



CORPORACIÓN
UNIVERSITARIA
REMINGTON

ESCUELA DE CIENCIAS BÁSICAS E INGENIERÍA
Ingeniería de Sistemas
ASIGNATURA: Lenguaje de Programación I

CORPORACIÓN UNIVERSITARIA REMINGTON
DIRECCIÓN PEDAGÓGICA

Este material es propiedad de la Corporación Universitaria Remington (CUR), para los estudiantes de la CUR
en todo el país.

2011

CRÉDITOS



El módulo de estudio de la asignatura Lenguaje de Programación I del Programa Ingeniería de Sistemas es propiedad de la Corporación Universitaria Remington. Las imágenes fueron tomadas de diferentes fuentes que se relacionan en los derechos de autor y las citas en la bibliografía. El contenido del módulo está protegido por las leyes de derechos de autor que rigen al país.

Este material tiene fines educativos y no puede usarse con propósitos económicos o comerciales.

AUTOR

César Augusto Jaramillo Henao
Tecnólogo en Sistemas
cesar.jaramillo@remington.edu.co

Nota: el autor certificó (de manera verbal o escrita) No haber incurrido en fraude científico, plagio o vicios de autoría; en caso contrario eximió de toda responsabilidad a la Corporación Universitaria Remington, y se declaró como el único responsable.

RESPONSABLES

Escuela de Ciencias Básicas e Ingeniería
Director Dr. Mauricio Sepúlveda

Director Pedagógico
Octavio Toro Chica
dirpedagogica.director@remington.edu.co

Coordinadora de Medios y Mediaciones
Angélica Ricaurte Avendaño
mediaciones.coordinador01@remington.edu.co

GRUPO DE APOYO

Personal de la Unidad de Medios y Mediaciones
EDICIÓN Y MONTAJE
Primera versión. Febrero de 2011.

Derechos Reservados



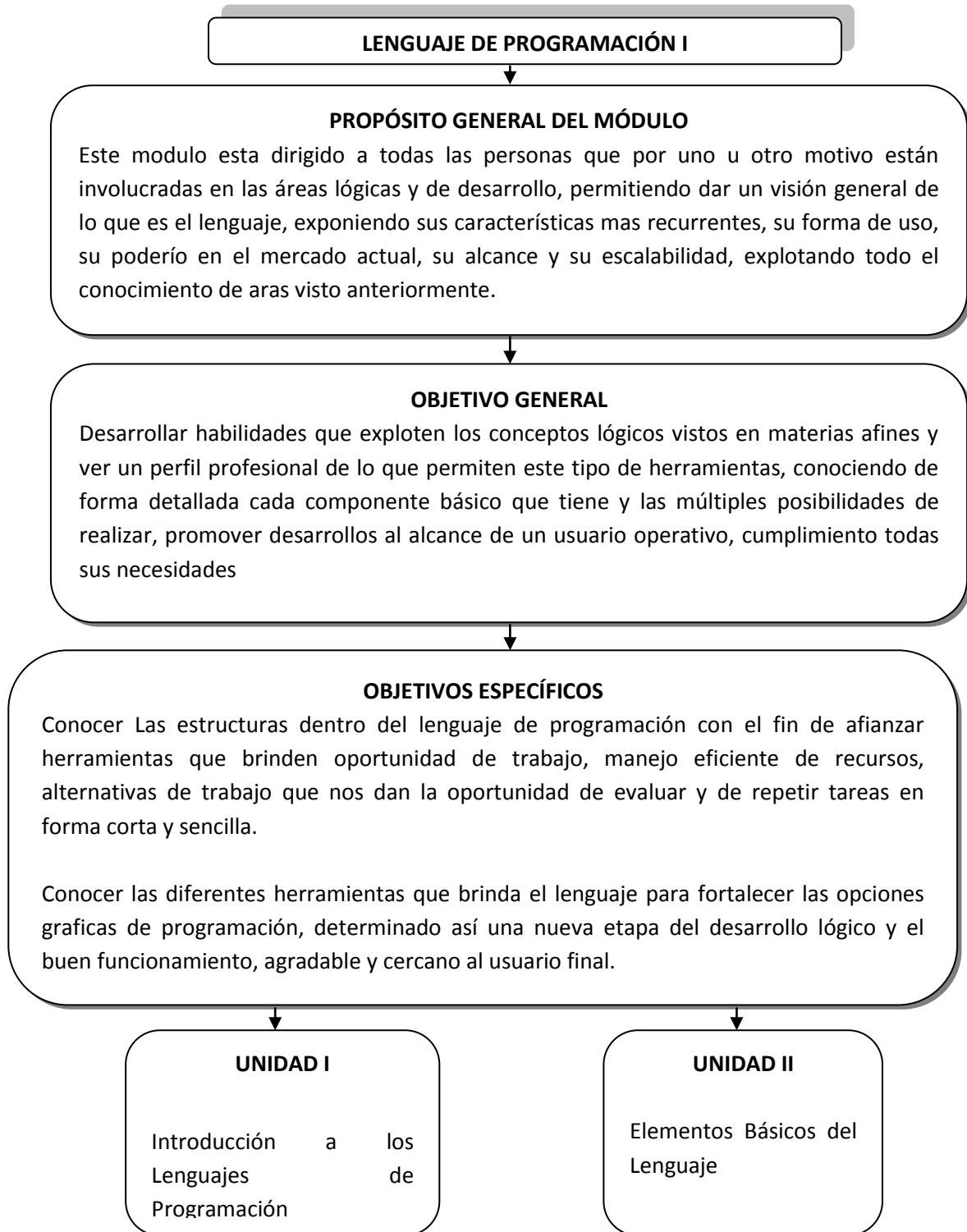
Esta obra es publicada bajo la licencia Creative Commons. Reconocimiento-No Comercial-Compartir Igual 2.5 Colombia.

TABLA DE CONTENIDO

1.	MAPA DE LA ASIGNATURA.....	8
2.	INTRODUCCIÓN A LOS LENGUAJES DE PROGRAMACIÓN	9
2.1.	Introducción a los Lenguajes	9
2.1.1.	Que es un Programa.....	9
2.2.	Conceptos de la Programación Orientada a Objetos	11
2.2.2.	Los métodos	19
2.3.	Desarrollo de un Programa en Java	20
2.3.1.	Que es java	20
2.3.2.	Adquirir e instalar Java	20
2.3.3.	Las variables Path y ClassPath	21
2.4.	Desarrollo de un programa en java.....	21
2.4.1.	Tipos de programas en java	21
2.4.2.	Estructura de un programa	22
2.4.3.	Edición del programa fuente	23
2.4.4.	Compilación del programa	24
3.	ELEMENTOS BÁSICOS DEL LENGUAJE.....	30
3.1.	Elementos Básicos.....	30
3.1.1.	Los tipos de datos primitivos.....	30
3.1.2.	Las palabras claves	33
3.1.3.	Las variables	34
3.1.4.	Los operadores	37
3.2.	Estructura General de una Clase	38
3.2.1.	Paquetes e importación de clases.....	38
3.2.2.	Uso de this y super	39
3.3.	Clases de uso común	39
3.3.1.	Datos numéricos – clase math	39
3.3.2.	Cadenas de caracteres – clase string	40

3.3.3. Introducción a las excepciones	40
4. ESTRUCTURAS DE CONTROL.....	49
4.1. Estructuras Básicas.....	49
4.1.1. Ciclos	59
4.2. Arreglos	67
5. APPLETS AWT Y SWING.....	87
5.1. Applets	87
5.1.1. Definición	87
5.1.2. Crear un applet.....	88
5.1.3. Ciclo de vida de un applet.....	90
5.2. Interfaces Gráficas AWT (SWT) Swing.....	91
5.2.1. El awt.....	91
6. FUENTES.....	108
6.1. Libros	108
6.1.1. Presencial	109
6.1.2. Distancia.....	109
6.1.3. Evaluación	110

1. MAPA DE LA ASIGNATURA



2. INTRODUCCIÓN A LOS LENGUAJES DE PROGRAMACIÓN

OBJETIVO GENERAL

Conocer los pormenores de los aspectos lógicos aplicados mediante lenguaje de programación, identificando sus características mas relevantes, al igual que la importancia que este presta a la industria tecnología actual, esta breve introducción nos dara una nueva perspectiva de los conceptos y los tipos de lenguajes que existen en el mercado y las caractereristicas mas importantes que estos puede ofrecer.

OBJETIVOS ESPECÍFICOS

- ◆ Identificar los conceptos de la programación aplicada según la lógica vista en semestres previos
- ◆ Conocer las características de los lenguajes de programación ,su alcance y limitaciones
- ◆ Dar a conocer los conceptos básicos de la programación orientada a objetos y la importancia que esta tiene en la actualidad

Prueba inicial

1. Aplique en un diagrama lógico los conceptos mas importes que este puede manejar, las características y la identifiacas de procesos para que este funcione adecuadamente, recuerde aplicar la prueba de escritorio para la verifacas de los datos, procesos y resultados.

2.1. Introducción a los Lenguajes

2.1.1. Que es un Programa

Un programa es una serie de instrucciones lógicas interpretadas por el PC para realizar un conjunto de operaciones o tareas y arrojar resultados según sea la necesidad del usuario, además de complementar lo visto en semestres anteriores en temas relacionados con la lógica, permitirá complementar y ampliar conceptos lógicos aplicados que permiten tener una nueva visión en la solución de problemas.

2.1.1.1 Lenguajes de Programación, Tipos según se evolución

Existen diferentes tipos de lenguaje y muchos de ellos dependiendo de la época evolutiva de estos, se encuentran lenguajes de bajo nivel, de alto nivel, interpretados, además de existir métodos que permiten un alcance mayor o menor según su uso y su necesidad, dentro de estos se pueden mencionar ambientes:

Orientados Procedimientos:

Con la programación procedimental realizamos tareas lógicas en un mismo proceso, archivo o ambiente lógico, la invocación de un procedimiento se utiliza para llamar a los subprocesos, después estas secuencia son procesada, el flujo de control continua igual después de la última posición donde la llamada fue realizada.

Orientados a Eventos:

En la programación orientada a eventos es el programador quien especifica la secuencia del programa, Aunque en la programación secuencial puede haber intervención de procesos externos al él, estas intervenciones ocurrirán cuando el desarrollador lo haya especificado, y no en cualquier momento como puede ser en el caso de la programación orientada por eventos.

Orientados a Objetos:

Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, reutilización, hilos y encapsulamiento. Su uso se hizo frecuente en los años 90's. En la actualidad existen muchos lenguajes de programación que integran la orientación a objetos por su poder y alcance que tiene y sobre todo por su flexibilidad y reutilización.

Orientados a Aspectos:

Es un tipo de programación muy reciente, tiene como condición permitir una adecuada modularización de las aplicaciones y posibilita una mejor separación de procesos. Gracias a la Programación Orientada a Aspectos (POA), se pueden manejar los diferentes conceptos que componen una aplicación bien definida, eliminando las dependencias entre cada uno de los diferentes módulos existentes. De esta forma se consiguen mejores conceptos de programación.

Orientados a Servicios:

Es un concepto de la arquitectura de software que utiliza los servicios para dar soporte a los requisitos del usuario. Permitiendo la creación de sistemas escalables que reflejan el negocio de la empresa, lo cual facilita la interacción entre diferentes sistemas propios y/o de terceros.

2.1.1.2 Los ByteCodes

El lenguaje de programación I (java), tiene por particularidad que es un ambiente multiplataforma, esto nos indica que puede ser creado e interpretado por diferentes sistemas operativas, los bytecodes, es la mejor forma de indicar que este aplicativo sea usado en varias sistemas Operativos sin cambio en su codificación, dando así un mayor alcance en su uso, aunque con limitantes en su ejecución por el tiempo de respuesta que este puede requerir.

2.2. Conceptos de la Programación Orientada a Objetos

La Programación Orientada a Objetos es un paradigma de programación diferente a las demás alternativas de programación, se basa en casos del común, que nos permite como principal aporte la reutilización de los procesos, haciendo que programar sea mas simple, menos extenso y de mayor alcance.

La el principal componente de este tipo de programación son los objetos y estos objetos tienen ciertas características que los hacen únicos y esas características puede ser atributos, descripción y uso.

Java es un lenguaje que nos ayuda a entender mucho mejor el paradigma Orientado a Objetos de una manera mas sencilla y natural, dándole a la tecnología un gran aporte en alcance y potencia.

Clases y Objetos

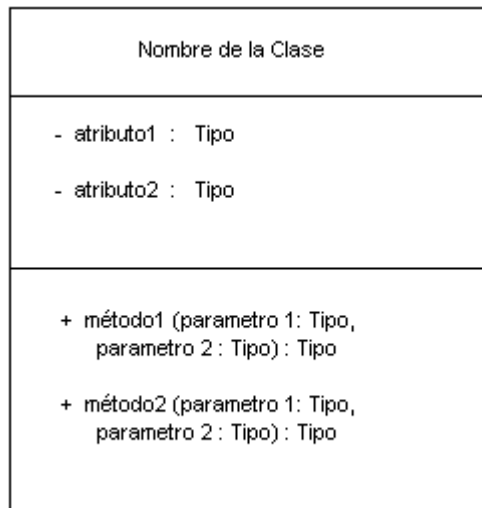
Una Clase es una plantilla o patrón que se emplea para instanciar un objeto a partir de ella y que define los datos que contendrán el estado del objeto y los métodos que implementaran el funcionamiento del mismo. Una vez definida una clase podemos usarla, como si fuera un tipo de datos definidos por el usuario.

1. **Identificar los atributos de cada clase.** Describe aquellos datos propios de cada una de las clases que se pretendan trabajar sin dejar procesos sueltos o al azar.

2. **Identificar los métodos de cada clase.** identificar las acciones que deben ser realizadas por los objetos de la clase, sean estas acciones que no tengan relación con otras clases o procesos donde se involucren otras clases.

Definición de una Clase

Existen múltiples formas de definir una clase, dentro de los ambientes modernos de programación se encuentra UML (Lenguaje de Modelado Unificado), que permite entre otras funciones expresar todos los procesos, requerimiento, métodos, objetos que se puedan tener, observe un ejemplo de esta representación.



Este proceso nos ayuda a entender visualmente la manera en la que una clase esta compuesta.

Dentro del ambiente de programación Java, existen múltiples clases del sistema, además de las que el usuario puede crear según la necesidad, obsérvese un grupo de las clases comunes que provee el lenguaje.

Tipo primitivo	Clase de Envoltura
Byte	java.lang.Byte
Short	java.lang.Short
Integer	java.lang.Integer
Char	java.lang.Char
Flota	java.lang.Float
Double	java.lang.Double
Boolean	java.lang.Boolean

2.2.1.1 Que es herencia

Al crear una clase, se definen las características, métodos y comportamiento que pueden tener todos los objetos que esta tenga.

Hay que tener en cuenta que las clases requieren de una definición muy especializada, para poder definir atributos, variables y métodos que son específicos de los objetos que esas clases más especializadas.

Observemos un ejemplo del uso de la herencia en Java, si tenemos la clase Persona, esta clase puede tener solamente el nombre y el sexo, con sus respectivos constructores (procesos que determinan el inicio de un proceso) y métodos (procesos que se realizaran), Observe un ejemplo que representa esto:

```
public class EjemploPersona {
    private String Nombre;
    private char Sexo;
    public EjemploPersona() {
        Nombre = new String();
        Sexo = ' ';
    }
    public String ObtenNombre() {
        return Nombre;
    }
    public void CambiaNombre(String Nombre) {
        this.Nombre = Nombre;
    }

    public char ObtenSexo() {
        return Sexo;
    }
    public void CambiaSexo(char Sexo) {
        this.Sexo = Sexo;
    }

    public String toString() {
        return "" + Nombre + " " + Sexo;
    }
}
```

Creemos una nueva Clase que represente a un Alumno, en la cual definimos el nombre y sexo de este. Cuando realizamos esto no debemos empezar desde cero, indicamos que la clase Alumno sería una clase que hereda de la clase anteriormente creada. Solamente que el Alumno puede adicionar nuevas características que serán solo de su interes, añadiremos los métodos específicos de la clase Alumno, tendríamos lo siguiente:

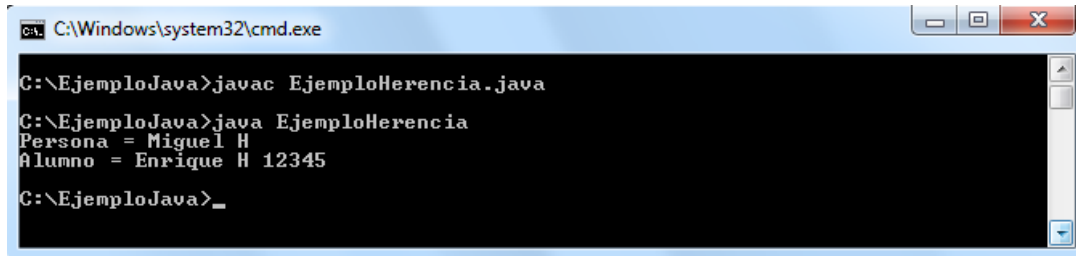
```
public class Alumno extends Persona {  
    private int matricula;  
  
    public Alumno() {  
        super();  
        matricula = 0;  
    }  
  
    public int obtenMatricula() {  
        return matricula;  
    }  
  
    public void cambiaMatricula(int matricula) {  
        this.matricula = matricula;  
    }  
  
    public String toString() {  
        return "" + super.toString() + " " + matricula;  
    }  
}
```

Podemos ver que la clase Hija (clase que hereda), que tiene un constructor se utiliza el método `super()`, pero en ninguna parte está definido este método, la palabra `super()` es utilizada para llamar al constructor vacío de la clase Padre (clase de la cual se está heredando).

La instrucción *extends* nos permite Definir herencia, entonces cuando decimos `public class EjemploAlumno extends EjemploPersona`, estamos indicando que la clase `EjemploAlumno` hereda de la clase `EjemploPersona`. Un ejemplo que nos represente esto es la siguiente:

```
import java.io.*;  
  
public class EjemploHerencia {  
    public static void main(String[] args) {  
        EjemploPersona Miguel = new EjemploPersona();  
        Miguel.CambiaNombre("Miguel");  
        Miguel.CambiaSexo('H');  
        System.out.println("Persona = " + Miguel.toString());  
    }  
}
```

```
EjemploAlumno Enrique = new EjemploAlumno();
Enrique.CambiaNombre("Enrique");
Enrique.CambiaSexo('H');
Enrique.CambiaMatricula(12345);
System.out.println("Alumno = " + Enrique.toString());
}
}
```



```
C:\Windows\system32\cmd.exe

C:\EjemploJava>javac EjemploHerencia.java
C:\EjemploJava>java EjemploHerencia
Persona = Miguel H
Alumno = Enrique H 12345
C:\EjemploJava>_
```

Así podemos ver que la clase `EjemploAlumno` está heredando de la clase `EjemploPersona` y todos los criterios y métodos de la `Persona`, están siendo utilizados por el `Alumno`, ya que automáticamente son de él, al heredarlos, tengase muy presente que la herencia solo se da de padres a hijos y no al contrario.

Polimorfismo

Polimorfismo, en general, es la característica de un lenguaje orientado a objetos que permite que un mismo identificador de método tenga significados diferentes en diferentes contextos. En Java, igual que en C++, es posible definir con un mismo nombre de método varias implementaciones distintas, que pueden corresponder a clases diferentes o a la misma clase.

