ✓ Con los valores primitivos como parametro
- ✓ Con los valores primitivos como cadena de caracteres, como parametro

Tienen un metodo estatico valueOf() que retorna un objeto Wrapper.

6.3.2. La clase Integer

Utilizada para envolver el tipo de dato primitivo int. Es una subclase de java.lang.Number.

Ejemplo en java:

```
Integer miEntero = new Integer(50);  
Integer miOtroEntero = Integer.valueOf("50");
```

6.3.3. La clase Float

Utilizada para envolver el tipo de dato primitivo float. Es una subclase de java.lang.Number.

Ejemplo en Java:

```
Float miPuntoFlotante = new Float(45.67);  
Float miOtroPuntoFlotante = Float.valueOf(45.67);
```

6.3.4. La clase Number

Representa el concepto abstracto de un numero en Java. Tiene como objetivo ser la superclase de los Wrappers de los tipos de dato numericos primitivos. Es una clase abstracta, es decir que no se puede instanciar.

6.4. Comparación entre objetos

6.4.1. El operador ==

Este operador esta sobrecargado para operar con objetos. Se utiliza para saber si dos punteros apuntan al mismo objeto, es decir que son dos referencias de la misma instancia de objeto. Retorna true o false segun el resultado de la comparacion.

Ejemplo:

```
String c1 = new String("Juan");  
String c2 = c1;  
if(c1 == c2){
```

```
System.out.println("c1 y c2 apuntan al mismo objeto");  
}
```

6.4.2. El método equals()

Se utiliza para saber si dos objetos son iguales, si el contenido es igual. Esta redefinido por subclase que lo utiliza. En la clase String se utiliza para saber si dos objetos String son iguales.

En la clase Integer se utiliza para saber si dos objetos Integer son iguales. Retorna true o false segun el resultado de la comparacion.

Ejemplo:

```
String c1 = new String("Juan");  
String c2 = new String("Pedro");  
if(c1.equals(c2)){  
    System.out.println("c1 y c2 son iguales");  
}
```

7. Contenedores

7.1. Que son los contenedores

7.1.1. Definición

Los contenedores son el modo de agrupar objetos. Tambien llamados colecciones (Collection). Representa a un conjunto de items, un conjunto de objetos, que pueden ser homogeneos o no. Por ejemplo, una agenda es una colección de datos de personas.

7.1.2. La interfaz Collection

La interfaz collection es la superinterfaz de donde heredan las mayoria de las interfaces utilizadas para el manejo de las colecciones. Es la interfaz raiz de la jerarquia de interfaces.

La interfaz Collection forma parte del Collection Framework, un conjunto de interfaces y clases que representan distintos modos de agrupar objetos, segun distintas politicas de manejo de memoria o acceso a ellos.

Representan un conjunto de objetos, tambien llamados elementos. Una clase que quiera comportarse como una Collection debera implementar esta interfaz, por lo tanto sus metodos.

7.2. Listas

7.2.1. La interfaz List

Es una subinterfaz de Collection. Tambien es llamada Secuencia. Puede contener Objetos duplicados. Puede contener elementos Nulos. Soporta manipulacion de elementos via indices a traves del metodo: Object get(int indice), que permite obtener el elemento en la posicion indice de la lista.

Tambien puede obtenerse el indice en el cual esta almacenado un determinado objeto, mediante el metodo: int indexOf(unObjeto).

Tambien permite trabajar con un subconjunto de elementos de la lista con el metodo: List subList(int indiceDesde, int indiceHasta).

En Java, las listas mas conocidas son ArrayList, Vector y Stack.

7.2.2. ArrayList

Es una clase que implementa la interfaz Collection. Representa a un arreglo de tamano variable. Los metodos mas comunmente utilizados son add(), get(), size() y remove(). No maneja sincronizacion.

```
public class Auto{  
    public Auto(String patente){  
        setPatente(patente);  
    }  
    private String patente;  
    public void setPatente(String patente){  
        this.patente = patente;  
    }  
    public String toString(){  
        return patente;  
    }  
}  
  
public class Programa{
```

```
public Programa(){  
}  
  
public static void main(String[] args){  
    ArrayList lista = new ArrayList();  
    lista.add(new Auto("123"));  
    lista.add(new Auto("456"));  
    lista.add(new Auto("789"));  
    for(int i = 0; i<lista.size(); i++){  
        System.out.println(lista.get(i));  
    }  
}
```

7.2.3. Vector

Es semanticamente igual a la clase ArrayList. Forma parte de una de las primeras versiones de la API de Java. A diferencia de la clase ArrayList, maneja sincronización.

```
public class Programa{  
    public Programa(){  
    }  
  
    public static void main(String[] args){  
        Vector lista = new Vector();  
        lista.add(new Auto("123"));  
        lista.add(new Auto("456"));  
        lista.add(new Auto("789"));  
        for(int i = 0; i<lista.size(); i++){  
            System.out.println(lista.get(i));  
        }  
    }  
}
```

7.3. Iteradores

7.3.1. Definición

Es un patron de diseño utilizado para recorrer las colecciones, abstrayendo al usuario de la implementación de la colección. En Java es una interfaz denominada Iterator. Esta formado por tres métodos:

- ✓ `boolean hasNext()`, retorna true en caso de haber mas elementos y false en caso de llegar al final del iterador.
- ✓ `Object next()`, retorna el siguiente elemento en la iteración.
- ✓ `void remove()`, remueve un objeto de la colección.

En las clases Vector, ArrayList, HashSet y TreeSet un iterador se consigue a través del método: Iterator iterator()

7.3.2. Utilización

Construye un contenedor, en este caso un ArrayList

Ejemplo:

```
ArrayList lasPersonas = new ArrayList();
lasPersonas.add("Pepe");
lasPersonas.add("Juan");
lasPersonas.add("Sabrina");
lasPersonas.add("Cecilia");
Iterator it = lasPersonas.iterator();
while(it.hasNext()){
    String unaPersona = (String)it.next();
}
```

8. Excepciones

8.1. Que es una excepcion

8.1.1. Definicion

Una excepcion que no fue capturada correctamente hace que el sistema se caiga. Una excepcion es un evento que ocurre durante la ejecucion de un programa, que interrumpe el flujo normal de ejecucion.

Java utiliza excepciones para el manejo de errores, error handling. Es un sinonimo de eventos excepcionales. Un buen manejo de Excepciones hace a un sistema robusto y estable, confiable.

8.1.2. Bloques try, catch y finally

Dentro del bloque try se pondra todo el codigo que puede arrojar alguna excepcion. Los bloques catch son los encargados de atrapar las excepciones arrojadas por alguna sentencia del bloque try. El bloque finally se ejecuta siempre despues del try y del catch.

Ejemplo:

```
try  
{  
    ...  
}  
  
catch (ClassNotFoundException excep1)  
{
```

```
    ...
}

catch (Exception excep2)

{
    ...
}

finally
{
    ...
// limpieza de código
}
```

8.2. Tipos de Excepciones

8.2.1. Unchecked Exceptions

Son las excepciones que tienen como superclase a la clase `RuntimeException`. No hay necesidad de capturarlas, es decir que no se necesita utilizar el bloque `try/catch/finally`, pero al saltar una excepcion de este tipo, como todas las excepciones corta el flujo de ejecucion.

Pueden ser dificiles de detectar, pero finalmente hacen que el sistema se caiga si no fueron tomadas en cuenta.

Ejemplo: Cuando se realiza una division por cero se lanza automaticamente una `ArithmeticException`. Cuando se quiere acceder a un objeto que apunta a null se lanza automaticamente una `NullPointerException`. Cuando se quiere acceder a un array con un indice que es mayor al tamano del array, se lanza automaticamente una `ArrayIndexOutOfBoundsException`.

```
int unValor = 0; // Este valor se suponia que no podia ser Cero,
                  // pero en algun momento se convirtio en Cero
```

```
try{
```

```

        int resultado = 10/unValor

    }catch(ArithmeticException ae){
        // Aqui tengo el control del flujo por el error de dividir por Cero
        // Asi que informo al usuario que hubo un error.

    }finally{
        // Y finalmente continuo con el sistema, reestableciendo el control.
    }
}

```

8.2.2. Checked Exceptions

Son las excepciones que tienen como superclase a la clase Exception. Necesitan ser capturadas, caso contrario no se podra compilar el codigo.

En el caso del FileReader se esta accediendo a lo que se llama un recurso externo al sistema en si. El sistema consta de las variables, las clases, los packages y los metodos que definimos, pero un archivo es un recurso externo, es decir que puede no existir al momento de ejecutar nuestro programa, o ser borrado mientras nuestro programa esta funcionando.

Tambien llamamos recurso externo a una conexion con la base de datos, porque depende de una conexion de red. Todos estos artefactos que no son parte del programa sino que se interactua con ellos, pueden traer algunas complicaciones como, no existir, que la red no este conectada, que no haya mas espacio en disco, que la red se caiga en medio de una corrida del programa.

Todos estos motivo llevan a la necesidad de evaluar si se pueden realizar correcta y completamente las instrucciones del programa. Por lo tanto si algo sale mal, lo que no puede ocurrir es que dejemos una conexion tomada, o un archivo sin que otro sistema pueda acceder a el.

Aqui es donde tenemos que prestar mucha atencion para que nuestro sistema no traiga problemas, simplemente verificar en el finally que se han liberado los recursos.

Ejemplo: La clase FileReader se utiliza para el acceso a disco. Cuando utilizo un acceso a disco con metodos de la clase FileReader, debo capturar la excepcion IOException.

```

FileReader archivo = null;

String nombreDeArchivoSinAcceso = "archivito.txt";

```

```
try{  
    archivo = new FileReader( new File( nombreDeArchivoSinAcceso ) );  
    // Aqui intento de hacer algo con el archivo  
    // puede ser que tenga tomado el archivo o  
    // puede ser que no se llegue porque salta la excepcion  
} catch(IOException ioe){  
    // Aqui informo al usuario que Ocurrio un error de Acceso al Recurso  
    Externo.  
}  
finally{  
    // Finalmente libero el recurso externo, si fue tomado.  
    if( archivo!=null ) {  
        try{  
            archivo.close();  
        } catch(Exception e){  
        }  
    }  
}
```

educación

8.3. La sentencia “throw”

8.3.1. Que es

La sentencia throw se utiliza para arrojar excepciones. Requiere un único argumento, una instancia de la clase `java.lang.Throwable`, que es implementada por la clase `Exception` y errores.