Observese un ejemplo de este tema.

Clase Animal

```
public class Animal {
    private int Peso = 0;

    public void CambiaPeso(int Peso) {
        this.Peso = Peso;
    }

    public int ObtenPeso() {
        return Peso;
    }
}
```

```
public String Habla() {  
    return "Los animales no hablan";  
}  
}
```

Clase Vaca

```
public class Vaca extends Animal {  
  
    public String Habla() {  
  
        return "MUU";  
  
    }  
}
```

Clase Cerdo

```
public class Cerdo extends Animal {  
  
    public String Habla() {  
  
        return "OINC";  
  
    }  
}
```

Clase Serpiente

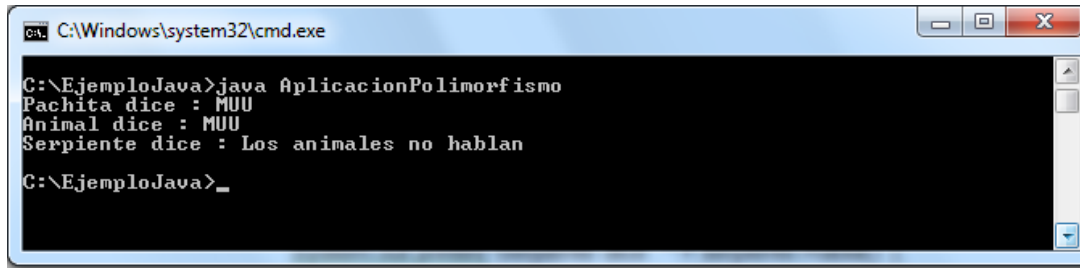
```
public class Serpiente extends Animal {  
    }  
}
```

Aplicación Herencia con Polimorfismo

```
import java.io.*;
```

```
public class AplicacionPolimorfismo {  
  
    public static void main(String[] args) {  
  
        Vaca Pachita = new Vaca();  
        Animal animal = new Vaca();  
  
        Serpiente serpiente = new Serpiente();  
        System.out.println("Pachita dice : " + Pachita.Habla() );  
        System.out.println("Animal dice : " + animal.Habla() );  
        System.out.println("Serpiente dice : " + serpiente.Habla() );  
    }  
}
```

```
}  
}
```



The screenshot shows a Windows command prompt window titled "cmd.exe" with the path "C:\Windows\system32\cmd.exe". The prompt is at "C:\EjemploJava>". The user has entered the command "java AplicacionPolimorfismo". The output of the program is displayed as follows:
Pachita dice : MUU
Animal dice : MUU
Serpiente dice : Los animales no hablan
The prompt is now "C:\EjemploJava>_".

Clases Abstractas

Mediante los mecanismos de herencia se organizan las clases en una jerarquía, lo normal es que las clases que representan los conceptos mas abstractos ocupen un lugar mas alto en la misma. Por ejemplo, se podría concebir una clase Figura que tuviera las subclases Cuadrado y Rectángulo. Una acción común a todas las figuras geométricas puede ser dibujarlas en la pantalla, pero los detalles concretos de cómo se va a dibujar cada una dependerá del tipo de figura de que se trate.

Clase Figura

```
public abstract class Figura{  
    protected int X, Y, Ancho, Alto;  
  
    public void CambiaX(int X) {  
        this.X = X;  
    }  
  
    public void CambiaY(int Y) {  
        this.Y = Y;  
    }  
  
    public void CambiaAncho(int Ancho) {  
        this.Ancho = Ancho;  
    }  
  
    public void CambiaAlto(int Alto) {  
        this.Alto = Alto;  
    }  
  
    public abstract float ObtenPerimetro();  
  
    public abstract float ObtenArea();  
}
```

Clase Cuadrado

```
public class Cuadrado extends Figura {
```

```
    public float ObtenPerimetro() {  
        return 4 * Ancho;  
    }
```

```
    public float ObtenArea() {  
        return Ancho * Ancho;  
    }
```

```
}
```

Clase Rectangulo

```
public class Rectangulo extends Figura {
```

```
    public float ObtenPerimetro() {  
        return 2 * Ancho + 2 * Alto;  
    }
```

```
    public float ObtenArea() {  
        return Ancho * Alto;  
    }
```

```
}
```

Aplicacion Herencia 2

```
public class AplicacionAbstracta {
```

```
    public static void main(String[] args) {  
        Cuadrado C = new Cuadrado();
```

```
        C.CambiaAncho(10);
```

```
        C.CambiaAlto(10);
```

```
        Rectangulo R = new Rectangulo();
```

```
        R.CambiaAncho(20);
```

```
        R.CambiaAlto(30);
```

```
        System.out.println("Perimetro Cuadrado = " + C.ObtenPerimetro());
```

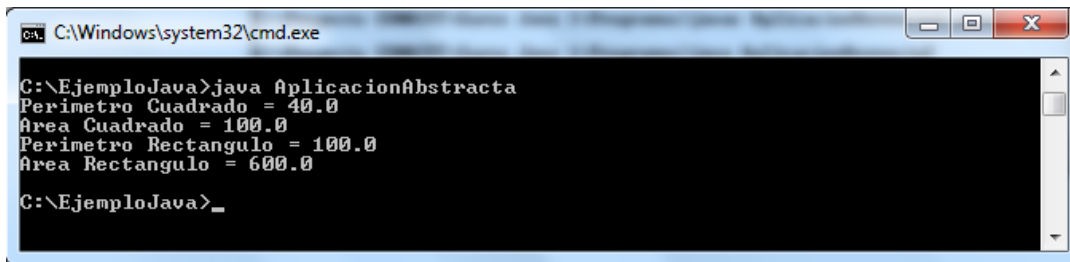
```
        System.out.println("Area Cuadrado = " + C.ObtenArea());
```

```
        System.out.println("Perimetro Rectangulo = " + R.ObtenPerimetro());
```

```
        System.out.println("Area Rectangulo = " + R.ObtenArea());
```

```
    }
```

```
}
```



```
C:\Windows\system32\cmd.exe
C:\EjemploJava>java AplicacionAbstracta
Perimetro Cuadrado = 40.0
Area Cuadrado = 100.0
Perimetro Rectangulo = 100.0
Area Rectangulo = 600.0
C:\EjemploJava>_
```

De esta manera observamos como al definir ambos métodos en la clase Particular esta quita la abstracción.

2.2.2. Los métodos

Los métodos de una clase son los que contienen el código que implementa la conducta de los objetos instanciados a partir de ella, en Java todo el código de un programa está contenido en los métodos de la clase que lo componen.

```
public int ObtenValor() {
    return Valor;
}
public double ObtenSaldo() {
    return Saldo;
}
public void CambiaSaldo(Saldo) {
    this.Saldo = Saldo;
}
```

Métodos Constructores

Los constructores son métodos especiales cuya misión es inicializar cada objeto que se instancia a partir de una clase. Los constructores se definen igual que otros métodos de la clase, pero con 2 características distintivas. Como primera característica, es preceptivo que los constructores tengan un mismo nombre que la clase a la que pertenece, como segunda instancia los constructores son los únicos métodos que no tienen definido un tipo de retorno, de hecho, el intento de asignar un tipo de retorno a un constructor produciría un error de compilación.

2.3. Desarrollo de un Programa en Java

2.3.1. Que es java

Java es hoy en día una de las plataformas mas versátiles, completas y poderosas herramientas de programación profesional del mercado, con java se podrán crear aplicativos desde lo personal, las redes de datos y los dispositivos móviles, pasando por aplicativos web, tiene características de ser multiplataforma y de propósito general, con lo cual no debería ser limitante para cualquier tipo de aplicación que deseemos crear. Esto nos da pie a decir que permite explotar todo su potencial de manera muy organizada y de una manera relativamente fácil.

2.3.1.1 Un poco de historia

Java se crea en los años 90's, con el fin de amplia variedad de dispositivos de red y sistemas embebidos. El propósito de este era una plataforma sencilla, segura, portable, distribuida y de tiempo real.

Cuando este se inicio, C++ era el lenguaje mas popular del momento. Pero con el tiempo se descubrió que existían dificultades encontradas con C++, esto llevo a que se propusieran herramientas y opciones que mejoraran y corregirán las fallas vistas en C++.

Dentro de los modelos que se tuvieron en cuenta para la construcción fueron Eiffel, SmallTalk, Objective C y Cedar/Mesa. Como resultado se tiene un lenguaje que se ha mostrado ideal para desarrollar aplicaciones de usuario final seguras, distribuidas y basadas en red con un amplio rango de entornos desde los dispositivos de red embebidos hasta su uso para soluciones en Internet.

2.3.2. Adquirir e instalar Java

El lenguaje de programación Java o el JDK, para usuarios tradicionales de desarrollo de software tendrá la facilidad de ser descargado en forma gratuita y de realizar una instalación mediante asistente de una forma muy rápida y fácil. Para descargar la plataforma consta solo de visitar el sitio oficial www.sun.com, y en la opción de descargas seleccionar la herramienta adecuada, teniendo en cuenta que esta primera etapa de desarrollo se centrara en el ambiente J2SE, que nos

indica que es la segunda versión general de Java es su distribuciones Stantard Edition, esta plataforma está diseñada para equipos personales o redes de computación.

Para la descarga adecuada encontrara archivos con la siguiente descripción, **jdk-6u17-nb-6_8-windows-m1**, el JDK, visto anteriormente es la descripción del ambiente de desarrollo de Java, versión 6 actualización 17, con ambiente de desarrollo netbeans versión 6.8 para Windows multilenguaje. Es importante conocer el significado de este archivo dado que con el determinaremos que tan actualizado esta nuestra descarga

2.3.3. Las variables Path y ClassPath

Los diferentes sistemas operativas tiene formas diversas de interpretar los lenguajes de programación, java no es la excepción, y tiene unas variables que al configurar adecuadamente se podrá acceder desde cualquier ubicación del PC y darle uso al aplicativo desarrollado.

Para esta tarea se debe ubicar en la configuración del sistema, una forma sencilla de llegar a esta es con doble pulsación en mi PC o mi equipo según la versión de Windows utilizada, botón derecho en cualquier área y elija la opción de propiedades, seleccione configuración avanzada del sistema y dentro de este selección variables de entorno, en esta, seleccione variables de usuario y agregue o modifique la opción PATH y en valor agregue la ruta de su entorno de desarrollo, ejemplo, C:\Program Files\Java\jdk1.6.0_17\bin, y agregue o modifique la opción CLASSPATH, en esta solo debe especificar un (.), para la ruta de trabajo que requiere, esta misma tarea debe repetirse en las variables del sistema.

2.4. Desarrollo de un programa en java

2.4.1. Tipos de programas en java

Java tiene una gran variedad de aplicativos que puede desarrollar, teniendo claro que este lenguaje es de propósito general, existe una clasificación que nos orientara en los tipos según la necesidad o el dispositivo que se requiera.

Aplicación de sobre mesa:

Son los aplicativos que se desarrollan para un pc tradicional, este puede ser de uso único o de uso compartido mediante una red de computadores debidamente configurada para tal fin, de los tipos de aplicativos puede ser el mas común de los procesos.

Aplicación web:

Son aplicativos que esta en un servidor web, con características similares las de sobremesa, pero con mayor versatilidad, son en esencia aplicativos compartidos que permiten el acceso desde cualquier lugar en el mundo, siempre que se tenga acceso a la tecnología web

Aplicativos móviles:

Son herramientas de un gran crecimiento, que dan al usuario como su nombre lo indica de movilidad, desde dispositivos como el celular, la pocket pc, palm, etc, se podrá acceder a un sin numero de aplicativos de una manera fácil y sencilla y realizar las tareas cotidianas, cabe anotar que también se desarrollan aplicativos de entretenimiento como juegos que están ubicados en la mayor parte de estos dispositivos.

2.4.2. Estructura de un programa

```
import biblioteca.paquete.Clase; //Permite el acceso a las clases JDK
```

```
class NOMBRE{
```

```
    declaración de variables;
```

```
        declaración de objetos o instancias de clases;
```

```
        NOMBRE (tipos y parametos del constructor) {
```

```
            inicialización de variables y objetos;
```

```
        }
```

```
        retorno método1(tipo parámetro) {
```

```
            Sentencia1;
```

```
                return this;
```

```
        }
```

```
        retorno método2() {
```

```
            Sentencias;
```

```
        }
```

```
    }
```

```
class NOMBRE2 extends NOMBRE{
```

```
    declaración de variables;
```

```
    declaración de objetos;
```

```
        NOMBRE2 (tipos parámetros del constructor) {
```

```
            super(parámetros del constructor);
```

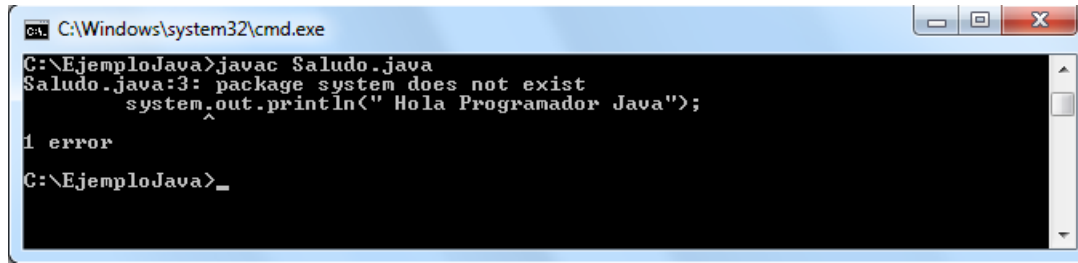
```
                this.Método();
```

```
    }  
    retorno método()    {  
super.métodoS();  
        sentencias;  
        return this;  
    }  
    public static void main(String Params[]) {  
NOMBRE2 Objeto=new NOMBRE2(tipos y parámetros de inicialización);  
        Sentencias... ;  
        Objeto.método(parámetros,,);  
        Sentencias...;  
    }  
}
```

2.4.3. Edición del programa fuente

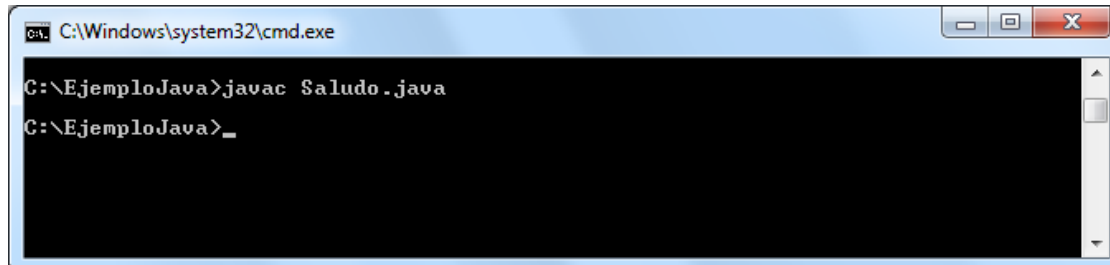
Editar un programa en java es un proceso relativamente sencillo, después del proceso de compilación, este arrojará el o los errores que puedan presentarse en el código digitado, estos errores son netamente de digitación o por falta de componentes que se requieren en el sistema, ubicado el error, será cuestión de seleccionar el archivo afectado y abrirlo en el block de notas o el editor que se tenga para la programación, se ubicará la línea se corregirá y luego se realizará nuevamente la compilación, este proceso puede tardar varios minutos si el número de errores es muy alto.

Veamos un ejemplo.



```
C:\Windows\system32\cmd.exe
C:\EjemploJava>javac Saludo.java
Saludo.java:3: package system does not exist
    system.out.println(" Hola Programador Java");
    ^
1 error
C:\EjemploJava>_
```

Observese que el sistema en su etapa de compilación, arroja el numero de errores y la posible ubicación de este, ingresando al block de notas y abriendo el archivo, se podrá corregir, en este caso particular, la palabra System, debe ir en mayúscula inicial, se guarda el archivo y se procede con la compilación nuevamente hasta que no muestre ningún mensaje.



```
C:\Windows\system32\cmd.exe
C:\EjemploJava>javac Saludo.java
C:\EjemploJava>_
```

En esta imagen, observamos que se compilo y no presento ningún error, esto indica que el programa fue recibido sin inconvenientes en escritura ni faltantes adicionales, este proceso nos crea un archivo .class, que será interpretado por el pc.

2.4.4. Compilación del programa

Los compiladores o intérpretes, nos permite la adecuada revisión de los procesos lógicos expuestos en un lenguaje de programación, el análisis de la sintaxis es fundamental en los procesos realizados y esto lo determina el lenguaje la buena o no especificación de los conceptos lógicos.

Digítese este bloque de código, creando un archivo en un editor para ambiente Java o en block de notas, el archivo debe tener por nombre EjemploHora.java, recuerde conservar la escritura que este tiene.

```
import java.applet.Applet;
```

```
import java.util.Date;

public class EjemploHora extends Applet implements Runnable {
    Date d;
    Thread t;
    boolean activo;

    public void run() {
        while ( activo ) {
            repaint();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) { e.printStackTrace(); }
        }
    }

    public void init() {
        t = new Thread(this);
    }

    public void start() {
        activo = true;
        t.start();
    }

    public void stop() {
        activo = false;
    }

    public void paint(java.awt.Graphics g) {
        d = new Date();
        g.drawString( d.getHours()+":"+ d.getMinutes()+
            ":"+d.getSeconds(), 10, 10);
    }
}
```

Este ejemplo se denomina applet, y requiere un archivo HTML, que debe de crear un el block de notas con el nombre Hora.html.

```
<html>
<title>Ejemplo de reloj</title>
```

```
<body>

<H1 align="center">Ejemplo de reloj</h1>
<hr>
<br>
<div align="center">

<applet code="EjemploHora.class" width=100 height=50>
</applet>
</div>

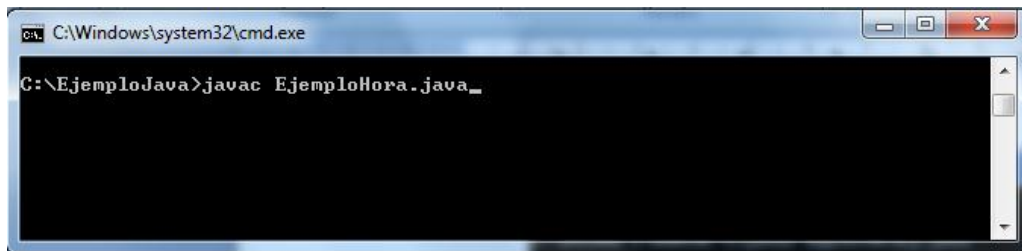
</body>
</html>
```

Para compilar el aplicativo y generar el código que se pueda interpretar, se puede utilizar la ventana de comandos del DOS y usar el comando javac.

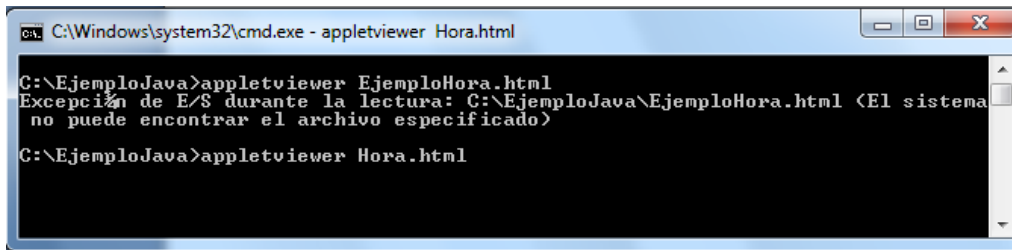
javac es un archivo ejecutable del lenguaje Java, dicho archivo, este archivo se encarga de revisar línea por línea, procesos por proceso todo lo digitado por el usuario desarrollador y determinar su compatibilidad y su uso.

```
javac EjemploHora.java
```

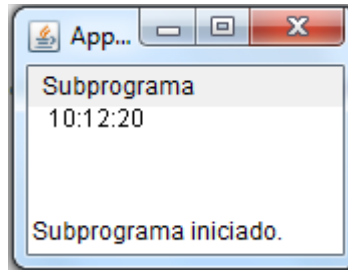
Después de utilizar dicho comando en la ventana del DOS, como lo muestra la gráfica:



Est applet será compilado y únicamente así podrás ver mediante el archivo HTML, dicho visualizador lo puede utilizar con el comando Appletviewer Hora.html



Y tenemos como resultado este archivo.



Tenga presente que los dos archivos creados estén en la misma ubicación.

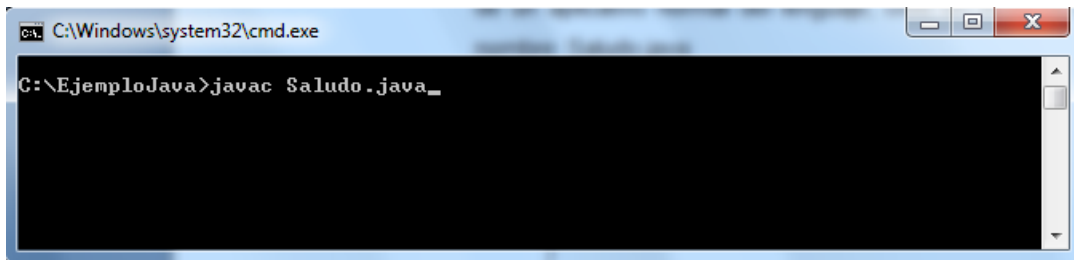
En este ejemplo se observa que al utilizar el comando Javac, no arrojo ningún resultado, esto indica que el proceso es completamente correcto, si arrojara un mensaje de error se debe revisar los procesos lógicos de este o de sintaxis que se puedan tener.

Para ejecutar una aplicación en Java, podemos usar el siguiente ejemplo sencillo:

Tomemos un nuevo ejemplo para observar nuevamente la compilación y ejecución de un aplicativo normal del lenguaje, cree en el block de notas el archivo con el nombre Saludo.java

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

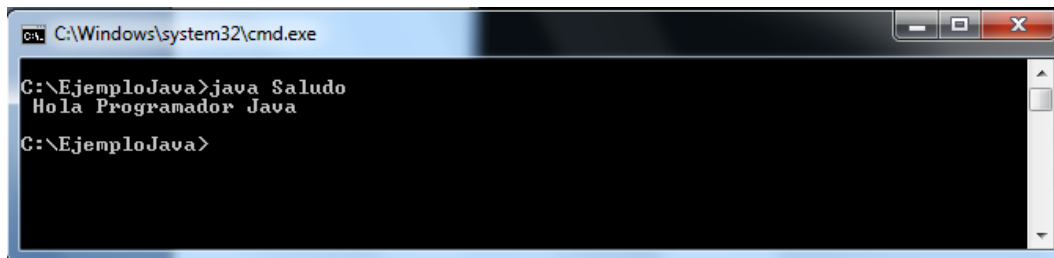
Después de la creación se compílala utilizando el comando javac como se utilizo anteriormente:



```
C:\Windows\system32\cmd.exe

C:\EjemploJava>javac Saludo.java_
```

Suponiendo que el archivo no tiene inconvenientes, se procede a la ejecución, debe utilizar el comando `java Saludo`, este comando se utiliza para interpretar y ejecutar los archivos compilados en java.



```
C:\Windows\system32\cmd.exe

C:\EjemploJava>java Saludo
Hola Programador Java

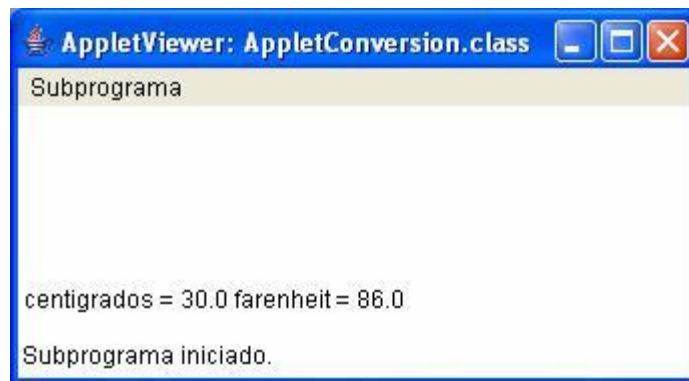
C:\EjemploJava>
```

EJERCICIOS DE AUTOEVALUACIÓN

1. Describa 5 características que describan el lenguaje de programación Java
2. Cree una situación familiar en la que plantee el concepto de clase, objeto, método, herencia, polimorfismo
3. Cree un pequeño ejemplo en el block de notas, que muestre su nombre, sexo y edad, compílelo, y ejecútelo.

Prueba Final

Practicar haciendo un Applet llamado `AppletConversion` que despliegue 30 grados centígrados, calcule y despliegue los grados Fahrenheit correspondientes, utilizando internamente un cálculo para la conversión. El Applet deberá presentar algo parecido a lo siguiente



Actividad

Crear un aplicativo que al ingresar un valor entero, determine si este es un valor par o impar.

3. ELEMENTOS BÁSICOS DEL LENGUAJE

OBJETIVO GENERAL

Conocer las características primordiales del lenguaje referenciado a su estructura, sus normas, su condición de uso, sus fortalezas y debilidades, esto con el fin de brindar seguridad al usuario desarrollador y conocer los pormenores que este presenta de modo que al crear un aplicativo se tenga la confianza y la seguridad de los resultados que se desean obtener.

OBJETIVOS ESPECÍFICOS

- ◆ Conocer los elementos básicos que componen el lenguaje de programación
- ◆ Entrar en detalle en las características de una clase, sus componentes y su propósito
- ◆ Conocer cuales son las clases mas comunes dentro de la programación

Prueba Inicial

Determine cuales son las características que debe tener un programa después de la parte lógica y que características tiene.

3.1. Elementos Básicos

3.1.1. Los tipos de datos primitivos

Los tipos primitivos son los que permiten manipular valores numéricos (con distintos grados de precisión), caracteres tipo texto y valores booleanos (verdadero / falso), además de formatos especiales, Los Tipos Primitivos son:

- ◆ **boolean**: Puede contener los valores **true** o **false**.
- ◆ **byte**: Enteros. Tamaño 8-bits. Valores entre -128 y 127.
- ◆ **short**: Enteros. Tamaño 16-bits. Entre -32768 y 32767.
- ◆ **int**: Enteros. Tamaño 32-bits. Entre -2147483648 y 2147483647.
- ◆ **long**: Enteros. Tamaño 64-bits. Entre -9223372036854775808 y 9223372036854775807.
- ◆ **float**: Números en coma flotante. Tamaño 32-bits.
- ◆ **double**: Números en coma flotante. Tamaño 64-bits.

- ◆ **char**: Caracteres. Tamaño 16-bits. Unicode. Desde '\u0000' a '\uffff' inclusive. Esto es desde 0 a 65535

Ejemplos más comunes

Enteros: Estos tipos son byte, short, int y long, que guardan el signo valor, estos representan un número y no pueden representar elementos fraccionarios.

```
public class enteros
{
    public static void main(String[] arg)
    {
        byte midato1 = 1;
        short midato2 = 100;
        int midato3 = 10000;
        long midato4 = 100000000;
        System.out .println("midato = " + midato1);
    }
}
```

Números en coma flotante: Estos son float y double y pueden almacenar números en coma flotante y con signo.

Todos los literales de coma flotante son del tipo double salvo que se especifique lo contrario, por eso si se intenta asignar un literal en coma flotante a una variable de tipo float el compilador nos dará un.:

```
public class tipoDecimales
{
    public static void main(String[] arg)
    {
        float valor;
        valor = 2.6;
        System.out .println("Valor del dato= " + valor); //esto dará un error.
    }
}
```

El tipo Carácter: Estos son de tipo char, que almacena la representación de los caracteres (letras o números y símbolos), un carácter esta almacenado en 16 bits.

Los caracteres en Java se pueden especificar de forma normal o con secuencias de escape, utilizando el carácter backslash "\" seguida de una letra:(\r) o utilizando el backslash con una "u" seguida de un número hexadecimal (\u0000d), en este caso hemos especificado la secuencia de escape \r y su código correspondiente del retorno de carro.

```
public class tipoCaracter
{
    public static void main(String[] arg)
    {
        char valor, valor1, valor2;
        valor = 'a'; // el literal de tipo carácter tiene que estar encerrado entre comillas simples.
        valor1 = 65;
    }
}
```

Ejemplo de secuencias de escape:

```
\'..... comillas simples.
\".....comillas dobles.
\\.....barra invertida.
\b.....espacio en blanco.
\ddd.....carácter octal.
\f.....avance.
\n.....nueva línea.
\r.....retorno de carro.
\t.....tabulador.
\uxxxx.....carácter Unicoide
```

El tipo booleano: Este solo guarda dos valores: verdadero (true) o falso (false), y no como ocurre en otros lenguajes que toman los valores 0 y 1. Generalmente su utilización es muy frecuente para determinar el flujo de los programas:

```
public class tipoBoolean
{
    public static void main(String[] arg)
    {
        boolean valor1, valor2 ;
        valor = true;
        valor1 = false;
        if (valor1){
            System.out .println("valor1 = verdadero");
        } else {
            System.out .println("valor1 = falso");
        }
    }
}
```

Tabla tipos de datos:

Tipos de datos	Rango de valores	Descripción
Números enteros		
byte	8-bit complemento a 2	Entero de un Byte
short	16-bit complemento a 2	Entero corto
int	32-bit complemento a 2	Entero
long	64-bit complemento a 2	Entero largo
Números reales		
float	32-bit IEEE 754	Coma flotante de precisión simple
double	64-bit IEEE 754	Coma flotante de precisión doble
otros tipos		
char	16-bit Carácter	Un sólo carácter
boolean	true o false	Un valor booleano (verdadero o falso)

3.1.2. Las palabras claves

En el siguiente cuadro se listan las palabras reservadas o claves, aquellas que emplea el lenguaje Java, y que el programador no puede utilizar como variables propias. Algunas de estas palabras le resultarán familiares al programador del lenguaje C/C++. Las palabras reservadas señaladas con un asterisco (*) no se utilizan.

abstract	boolean	break	byte	byvalue*
case	cast*	catch	char	class
const*	continue	default	do	double
else	extends	false	final	finally
float	for	future*	generic*	goto*
if	implements	import	inner*	instanceof
int	interface	long	native	new
null	operator*	outer*	package	private
protected	public	rest*	return	short
static	super	switch	synchronized	this
throw	transient	true	try	var*
void	volatile	while		

3.1.3. Las variables

◆ Variables

Una variable es un área en memoria que tiene un nombre y un Tipo asociado. El Tipo es o bien un Tipo primitivo o puede ser una Referencia.

Es obligatorio declarar las variables antes de usarlas. Para declararlas se indica su nombre y su Tipo, de la siguiente manera:

tipo_variable nombre ;

Ejemplos:

int i; // Declaracion de un entero

char letra; // Declaracion de un carácter

boolean flag; // Declaracion de un booleano

- ◆ El ; es el separador de instrucciones en Java.
- ◆ El símbolo // indica comentarios de línea, no se tendrá en cuenta durante la compilación
- ◆ En Java las mayúsculas y minúsculas son significativas, esto indica que un mismo nombre de variable con distinta escritura equivale a procesos distintos
- ◆ Todas las palabras reservadas del lenguaje van en minúsculas.

Se pueden asignar valores a las variables mediante el signo de asignación (=). Ejemplo:

i = 5; // a la variable i se le asigna el valor 5

letra = 'c'; // a la variable letra se le asigna el valor 'c'

flag = false; // a la variable flag se le asigna el valor false

La declaración y la asignación se pueden combinar en una sola expresión:

int i = 5;

char letra = 'c';

boolean flag = false;

Operaciones Básicas

En java al igual que en C++ se tienen una serie de operadores que ayudan a obtener cálculos, Java trabaja con los siguientes operadores comunes:

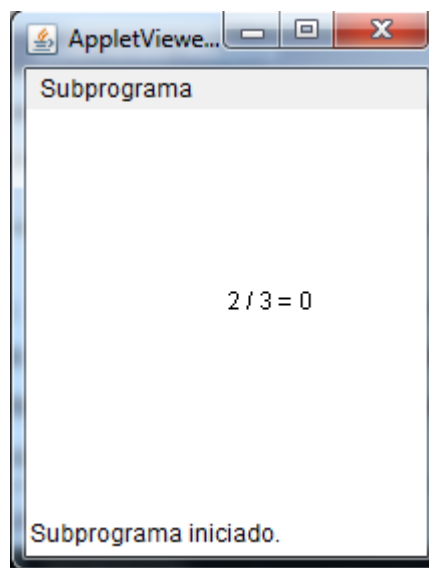
Aritméticos

Operador en Java	Significado
+	suma
-	resta
*	multiplicación
/	división
%	residuo

Ejemplo de operadores.

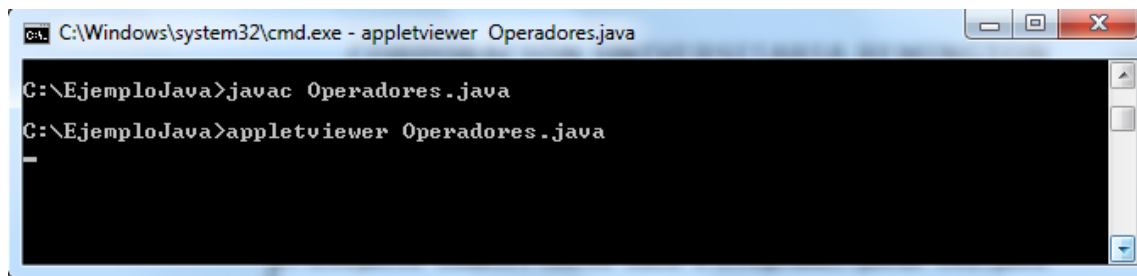
```
import java.awt.*;  
import java.applet.*;  
// <applet width="200" height="200" code="Operadores"></applet>  
public class Operadores extends Applet {  
    public void paint(Graphics g) {  
        int x;  
        x = 2 / 3;  
        g.drawString("2 / 3 = "+x, 100, 100);  
    }  
}
```

El resultado será el siguiente:



Observa como utilizamos dentro del applet Operadores.java el comentario
// <applet width="200" height="200" code="Operadores"></applet>

Esto lo hacemos para no tener que crear un archivo de tipo HTML independiente, ya que un navegador o visualizador de applets lo que requiere para ejecutar el applet es solamente la directiva <applet.



Al estar haciendo operaciones, si hay operandos de diferentes tipos de datos, se convierten al tipo de datos más amplio y el tipo del valor resultante es del tipo más amplio. Por ejemplo, si hay enteros y flotantes, todos los números se convierten a flotantes y el resultado se calcula como flotante.

Por ejemplo:

4/3.0 da como resultado 1.3333

El operador % calcula el residuo de la división entera y sólo existe para datos de tipo entero.

Por ejemplo:

10%3 da como resultado 1

Otros operadores de Asignación

En Java, como en C++, es posible abreviar algunas expresiones de asignación, esto permite trabajar de la forma tradición lógica, o tener una herramienta que nos de un trabajo mas corto pero igual de eficiente.

Operador	Expresión equivalente
$v += e$	$v = v + e$
$v -= e$	$v = v - e$
$v *= e$	$v = v * e$
$v /= e$	$v = v / e$
$v \% = e$	$v = v \% e$

Otros Operadores aritméticos

En Java, existen también los siguientes operadores aritméticos:

++ Incremento unitario

-- decremento unitario

Es decir:

$x++$ ó $++x$ es equivalente a $x = x+1$

$x--$ ó $--x$ es equivalente a $x = x-1$

Jerarquía de los operadores aritméticos

Prioridad	Operadores	Asociatividad
1	()	Empezando por los paréntesis más internos
2	$++$, $--$, $+(positivo)$, $-(negativo)$	De derecha a izquierda, $++$ y $--$ dependiendo de la posición
3	$*$, $/$, $\%$	De izquierda a derecha
4	$+$, $-$	De izquierda a derecha
5	$=$, $+=$, $-=$, $*=$, $/=$, $\%=$	De derecha a izquierda

3.1.4. Los operadores

Jerarquía de los operadores aritméticos

Prioridad	Operadores	Asociatividad
1	()	Empezando por los paréntesis más internos
2	$++$, $--$, $+(positivo)$, $-(negativo)$	De derecha a izquierda, $++$ y $--$ dependiendo de la posición
3	$*$, $/$, $\%$	De izquierda a derecha
4	$+$, $-$	De izquierda a derecha
5	$=$, $+=$, $-=$, $*=$, $/=$, $\%=$	De derecha a izquierda

La jerarquía en los operadores nos ayuda a poder definir la manera adecuada en la que una instrucción de cálculo debe ser escrita en el lenguaje de programación.

3.1.4.1 Conversión entre tipos de datos

Las conversiones de tipos permiten adaptar resultados de un tipo de datos a otro en función de las necesidades de cada parte del programa. Cuando estas conversiones son forzadas explícitamente por el programador reciben el nombre de cast.

En java, al igual que en C++, los casts se realizan metiendo entre paréntesis nombre del tipo destino y anteponiéndolo a la expresión que se desea convertir.

Observemos

```
Int Entero = 4;
```

```
Float real;
```

```
Real = float(entero);
```

El cast dejaría en real el valor 4.0, resultado de convertir entero en punto flotante.

3.2. Estructura General de una Clase

3.2.1. Paquetes e importación de clases

Cuando un lenguaje se ha de usar en proyectos grandes son precisos unos mecanismos que permitan hacer una partición del programa en modulos tan independientes entre si como sea posible, de manera que tengan una gran coherencia interna y unos mecanismos de comunicación bien definidos y poco numerosos.

Si todos los nombres de clase son accesibles desde cualquier parte del programa se produce con gran rapidez una saturación de espacio de nombres. Si además existen varias personas o equipos trabajando de forma independiente en partes distintas del desarrollo, los conflictos de nombres se vuelven casi inevitables.

Con el fin de evitar los problemas de este tipo, java proporciona los llamados paquetes o package, que permiten ocultar un conjunto de clases interrelacionadas e interfaces, de modo que estas clases no son accesibles desde el exterior de cada paquete.

3.2.2. Uso de this y super

Toda clase java tiene definidas implícitamente estas dos variables, que sirven respectivamente, para referenciar el objeto actual y para acceder a métodos o variables de la clase base.

La variable this se refiere a la instancia actual de la clase, y es accesible poro tanto solo desde métodos no estáticos, ya que los métodos estáticos solo pueden acceder a variables de clase.

La variable supera, permite acceder desde una subclase a métodos de su superclase inmediata. Esto tiene interés en el caso de que se redefina un método NombreDeMetodo en la clase derivada, así, NombreDeMetodo (...), hace referencia al método de la clase actual, mientras que super.NombreDeMetodo(...), hace referencia al método original de la clase base.