8.3.2. Utilización

Ejemplo:

```
boolean hayError = true;  
if(hayError){  
    throw new Exception();  
}
```

Esto significa que luego de evaluar la cláusula `hayError` y el hecho que de verdadero, significa que debe cortarse el flujo de corrida del programa, y desde el método que llama a este método debe capturarse este error, informar al usuario que ocurrió el error y luego reestablecer el sistema.

8.4. Creación de excepciones propias

8.4.1. La clase `Exception` como superclase

Por convención, su nombre debería terminar con la palabra `Exception`. Consiste en crear una clase que hereda de `java.lang.Exception`. Lo conveniente es ir identificando qué tipo de excepciones, en el flujo de información de la vida real, pueden ser representadas por una excepción creada a medida, de forma tal de poder entender de forma inmediata un bloque de código.

El manejo de excepciones hace que el código sea legible y fácilmente comprensible.

Ejemplo:

```
public class MiExcepcionException extends Exception{  
    // Esto es un comentario. Aqui va la definicion de mi clase propia de  
    excepcion.  
}
```

8.4.2. La keyword “throws”

La palabra clave throws se utiliza en la firma de los métodos que pueden lanzar excepciones. Una buena combinación de Creación de Excepciones propias del negocio que estamos modelando con el sistema, y la declaración en los métodos indicados que pueden lanzar este tipo de excepciones, hace que sea fácil programar y compartir clases entre distintos equipos de desarrollo.

Estas prácticas facilitan mucho el compartir código y queda uno siempre obligado a capturar excepciones que permiten un buen control del flujo de datos dentro del sistema, y lo hacen robusto.

Ejemplo:

```
public class Hombre{  
    public void comer() throws MiExcepcion{  
        // Esto es un comentario. Aqui va algo de código  
  
        // Esto es otro comentario. Aquí evaluamos una condición que hace que  
        // arrojemos una excepción  
  
        throw new MiExcepcion();  
    }  
}
```

9. Streams

9.1. Definición

9.1.1. Que es un Stream

Un Stream es un medio utilizado para leer datos de una fuente, y para escribir datos en un destino. Tanto la fuente como el destino pueden ser archivos, sockets, memoria, cadenas de caracteres, y tambien procesos.

Los Streams se caracterizan por ser unidireccionales, es decir que un Stream se utilizara solo para leer, o solo para escribir, pero no ambas acciones al mismo tiempo.

Para utilizar una Stream, el programa a realizar deberá construir el Stream relacionandolo directamente con una fuente o con un destino, dependiendo si se necesita leer o escribir informacion.

La accion de leer informacion de una fuente es conocida tambien como *input*, y la accion de escribir informacion en un destino es conocida como *output*. Dentro de Java, todas las clases utilizadas tanto para el input como para el output están incluidas en el paquete java.io

9.1.2. Algoritmo de Lectura

Para obtener información, un programa deberá abrir un stream sobre una fuente y leer la información de forma secuencial.

Independientemente del tipo de información, el algoritmo de lectura es siempre el mismo:

```
    Abrir un stream  
    Mientras haya mas información  
    {  
        Leer información
```

```
    }  
    Cerrar stream
```

9.1.3. Algoritmo de Escritura

Para escribir información, un programa deberá abrir un stream sobre un destino y escribir la información de forma secuencial.

Independientemente del tipo de información, el algoritmo de escritura es siempre el mismo:

```
Abrir un stream  
Mientras haya mas información  
{  
    Escribir información  
}  
Cerrar stream
```

9.2. Tipos de Streams

9.2.1. Organizacion

La tecnología Java contiene distintos tipos de Streams, los cuales están organizados en dos grandes grupos:

- Streams orientados a Carácter (Character Streams)
- Streams orientados a Byte (Byte Streams)

9.2.2. Streams orientados a Carácter

Los *Streams orientados a Carácter* operan con caracteres como unidad de trabajo. Los caracteres a leer están formados por 2 bytes (es decir 16 bits por carácter).

Son utilizados para leer y escribir información que está almacenada en forma de texto, como por ejemplo archivos de extensión txt, ini, csv, etc.

La superclase utilizada para leer streams orientados a carácter es la clase *Reader*. A partir de esta clase – la cual es abstracta – heredan todas las clases concretas que se utilizan para leer información en forma textual.

Por otra parte, la superclase utilizada para escribir streams orientados a carácter es la clase *Writer*. A partir de esta clase – la cual es abstracta – heredan todas las clases concretas que se utilizan para escribir información en forma textual.

9.2.3. Streams orientados a Byte

Los *Streams orientados a Byte* operan con bytes como unidad de trabajo. Los bytes a leer se leen en forma unitaria (es decir 8 bits por byte).

Son utilizados para leer y escribir información que está almacenada en forma binaria, como por ejemplo archivos de extensión jpeg, mpeg, xls, etc.

La superclase utilizada para leer streams orientados a byte es la clase *InputStream*. A partir de esta clase – la cual es abstracta – heredan todas las clases concretas que se utilizan para leer información en forma binaria.

Por otra parte, la superclase utilizada para escribir streams orientados a byte es la clase *OutputStream*. A partir de esta clase – la cual es abstracta – heredan todas las clases concretas que se utilizan para escribir información en forma binaria.

9.3. Que es un File Stream

9.3.1. Definición

Los File Streams son los streams utilizados para lectura y escritura de (particularmente) archivos, es una categoría que agrupa tanto a los streams orientados a carácter como a los streams orientados a byte.

En general se utilizan en conjunto con un objeto del tipo *File*, que es una representación abstracta de un archivo. La clase *File* modela tanto archivos como directorios.

9.3.2. Lectura de un Archivo de Texto

La clase *FileReader* es una clase concreta utilizada para generar streams orientados a carácter, y es la encargada de realizar la lectura de archivos en forma de texto.

A continuación se presenta un ejemplo de lectura de un archivo llamado fuente.txt:

```
import java.io.*;  
  
public class Lector  
{  
    public static void main(String[] args) throws IOException  
    {  
        // Define el archivo a utilizar  
        File archivoEntrada = new File("fuente.txt");  
  
        // Abre el stream necesario  
        FileReader in = new FileReader(archivoEntrada);  
  
        int unCaracter;  
  
        // Lee el archivo  
        while ( (unCaracter = in.read()) != -1)  
            System.out.print((char)unCaracter);  
  
        // Cierra el stream  
        in.close();  
    }  
}
```

9.3.3. Escritura de un Archivo de Texto

La clase *FileWriter* es una clase concreta utilizada para generar streams orientados a carácter, y es la encargada de realizar la escritura de archivos en forma de texto.

A continuación se presenta un ejemplo de escritura de un archivo llamado destino.txt:

```
import java.io.*;

public class Escritor
{
    public static void main(String[] args) throws IOException
    {
        // Define el archivo a utilizar
        File archivoSalida = new File("destino.txt");

        // Abre el stream necesario
        FileWriter out = new FileWriter(archivoSalida);

        // Define la información a guardar en el archivo
        String info = "Soy la información";

        // Escribe el archivo con la información
        for(int i=0; i<info.length(); i++)
            out.write( info.charAt(i) );

        // Cierra los streams
        out.close();
    }
}
```

edudación

9.3.4. Lectura y Escritura de Archivos Binarios

La clase *FileInputStream* es una clase concreta utilizada para generar streams orientados a byte, y es la encargada de realizar la lectura de archivos en forma binaria.

La clase *FileOutputStream* es una clase concreta utilizada para generar streams orientados a byte, y es la encargada de realizar la escritura de archivos en forma binaria.

A continuación se presenta un ejemplo de lectura de un archivo llamado *fuente.gif* y la escritura de dicho contenido en un archivo llamado *destino.gif*. La clase *Copificador* presentada a continuación lee la imagen, y genera un duplicado de la misma:

```
import java.io.*;
public class Copificador
{
    public static void main(String[] args) throws IOException
    {
        // Define los archivos a utilizar
        File archivoEntrada = new File("fuente.gif");
        File archivoSalida = new File("destino.gif");

        // Abre los streams necesarios
        FileInputStream in =
            new FileInputStream(archivoEntrada);
        FileOutputStream out =
            new FileOutputStream(archivoSalida);

        int unCaracter;

        // Copia el archivo fuente en el archivo destino
        while ( (unCaracter = in.read()) != -1)
            out.write(unCaracter);

        // Cierra los streams
        in.close();
        out.close();
    }
}
```

9.4. Que son los Buffers

9.4.1. Definicion

Los *buffers* son una alternativa a las clases básicas de entrada y salida. Son subclases de las clases básicas correspondientes a la clase Reader, Writer, InputStream y OutputStream.

Cumplen el mismo objetivo que dichas clases, pero son eficientes ya que tienen como objetivo guardar en un buffer los caracteres leídos / por escribir para lograr un mejora sustancial en la lectura / escritura.

Son utilizados como wrappers (envoltorios) para envolver las clases básicas.

9.4.2. La clase BufferedReader

La clase *BufferedReader* es una clase que hereda de la clase Reader y se utiliza para envolver a otras subclases del tipo Reader, tales como la clase FileReader.

Entre los métodos más utilizados se encuentra el método *readLine()*, que permite leer un conjunto de caracteres retornado en forma de String, en lugar de leer carácter a carácter.

La clase BufferedReader puede ser utilizada de la siguiente manera:

```
// Instancia un objeto del tipo BufferedReader llamado in
File archivo = new File("fuente.txt");
BufferedReader in = new BufferedReader(
                    new FileReader(archivo) );

// Lee la primer linea del archivo fuente.txt a traves del stream
// denominado in
String lineaLeida = readerMejorado.readLine();

// Cierra el stream y libera recursos
in.close();
```

9.4.3. La clase BufferedWriter

La clase *BufferedWriter* es una clase que hereda de la clase *Writer* y se utiliza para envolver a otras subclases del tipo *Writer*, tales como la clase *FileWriter*.

Entre los métodos más utilizados se encuentra el método *write(String s, int offset, int length)*, que permite para escribir un conjunto de caracteres en lugar de leer carácter a carácter.

Adicionalmente, posee un método denominado *newLine()* que se utiliza para escribir en el stream la representación de una nueva línea, es decir la tecla “enter”, entendiendo que cada sistema operativo puede representarla con distintos caracteres.

La clase *BufferedWriter* puede ser utilizada de la siguiente manera:

```
// Instancia un objeto del tipo BufferedWriter llamado out
File archivo = new File("destino.txt");
BufferedWriter out = new BufferedWriter(
    new FileWriter(archivo) );

// Línea a escribir
String linea1 = "Hola, soy una línea a escribir en el archivo";

// Escribe la cadena de caracteres en el archivo
writerMejorado.write(linea1, 0, linea1.length());

// Escribe un "enter" en el archivo
writerMejorado.newLine();

// Cierra el stream y libera recursos
out.close();
```

9.4.4. La clase BufferedInputStream

El objetivo de la clase *BufferedInputStream* es el mismo que el de la clase *BufferedReader*, la diferencia radica en que el tratamiento de la informacion es a nivel bytes, y no a nivel caracteres.

9.4.5. La clase BufferedOutputStream

El objetivo de la clase *BufferedOutputStream* es el mismo que el de la clase *BufferedWriter*, la diferencia radica en que el tratamiento de la informacion es a nivel bytes, y no a nivel caracteres.

10. Base de datos

10.1. El lenguaje SQL

El Lenguaje de Consulta Estructurado (SQL= Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar información de interés de una base de datos, de una forma sencilla.

El SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones sobre los mismos. Es un lenguaje declarativo de alto nivel, que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación.

Es el lenguaje utilizado universalmente para interactuar con base de datos, permitiendo realizar consultas, inserciones, actualizaciones y eliminaciones de datos, como así también de base de datos.

10.1.2. Donde se utiliza

En la actualidad el SQL es el estándar de facto de la inmensa mayoría de los Administradores de Base de Datos comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio. El soporte estándar se denomina ANSI SQL.

Entre los sistemas de gestión de base de datos con soporte SQL más utilizados se encuentran los siguientes:

- DB2
- Oracle
- SQL Server
- MySQL
- PostgreSQL
- Informix

10.2. MySQL como Data Base Management System

10.2.1. Introducción

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario. Inicialmente, carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. A pesar de ello, atrajo a los desarrolladores de páginas web con contenido dinámico, justamente por su simplicidad y velocidad.

Poco a poco los elementos de los que carecía MySQL están siendo incorporados tanto por desarrollos internos, como por desarrolladores de software libre.

10.2.2. Características

Entre las características disponibles en las últimas versiones se puede destacar:

- Amplio subconjunto del lenguaje SQL. Algunas extensiones son incluidas igualmente.
- Disponibilidad en gran cantidad de plataformas y sistemas.
- Diferentes opciones de almacenamiento según si se desea velocidad en las operaciones o el mayor número de operaciones disponibles.
- Transacciones y claves foráneas.

- Conectividad segura.
- Replicación.
- Búsqueda e indexación de campos de texto.

La licencia GNU GPL de MySQL obliga a distribuir cualquier producto derivado (aplicación) bajo esa misma licencia. Si un desarrollador desea incorporar MySQL en su producto pero no desea distribuirlo bajo licencia GNU GPL, puede adquirir la licencia comercial de MySQL que le permite hacer justamente eso.

10.3. Qué es DDL?

El lenguaje de Definición de datos, en inglés Data Definition Language (DDL), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Algunas de sus operaciones básicas son: CREATE, ALTER y DROP.

Involucra los comandos necesarios para crear, modificar y eliminar una tabla y una base de datos, como así tambien permite crear claves primarias, indices y restricciones.

10.3.1. La operacion CREATE

Este comando crea un objeto dentro de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte.

A continuacion un ejemplo de utilizacion para la creacion de una tabla:

```
CREATE TABLE TABLA_NOMBRE (
    my_field1  INT UNSIGNED,
    my_field2  VARCHAR (50),
    my_field3  DATE NOT NULL,
    PRIMARY KEY (my_field1, my_field2)
)
```

10.3.2. La operacion ALTER

Este comando permite modificar la estructura de un objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.

A continuacion un ejemplo de utilizacion para agregar una columna a una tabla:

```
ALTER TABLE TABLA_NOMBRE (
    ADD NUEVO_CAMPO INT UNSIGNED
)
```

10.3.3. La operacion DROP

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte. Se puede combinar con la sentencia ALTER.

A continuacion un ejemplo de utilizacion para agregar eliminar una tabla:

```
DROP TABLE TABLA_NOMBRE
```

10.4. Qué es DML?

DML significa Data Manipulation Language o Lenguaje de Manipulación de Datos, y corresponde las sentencias del SQL que se utilizan para manejar los datos de la base de datos (select, insert, update, delete, etc).

Involucra los comandos necesarios para hacer consultas, inserciones, modificaciones y eliminaciones. Dichos comandos estan presentados de la siguiente manera:

- SELECT – obtiene información de una base de datos
- INSERT INTO – inserta información en una base de datos
- UPDATE - actualiza información de una base de datos
- DELETE – elimina información de una base de datos

10.4.1. El comando SELECT

El comando SELECT se utiliza para seleccionar información de una tabla. Para seleccionar todas la columnas se utiliza el * (asterisco), y la clausula WHERE se utiliza para establecer un criterio de búsqueda.

Para traer todos los datos de una tabla, se realiza la siguiente consulta:

```
SELECT * FROM tabla_nombre
```

Si es necesario traer únicamente los datos de una columna, se realiza de la siguiente manera:

```
SELECT nombre_columna(s) FROM tabla_nombre
```

Para traer los registros segun una condicion, se realiza de la siguiente manera:

```
SELECT nombre_columna(s) FROM tabla_nombre WHERE campo1 = valor1
```

10.4.2. El comando INSERT

El comando INSERT se utiliza para insertar datos en una tabla. La insercion se realiza de la siguiente manera:

```
INSERT INTO "nombre_tabla" ("columna1", "columna2", ...)  
VALUES ("valor1", "valor2", ...)
```

Se pueden agregar datos en grupo o especificando la columna donde es necesario ingresar la informacion. Para agregar informacion a todos los campos, el uso es el siguiente:

```
INSERT INTO nombre_tabla  
VALUES (valor1, valor2,...)
```

Para agregar valores en cada columna de manera particular, se realiza de la siguiente forma:

```
INSERT INTO nombre_tabla(columna1, columna2, ...)  
VALUES (valor1, valor2,...)
```

10.4.3. El comando UPDATE

El comando UPDATE se utiliza para actualizar registros, y su forma de utilizacion es la siguiente:

```
UPDATE nombre_tabla  
SET nombre_columna1 = nuevo_valor, nombre_columna2 = otro_valor WHERE  
nombre_columna = algun_valor
```

10.4.4. El comando DELETE

El comando DELETE se utiliza para eliminar registros, su forma de utilizacion es la siguiente:

```
DELETE FROM nombre_tabla WHERE nombre_columna = algun_valor
```

educationIT

11. JDBC: Conexion con Base de Datos

11.1. Introducción

11.1.1. Que es JDBC

JDBC es el acrónimo de Java Database Connectivity, una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede utilizando el lenguaje SQL del modelo de base de datos que se utilice.

Esta conformada por diversas clases e interfaces ubicadas en el paquete *java.sql*.

11.1.2. La necesidad de una librería

Al trabajar con JDBC resulta necesario agregar un jar al proyecto que contiene las clases necesarias que se utilizan para “dialogar” con un DBMS. Cada DBMS tiene su propio archivo jar. Estos archivos se pueden obtener de:

- <http://developers.sun.com/product/jdbc/drivers>

Tambien es posible conseguir estos archivos jar en la pagina web correspondiente a cada DBMS, por ejemplo, en caso de usar un DBMS Oracle es posible buscarlo en la pagina de Oracle: www.oracle.com

11.2. Conexión con la base de datos

11.2.1. La interfaz Connection

Para poder trabajar con una base de datos, el punto de partida siempre es conseguir una conexión, es decir un objeto del tipo Connection (este objeto pertenece a una clase que implementa la interfaz Connection).

11.2.2. Construcción de un Administrador de Conexiones

Para poder obtener una conexión, una forma simple y comoda de trabajar es armar una clase llamada, por ejemplo, *AdministradorDeConexiones*, que contenga dentro de un método *obtenerConexion()* el código necesario para obtenerla. A continuación se presenta un ejemplo de la clase con su correspondiente método:

```
package ar.com.educacionIT.database;

import java.sql.Connection;
import java.sql.DriverManager;

public abstract class AdministradorDeConexiones {

    public static Connection obtenerConexion() throws Exception {
        // Establece el nombre del driver a utilizar
        String dbDriver = "com.mysql.jdbc.Driver";

        // Establece la conexión a utilizar
        String dbConnString = "jdbc:mysql://localhost/baseDatos";

        // Establece el usuario de la base de datos
        String dbUser = "root";

        // Establece la contraseña de la base de datos
        String dbPassword = "";

        // Establece el driver de conexión
        Class.forName(dbDriver).newInstance();

        // Retorna la conexión
        return DriverManager.getConnection(
            dbConnString, dbUser, dbPassword);
    }
}
```

11.3. Como consultar datos

11.3.1. El metodo createStatement()

El metodo *createStatement()* se utiliza para crear un objeto que modela a una sentencia SQL. Es un objeto del tipo de una clase que implementa la interfaz Statement, y provee la infraestructura para ejecutar sentencias SQL sobre una conexión con una base de datos.

La forma de construir un objeto de este tipo es:

```
Statement stmtConsulta = laConexion.createStatement();
```

11.3.2. El metodo executeQuery()

El metodo *executeQuery()* se utiliza para ejecutar una sentencia SQL y obtener el resultado correspondiente dentro de un objeto del tipo ResultSet. Este objeto representa un conjunto de resultados que se obtienen como consecuencia de ejecutar la sentencia SQL del tipo SELECT a través de la conexión.

La forma de generar un objeto de este tipo es:

```
ResultSet rs = stmtConsulta.executeQuery(laConsulta);
```

11.3.3. Como realizar una consulta

```
// Define la conexión
Connection laConexion = AdministradorDeConexiones.getConnection();

// Arma la consulta y la ejecuta
String laConsulta = "SELECT * FROM alumnos";
Statement stmtConsulta = laConexion.createStatement();
ResultSet rs = stmtConsulta.executeQuery(laConsulta);

// Muestra los datos
while( rs.next() ){
    System.out.println("ID: " + rs.getInt("alu_id") + " -- " +
        "Nombre: " + rs.getString("alu_nombre") + " -- " + "Apellido: " +
        rs.getString("alu_apellido"));
}

// Cierra el Statement y la Connection
stmtConsulta.close();
laConexion.close();
```

educación

11.4. Como insertar datos

11.4.1. El metodo createStatement()

El metodo `createStatement()` es el mismo presentado en la sección Consulta.