3.3. Clases de uso común

3.3.1. Datos numéricos – clase math

La clase math, se utiliza normalmente para el manejo de procesos matemáticos no tradicionales, dentro de estas opciones tenemos.

Trigonométricas

Sin : seno del angulo x radianes

Cos : coseno del angulo x radianes

Tan : tangente del angulo x radianes

Trigonométricas inversas

Asin : arco seno del parámetro

Acos : arco coseno del parámetro

Atan : arco tangente del parámetro

Atan2 : arco tangente del a/b

Pow : devuelve valor elevado a la potencia

Exp : devuelve valor elevado

Log : devuelve el logaritmo natural

Sqrt: devuelve la raíz cuadrada de un valor

Ceil: devuelve el número entero mas pequeño mayor o igual

Floor: devuelve número entero menor o igual

3.3.2. Cadenas de caracteres – clase string

◆ Clase String

Una String es una variable referenciada asociada a un objeto de la clase java.lang.String. Se emplea para almacenar cadenas de caracteres.

Las Strings tienen una característica que las diferencia del resto de objetos: son inmutables, es decir, cuando se intenta modificarlas, por ejemplo al aplicarles un método, no se modifican sino que se crea otra String nueva.

Métodos principales

int length(): devuelve la longitud de la String

int indexOf(String str, int indice): devuelve el índice en el que aparece por primera vez la String del primer argumento en la que se aplica el método

String toLowerCase(): devuelve una nueva String convirtiendo todos los caracteres de la String sobre la que se aplica el método, en minúsculas.

String toUpperCase(): devuelve una nueva String convirtiendo todos los caracteres de la String sobre la que se aplica el método, en mayúsculas

3.3.3. Introducción a las excepciones

El término **excepción** se utiliza en Java cuando algo salió de forma incorrecta. Existe un gran número de situaciones por las que el software puede fallar, pero si queremos hacer software de calidad, debemos tomar conciencia que las clases de aplicaciones que desarrollemos, en Java, deben manejar las excepciones.

En Java una excepción es un objeto que define una situación inusual.

Normalmente Java nos detecta algunos errores en los que pudiera suceder una excepción y nos pide que declaremos el lugar en el que puede ocurrir, de esta manera un programa puede estar diseñado para procesar las excepciones en tres formas distintas:

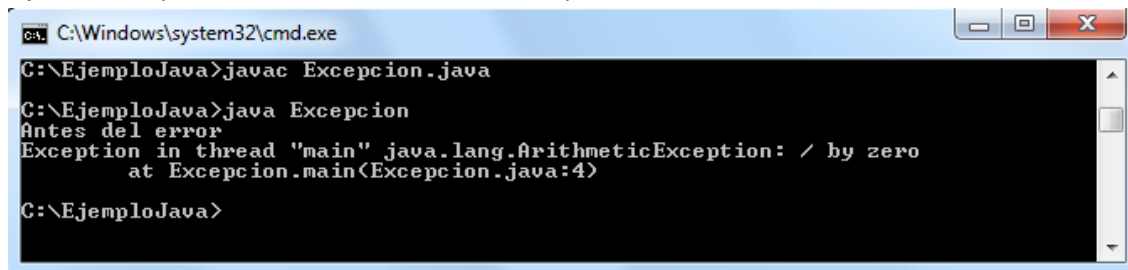
No manejarla
manejar la excepción cuando ocurre
manejar la excepción en algún otro punto del programa

Observemos el siguiente ejemplo:

```
public class Excepcion {  
  
    public static void main(String args[]) {  
        System.out.println("Antes del error");  
        System.out.println("Division por cero = " + 3 / 0);  
        System.out.println("Despues del error");  
    }  
}
```

En este ejemplo vemos como primero se despliega un letrero “Antes del error”, después se despliega un cálculo en el que planificamos que habría un error y finalmente se despliega otro letrero “Después del error”.

Al ejecutar la aplicación anterior observamos lo que sucede:



Vemos como se despliega el letrero “Antes del error”, pero después observamos el error de ejecución denominado en Java una excepción, la Arithmetic Exception, esta excepción sucede cuando tratamos de dividir un número por cero, por eso mismo no alcanzamos a ver el mensaje “Después del error”, hay que tener presente que al presentarse el error de ejecución, se termina de usar la aplicación.

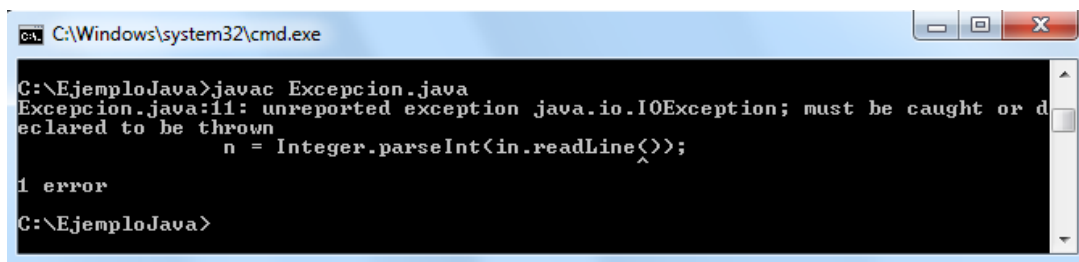
Hay excepciones las cuales el compilador las marca desde un inicio y nos pide que nos protejamos para eso.

```
import java.io.*;
```

```
public class Excepcion {  
    public static void main(String args[]) {
```

```
        BufferedReader in =  
            new BufferedReader(new InputStreamReader(System.in));  
  
        int n;  
        System.out.println("Da el numero");  
        n = Integer.parseInt(in.readLine());  
        System.out.println("El cuadrado del numero = " + n*n);  
    }  
}
```

En este ejemplo observamos que en el momento de compilar se produce la excepción, muy distinto al primer caso en el que vemos la excepción luego de ejecutar el aplicativo

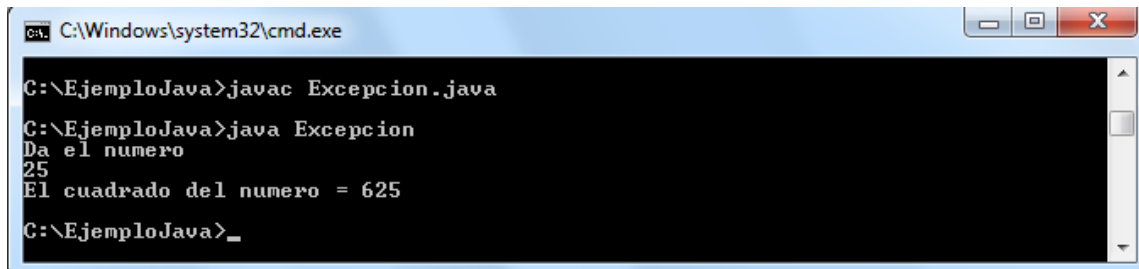


Nos está diciendo que al momento de tratar de leer un valor, puede haber un error de ejecución el cual debe ser capturado para que la aplicación no falle.

La aplicación se compila entonces marcando la excepción que puede lanzar de la siguiente manera:

```
import java.io.*;
```

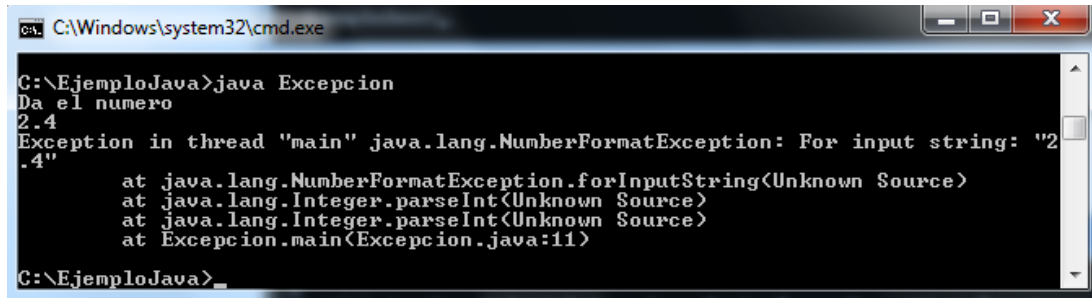
```
public class Excepcion {  
    public static void main(String args[]) throws IOException {  
  
        BufferedReader in =  
            new BufferedReader(new InputStreamReader(System.in));  
  
        int n;  
        System.out.println("Da el numero");  
        n = Integer.parseInt(in.readLine());  
        System.out.println("El cuadrado del numero = " + n*n);  
    }  
}
```



```
C:\Windows\system32\cmd.exe

C:\EjemploJava>javac Excepcion.java
C:\EjemploJava>java Excepcion
Da el numero
25
El cuadrado del numero = 625
C:\EjemploJava>_
```

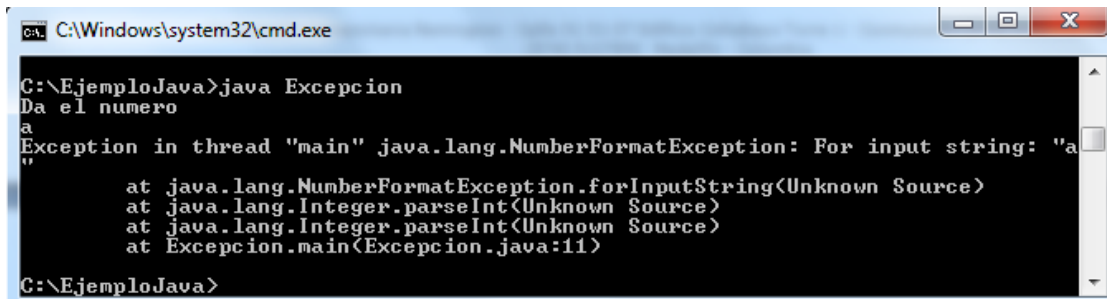
El aplicativo tiene un buen funcionamiento temporal, pero si ingresamos un valor flotante, tendríamos todavía un error de manejo.



```
C:\Windows\system32\cmd.exe

C:\EjemploJava>java Excepcion
Da el numero
2.4
Exception in thread "main" java.lang.NumberFormatException: For input string: "2.4"
    at java.lang.NumberFormatException.forInputString(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at Excepcion.main(Excepcion.java:11)
C:\EjemploJava>_
```

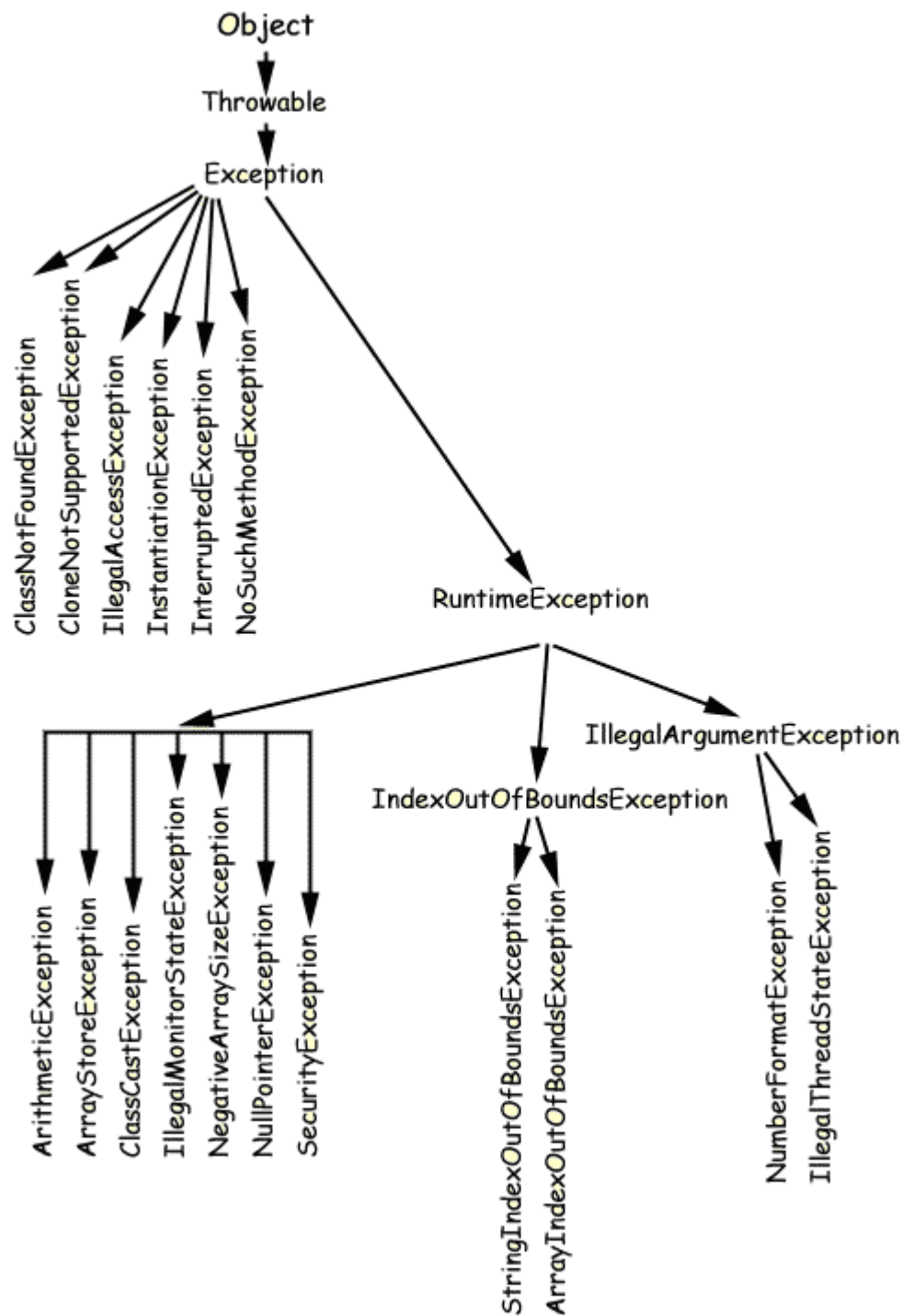
Otra forma de probar esta aplicación es ingresando valores que no son numéricos



```
C:\Windows\system32\cmd.exe

C:\EjemploJava>java Excepcion
Da el numero
a
Exception in thread "main" java.lang.NumberFormatException: For input string: "a"
    at java.lang.NumberFormatException.forInputString(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at Excepcion.main(Excepcion.java:11)
C:\EjemploJava>
```

Observe un cuadro de excepciones, en el se vera la cantidad y variedad de opciones a tener presente cuando se elabora un aplicativo.



Cuando tenemos un aplicativo no es compuesto solo por la parte lógica de funcionamiento, es tener presente todos los sucesos que pueden ocurrir en el manejo de este.

La única manera de hacer que el software no tenga errores, es utilizando la instrucción try/catch, esta es la instrucción en Java que nos permiten detectar y corregir estos errores, veamos primero el formato del try/catch:

```
try
{
    instrucciones que pueden lanzar una excepción
}
catch (Excepción1 e1)
{
    instrucciones para el error 1
}
catch (Excepción2 e2)
{
    instrucciones para el error 2
}
finally // bloque opcional
{
    instrucciones que se hacen se haya encontrado error o no
}
```

Revisando Excepciones Existentes

Entre algunas de las excepciones ya definidas en el sistema más conocidas están:

NullPointerException

Se produce cuando se intenta acceder a una variable antes de ser definido:

```
public class Ejemplo {
    String hola;

    public static void main(String[] args) {
        System.out.println(hola);
    }
}
```

IncompatibleClassChangeException

El intento de cambiar una clase afectada por referencias en otros objetos, específicamente cuando esos objetos todavía no han sido recompilados.

ClassCastException

El intento de convertir un objeto a otra clase que no es válida.

y = (ClaseA)x; // donde x no puede ser de tipo ClaseA

NegativeArraySizeException

Puede ocurrir si hay un error aritmético al cambiar el tamaño de un arreglo.

OutOfMemoryException

sucede con el intento de crear un objeto con el operador new y este ha fallado por falta de memoria. Y siempre tendría que haber memoria suficiente porque el garbage collector se encarga de proporcionarla al ir liberando objetos que no se usan y devolviendo memoria al sistema.

NoClassDefFoundException

Se hizo referencia a una clase que el sistema es incapaz de encontrar.

ArrayIndexOutOfBoundsException

Es la excepción que más frecuentemente se produce. Se genera al intentar acceder a un elemento de un arreglo más allá de los límites definidos inicialmente para ese arreglo. Ejemplo:

```
int arreglo[] = new int[5];  
arreglo[5] = 100; // no puede ser ya que solo existen del cero al cuatro
```

Creando Excepciones

Un programador puede crear excepciones propias en Java, estas pueden ser usadas para casos especiales si el sistema no las proporciona.

Las Excepciones se pueden crear, esto puede ser definido en alguna compañía en la que trabajemos, o por un mismo programador que desarrolle software en el que se desee reutilizar código.

Una manera mas sencilla de crear excepciones nuevas sería la de siempre hacer las subclases de Throwable, como se muestra a continuación:

```
public class MiExcepcion extends Throwable {  
  
    public MiExcepcion() {  
        System.out.println("Se arrojó excepción mía");  
    }  
}
```

Con esto, las excepciones de **MiExcepcion** pueden ser activadas, declaradas y atrapadas como en el siguiente ejemplo:

```
import java.io.*;

public class Excepcion {

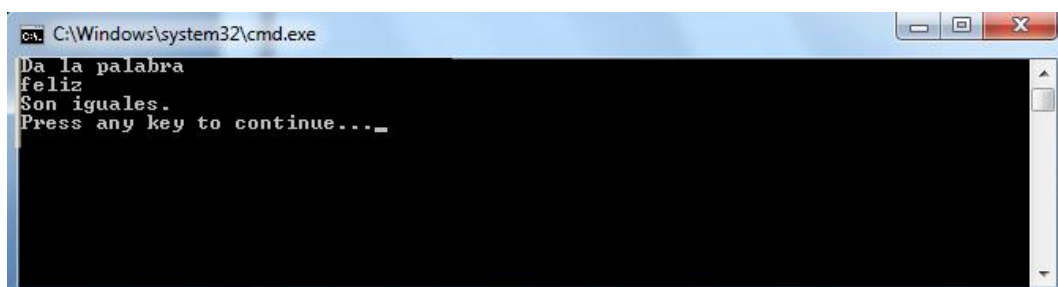
    public static void metodo(String s) throws MiExcepcion {
        if ("feliz".equals(s)) {
            System.out.println("Son iguales.");
        } else {
            throw new MiExcepcion(); //se lanza
        }
    }

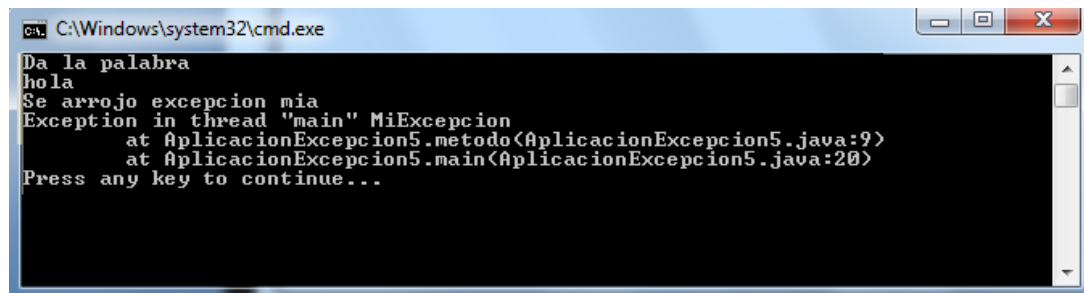
    public static void main(String[] args) throws IOException,
    MiExcepcion {

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

        String s;
        System.out.println("Da la palabra");
        s = in.readLine();
        metodo(s);
    }
}
```

En este ejemplo vemos que si la palabra dada por el usuario es igual a feliz entonces se desplegará el mensaje son iguales, y la aplicación terminará normalmente, pero lado si la palabra no es igual a feliz, entonces aparecerá excepción MiExcepcion y el mensaje se arrojó excepción mía será desplegado, veamos la ejecución en ambos casos:





```
C:\Windows\system32\cmd.exe
Da la palabra
hola
Se arrojo excepcion mia
Exception in thread "main" MiExcepcion
    at AplicacionExcepcion5.metodo(AplicacionExcepcion5.java:9)
    at AplicacionExcepcion5.main(AplicacionExcepcion5.java:20)
Press any key to continue...
```

Throw

La instrucción throw puede ser usada para activar una excepción sea esta propia del programador o sea del sistema, se podría decir que throw ArithmeticException, así como establecimos throw MiExcepcion.

EJERCICIOS DE AUTOEVALUACIÓN

1. Crear un diagrama y determinar cuales variables son y de que tipo seria el ideal de ellas.
2. Determine de las siguientes operación cuales requiere math (suma, raíz, potencia, seno, multiplicación)

Prueba Final

Crear un aplicativo que lea su edad y salario, y realizar excepción para los tipos de datos ingresados.

Actividad

Crear un aplicativo que ingrese su nombre, estatura, utilice las clases math y string y aplique excepciones propias para validar los datos ingresados.

4. ESTRUCTURAS DE CONTROL

OBJETIVO GENERAL

Conocer Las estructuras dentro del lenguaje de programación con el fin de afianzar herramientas que brinden oportunidad de trabajo, manejo eficiente de recursos, alternativas de trabajo que nos dan la oportunidad de evaluar y de repetir tareas en forma corta y sencilla.

OBJETIVOS ESPECÍFICOS

- ◆ Conocer las estructuras evaluativas del lenguaje
- ◆ Establecer conceptos de procesos repetitivos

Prueba inicial

Determinar mediante procesos lógicos las características que puede tener un juego como el triqui (tres en línea), y especificar si requiere procesos repetitivos y evaluativos.

4.1. Estructuras Básicas

Condiciones, selectores múltiples

if

Esta instrucción permite realizar procesos básicos de comparación entre diferentes procesos.

Sintaxis

```
if ( condición
    estatuto;
else
    estatuto; // la parte else es opcional
```

En caso de requerir más de un proceso comparativo es necesario usar llaves.

```
if ( condición ) {
    bloque de estatutos;
}
```

```
else {                                // la parte else es opcional
    bloque de estatutos;
}
```

Una condición es la comparación de una variable/constante/expresión-numérica contra otra variable/constante/expresión-numérica. A este tipo de condición se le llama condición simple, la condición compleja se manejarán mas adelante:

Operadores Relacionales

Los operadores relacionales que tiene Java son:

Operador en Java	Significado
==	Igual
!=	Diferente
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que

Un ejemplo puede ser que X sea igual a 10 la condición quedaría como

If (x == 10)

Otro podría ser comparar si el área del círculo con radio r es menor a 100

If (Math.PI() * Math.pow(r, 2) < 100)

Etc.

Ejecución del if

La estructura if, ejecuta una instrucción solo cuando la condición es verdadera; en caso de que sea falsa brinca el proceso representado con else, aunque este es opcional.

La estructura de selección if / else , ejecuta las condiciones después de la parte if cuando la condición es verdadera; en caso de que sea falsa ejecuta las acciones que están después del else.

Ejemplo

```
if (promedio >= 70)
    t.setText(" Aprobado");
else
    t.setText("Reprobado");
```

Anidamiento de condiciones

Se dice que hay if anidados cuando existe un if/else dentro de otra estructura if/else

Ejemplo:

Determinar si un número es positivo, cero o negativo.

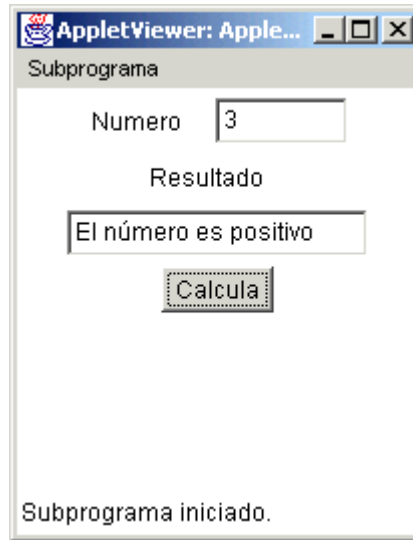
```
if (num > 0)
    t.setText("El número es positivo");
else if (num == 0)
    t.setText("El número es cero");
else
    t.setText("El número es negativo");
```

El applet quedaria como sigue:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
// <applet width="200" height="200" code="AppletDecisiones1"></applet>
public class AppletDecisiones1 extends Applet implements ActionListener {
    Label l1, l2;
    TextField t1, t2;
    Button b;

    public AppletDecisiones1() {
        l1 = new Label("Numero");
        t1 = new TextField(6);
        l2 = new Label("Resultado");
        t2 = new TextField(18);
        b = new Button("Calcula");
        add(l1);
        add(t1);
        add(l2);
        add(t2);
        add(b);
        b.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae) {
        int num = Integer.parseInt(t1.getText());
        if (num > 0)
            t2.setText("El número es positivo");
        else if (num == 0)
            t2.setText("El número es cero");
        else
            t2.setText("El número es negativo");
    }
}
```

El applet en ejecución se muestra a continuacion:



En el ejemplo anterior se realizó el uso del constructor `TextField(entero)` donde `entero` es un número entero para definir el número de espacios que se quieren usar en el campo de texto creado.

Si tenemos el siguiente fragmento de código:

```
if (condición 1)
    if (condición 2)
        estatuto;
    else // este else pertenece al if de la condición 2, pues se asocia al if más cercano
        estatuto;
```

Y queremos que el `else` pertenezca al primer `if` debemos poner `{}` para determinar donde termina el segundo `if`:

```
if (condición 1)
{
    if (condición 2)
        estatuto;
}
else // con el uso de llaves cerramos el if anidado y el else
    estatuto; // pertenece al primer if
```

Ejemplos de Programas

Ejemplo I: Programa que calcula la energía necesaria para la combustión de un compuesto dada la cantidad de átomos

"x" y "y" de dos diferentes elementos, de acuerdo a la siguiente fórmula:

$$4x + 3x^2y - 2y \text{ si } x > 0, y \geq 0$$

$$e(x,y) = x^2 - 4(y - x) + y^2 \text{ si } x < 0, y \geq 0$$

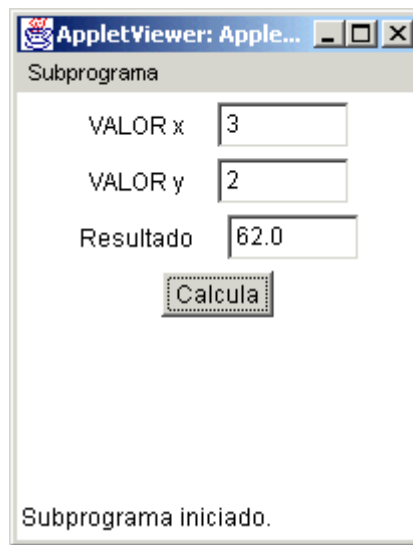
$$x^2 + y + y(x - 2) \text{ en cualquier otro caso}$$

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.lang.Math;
// <applet width="200" height="200" code="AppletDecisiones2"></applet>
public class AppletDecisiones2 extends Applet implements ActionListener {
    Label l1, l2, l3;
    TextField t1, t2, t3;
    Button b;

    public AppletDecisiones2() {
        l1 = new Label("VALOR x");
        t1 = new TextField(6);
        l2 = new Label("VALOR y");
        t2 = new TextField(6);
        l3 = new Label("Resultado");
        t3 = new TextField(6);
        b = new Button("Calcula");
        add(l1);
        add(t1);
        add(l2);
        add(t2);
        add(l3);
        add(t3);
        add(b);
        b.addActionListener(this);
    }

    public void actionPerformed(ActionEvent ae) {
        double x = Double.parseDouble(t1.getText());
        double y = Double.parseDouble(t2.getText());
        double res;
        if (y >= 0)
```

```
if ( x > 0)
    res = 4 * x + 3 * Math.pow (x,2) * y - 2 * y;
else
    res = Math.pow(x,2) - 4* (y - x) + Math.pow (y,2);
else
    res = Math.pow (x,2) + y + y * (x-2);
t3.setText("" + res);
}
}
```



Ejemplo II: Programa que lee 3 números enteros diferentes y los despliega de mayor a menor.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
```

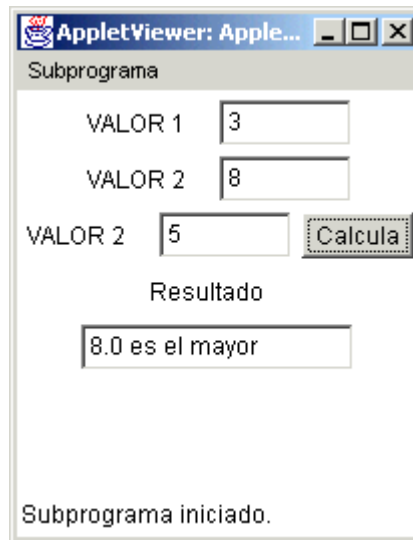
```
// <applet width="200" height="200" code="AppletDecisiones3"></applet>
```

```
public class AppletDecisiones3 extends Applet implements ActionListener {
    Label l1, l2, l3,l4;
    TextField t1, t2, t3,t4;
    Button b;
```

```
public AppletDecisiones3() {
    l1 = new Label("VALOR 1");
    t1 = new TextField(6);
    l2 = new Label("VALOR 2");
    t2 = new TextField(6);
```

```
l3 = new Label("VALOR 2");
t3 = new TextField(6);
l4 = new Label("Resultado");
t4 = new TextField(16);
b = new Button("Calcula");
add(l1);
add(t1);
add(l2);
add(t2);
add(l3);
add(t3);
add(b);
add(l4);
add(t4);
b.addActionListener(this);
}
public void actionPerformed(ActionEvent ae) {
    double a = Double.parseDouble(t1.getText());
    double b = Double.parseDouble(t2.getText());
    double c = Double.parseDouble(t3.getText());
    double res;
    if (a >= b)
        if (a >= c)
            t4.setText("" + a + " es el mayor");
        else
            t4.setText("" + c + " es el mayor");
    else
        if (b >= c)
            t4.setText("" + b + " es el mayor");
        else
            t4.setText("" + c + " es el mayor");
    }
}
```

La aplicación ejecutando funcionaría como se observa:



Estatuto Switch o Selector Multiple

Se utiliza para ejecutar condiones de multiples valores, en el caso anterior se evalua una intruccion y se determinan 2 posibles respuestas, ahora tenemos la posibilidad de evaluar multiples condiciones en un mismo procedimiento..

Consiste en una serie de etiquetas case y un case por omisión (por defecto)

Sintaxis

switch (variable) la variable es de tipo entero o carácter

```
{  
    case valor1 : accion1; break;  
    case valor2 : accion2; break;  
    .  
    .  
    case valor n : accionn; break;  
    default: accionD;  
};
```

La acción 1 se ejecuta si la variable adquiere el valor1.

La acción 2 se ejecuta si la variable adquiere el valor 2.

La acción n se ejecuta si la variable adquiere el valor n.

Cualquier otro valor de la variable conduce a la realización de la secuencia accionD, indicada por la palabra reservada default.

esta instrucción default puede obviarse según el tipo de condicionales que requerimos.

Instrucción break

Cuando se encuentra una sentencia case que concuerda con el valor del switch se ejecutan las sentencias que le siguen y todas las demás a partir de ahí, la instrucción break se encarga de realizar un rompimiento que indica que el procedimiento finalizo.

Ejemplo: Programa que pide un número de mes y escribe la cantidad de días que tiene.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

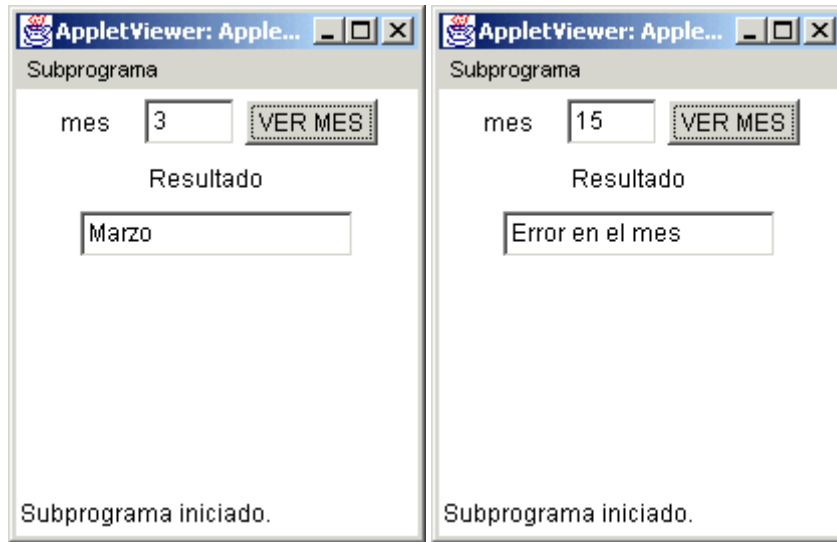
// <applet width="200" height="200" code="AppletDecisiones4"></applet>
public class AppletDecisiones4 extends Applet implements ActionListener {
    Label l1, l2;
    TextField t1, t2;
    Button b;

    public AppletDecisiones4() {
        l1 = new Label("mes");
        t1 = new TextField(3);
        l2 = new Label("Resultado");
        t2 = new TextField(16);
        b = new Button("VER MES");
        add(l1);
        add(t1);
        add(b);
        add(l2);
        add(t2);
        b.addActionListener(this);
    }

    public void actionPerformed(ActionEvent ae) {
        int mes = Integer.parseInt(t1.getText());
        switch (mes)
        {
            case 1:
                t2.setText("Enero");
                break;
            case 2:
                t2.setText("Febrero");
                break;
            case 3:
```

```
        t2.setText("Marzo");  
        break;  
    case 4:  
        t2.setText("Abril");  
        break;  
    case 5:  
        t2.setText("Mayo");  
        break;  
    case 6:  
        t2.setText("Junio");  
        break;  
    case 7:  
        t2.setText("Julio");  
        break;  
    case 8:  
        t2.setText("Agosto");  
        break;  
    case 9:  
        t2.setText("Septiembre");  
        break;  
    case 10:  
        t2.setText("Octubre");  
        break;  
    case 11:  
        t2.setText("Noviembre");  
        break;  
    case 12:  
        t2.setText("Diciembre");  
        break;  
    default:  
        t2.setText("Error en el mes");  
    }  
}  
}
```

Algunos ejemplos de esta aplicación son:



4.1.1. Ciclos

Bucle o Ciclo for Sintaxis

```
for (inicialización ; condición ; acción )  
for (inicialización ; condición ; acción )  
{  
    bloque de estatutos;  
}
```

Funcionamiento del For

1. Ejecuta la o las instrucción de inicialización
2. Evalúa la condición, si es verdadera entra al ciclo
3. Ejecuta los procesos
4. Ejecuta la o las acciones y regresa al paso 2

Notas sobre el For

Las 3 partes del for son opcionales, si no se pone condición se toma como verdadero.

Si no se incluye la inicialización o condición, los ; deben de ir.

Ejemplo: for (; a > 10 ; a--)

Si la primera vez la condición es falsa no se ejecuta ningún estatuto y termina el for

Una variable puede declararse en la sección de inicialización, solo hay que tomar en cuenta que esta variable solo es reconocida dentro del ciclo.

Ejemplo: for (int num = 1; num <= 10; num++)

Ejemplo I: Mostrar los N primeros números de la serie de Fibonacci. La serie es 1,1,2,3,5,8,13....

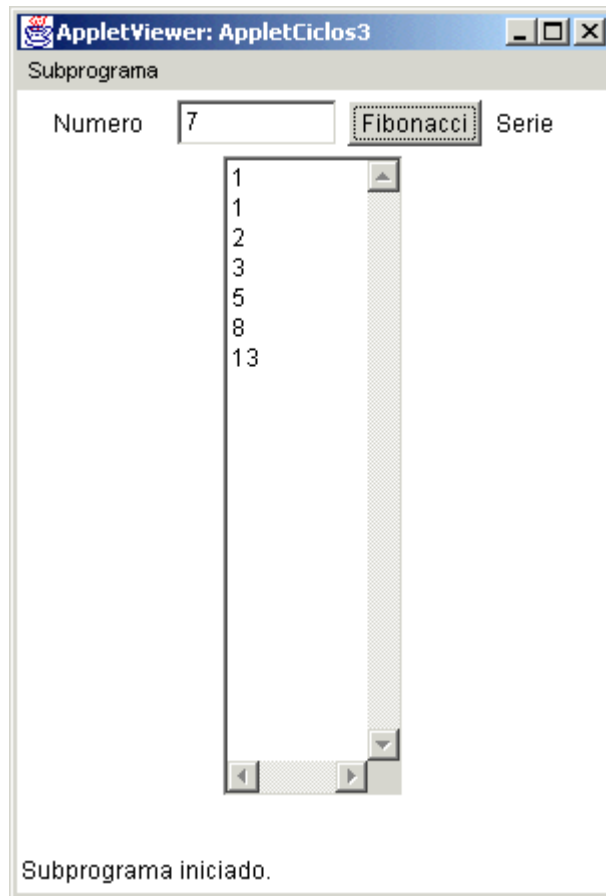
```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

// <applet width="300" height="400" code="AppletCiclos3"></applet>
public class AppletCiclos3 extends Applet implements ActionListener {
    Label l1, l2;
    TextField t1;
    TextArea t;
    Button b;

    public AppletCiclos3() {
        l1 = new Label("Numero");
        t1 = new TextField(8);
        l2 = new Label("Serie");
        t = new TextArea(20,10);
        b = new Button("Fibonacci");
        add(l1);
        add(t1);
        add(b);
        add(l2);
        add(t);
        b.addActionListener(this);
    }

    public void actionPerformed(ActionEvent ae) {
        int n = Integer.parseInt(t1.getText());
        t.setText("1" + "\n");
        t.append("1" + "\n");
        int a = 1, b = 1, fibo;
        for (int i = 3; i <= n; i++) // empiezo i en 3 porque ya mostre los 2 primeros
        {
            fibo = a + b;
```

```
t.append("" + fibo + "\n");  
a = b;  
b = fibo;  
}  
}  
}
```

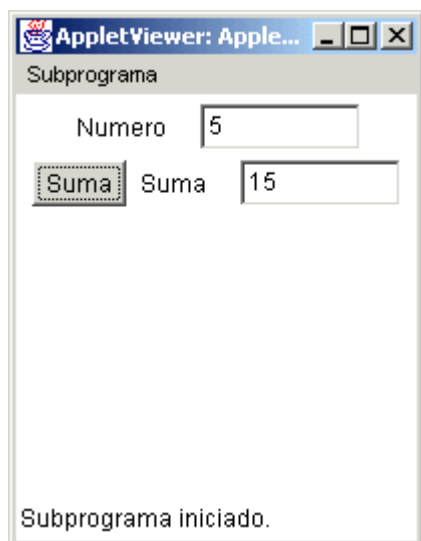


Ejemplo II: Sumar todos los números impares desde 1 hasta el número dado por el usuario

```
import java.awt.*;  
import java.applet.*;  
import java.awt.event.*;  
  
// <applet width="200" height="200" code="AppletCiclos4"></applet>  
public class AppletCiclos4 extends Applet implements ActionListener {  
    Label l1, l2;  
    TextField t1, t2;  
    Button b;
```

```
public AppletCiclos4() {  
    l1 = new Label("Numero");  
    t1 = new TextField(8);  
    l2 = new Label("Suma");  
    t2 = new TextField(8);  
    b = new Button("Suma");  
    add(l1);  
    add(t1);  
    add(b);  
    add(l2);  
    add(t2);  
    b.addActionListener(this);  
}  
  
public void actionPerformed(ActionEvent ae) {  
    int n = Integer.parseInt(t1.getText());  
    int suma = 0;  
    for (int i = 1; i <= n; i++) {  
        suma += i;  
    }  
    t2.setText("" + suma);  
}  
}
```

La ejecución del siguiente applet quedaría como:



Ciclo o Bucle while

Sintaxis

```
while (condición)
    estatuto;
```

Si se requiere realizar más de una instrucción se deben utilizar llaves.

```
while ( condición )
{
    bloque de estatutos;
}
```

Si la condición es falsa la primera vez nunca se ejecutan las instrucciones contenidas.