11.4.2. El metodo execute()

El metodo `execute()` se utiliza para ejecutar sentencias SQL del tipo INSERT, UPDATE o DELETE, y a diferencia del metodo `executeQuery()` no retorna un conjunto de resultados.

La forma de utilizar el metodo `execute()` es:

```
String laInsercion =  
    "INSERT INTO alumnos (alu_id, alu_nombre, alu_apellido) VALUES  
    (101, 'Manuel', 'Santos')";  
  
stmtInsercion.execute(laInsercion);
```

11.4.3. Como realizar una insercion

```
// Define la conexion  
Connection laConexion = AdministradorDeConexiones.getConnection();  
  
// Arma la sentencia de insercion y la ejecuta  
String laInsercion = "INSERT INTO alumnos (alu_id, alu_nombre,  
    alu_apellido) VALUES (101, 'Manuel', 'Santos')";  
Statement stmtInsercion = laConexion.createStatement();  
stmtInsercion.execute(laInsercion);  
  
// Cierra el Statement y la Connection  
stmtInsercion.close();  
laConexion.close();  
  
// Informa que la insercion ha sido realizada con exito  
System.out.println("La insercion ha sido realizada con exito...");
```

11.5. Como actualizar datos

11.5.1. El metodo createStatement()

El metodo *createStatement()* es el mismo presentado en la sección Consulta.

11.5.2. El metodo execute()

El metodo *execute()* es el mismo presentado en la sección Insercion.

11.5.3. Como realizar una actualizacion

```
// Define la conexion
Connection laConexion = AdministradorDeConexiones.getConnection();

// Arma la sentencia de actualización y la ejecuta
String laActualizacion = "UPDATE alumnos SET alu_apellido =
    'Troppiani' WHERE alu_id = 101";
Statement stmtActualizacion = laConexion.createStatement();
stmtActualizacion.execute(laActualizacion);

// Cierra el Statement y la Connection
stmtActualizacion.close();
laConexion.close();

// Informa que la actualización ha sido realizada con éxito
System.out.println("La actualización ha sido realizada con
    éxito...");
```

11.6. Como eliminar datos

11.6.1. El metodo createStatement()

El metodo *createStatement()* es el mismo presentado en la sección Consulta.

11.6.2. El metodo execute()

El metodo *execute()* es el mismo presentado en la sección Insercion.

11.6.3. Como realizar una eliminacion

```
// Define la conexion
Connection laConexion = AdministradorDeConexiones.getConnection();

// Arma la sentencia de eliminación y la ejecuta
String laEliminacion = "DELETE FROM alumnos WHERE alu_id = 101";
Statement stmtEliminacion = laConexion.createStatement();
stmtEliminacion.execute(laEliminacion);

// Cierra el Statement y la Connection
stmtEliminacion.close();
laConexion.close();

// Informa que la eliminación ha sido realizada con éxito
System.out.println("La eliminación ha sido realizada con éxito...");
```

11.7. Transacciones

11.7.1. Que es un DAO

Cuando se pretende modelar con objetos un modelo de datos, es decir las tablas y sus relaciones, es muy comun utilizar una propuesta simple que consiste en armar una clase por cada tabla que existente.

Dicha clase tendra los metodos de acceso a la tabla correspondiente, entre ellos la insercion, modificacion y la eliminacion. Este tipo de clases suelen conocerse como DAOs, ya que objetos de esta clase se utilizaran para realizar operaciones de datos.

De esta forma, si existe la tabla autos, es posible construir una clase denominada Auto, que siguiendo la especificacion de un DAO, deberia tener los metodos insertar, modificar, eliminar, y metodos con las consultas que resulten necesarias.

11.7.2. Que es una transaccion

Una transacción en SQL es una colección de sentencias DML que forman una unidad lógica de trabajo o procesamiento, con propiedades bien definidas. De esta manera será un conjunto de operaciones sobre los datos en una base de datos que o se ejecuta entera o no se ejecuta ninguna de sus sentencias.

JDBC permite que las declaraciones de SQL sean agrupadas juntas en una sola transacción. De esta manera, es posible asegurar la atomicidad y consistencia de datos, usando características transaccionales de JDBC.

11.7.3. El metodo setAutoCommit()

El control de la transacción es realizado por el objeto de la conexión. Cuando se crea una conexión, por defecto está en el modo activado. Esto significa que cada declaración individual del SQL es tratada como transacción por sí mismo, y será comprometida tan pronto como finalice la ejecución.

El metodo setAutoCommit es el encargado de establecer si se trabajara agrupando varias sentencias en una transaccion, o por el contrario cada sentencia SQL sera una transaccion independiente.

Para trabajar con varias sentencias SQL y ejecutarlas como una transaccion es necesario establecer el auto-commit en false:

```
unaConexion.setAutoCommit(false);
```

Si no es necesario trabajar con transacciones, sera necesario establecer el auto-commit en true:

```
unaConexion.setAutoCommit(true);
```

Por defecto el auto-commit esta seteado en true.

11.7.4. El metodo commit()

Cuando el auto-commit esta en falso, para poder comprometer o impactar los cambios en la base de datos, es necesario llamar al metodo commit(). Si no se llama al metodo commit(), los cambios no seran reflejados en la base de datos aun cuando se hayan ejecutados una o mas sentencias SQL.

El metodo commit() pertenece a la conexion, y la forma de invocarlo es la siguiente:

```
unaConexion.commit();
```

11.7.5. El metodo rollback()

Si en algun punto antes de invocar el metodo commit(), el metodo rollback() es invocado, todas las sentencias que se hayan ejecutado quedaran sin efecto, es decir que el rollback() vuelve atras todos los cambios realizados sobre los datos desde el ultimo commit() realizado

El metodo rollback() tambien pertenece a la conexion, y la forma de invocarlo es la siguiente:

```
unaConexion.rollback();
```

11.7.6. Utilizacion de transacciones

```
// Declara la conexión  
Connection conn = null;  
  
// Define la conexión  
conn = AdministradorDeConexiones.getConnection();  
  
// Setea el auto-commit en falso  
conn.setAutoCommit(false);  
  
// Setea el auto-commit en falso  
Alumno a1 = new Alumno("Juan");  
Alumno a2 = new Alumno("Mario");  
Alumno a3 = new Alumno("Pepe");  
  
// Acá comienza la transacción  
alumno1.insertar(conn);  
alumno2.insertar(conn);  
alumno3.insertar(conn);  
  
// Confirma los cambios  
conn.commit();  
  
// Cierra la conexión  
conn.close()
```

educación

11.7.7. Utilizacion de transacciones con manejo de excepciones

```
// Declara la conexion  
Connection conn = null;  
  
try  
{  
    // Define la conexion  
    conn = AdministradorDeConexiones.getConnection();  
  
    // Setea el auto-commit en falso  
    conn.setAutoCommit(false);  
  
    // Acá comienza la transacción  
    alumno1.insertar(conn);  
    alumno2.insertar(conn);  
    alumno3.insertar(conn);  
  
    // Confirma los cambios  
    conn.commit();  
}  
  
catch(Exception e)  
{  
    try  
    {  
        // Vuelve atras los cambios  
        conn.rollback();  
    }  
    catch(Exception ee){ // Manejo de errores }  
}  
  
finally  
{  
    try  
    {  
        // Cierra la conexión  
        if( conn != null ) conn.close();  
    }  
    catch(Exception e){ // Manejo de errores }  
}
```

educaciónIT

12. Laboratorios

12.1. Lab #1 - Conceptos Basicos de JAVA

12.1.1. Ejercicio #1

Construir un vector de enteros llamado vecNumeros e inicializarlo con los numeros 11, -22, 33, -44, 55, -66, 77, -88, 99.

Construir un vector de enteros llamados vecPositivos, de longitud 10, que contenga los numeros positivos de vecNumeros.

Construir un vector de enteros llamado vecNegativos, de longitud 10, que contenga los numeros negativos de vecNumeros.

Acumular la suma de los valores positivos en una variable totalPositivos.

Acumular la suma de los valores negativos en una variable totalNegativos.

Imprimir en pantalla:

- ✓ el contenido de vecNumeros
- ✓ el contenido de vecPositivos
- ✓ el contenido de vecNegativos
- ✓ los valores de totalPositivos y totalNegativos

Imprimir la suma totalPositivos y totalNegativos, sin utilizar una variable extra para sumarizarlos.

Imprimir los valores 1000 2000 3000 pasados como argumentos del programa principal.