Ejemplo: Applet que toma la cantidad de dinero a invertir, el porcentaje de inversión mensual y el dinero que se quiere tener invertido finalmente y va desplegando en un campo (TextArea) el nuevo saldo mes tras mes.

```
import java.awt.*;
```

```
import java.applet.*;
```

```
import java.awt.event.*;
```

```
// <applet width="300" height="400" code="AppletCiclos1"></applet>
```

```
public class AppletCiclos1 extends Applet implements ActionListener {
```

```
    Label l1, l2, l3, l4;
```

```
    TextField t1, t2, t3;
```

```
    TextArea ta;
```

```
    Button b;
```

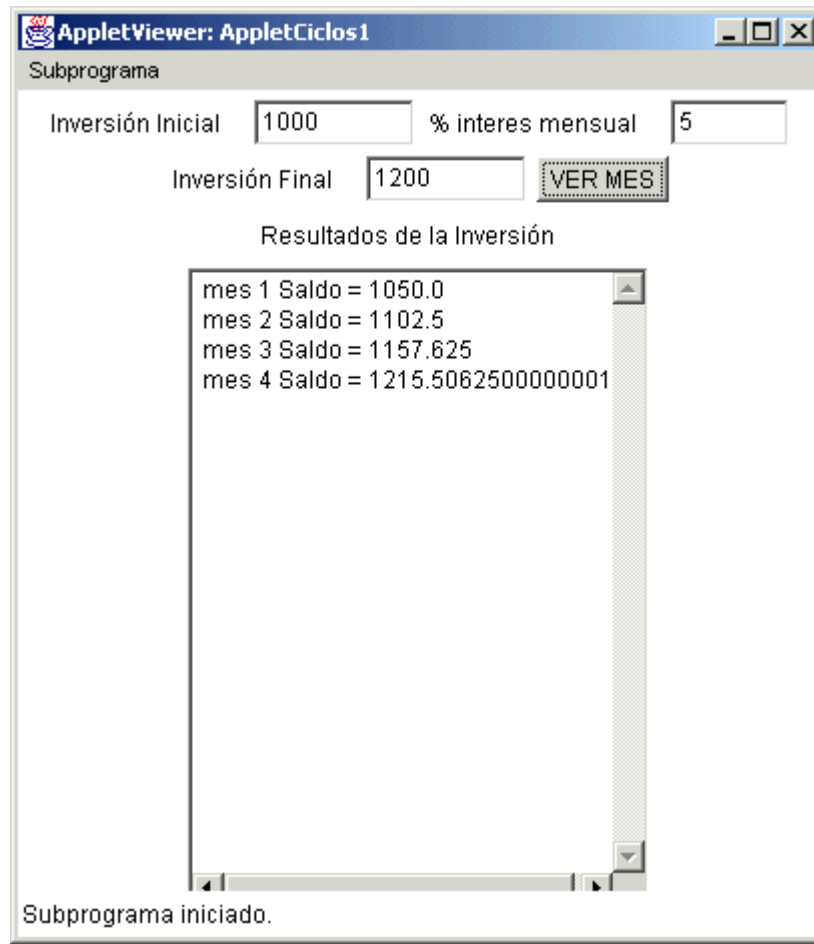
```
    public AppletCiclos1() {
```

```
        l1 = new Label("Inversión Inicial");
```

```
t1 = new TextField(8);
l2 = new Label("% interes mensual");
t2 = new TextField(5);
l3 = new Label("Inversión Final");
t3 = new TextField(8);
l4 = new Label("Resultados de la Inversión");
ta = new TextArea(20,30);
b = new Button("VER MES");
add(l1);
add(t1);
add(l2);
add(t2);
add(l3);
add(t3);
add(b);
add(l4);
add(ta);
b.addActionListener(this);
}

public void actionPerformed(ActionEvent ae) {
    double invinicial = Double.parseDouble(t1.getText());
    double interes = Double.parseDouble(t2.getText());
    double invfinal = Double.parseDouble(t3.getText());
    ta.setText("");
    int mes = 1;
    double saldo = invinicial;
    while (saldo < invfinal) {
        saldo = saldo * (1 + interes/100);
        ta.append(" mes " + mes + " Saldo = " + saldo + "\n");
        mes ++;
    }
}
}
```

Un ejemplo de la ejecución de este applet es:



En este applet hemos utilizado el texto de área `TextArea` `t`, el cual nos ayuda a mostrar la información por línea, haciendo uso del método `append()`, es importante también notar que dentro del método `append` hemos concatenado el caracter `"\n"`, el cual nos sirve para saltar de línea dentro del objeto `TextArea`, ya que con el `append` añadimos caracteres que son concatenados, pero nunca se salta de línea.

Ciclo o Bucle `do .. while`

Sintaxis

`do`

`estatuto;`

`while` (condición);

Si se requiere realizar más de una instrucción se deben utilizar llaves.

`do`

{

```
    bloque de estatutos;  
}  
while ( condición );    // nota que lleva ;
```

Se realizan las instrucciones y se verifica la condición, mientras sea verdadera se sigue ejecutando; al momento de ser falsa termina El ciclo.

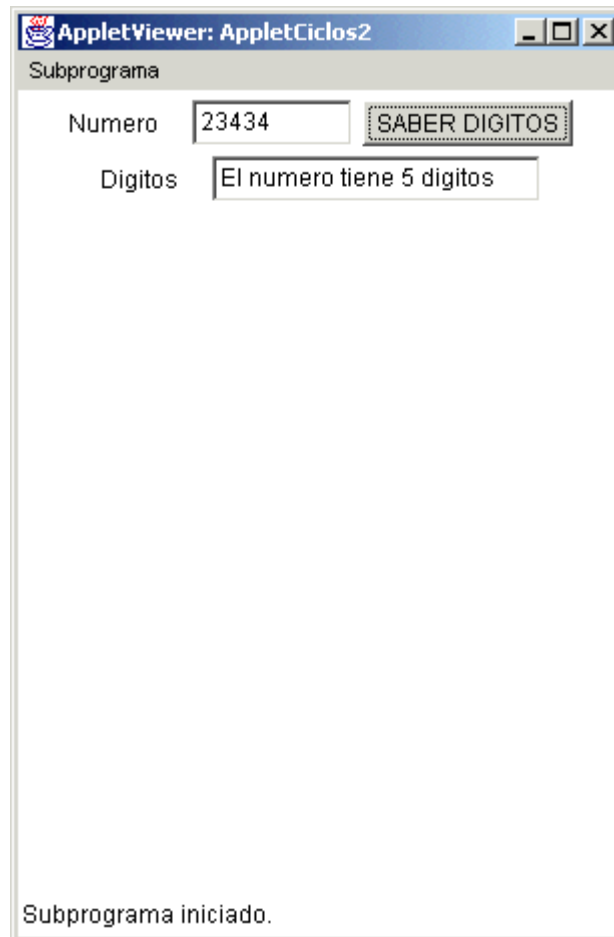
Dado que la condición se revisa al final del ciclo el (los) estatuto (s) se realizan al menos una vez a diferencia del while

Ejemplo: Dado un número en un campo texto, desplegar en otro el número de dígitos del primero

```
import java.awt.*;  
import java.applet.*;  
import java.awt.event.*;  
  
// <applet width="300" height="400" code="AppletCiclos2"></applet>  
public class AppletCiclos2 extends Applet implements ActionListener {  
    Label l1, l2;  
    TextField t1, t2;  
    Button b;  
  
    public AppletCiclos2() {  
        l1 = new Label("Numero");  
        t1 = new TextField(8);  
        l2 = new Label("Digitos");  
        t2 = new TextField(20);  
        b = new Button("SABER DIGITOS");  
        add(l1);  
        add(t1);  
        add(b);  
        add(l2);  
        add(t2);  
        b.addActionListener(this);  
    }  
  
    public void actionPerformed(ActionEvent ae) {  
        int x = Integer.parseInt(t1.getText());  
        int cant = 0;  
        do  
        {  
            x = x / 10;  
            cant++;  
        } while (x > 0);  
    }  
}
```

```
t2.setText("El numero tiene " + cant + " digitos");  
}  
}
```

La visualización de este applet queda de la siguiente manera:



4.2. Arreglos

Introducción

¿Que es un arreglo o Array?

Un arreglo es un tipo de dato estructurado que permite guardar colecciones de elementos del mismo tipo, estos se almacenan en secuencia y se denomina manejo estatico de la memoria.

Arreglo

--	--	--	--	--

Esto representa un lugar en memoria en el que se pueden guardar más de un valor en una misma variable, solo utilizando el índice en el que se encuentra el valor deseado.

Declaración de arreglos

Para declarar un arreglo se utiliza el siguiente formato:

```
tipo nombre_arreglo [] = new tipo[tamaño];
```

Donde tipo es el tipo de los datos que almacenará el arreglo. Es importante mencionar que se pueden declarar arreglos de los tipos primitivos de Java (int, double, char, etc) o bien de tipos definidos por el usuario (Persona, Estudiante, etc).

Tamaño representa la cantidad de posiciones que se reservan para el arreglo. En Java todos los arreglos empiezan en la posición 0 y llegan la posición tamaño-1.

Por ejemplo:

```
int arr[] = new int[6]; // arreglo de 6 elementos enteros  
char cad[] = new char[10]; // arreglo de 10 elementos de tipo carácter
```

En los ejemplos anteriores, arr es un arreglo entero de 6 elementos, cuyo índice inicial es cero y el último es 5, cad es un arreglo de caracteres, que contiene 10 diferentes caracteres, desde el cero hasta el 9.

Uso de los elementos del arreglo

Para usar los elementos individuales de un arreglo se usa el siguiente formato:

arreglo[posición]

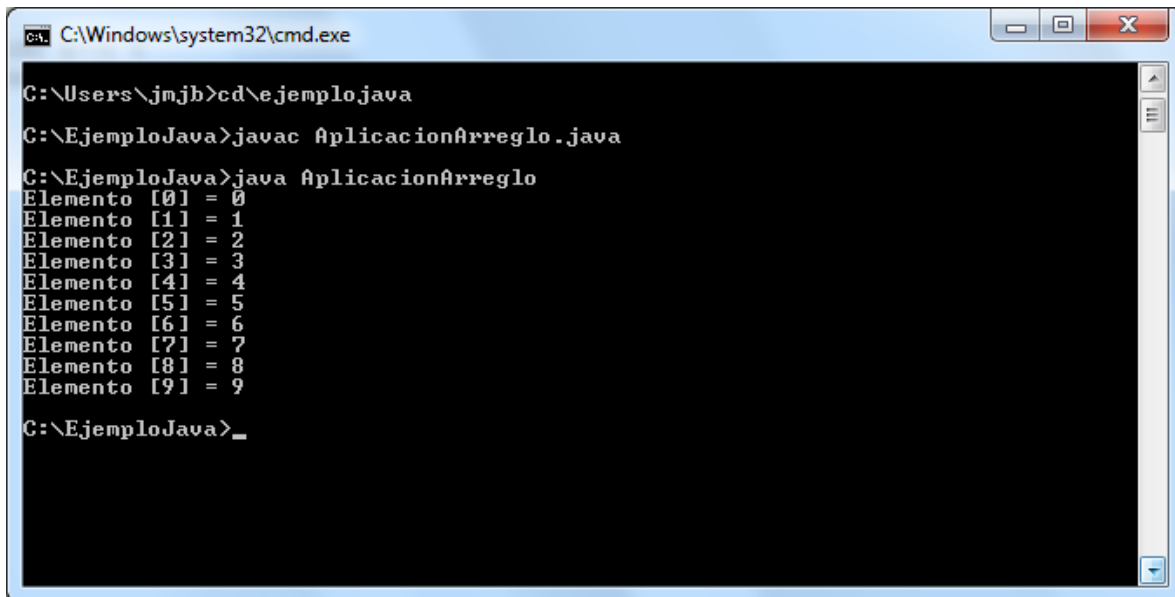
Como un elemento de un arreglo es un dato, se puede usar como cualquier variable de ese tipo.

```
int arr[] = new int [4];  
arr[3] = 8;  
arr[2]= Integer.parseInt(t1.getText());  
t2.setText("" + arr[3]);  
arr[0] = arr[1] + arr[2];  
int k = 2;  
arr[k+1] = 20;
```

Ejemplo:

En este siguiente ejemplo, tenemos una aplicación que define un arreglo de 10 enteros y los inicializa con el valor de 0 a 9 correspondientemente el valor de cada índice, es decir que el elemento cero tiene un cero, el elemento 1, tiene 1, y así sucesivamente hasta 9. Finalmente se despliegan los valores.

```
public class AplicacionArreglo {  
  
    public static void main(String[] args) {  
        int arreglo[] = new int [10];  
  
        for (int i=0; i<10; i++) {  
            arreglo [i] = i;  
        }  
  
        for (int i=0; i<10; i++) {  
            System.out.println("Elemento [" + i + "] = " + arreglo[i]);  
        }  
    }  
}
```



```
C:\Windows\system32\cmd.exe  
C:\Users\jmjb>cd\ejemplo.java  
C:\EjemploJava>javac AplicacionArreglo.java  
C:\EjemploJava>java AplicacionArreglo  
Elemento [0] = 0  
Elemento [1] = 1  
Elemento [2] = 2  
Elemento [3] = 3  
Elemento [4] = 4  
Elemento [5] = 5  
Elemento [6] = 6  
Elemento [7] = 7  
Elemento [8] = 8  
Elemento [9] = 9  
C:\EjemploJava>_
```

Inicializar arreglos en la declaración

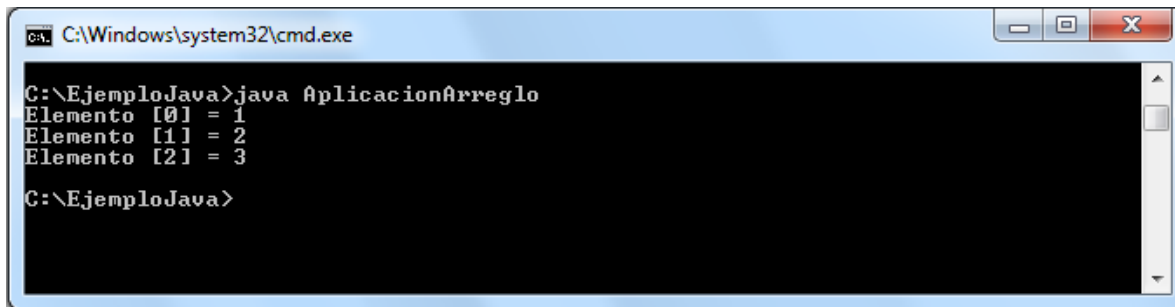
Java tiene la particularidad de que puede iniciar arreglos de forma predefinida, sin intervención del usuario, así:

```
double arreglo[] = { 23.5, 54.22, 78};
```

```
char cadena[] = {'a', 'b', 'c', 'd'};
```

esto da pie a que utilicemos propiedades del arreglo que determinen el tamaño o la cantidad de posiciones según se tengan almacenadas, la palabra `length`, como se muestra en el siguiente ejemplo, determina cuantos elementos hay dentro del arreglo creado.

```
public class AplicacionArreglo {  
  
    public static void main(String[] args) {  
        int arreglo[] = {1,2,3};  
  
        for (int i=0; i<arreglo.length; i++) {  
            System.out.println("Elemento [" + i + "] = " + arreglo[i]);  
        }  
    }  
}
```



```
C:\Windows\system32\cmd.exe  
C:\EjemploJava>java AplicacionArreglo  
Elemento [0] = 1  
Elemento [1] = 2  
Elemento [2] = 3  
C:\EjemploJava>
```

Es muy sencillo tomar datos y agregarlos a un arreglo, como lo puede ver en el siguiente ejemplo, este applet, tomará un dato de texto y lo añadirá en un arreglo de números, además desplegará lo que tiene el arreglo en memoria, para ser desplegado en el texto de área.

```
import java.awt.*;
```

```
import java.applet.*;
```

```
import java.awt.event.*;
```

```
// <applet width="400" height="200" code="AppletArreglos1"></applet>
```