12.2. Lab #2 - Programacion Orientada a Objetos

12.2.1. Ejercicio #1

- o Crear la clase *abstracta* Persona con las siguientes características:
 - nombre (*privado* y de la clase java.lang.String)
 - fechaDeNacimiento (*privado* y de la clase java.util.Date)
 - métodos de acceso (setters y getters) correspondientes
- o Crear la clase Zoologico que tenga las siguientes características:
 - Atributos de clase públicos y constantes CANTIDAD_ANIMALES = 25 y RACIONES_POR_ANIMAL = 5 del tipo entero
 - Atributo *privado* abierto del tipo boolean representando si el zoologico está abierto o cerrado
 - Método abrir() para abrir el zoológico
 - Método alimentarAnimales(int unaCantidadDeRaciones) → deberá instanciar a un Cuidador pasándole la cantidad de raciones en el constructor del mismo
 - Método cerrar() para cerrar el zoológico
- o Crear la clase Cuidador que tenga las siguientes características:
 - Es el responsable de alimentar a los animales
 - Es una subclase de Persona
 - Posee un atributo *entero privado* cantidadDeRaciones que se setea cuando se construye el objeto
 - Método alimentarAnimales() que informará si la cantidad de raciones que le entregaron al cuidador fue suficiente para alimentar a todos los animales
- o Crear la clase Programa que la utilizaremos como punto de entrada principal
 - Deberá crear una instancia de Zoologico para abrir el zoológico, alimentar a los animales y cerrar el zoológico

12.3. Lab #3 – Conceptos Generales

12.3.1. Ejercicio #1

Analizar el comportamiento de la clase TestSystem ubicada en el paquete ar.com.educacionit.lab3.ejercicio1

Debatir acerca del uso de los métodos

- ✓ getProperty()
- ✓ getProperties()
- ✓ exit()
- ✓ gc()

12.3.2. Ejercicio #2

Analizar el comportamiento de la clase Comparable ubicada en el paquete ar.com.educacionit.lab3.ejercicio2

Ejecutar la clase Comparable utilizando los diferentes casos de prueba y evaluar el comportamiento del operador == y el método equals

12.4. Lab #4 - Colecciones

12.4.1. Ejercicio #1

- Generar la clase Empleado con las siguientes características:
 - atributos nombre y dni de tipo String
 - atributo edad de tipo int
 - un constructor de que soporte pasar el nombre, dni y edad
 - respetar el concepto de encapsulamiento
- Armar una clase Empresa, donde el metodo main deberá tener la siguiente secuencia de actividades:
 - a. instanciar N (al menos 6) empleados con sus nombres, dni y edad
 - b. agregar los N empleados a una ArrayList llamado losEmpleados
 - c. recorrer el ArrayList losEmpleados e imprimir en pantalla los datos (nombre / dni / edad) de cada empleado
 - d. imprimir en pantalla la cantidad de empleados que tiene la empresa
 - e. recorrer el ArrayList losEmpleados e imprimir en pantalla los datos (nombre / dni / edad) de cada empleado menor de 30 años
 - f. para realizar el punto c. armar un método con la siguiente firma:

```
public static void informarDatosDeEmpleados (ArrayList  
losEmpleados)
```
 - g. para realizar el punto e. armar un método con la siguiente firma:

```
public static ArrayList  
obtenerDatosDeEmpleadosMenosDe30(ArrayList losEmpleados)
```
 - De ser posible reutilizar el método informarDatosDeEmpleados()

12.4.2. Ejercicio #2

- Título: La clase *Vector* como contenedor
- Fuente: cursojava.ejercicios.contenedores.vectors.*

Reemplazar la palabra *ArrayList* por *Vector* en el ejercicio anterior y analizar el funcionamiento

12.4.3. Ejercicio #3

- Construir una clase llamada Cliente que tenga como atributos “razon social” y “dirección”. Respetar el encapsulamiento
- Redefinir el método `toString()` para que retorne la razon social y la dirección
- Construir una clase llamada ImpresorDeClientes
- El código a desarrollar estará dentro del método main de esta clase
- Instanciar 5 clientes y agregarlos a un ArrayList denominado losClientes
Utilizar un iterador para recorrer losClientes e imprimir los datos de cada uno en pantalla

educación

12.5. Lab #5 - Excepciones

12.5.1. Ejercicio #1

Crear la clase Programa que recibe dos valores como argumentos en su método main(). La clase Programa deberá corroborar que se reciban únicamente 2 valores, y que ambos sean numéricos.

En caso de que la cantidad de argumentos sea distinta a dos, lanzar la excepción CantidadDeArgumentosException.

En caso de que al convertir un argumento a entero se lance una excepción, capturarla e imprimir el mensaje “Los dos valores deben ser numéricos” en pantalla.

La clase Programa deberá tener un método público, que recibe los dos argumentos, y que devuelve un float con el resultado del primer argumento dividido el segundo. En caso de que el divisor sea cero, el método deberá lanzar la excepción para ser atrapada por la clase Programa y mostrar el error.

Cualquier otra excepción que exista deberá ser también capturada.

Crear la excepción CantidadDeArgumentosException:

- ✓ con los atributos id, nivelDeError y mensajeError, todos de tipo String
- ✓ con un constructor que recibe los valores id, nivel de error y mensaje de error

Respetar el concepto de encapsulamiento.

12.6. Lab #6 - Streams

12.6.1. Ejercicio #1

Codificar la clase *Lector* presentada anteriormente en el manual. Construir un archivo denominado *fuente.txt* que sera el archivo a ser leido, y ubicarlo en la carpeta raiz del proyecto.

Al ejecutar la clase Lector, debera informar el contenido de fuente.txt en la consola de salida.

12.6.2. Ejercicio #2

Codificar la clase *Escritor* presentada anteriormente en el manual.

Al ejecutar la clase Escritor, debera escribir "Soy la informacion" en el archivo destino.txt.

12.6.3. Ejercicio #3

Armar una clase denominada Copiador. Dentro de su metodo main() debera leer un archivo denominado *fuente.gif*, y copiar su contenido a un archivo denominado *destino.gif*

12.6.4. Ejercicio #4

Construir la clase *LectorMejorado* la cual debera leer el contenido de un archivo denominado *fuente.txt*. Utilizar la clase *BufferedReader* como stream de lectura.

Al ejecutar la clase Lector, debera informar el contenido de fuente.txt en la consola de salida.

12.6.5. Ejercicio #5

Construir la clase *EscritorMejorado* la cual debera escribir informacion en un archivo denominado *destino.txt*. Utilizar la clase *BufferedWriter* como stream de escritura.

Al ejecutar la clase Escritor, debera escribir en el archivo destino.txt los siguientes strings: "Hola, soy la linea # 1", luego en una linea nueva "Como te va? Yo soy la linea # 2!", luego en una linea nueva "Y yo la linea # 3!!!".

12.6.6. Ejercicio #6

Armar una clase denominada *CopiadorMejorado*. Dentro de su metodo main() debera leer un archivo denominado *fuente.txt*, y copiar su contenido a un archivo denominado *destino.txt*

Utilizar la clase *BufferedReader* como stream de lectura y la clase *BufferedWriter* como stream de escritura

12.6.7. Ejercicio #7

Realizar un copiador visual a partir de la siguiente ventana:



Si lanza alguna excepcion, debera ser capturada y mostrar el error en una caja de dialogo

Si el archivo es copiado correctamente, mostrar en una caja de dialogo la leyenda "El archivo ha sido copiado con exito"

12.6.8. Ejercicio #8

Analizar la clase *StreamArchivoRemoto* que viene con la entrega del código fuente correspondiente a los resueltos.

Dicha clase se utilizara para conectarse contra un sitio web, y obtener la información del HTML en forma de texto utilizando un BufferedReader.

12.6.9. Ejercicio #9

Analizar la clase *LecturaDesdeClasspath* que viene con la entrega del código fuente correspondiente a los resueltos.

Dicha clase ejemplifica como leer un archivo de texto ubicado dentro de un paquete.

12.7. Lab #7 - Interfaz Grafica de Usuario

12.7.1. Ejercicio #1

El objetivo del ejercicio es modelar una institucion de educacion que posee *tres niveles*: Jardin, Primaria y Secundaria

El *cliente* de la institucion representa a un parent que tiene *hijos* cursando en los tres distintos niveles

Se debera construir una *interfaz grafica* que se utilizara para las modificaciones de un cliente, tomando como punto de partida que este ya ha sido dado de alta y sus hijos estan cursando

La construccion de la pantalla se deberá realizar en *cuatro etapas*:

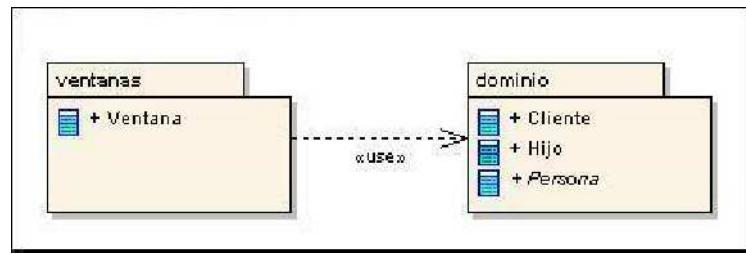
- ✓ 1. Diseño y Desarrollo de Clases
- ✓ 2. Construcción de la GUI
- ✓ 3. Inicialización de la pantalla
- ✓ 4. Manejo de Eventos

Las *etapas* se deberán realizar en *forma ordenada*, pasando a la siguiente etapa únicamente cuando la etapa actual haya sido finalizada

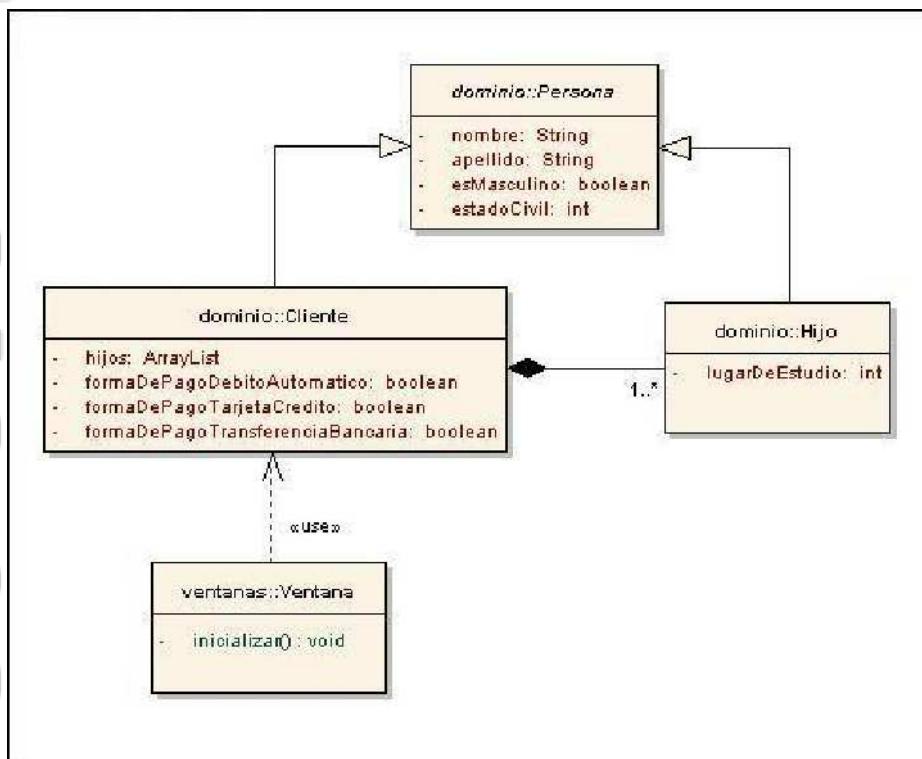
Etapa # 1 - Diseño y Desarrollo de Clases

Se deberá realizar la implementación de las siguientes clases surgidas de la etapa de Diseño, respetando nombres de clases, atributos, métodos y paquetes, según corresponda.

El *Diagrama de Paquetes* representa la organización de paquetes que se deberá realizar:



El *Diagrama de Clases* representa el aspecto estructural del sistema, por lo que se deberá seguir la siguiente especificación:



Etapa # 2 - Construcción de la GUI

Construir la siguiente interfaz gráfica de usuario:

Datos del Cliente

Nombre:

Apellido:

Sexo:

Estado Civil:

Formas de Pago

Tarjeta de crédito

Débito automático

Transferencia bancaria

Hijos en jardín

Hijos en primaria

Hijos en secundaria

Datos del hijo seleccionado

La caja de texto “Datos del hijo seleccionado” deberá ser del tipo “JTextArea”, y además deberá ser de solo lectura.

Etapa # 3 - Inicializacion de la pantalla

La inicializacion de la pantalla representa las acciones que deben ocurrir cuando la pantalla es ejecutada

La codificacion debera realizarse en el metodo *inicializar()* correspondiente a la clase Ventana. Dicho metodo debera ser invocado en el constructor de la clase Ventana

Se deberan realizar las siguientes actividades:

- ✓ Instanciar un cliente, y completar sus atributos
- ✓ Instanciar cinco hijos, y completar los atributos correspondientes. No olvidar establecer el lugar de estudio
- ✓ Instanciar un objeto del tipo ArrayList, agregar los hijos y vincularlo al cliente a traves del atributo hijos
- ✓ Llenar combo sexo con opciones “Masculino” y “Femenino”
- ✓ Llenar combo estado civil con opciones “Soltero”, “Casado” y “Divorciado”
- ✓ Establecer en las cajas de texto, combos y checkboxes los valores correspondientes al cliente
- ✓ Llenar los combos Jardin, Primario y Secundario con los hijos segun corresponda



Etapa # 4 - Manejo de Eventos

El manejo de eventos es la programacion de las acciones que deben ocurrir cuando se realiza algun evento, por ejemplo el evento click sobre un boton

Se debera realizar la programacion de los siguientes eventos:

- ✓ Al presionar el *boton “Promover”* que se encuentra *debajo del combo Jardín*, se deberá mover el hijo seleccionado en combo Jardin al combo Primario. En caso de no haber ningun hijo seleccionado en combo Jardin, enviar un mensaje de advertencia
- ✓ Al presionar el *boton “Promover”* que se encuentra *debajo del combo Primario*, se deberá mover el hijo seleccionado en combo Primario al combo Secundario. En caso de no haber ningun hijo seleccionado en combo Primario, enviar un mensaje de advertencia
- ✓ Al *seleccionar un hijo en un combo*, los datos del mismo deberán aparecer en la caja de texto “Datos del hijo seleccionado”
- ✓ Al *seleccionar un hijo en un combo*, los otros combos deben visualizarse con ningún item seleccionado
- ✓ Al presionar el *boton “Guardar cambios”* se deberá realizar la siguiente pregunta: “Confirma actualización de datos?”. La respuesta deberá ser SI / NO

Tip: se recomienda la utilizacion de los metodos `removeItem()`, `setSelectedItem()` y/o `setSelectedIndex()` para la gestion de los combos.



12.8. Lab #8 – Acceso a Base de Datos

12.8.1. Ejercicio #1

Analizar la organización del proyecto `educacionIT-Lab-08-AccesoBaseDatos`

Analizar la clase `AdministradorDeConexiones` ubicada en el paquete `ar.com.educacionit.lab8.ejercicio1.administrador`

Abrir el archivo `alumnos.sql` ubicado en el paquete `ar.com.educacionit.lab8.ejercicio1.ddl` y ejecutarlo como un script en MySQL Query Browser (File > > New Script Tab)

Una vez ejecutado el script, se habrá creado la base de datos y la tabla para utilizar en este tutorial

Recorrer, ejecutar y analizar los archivos ubicados dentro del paquete `ar.com.educacionit.lab8.ejercicio1.tutorial` en el siguiente orden:

- ✓ `SQLSelectSample`
- ✓ `SQLInsertSample`
- ✓ `SQLUpdateSample`
- ✓ `SQLDeleteSample`
- ✓ `ManejoDeExcepciones`
- ✓ `CommitRollback`

13. Proyecto Integrador

13.1. Fase #1 – Detección de clases y Construcción Base del proyecto

13.1.1. Requisitos

Para la construcción de la presente fase requiere:

- ✓ conocer que es una clase, un objeto, los constructores y los setters y getters
- ✓ conocer los tipos de datos
- ✓ conocer las palabras claves private y public

13.1.2. Objetivos

Los objetivos de la presente fase son:

- ✓ construir el proyecto para comenzar a trabajar
- ✓ construir las clases que formaran la base del proyecto

13.1.3. Especificación

- 
- Armar la clase Vehiculo con los siguientes atributos:
 - alto, de tipo int
 - ancho, de tipo int
 - largo, de tipo int
 - Construir los setters y los getters de forma automática
 - Armar un constructor que reciba como parámetros los valores correspondientes a alto, ancho y largo
 - Armar la clase Persona con los siguientes atributos:
 - nombre, de tipo String
 - apellido, de tipo String
 - numeroDocumento, de tipo String
 - Construir los setters y los getters de forma automatica
 - Armar un constructor que espere como parámetros los valores correspondientes a nombre, apellido y numeroDocumento
 - Construir la clase Programa y dentro del metodo main realizar lo siguiente:
 - ✓ instanciar un vehiculo utilizando el constructor de tres parametros
 - ✓ informar los valores de los atributos del vehiculo instanciado
 - ✓ instanciar una persona utilizando el constructor de tres parametros
 - ✓ informar los valores de los atributos de la persona instanciada

13.2. Fase #2 – Profesionalización de la organización del proyecto

13.2.1. Requisitos

Para la construcción de la presente fase requiere:

- ✓ conocer que es una clase abstracta
- ✓ manejar el concepto de herencia
- ✓ agrupar clases en paquetes
- ✓ conocer el polimorfismo con redefinición (method override)

13.2.2. Objetivos

Los objetivos de la presente fase son:

- ✓ mejorar el diseño de clases para maximizar su reutilización
- ✓ organizar las clases según funcionalidad

13.2.3. Especificación

- Para la construcción de la presente fase, tomar como base los fuentes de la fase anterior
- Armar los paquetes:
 - ✓ ar.com.educacionit.base.entidades (se utilizara para guardar clases reutilizables en distintos proyectos)
 - ✓ ar.com.educacionit.vehiculos.entidades (se utilizara para guardar clases propias del presente proyecto)
 - ✓ ar.com.educacionit.vehiculos.pruebas (se utilizara para armar clases para testear la aplicacion)
- Ubicar las clases Vehiculo y Persona en el paquete ar.com.educacionit.base.entidades
- Transformar las clases Vehiculo y Persona a clases abstractas
- Construir la clase Auto con las siguientes caracteristicas:
 - ✓ debera estar ubicado en el paquete ar.com.educacionit.vehiculos.entidades
 - ✓ debera heredar de la clase Vehiculo
 - ✓ atributos marca de tipo String, modelo de tipo String, color de tipo String
 - ✓ constructor que soporte como parametros la marca, modelo, color, largo, ancho y altura
 - ✓ redefinir el metodo `toString()` para que retorne una cadena de caracteres con la marca, modelo, color, largo, ancho y altura
- Construir la clase Comprador con las siguientes características:
 - ✓ debera estar ubicado en el paquete ar.com.educacionit.vehiculos.entidades
 - ✓ debera heredar de la clase Persona
 - ✓ atributo presupuesto de tipo double
 - ✓ setters y getters creados de forma automatica
 - ✓ constructor que soporte como parametros el nombre, apellido, numero de documento y presupuesto
 - ✓ redefinir el metodo `toString()` para que retorne una cadena de caracteres con el nombre, apellido, numeroDocumento y presupuesto

Construir la clase Vendedor con las siguientes características:

- ✓ deberá estar ubicado en el paquete ar.com.educacionit.vehiculos.entidades
- ✓ deberá heredar de la clase Persona
- ✓ atributo cantAutosVendidos de tipo int
- ✓ setters y getters creados de forma automática
- ✓ constructor que soporte como parámetros el nombre, apellido, número de documento y la cantidad de autos vendidos
- ✓ redefinir el método `toString()` para que retorne una cadena de caracteres con el nombre, apellido, número de documento y la cantidad de autos vendidos

Abrir la clase Programa creada en la fase anterior y ubicarla dentro del paquete ar.com.educacionit.vehiculos.pruebas. Dentro del método `main` realizar lo siguiente:

- ✓ Borrar (o comentar) el contenido existente dentro del método que pertenece a la fase anterior
- ✓ Instanciar un auto utilizando el constructor completo
- ✓ Informar el auto utilizando el método `toString()`
- ✓ Instanciar un comprador utilizando el constructor completo
- ✓ Informar el comprador utilizando el método `toString()`
- ✓ Instanciar un vendedor utilizando el constructor completo
- ✓ Informar el vendedor utilizando el método `toString()`

13.3. Fase #3 – Construcción de la Interfaz Gráfica de Usuario

13.3.1. Requisitos

Para la construcción de la presente fase requiere:

- ✓ conocer que es Swing
- ✓ conocer las clases JFrame y JInternalFrame
- ✓ conocer como trabajar con un menú de barras
- ✓ conocer como se utiliza Netbeans de forma visual