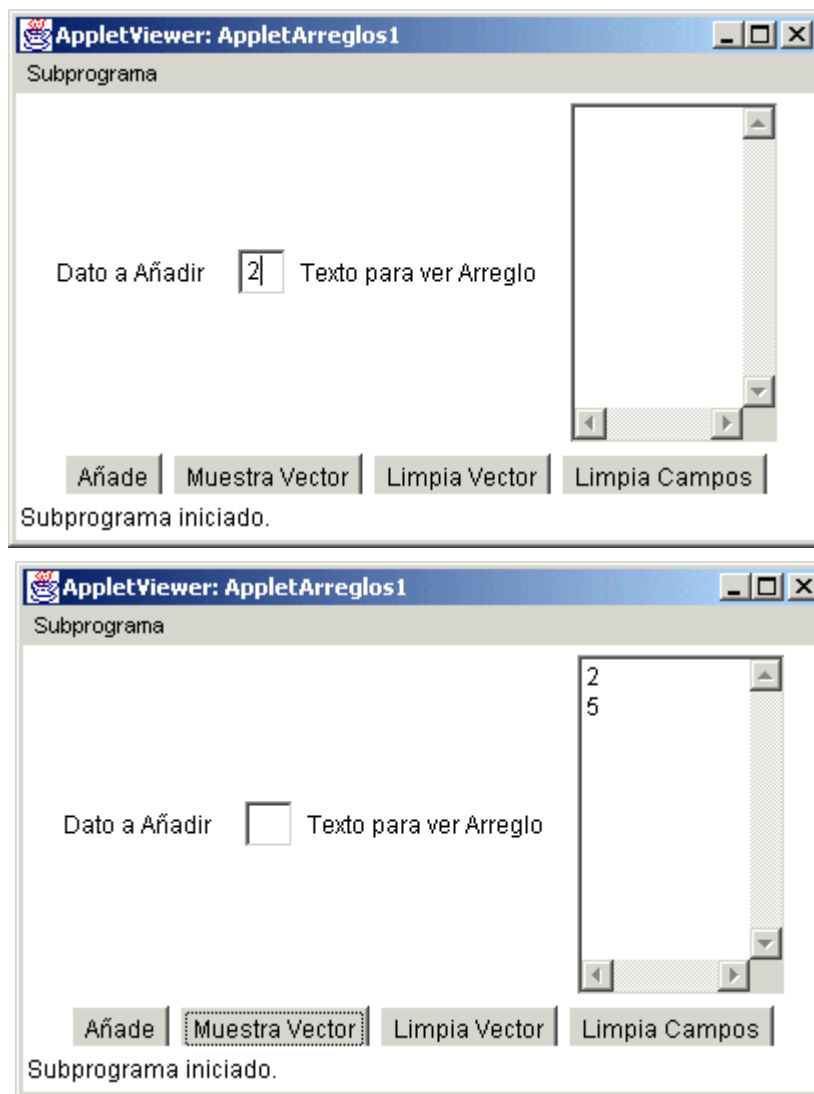
```
public class AppletArreglos extends Applet implements ActionListener{
```

```
Label l1, l2;
    Button b1, b2,b3,b4;
    TextField t1;
    TextArea ta1;
    int arreglo[];
    int conta;

public AppletArreglos1() {
    l1 = new Label("Dato a Añadir");
    l2 = new Label("Texto para ver Arreglo");
    t1 = new TextField();
    ta1 = new TextArea(10,12);
    b1 = new Button("Añade");
    b2 = new Button("Muestra Vector");
    b3 = new Button("Limpia Vector");
    b4 = new Button("Limpia Campos");
    add(l1);
    add(t1);
    add(l2);
    add(ta1);
    add(b1);
    add(b2);
    add(b3);
    add(b4);
    b1.addActionListener(this);
    b2.addActionListener(this);
    b3.addActionListener(this);
    b4.addActionListener(this);
    arreglo = new int[100];
    conta=0;
}

public void actionPerformed(ActionEvent ae) {
    if (ae.getSource() == b1) {
        if (conta > arreglo.length) {
            ta1.setText("No se puede añadir otro elemento");
        }
        else {
            arreglo[conta++] = Integer.parseInt(t1.getText());
            t1.setText("");
        }
    }
}
```

```
    }  
  }  
  if (ae.getSource() == b2) {  
    ta1.setText("");  
    for (int i=0; i < conta; i++) {  
      ta1.append("" + arreglo[i] + "\n");  
    }  
  }  
  if (ae.getSource() == b3) {  
    conta = 0;  
    arreglo = new int[100];  
  }  
  if (ae.getSource() == b4) {  
    t1.setText("");  
    ta1.setText("");  
  }  
}  
}
```

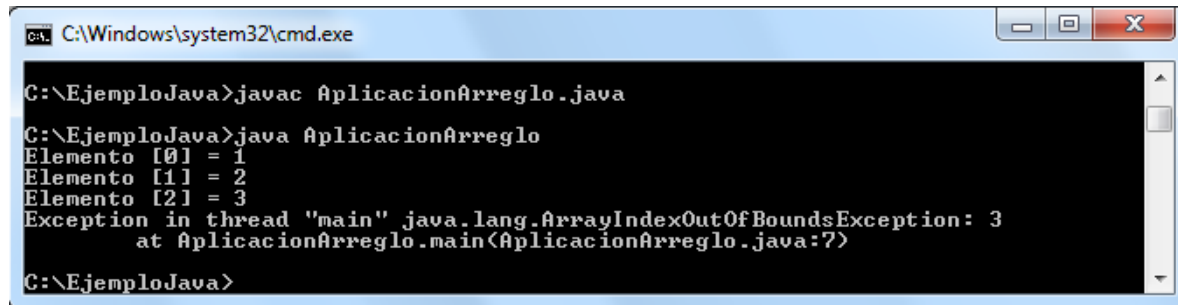


Usando mal los Índices

Cuando un subíndice está mal utilizado, haciendo referencia a un elemento en el arreglo que no existe, Java arroja un error de ejecución llamado de excepción (temas visto anteriormente), el cual es `ArrayIndexOutOfBoundsException`, se debe tener cuidado en esto, pues la aplicación se cancela y no continua, como se muestra en la siguiente aplicación:

```
public class AplicacionArreglo {  
  
    public static void main(String[] args) {  
        int arreglo[] = {1,2,3};
```

```
for (int i=0; i<arreglo.length+1; i++) {  
    System.out.println("Elemento [" + i + "] = " + arreglo[i]);  
}  
}
```



```
C:\Windows\system32\cmd.exe  
C:\EjemploJava>javac AplicacionArreglo.java  
C:\EjemploJava>java AplicacionArreglo  
Elemento [0] = 1  
Elemento [1] = 2  
Elemento [2] = 3  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3  
    at AplicacionArreglo.main(AplicacionArreglo.java:7)  
C:\EjemploJava>
```

Se puede ver como la aplicación alcanza a mostrar el último valor posible, pero cuando hace referencia al elemento 3 (ya que en la aplicación la condición es $i < \text{arreglo.length} + 1$, se sale del rango y arroja la excepción).

¿Que es un arreglo de dos dimensiones?

Un arreglo de dos dimensiones es una colección de datos para una misma variable en dos dimensiones comúnmente llamados renglones y columnas, también se conoce como matriz bidimensional.

Arreglo

12	-4	0	28	-3
-3	-5	2	189	-2
1	0	9	-4	12

Para poder guardar un valor u obtener alguno del arreglo de dos es ahora necesario utilizar dos dimensiones distribuidos en filas y columnas (muy similar a lo utilizado por Excel en sus hojas de trabajo)

Declaración de arreglos de dos dimensiones

Para declarar un arreglo de dos dimensiones se utiliza el siguiente formato:

tipo nombre_arreglo [][] = new tipo[numero renglones][numero columnas];

Donde tipo es el tipo de los datos que almacenará el arreglo de dos dimensiones. Es importante recordar que se pueden declarar arreglos de los tipos primitivos de Java (int, double, char, etc) o

bien de tipos definidos por el usuario (Cuenta, Alumno, etc), igual que los arreglos de una sola dimension.

El tamaño representa la cantidad de posiciones que se reservan para el arreglo. En Java todos los arreglos empiezan en el subíndice 0 y llegan al subíndice tamaño-1.

Por ejemplo:

```
int arr[][] = new int[5] [6]; // arreglo de 5 renglones y 6 columnas enteros
char cad[][] = new char[10] [5]; // arreglo de 10 renglones por 5 columnas tipo carácter
```

Uso e inicialización de los elementos de una Matriz

Para usar los elementos individuales de un arreglo se usa el siguiente formato:

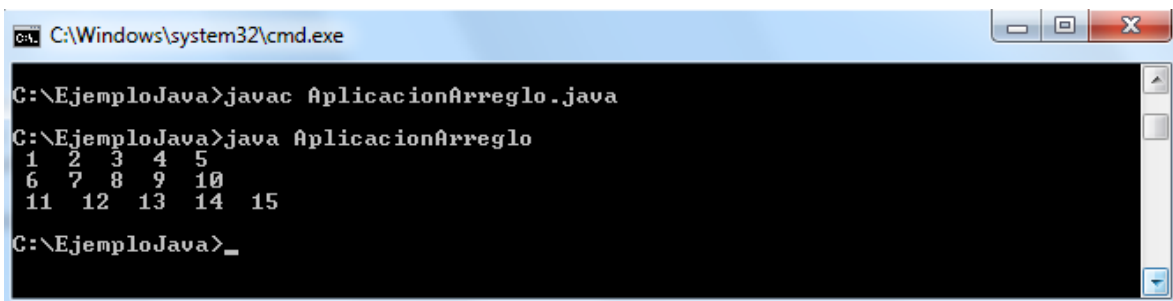
arreglo[subíndice-renglon] [subíndice-columna]

Como un elemento de un arreglo de dos dimensiones es también un dato, se puede usar como cualquier variable de ese tipo, debemos recordar que tanto el índice de renglón como el índice de columna toman como base el primer elemento cero:

```
int arr[][] = new int [2][5];
arr[3][4] = 12;
arr[1][0]= Integer.parseInt(t1.getText());
t2.setText("" + arr[0][1]);
arr[0][0] = arr[0][1] + arr[0][2];
int k = 2;
int l = 3
arr[k+1][l] = 20;
```

Ejemplo:

En este siguiente ejemplo, tenemos una aplicación que define un arreglo de enteros de dos dimensiones, con 3 renglones y 5 columnas, y los inicializa con el valor de 1 a 15, de acuerdo a cada renglón, empezando por 1 en el renglón 1, luego por 6 en el renglón 2 y 11 en el renglón 3. Después de inicializar la matriz, la despliega, desplegando los valores de un mismo renglón en la misma línea, como lo muestra la figura:



```
C:\Windows\system32\cmd.exe

C:\EjemploJava>javac AplicacionArreglo.java
C:\EjemploJava>java AplicacionArreglo
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
C:\EjemploJava>_
```


La aplicación es como se muestra a continuación:

```
public class AplicacionArreglo {

    public static void main(String[] args) {
        int arreglo[][] = new int [3][5];

        for (int i=0; i<3; i++) {
            for (int j=0; j<5; j++) {
                arreglo [i][j] = i*5+j+1;
            }
        }

        for (int i=0; i<3; i++) {
            for (int j=0; j<5; j++) {
                System.out.print(" " + arreglo[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Inicializar matriz con datos predefinidos

En Java es posible inicializar una matriz al declararla, tal y como sucede con los arreglos de una dimension; esto se hace sin definir el número de filas y columnas, colocando un operador de asignación y después entre llaves la lista de valores para cada renglón del arreglo, el arreglo separando con llaves para cada valor por columna, separado por comas, veamos los siguientes ejemplos:

```
double arreglo[][] = { {3.5, 5.4, -2.3 }, {22.3, 78.5, -9.4}};
```

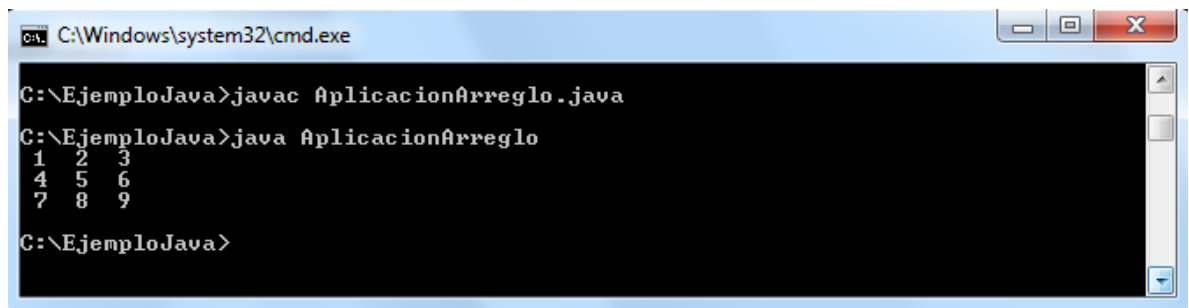
```
char cadena[][] = {{'a', 'g', 'u', 'a'}, {'r', 'o', 'j', 'a'}};
```

si se desea conocer el tamaño de una matriz sea el total de filas y/o columnas se utiliza la instrucción length.

```
public class AplicacionMatrices2 {

    public static void main(String[] args) {
        int arreglo[][] = {{1,2,3}, {4,5,6}, {7,8,9}};
```

```
for (int i=0; i<arreglo.length; i++) {  
    for (int j=0; j<arreglo[0].length; j++) {  
        System.out.print(" " + arreglo[i][j] + " ");  
    }  
    System.out.println();  
}  
}  
}
```



```
C:\Windows\system32\cmd.exe  
  
C:\EjemploJava>javac AplicacionArreglo.java  
C:\EjemploJava>java AplicacionArreglo  
1 2 3  
4 5 6  
7 8 9  
C:\EjemploJava>
```

Es importante observar como se escriben las filas por columna, recordando que se utiliza el print() para mostrar el valor, esto permitirá no cambiar de fila, pero tampoco saldrá a la pantalla hasta encontrar un println(), el cual se hace después del ciclo de adentro que despliega todos los renglones de una columna.

Esta instrucción println(), parte una línea similar a un enter para representar los datos como se ven en el ejemplo.

Ver Ejemplo que muestra el ingreso de los elementos por parte del usuario.

```
import java.io.*;
```

```
public class AplicacionMatrices3 {
```

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

```
        int arreglo[][] = new int[3][3];
```

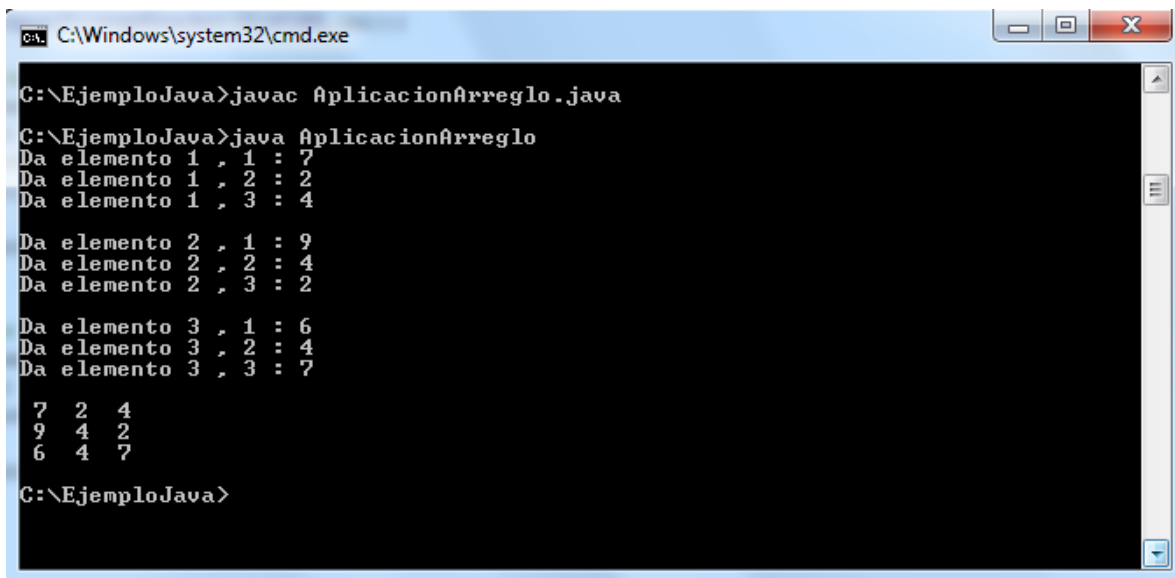
```
        // definiendo un objeto de entrada para tomar datos del teclado
```

```
        BufferedReader in =
```

```
        new BufferedReader(new InputStreamReader(System.in));
```

```
        // pidiendo los datos del teclado
```

```
for (int i=0; i<arreglo.length; i++) {  
    for (int j=0; j<arreglo[0].length; j++) {  
        System.out.print("Da elemento " + (i+1)+ " , " + (j+1) + " : ");  
        arreglo[i][j] = Integer.parseInt(in.readLine());  
    }  
    System.out.println();  
}  
  
//desplegando los valores por renglon  
for (int i=0; i<arreglo.length; i++) {  
    for (int j=0; j<arreglo[0].length; j++) {  
        System.out.print(" " + arreglo[i][j] + " ");  
    }  
    System.out.println();  
}  
}
```



```
C:\Windows\system32\cmd.exe  
  
C:\EjemploJava>javac AplicacionArreglo.java  
  
C:\EjemploJava>java AplicacionArreglo  
Da elemento 1 , 1 : 7  
Da elemento 1 , 2 : 2  
Da elemento 1 , 3 : 4  
  
Da elemento 2 , 1 : 9  
Da elemento 2 , 2 : 4  
Da elemento 2 , 3 : 2  
  
Da elemento 3 , 1 : 6  
Da elemento 3 , 2 : 4  
Da elemento 3 , 3 : 7  
  
7 2 4  
9 4 2  
6 4 7  
  
C:\EjemploJava>
```

Esta aplicación utiliza primero un ciclo anidado (que representa las filas) para pedir los datos, utilizando para ello el objeto `in`, solo debes utilizar el objeto tal y como esta declarado y cada vez que quieras leer algo del teclado utilizar `in.readLine()`, esto substituye al `t.getText()` de un applet, por eso tenemos

```
arreglo[i][j] = Integer.parseInt(in.readLine());
```

Ya que estamos tomando lo que ingreso por el teclado, se pasa a entero, ayudándonos por la clase Integer y finalmente se asigna esto en la matriz arreglo dentro del renglón (i-1) columna (j-1).

Usando mal los Índices

Al igual que en un arreglo de una sola dimensión o vector unidimensional, cuando utilizamos mal los índices de una matriz, Java arroja un error de ejecución llamado de excepción, el cual es `ArrayIndexOutOfBoundsException`, debemos tener cuidado en esto, pues la aplicación se cancela y no continua, como se muestra en la siguiente aplicación:

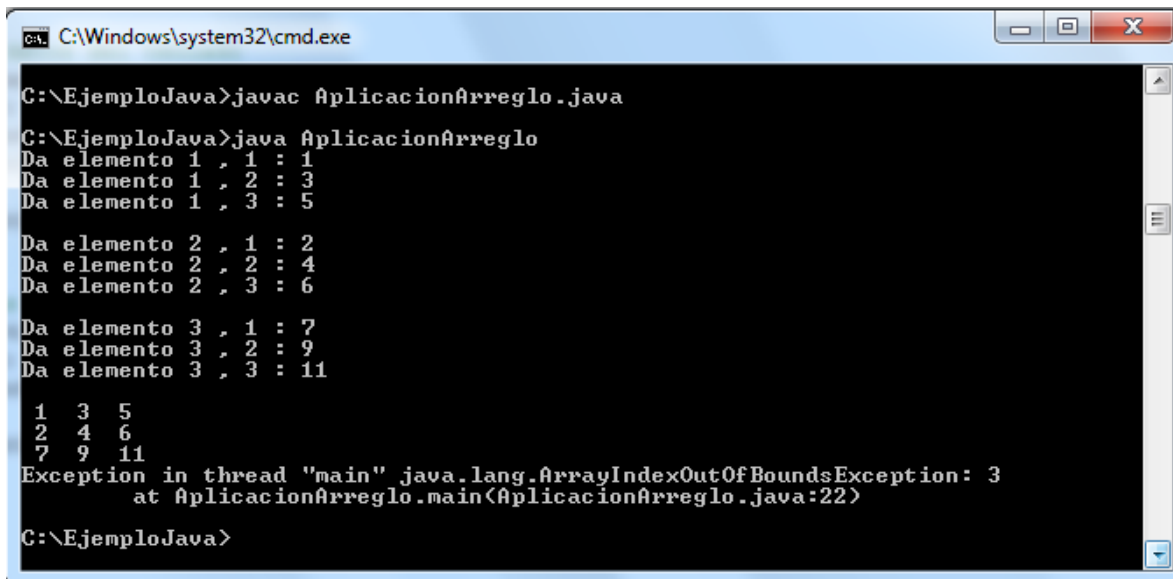
```
import java.io.*;

public class AplicacionMatrices4 {

    public static void main(String[] args) throws IOException {
        int arreglo[][] = new int[3][3];
        // definiendo un objeto de entrada para tomar datos del teclado
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

        // pidiendo los datos del teclado
        for (int i=0; i<arreglo.length; i++) {
            for (int j=0; j<arreglo[0].length; j++) {
                System.out.print("Da elemento " + (i+1)+ " , " + (j+1) + " : ");
                arreglo[i][j] = Integer.parseInt(in.readLine());
            }
            System.out.println();
        }

        //desplegando los valores por renglon
        for (int i=0; i<4; i++) {
            for (int j=0; j<3; j++) {
                System.out.print(" " + arreglo[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```



```
C:\Windows\system32\cmd.exe

C:\EjemploJava>javac AplicacionArreglo.java

C:\EjemploJava>java AplicacionArreglo
Da elemento 1 , 1 : 1
Da elemento 1 , 2 : 3
Da elemento 1 , 3 : 5

Da elemento 2 , 1 : 2
Da elemento 2 , 2 : 4
Da elemento 2 , 3 : 6

Da elemento 3 , 1 : 7
Da elemento 3 , 2 : 9
Da elemento 3 , 3 : 11

1 3 5
2 4 6
7 9 11
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
    at AplicacionArreglo.main(AplicacionArreglo.java:22)

C:\EjemploJava>
```

Podemos observar como la aplicación alcanza a mostrar todos los renglones desplegados, pero como queremos continuar con el renglón 3, el cual no existe pues el último fue 2 (empezando en cero), arroja la excepción `ArrayIndexOutOfBoundsException`.