13.3.2. Objetivos

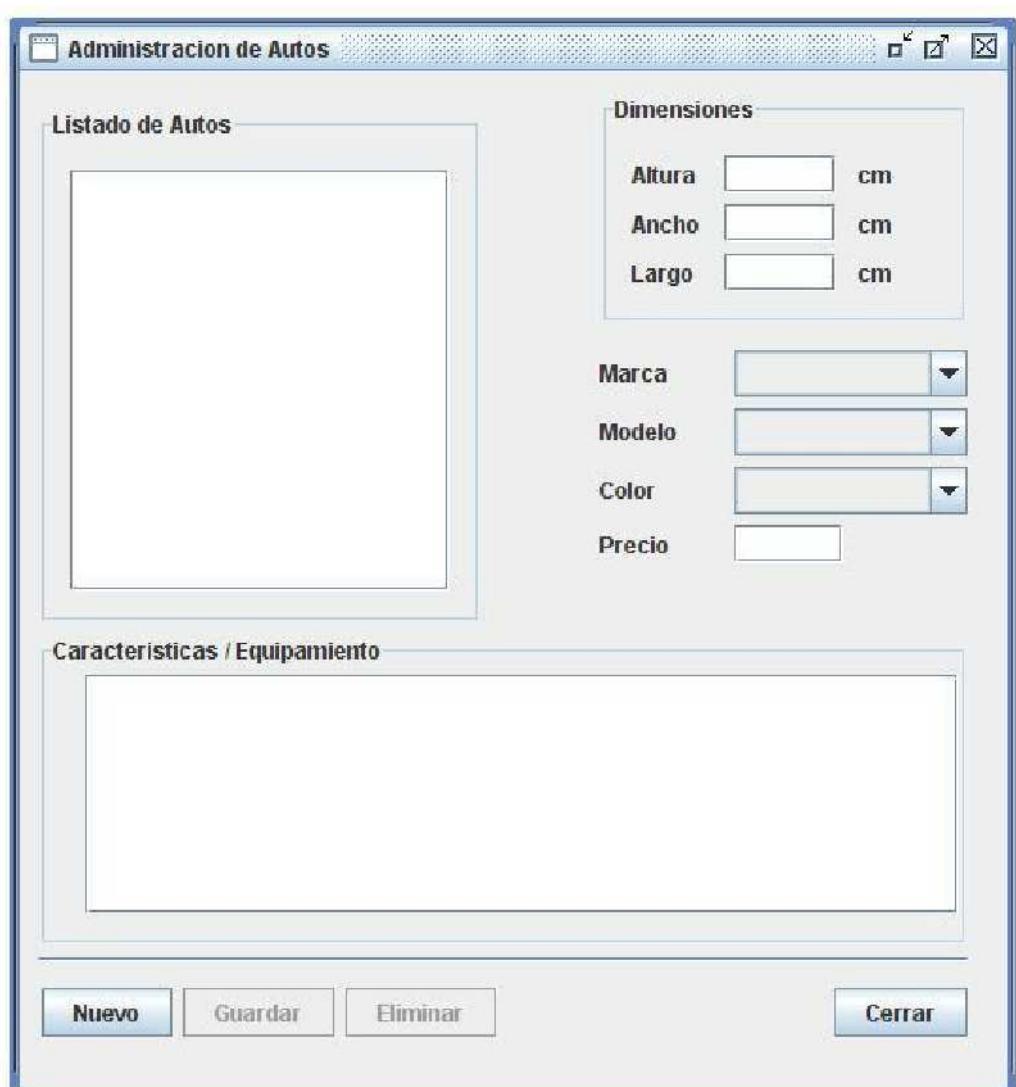
Los objetivos de la presente fase son:

- ✓ construir la ventana principal con su menú de barras
- ✓ construir las ventanas internas de la aplicación

13.3.3. Especificación

- Para la construcción de la presente fase, tomar como base los fuentes de la fase anterior
- Crear una paquete denominado ar.com.educacionit.vehiculos.ventanas para ubicar las clases a crear en esta fase
- Crear la clase VentanaMaestra del tipo JFrame
- La clase VentanaMaestra debe tener un menú de barras con las siguientes opciones de menu:
 - ✓ Abrir Ventana
 - ✓ Ayuda
- Dentro de la opcion de menú Abrir Ventana, debera tener los siguientes ítems:
 - ✓ Autos
 - ✓ Vendedores
 - ✓ Compradores
- Dentro de la opcion de menú Abrir Ventana, debera tener los siguientes ítems:
 - ✓ Ayuda on-line
 - ✓ Configuración
 - ✓ Salir

- Crear la clase AutosVentana del tipo JInternalFrame de la siguiente forma:



- Crear la clase VendedoresVentana del tipo JInternalFrame de la siguiente forma:

The screenshot shows a Java Swing application window titled "Administracion de Vendedores". On the left, there is a panel labeled "Lista de Vendedores" containing a large empty rectangular area. To the right of this panel are four text input fields with labels: "Nombre", "Apellido", "Nro Documento", and "Autos Vendidos". Below these fields is a horizontal line. At the bottom of the window are four buttons: "Nuevo", "Guardar", "Eliminar", and "Cerrar".

- Crear la clase CompradoresVentana del tipo JInternalFrame de la siguiente forma:

The screenshot shows a Java Swing application window titled "Administracion de Compradores". On the left, there is a panel labeled "Lista de Compradores" containing a large empty rectangular area. To the right of this panel are four text input fields with labels: "Nombre", "Apellido", "DNI", and "Presupuesto". Below these fields is a horizontal line. At the bottom of the window are four buttons: "Nuevo", "Guardar", "Eliminar", and "Cerrar".

13.3.4. Especificación – BONUS!

- Hacer las modificaciones que resulten necesarias para lograr que una misma ventana no pueda ser abierta mas de una vez

educación
IT

13.4. Fase #4 – Determinación de la Navegación

13.4.1. Requisitos

Para la construcción de la presente fase requiere:

- ✓ conocer el uso del metodo requestFocus() para establecer el foco sobre componentes
- ✓ conocer el metodo setEnabled(boolean) de los componentes para establecer la habilitacion / deshabilitacion de los mismos
- ✓ conocer la clase JOptionPane y sus metodos necesarios para poder crear cajas de dialogo
- ✓ conocer la clase System y su metodo getProperty()

13.4.2. Objetivos

Los objetivos de la presente fase son:

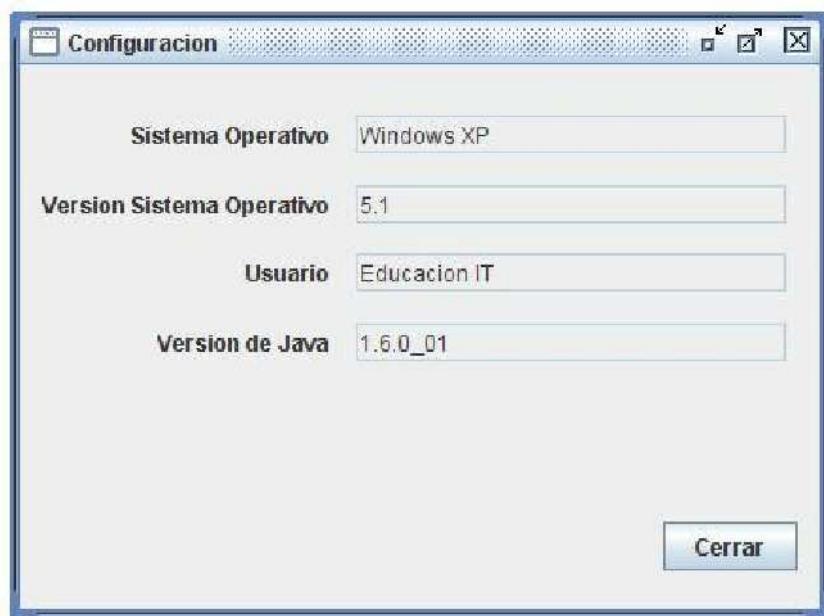
- ✓ construir la navegacion dentro de las ventanas para facilitar su utilizacion
- ✓ validar que las cajas de texto de las diferentes ventanas estan completas al guardar un elemento
- ✓ manejar la ubicacion del foco del cursor para aumentar la amigabilidad e intituividad de la interfaz grafica con el usuario
- ✓ obtener las propiedades del sistema para construir una ventana de configuracion

13.4.3. Especificación

- Para la construcción de la presente fase, tomar como base los fuentes de la fase anterior
- Establecer como deshabilitados los botones "eliminar" y "guardar" de todas las ventanas
- Programar el botón "cerrar" de cada ventana para que cierre la ventana en cuestión
- Al presionar el botón "Nuevo" de la Ventana Compradores se deberá:
 - ✓ limpiar todas las cajas de texto
 - ✓ deshabilitar botón "Nuevo"
 - ✓ poner el foco en la caja de texto "Nombre"
 - ✓ habilitar botón "Guardar"
- Al presionar el botón "Guardar" de la Ventana Compradores se deberá:
 - ✓ chequear que las cajas de texto nombre, apellido, dni, presupuesto no estén vacías
 - ✓ si alguna caja de texto está vacía, por ejemplo caja de texto nombre, deberá aparecer una caja de diálogo con la leyenda: "El nombre no puede estar vacío", y el ícono de Advertencia. Posteriormente ubicar el foco del cursor en esa caja de texto
 - ✓ si las 4 cajas de texto están completas, deberá aparecer una caja de diálogo con la leyenda: "El comprador ha sido guardado", y posteriormente deberán limpiarse todas las cajas de texto, habilitar el botón "Nuevo" y deshabilitar botón "Guardar"

13.4.4. Especificación – BONUS!

- Crear una ventana denominada VentanaConfiguración del tipo JInternalFrame con el siguiente contenido:



- Completar las cajas de texto de la ventana Configuracion cuando se abre la ventana obteniendo la informacion de la clase System a traves de los metodos getProperty() correspondientes
- Deshabilitar todas las cajas de texto dentro de la ventana
- Programar los botones "Nuevo" y "Guardar" de las ventanas de Autos y Vendedores bajo el mismo concepto que los botones de la ventana Compradores. En la ventana Autos no es necesario aun tener en cuenta la utilizacion de los combos correspondientes a marca, color y modelo.

13.5. Fase #5 – Validación y Manejo de Errores

13.5.1. Requisitos

Para la construcción de la presente fase requiere:

- ✓ conocer el uso del método `Integer.parseInt()` y `Double.parseDouble()`
- ✓ Conocer el manejo de excepciones

13.5.2. Objetivos

Los objetivos de la presente fase son:

- ✓ completar la validación de datos
- ✓ validar que las cajas de texto que sean numéricas de tipo entero reciban números enteros
- ✓ validar que las cajas de texto que sean numéricas de tipo punto flotante reciban números enteros

13.5.3. Especificación

- Para la construcción de la presente fase, tomar como base los fuentes de la fase anterior
- En la ventana Autos, validar que altura, ancho y largo sean enteros. En caso de que alguno no sea entero se pide:
 - ✓ levantar una caja de dialogo con la leyenda "La altura debe ser un numero entero"
 - ✓ setear el foco del cursor en la caja de texto correspondiente
- En la ventana Autos, validar que el precio sea un numero con decimales. En caso de que no sea un numero con decimales se pide:
 - ✓ levantar una caja de dialogo con la leyenda "El precio debe ser un numero con decimales"
 - ✓ setear el foco del cursor en la caja de texto precio

13.5.4. Especificación - BONUS!

- En la ventana Compradores, validar que el presupuesto sea un numero con decimales. En caso de que no sea un numero con decimales se pide:
 - ✓ levantar una caja de dialogo con la leyenda "El presupuesto debe ser un numero con decimales"
 - ✓ setear el foco del cursor en la caja de texto presupuesto
- En la ventana Vendedores, validar que la cantidad de autos sea un numero entero. En caso de que no sea un numero entero se pide:
 - ✓ levantar una caja de dialogo con la leyenda "La cantidad de autos debe ser un numero entero"
 - ✓ setear el foco del cursor en la caja de texto cantidad de autos

13.6. Fase #6 – Lectura de Recursos Adicionales

13.6.1. Requisitos

Para la construcción de la presente fase requiere:

- ✓ conocer el funcionamiento de los streams
- ✓ conocer la clase BufferedReader y su metodo readLine()
- ✓ conocer la utilización de combos con sus métodos addItem(), getSelectedItem(), getSelectedIndex()

13.6.2. Objetivos

Los objetivos de la presente fase son:

- ✓ completar los combos de marcas, colores y modelos
- ✓ ampliar la validación para que los combos de marcas, colores y modelos tengan un item seleccionado al guardar un auto

13.6.3. Especificaciones

- Para la construcción de la presente fase, tomar como base los fuentes de la fase anterior
- Crear el paquete ar.com.educacionit.vehiculos.recursos y ubicar dentro de este paquete:
 - ✓ un archivo marcas.txt que contenga diferentes marcas, una abajo de la otra
 - ✓ un archivo colores.txt que contenga diferentes colores, un abajo del otro
- Al inicializar la ventana de autos se deberá:
 - ✓ llenar el combo de marcas a partir del archivo marcas.txt
 - ✓ llenar el combo de colores a partir del archivo colores.txt
 - ✓ llenar el combo modelos con los años de 1985 al actual
- Para estos casos, se recomienda armar un método individual para cada caso - por ejemplo cargasMarcas() - que deberá invocarse en el constructor de la ventanaAutos
- Agregar en todos los combos como primera opción "Seleccione..."
- Al presionar el botón "Nuevo" (además de poner todas las cajas de texto en blanco, resuelto en fases previas) deberá seleccionar la opción "Seleccione..." en todos los combos
- Al presionar el botón "Guardar" (además de poner todas las cajas de texto en blanco, resuelto en fases previas) deberá seleccionar la opción "Seleccione..." en todos los combos
- Modificar la validación para que no permita guardar un auto si no tiene un color, marca y modelo seleccionado en los combos

13.7. Fase #7 – Utilización de Listas

13.7.1. Requisitos

Para la construcción de la presente fase requiere:

- ✓ conocer el uso del componente JList

13.7.2. Objetivos

Los objetivos de la presente fase son:

- ✓ agregar items en las listas al apretar el botón guardar
- ✓ quitar items de las listas al apretar el botón eliminar

13.7.3. Especificaciones

- Para la construcción de la presente fase, tomar como base los fuentes de la fase anterior
- Al presionar el botón "Guardar" se deberá:
 - ✓ agregar el auto en el listado de autos si este no se encuentra en la lista. Si se encuentra, entonces modificar sus datos con los nuevos
 - ✓ deshabilitar botón "Eliminar"
 - ✓ mostrar una caja de dialogo con la leyenda "El auto ha sido guardado"
- Al seleccionar un auto de la lista, se deberá:
 - ✓ habilitar el botón "Eliminar" y "Guardar"
 - ✓ completar las cajas de texto y los combos con los datos del auto seleccionado
- Al presionar el botón "Eliminar" se deberá:
 - ✓ realizar la pregunta "Desea eliminar el auto [datos del auto]?".
 - ✓ En caso afirmativo, quitarlo de la lista, deshabilitar el botón "Eliminar" y mostrar una caja de dialogo con la leyenda "El auto ha sido eliminado"

13.7.4. Especificación - BONUS!

- Realizar el mismo procedimiento para las ventanas de compradores y vendedores

13.8. Fase #8 – Conexión con Base de Datos

13.8.1. Requisitos

Para la construcción de la presente fase requiere:

- ✓ conocer SQL como lenguaje de consultar
- ✓ dominar los conceptos de INSERT, UPDATE, DELETE y SELECT
- ✓ conocer JDBC para conectarse a una base de datos
- ✓ saber manejar el administrador de conexiones
- ✓ entender el concepto de DAO (Data Access Object)

13.8.2. Objetivos

Los objetivos de la presente fase son:

- ✓ configurar el proyecto para soportar base de datos
- ✓ construir clases de acceso a datos que interactuaran con las tablas
- ✓ probar las clases construidas de manera independiente con la consola, sin la utilizacion de la interfaz grafica

13.8.3. Especificaciones

- Para la construcción de la presente fase, tomar como base los fuentes de la fase anterior
- Armar el esquema de base de datos j2se utilizando MySQL Query Browser
- Dentro del esquema j2se, construir la tabla "autos" con la siguiente especificacion:

Column Name	Datatype	NULL	AUTO	Flags	Default Value	Comment
au_id	INTEGER	✓	✓	<input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	HULL	
au_marca	VARCHAR(255)	✓		<input type="checkbox"/> BINARY		
au_modelo	VARCHAR(255)	✓		<input type="checkbox"/> BINARY		
au_precio	FLOAT	✓		<input type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	0	
au_color	VARCHAR(255)	✓		<input type="checkbox"/> BINARY		
au_largo	INTEGER	✓		<input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	0	
au_ancho	INTEGER	✓		<input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	0	
au_altura	INTEGER	✓		<input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	0	
au_equipamiento	VARCHAR(255)	✓		<input type="checkbox"/> BINARY		

- Configurar el proyecto para soportar acceso a datos. Para esto se requiere:
 - ✓ crear una carpeta dentro del proyecto llamada "jars", y ubicar el archivo .jar que se utiliza como conector a la base de datos
 - ✓ agregar como libreria el archivo .jar desde las propiedades del proyecto
 - ✓ copiar la clase AdministradorDeConexiones entregada en el tutorial de JDBC dentro del paquete ar.com.educacionIT.vehiculos.util. TIP: *se puede realizar utilizando drag & drop dentro de Netbeans entre proyectos*
 - ✓ configurar el usuario y contraseña dentro del AdministradorDeConexiones
- Agregar a la clase Auto un atributo privado "id" del tipo entero, junto con sus setters y getters

- Construir los siguientes metodos dentro de la clase Auto, que se utilizaran para realizar las operaciones con la tabla autos:

- ✓ insertar(), utilizado para realizar la insercion de un auto
- ✓ modificar(), utilizado para realizar la modificacion de un auto
- ✓ eliminar(), utilizado para realizar la eliminacion de un auto
- ✓ obtenerTodos(), utilizado para obtener todos los autos de la tabla

- Las firmas de los metodos presentados anteriormente deberan ser:

- ✓ public void insertar(Connection conn) throws Exception
- ✓ public void actualizar(Connection conn) throws Exception
- ✓ public void eliminar(Connection conn) throws Exception
- ✓ public static ArrayList obtenerTodos(Connection conn) throws Exception

Nota: *los metodos insertar, modificar y eliminar no poseen parametros adicionales debido a que deberan utilizar los atributos del objeto auto para realizar las respectivas operaciones.*

- Dentro del paquete ar.com.educacionit.vehiculos.pruebas construir la clase TestAutos y codificar dentro del metodo main() lo que resulte necesario para probar que los metodos de Auto funcionan correctamente.

13.8.4. Especificación - BONUS!

- Realizar el mismo procedimiento para las clases Comprador y Vendedor

13.9. Fase #9 – Integracion con Interfaz Grafica de Usuario

13.9.1. Requisitos

Para la construcción de la presente fase requiere:

- ✓ haber realizado todas las fases anteriores, ya que se realiza una integración

13.9.2. Objetivos

Los objetivos de la presente fase son:

- ✓ integrar las clases de acceso a datos con la interfaz gráfica de usuario

educación

13.9.3. Especificaciones

- Para la construccion de la presente fase, tomar como base los fuentes de la fase anterior
- Redefinir el metodo `toString()` de la clase Auto para que retorne marca, color y modelo
- Al cargar la ventana Autos, se deberan cargar en la vista de autos todos los autos que hay en la tabla de autos. TIP: *armar un metodo llenarListaDeAutos() que posee toda la logica necesaria para llenar la lista. Dicho metodo deberá ser invocado desde el constructor de VentanaAutos*
- Al hacer click sobre un auto de la lista (es decir, al seleccionar un auto) se deberan llenar todas las cajas de texto y los combos con los valores pertenecientes a ese auto, y tambien se debera habilitar el boton "Guardar"
- Al presionar el boton "Nuevo" en la lista no debe quedar ningun item seleccionado. Tambien se debera deshabilitar el boton "Guardar" (ya que podria estar habilitado)
- Al presionar el boton "Guardar" hay que tener en cuenta dos posibles casos:
 - ✓ si el auto es nuevo, hay que insertarlo en la tabla de autos y luego debe quedar agregado a la lista
 - ✓ si el auto ya esta en la lista de autos, hay que realizar la modificacion correspondiente en la tabla de autos, y luego reflejar los cambios
 - ✓ en ambos casos, luego de realizar la insercion o actualizacion, debera mostrarse una caja de dialogo avisando "El auto ha sido actualizado"
- Al presionar el boton "Eliminar" hay que eliminar se debera mostrar una caja de dialogo pidiendo la confirmacion si deseamos eliminarlo
- En caso de querer eliminarlo, realizar la eliminacion de la tabla y de la lista, y luego mostrar una caja de dialogo con la leyenda: "El auto ha sido eliminado"

13.9.4. Especificación - BONUS!

- Realizar el mismo procedimiento para las clases Comprador y Vendedor

educationIT