Operaciones con Matrices

Utilizar una matriz para hacer operaciones con ella, en la mayoría de las veces implica el uso de la instrucción de ciclos anidados, sea mientras o while, haga mientras o do.. while, pero la mayoría de las veces por comodidad este se realiza con el ciclo Para o for, pues para poder tomar o actualizar cada elemento de la matriz, es necesario utilizar índice por filas y el índice por columnas, y es por esto que el for es la instrucción ideal, un for para la fila y otro para la columna.

Por ejemplo como lo vimos en el tema anterior, para inicializar una matriz ya una vez definida podemos utilizar

```
int arreglo[][] = new int [2][5];

for (int i=0; i<2; i++) {
    for (int j=0; j<5; j++) {
        arreglo [i][j] = i*5 + j + 1;
    }
}
```

Pero también podemos utilizar la variable `length`, la cual es definida para todo arreglo en Java, y esta representa el número máximo de filas que tiene utilizado el arreglo.

```
int arreglo[] = new int [2][5];

for (int i=0; i<arreglo.length; i++) {
```

```
        for (int j=0; j<arreglo[0].length; j++) {  
            arreglo [i][j] = i*arreglo[0].length + j + 1;  
        }  
    }
```

Al hacer operaciones con arreglos es muy común que utilicemos también una constante para definir el máximo valor a utilizar en la matriz, es decir para el ejemplo anterior quedaría como:

```
final int REN = 2;  
final int COL = 5;  
int arreglo[] = new int [REN][COL];  
  
for (int i=0; i<REN; i++) {  
    for (int j=0; j<COL; j++) {  
        arreglo [i][j] = i*COL + j + 1;  
    }  
}
```

Donde REN y COL son constantes (definidas así al usar la cláusula final) que valdrán 2 y 5 correspondientemente durante la ejecución del método, clase o parte donde se encuentre definida, esto nos ayuda a tener orden en los procesos aunque no siempre deben determinar las constantes, también se podría utilizar un valor fijo o manejar otro tipo de instrucción que represente la cantidad de filas y columnas.

Ejemplos

Buscando el elemento mayor de una matriz

Cuando deseamos obtener el valor mayor de todos los valores definidos en una matriz, debemos recorrer toda la matriz y utilizar una variable que nos ayude en esta comparación. La mejor manera de inicializar esta variable es utilizar el primer valor de la matriz, por ejemplo (asumiendo que la matriz ya tiene valores):

```
int mayor = arreglo[0][0]; // se toma el primer valor como el mayor  
// se revisa cada elemento en la matriz  
for (int i=0; i < arreglo.length; i++) {  
    for (int j=0; j<arreglo[0].length; j++) {  
        // si el elemento de la matriz es mayor  
        if (arreglo[i][j] > mayor) {  
            // cambiamos el valor del mayor  
            mayor = arreglo[i][j];  
        }  
    }  
}
```

```
}  
    System.out.println("El valor mayor es " + mayor);
```

Tomando el índice en el que se encuentra este elemento

Para hacer esto definimos otra variable, la cual debe empezar con 0 y si el valor de la matriz es mayor, además de hacer el cambio de mayor, actualizamos la posición del renglón y la columna de donde se encontró el mayor, el ejemplo quedaría como sigue (asumiendo que la matriz ya tiene valores):

```
int posiconi = 0;  
int posiconj = 0;  
int mayor = arreglo[0][0]; // se toma el primer valor como el mayor  
// se revisa cada elemento en la matriz desde el segundo  
for (int i=0; i < arreglo.length; i++) {  
    for (int j=0; j<arreglo[0].length; j++) {  
        // si el elemento de la matriz es mayor  
        if (arreglo[i][j] > mayor) {  
            // cambiamos el valor del mayor  
            mayor = arreglo[i][j];  
            posiconi = i;  
            posiconj = j;  
        }  
    }  
}  
  
System.out.println("El valor mayor es " + mayor);  
System.out.println("Y esta en el renglón " + (posiconi +1));  
System.out.println("columna " + (posiconj +1));
```

Si queremos saber en que posición se encontró el mayor valor, entonces utilizamos dos variables, como se observó en la aplicación, una para la posición de la fila (posiconi) y otra para la posición de la columna (posiconj).

Buscando el Menor de una matriz y su posición

Para obtener el valor menor, solo se cambia la comparación y en lugar de comparar contra mayor, solo se compara contra menor, el ejemplo lo vemos como sigue (asumiendo que la matriz ya tiene valores):

```
int posiconi = 0;
```

```
int posicionj = 0;
int menor = arreglo[0][0]; // se toma el primer valor como el mayor
// se revisa cada elemento en la matriz desde el segundo
for (int i=0; i < arreglo.length; i++) {
    for (int j=0; j<arreglo[0].length; j++) {
        // si el elemento de la matriz es menor
        if (arreglo[i][j] < menor) {
            // cambiamos el valor del menor
            menor = arreglo[i][j];
            posicioni = i;
            posicionj = j;
        }
    }
}

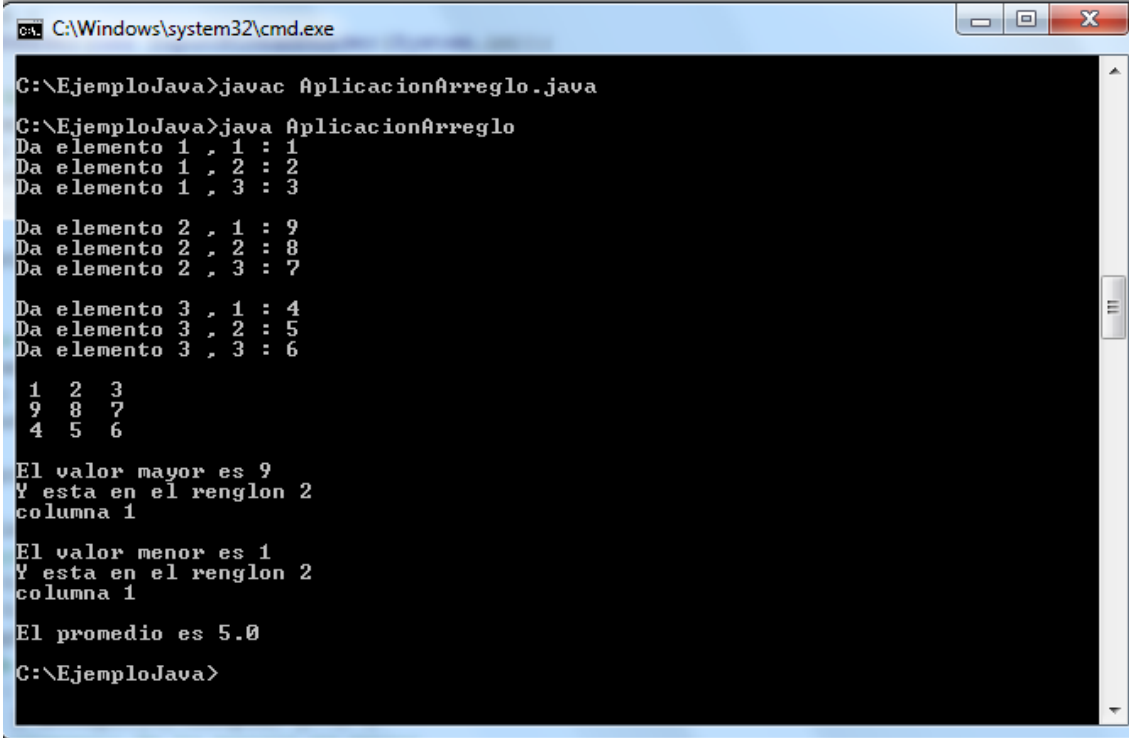
System.out.println("El valor menor es " + menor);
System.out.println("Y esta en el renglón " + (posicioni + 1));
System.out.println("columna " + (posicionj + 1));
```

Calculando promedio de la matriz

Para obtener el promedio de una matriz, se debe de sumar los elementos y dividir entre cuantos sean, el ejemplo lo vemos como sigue (asumiendo que la matriz ya tiene valores):

```
double promedio;
double suma = 0; // se inicializa la suma en cero
// se tomara cada elemento para sumarlo
for (int i=0; i < arreglo.length; i++) {
    for (int j=0; j<arreglo[0].length; j++) {
        suma += arreglo[i][j];
    }
}

promedio = suma / arreglo.length;
System.out.println("El promedio es " + promedio);
```

```
C:\Windows\system32\cmd.exe

C:\EjemploJava>javac AplicacionArreglo.java

C:\EjemploJava>java AplicacionArreglo
Da elemento 1 , 1 : 1
Da elemento 1 , 2 : 2
Da elemento 1 , 3 : 3

Da elemento 2 , 1 : 9
Da elemento 2 , 2 : 8
Da elemento 2 , 3 : 7

Da elemento 3 , 1 : 4
Da elemento 3 , 2 : 5
Da elemento 3 , 3 : 6

 1  2  3
 9  8  7
 4  5  6

El valor mayor es 9
Y esta en el renglon 2
columna 1

El valor menor es 1
Y esta en el renglon 2
columna 1

El promedio es 5.0

C:\EjemploJava>
```

```
import java.io.*;

public class AplicacionArreglo {
    public static void main(String[] args) throws IOException {
        int arreglo[][] = new int[3][3];
        // definiendo un objeto de entrada para tomar datos del teclado
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
        // pidiendo los datos del teclado
        for (int i=0; i<arreglo.length; i++) {
            for (int j=0; j<arreglo[0].length; j++) {
                System.out.print("Da elemento " + (i+1)+ " , " + (j+1) + " : ");
                arreglo[i][j] = Integer.parseInt(in.readLine());
            }
            System.out.println();
        }
        //desplegando los valores por renglon
        for (int i=0; i<arreglo.length; i++) {
            for (int j=0; j<arreglo[0].length; j++) {
                System.out.print(" " + arreglo[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

```
}  
System.out.println();  
// sacando el mayor  
int posiconi = 0;  
int posiconj = 0;  
int mayor = arreglo[0][0]; // se toma el primer valor como el mayor  
// se revisa cada elemento en la matriz desde el segundo  
for (int i=0; i < arreglo.length; i++) {  
    for (int j=0; j<arreglo[0].length; j++) {  
        // si el elemento de la matriz es mayor  
        if (arreglo[i][j] > mayor) {  
            // cambiamos el valor del mayor  
            mayor = arreglo[i][j];  
            posiconi = i;  
            posiconj = j;  
        }  
    }  
}  
  
System.out.println("El valor mayor es " + mayor);  
System.out.println("Y esta en el renglon " + (posiconi +1));  
System.out.println("columna " + (posiconj +1));  
System.out.println();  
  
//sacando el menor  
  
int menor = arreglo[0][0]; // se toma el primer valor como el mayor  
// se revisa cada elemento en la matriz desde el segundo  
for (int i=0; i < arreglo.length; i++) {  
    for (int j=0; j<arreglo[0].length; j++) {  
        // si el elemento de la matriz es mayor  
        if (arreglo[i][j] < menor) {  
            // cambiamos el valor del mayor  
            menor = arreglo[i][j];  
            posiconi = i;  
            posiconj = j;  
        }  
    }  
}  
  
System.out.println("El valor menor es " + menor);  
System.out.println("Y esta en el renglon " + (posiconi +1));
```

```
        System.out.println("columna " + (posicionj +1));  
        System.out.println();  
  
        //obteniendo el promedio de la matriz  
        double promedio;  
        double suma = 0; // se inicializa la suma en cero  
        // se tomara cada elemento para sumarlo  
        for (int i=0; i < arreglo.length; i++) {  
            for (int j=0; j<arreglo[0].length; j++) {  
                suma += arreglo[i][j];  
            }  
        }  
        promedio = suma / (arreglo.length * arreglo[0].length);  
        System.out.println("El promedio es " + promedio);  
    }  
}
```

EJERCICIOS DE AUTOEVALUACIÓN

1. Crear un arreglo unidimensional y determinar cual es elemento mayor y el menor con sus respectivas posiciones
2. Crear una matriz y determinar cuales son los elementos de su diagonal secundaria

Prueba Final

Crear una matriz cuadrada, y crear un algoritmo que permita ordenarla en forma ascendente.

Actividad

Escribir una aplicación llamado CuadradoMagico que pida los valores del teclado de una matriz en orden n por n, y que evalúe si esta matriz es un cuadrado mágico. Los cuadrados mágico es aquella matriz donde la suma de todos los filas y la suma de todas las columnas y la suma de las diagonales dan el mismo valor.

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

5. APPLETS AWT Y SWING

OBJETIVO GENERAL

Conocer las diferentes herramientas que brinda el lenguaje para fortalecer las opciones graficas de programación, determinado así una nueva etapa del desarrollo lógico y el buen funcionamiento, agradable y cercano al usuario final.

OBJETIVOS ESPECÍFICOS

- ◆ Conocer las características y componentes básicos de los applets
- ◆ Desarrollar aplicaciones de ambiente grafico con sus componentes mas comunes
- ◆ Dar un paso adelante en el ambiente gráficos desarrollando aplicativos con la tecnología swing

Prueba Inicial

Desarrollar un aplicativo con los conceptos previamente vistos que relacionen conceptos de una nomina, esta en la programación tradicional.

5.1. Applets

5.1.1. Definición

Un applet es una clase que posee interfaz gráfica y que está inmerso en un navegador, de manera que para poderlo utilizar requieres ejecutar el navegador o el visualizador de applets de Java.

Un applet esta compuesto por varios métodos que utilizamos de acuerdo a lo que deseamos realizar en nuestra aplicación en Web.

Existe una clase de java muy peculiar llamada Applet, esta clase se encarga de mostrar una aplicación en Web, esta clase ya esta previamente definida, y para poder hacer un applet es necesario que se defina una clase que herede de la clase Applet, de manera que el usuario solo se encarga de reescribir los métodos necesarios para definir su nueva aplicación en Web.

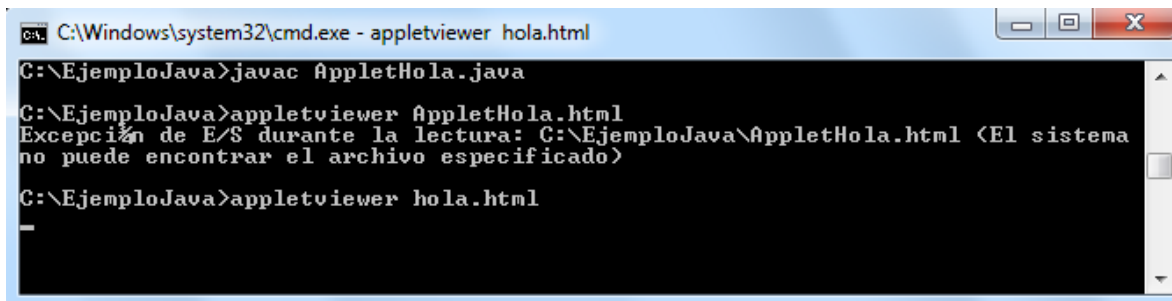
5.1.2. Crear un applet

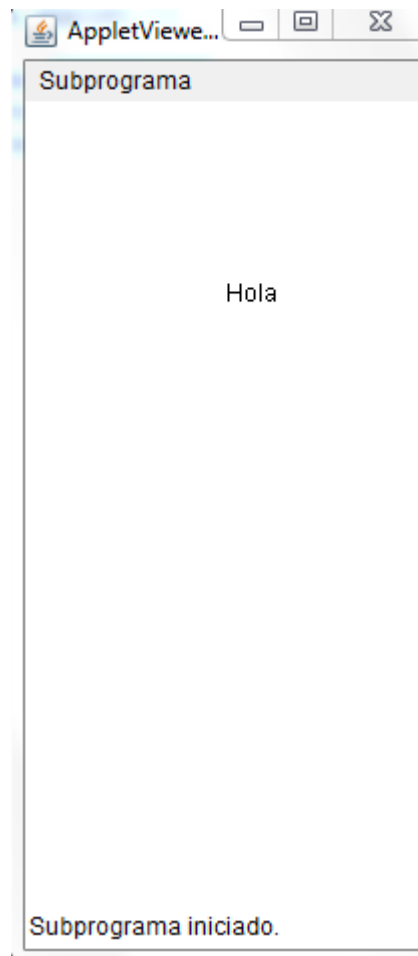
Veamos un ejemplo de un applet sencillo Hola y analicemos este applet (AppletHola.java):

```
import java.awt.*;  
import java.applet.*;  
public class AppletHola extends Applet {  
    public void paint(Graphics g) {  
        g.drawString("Hola", 100, 100);  
    }  
}
```

La manera de visualizar este applet es a través de una página de Web, como la siguiente (AppletHola.html):

```
<html>  
  <head>  
    <title>Applet Hola </title>  
  </head>  
  <body>  
    <applet width="200" height="400" code="AppletHola"></applet>  
  </body>  
</html>
```





A diferencia de una aplicación, un applet debe heredar de la clase Applet, en ella existen algunos métodos predefinidos a través de los cuales vamos a dibujar o desplegar los elementos gráficos en nuestra ventana, es por esto que utilizamos el método paint, en este método desplegamos lo que deseamos aparezca en la pantalla, y eso es a través de un objeto de la clase Graphics, entonces ahora entendemos que solo a través de este objeto es que podemos dibujar.

Normalmente se define el objeto de la clase Graphics como g, pero puede ser que tu uses otro, dentro del método paint, la manera de dibujar es utilizando los métodos de la clase Graphics, en este caso para dibujar letras podemos usar el método drawString(), el cual requiere se le defina un string (cadena de caracteres) y la coordenada en la cual se desea dibujar, empezando por la coordenada x y siguiendo la coordenada y.

5.1.3. Ciclo de vida de un applet

El código que introduzcamos en nuestro applet debe responder a los sucesos propios del ciclo de vida de la misma. Para ello, debemos redefinir al menos una parte de los métodos de la clase Applet que reflejan las fases de este ciclo de vida y que son los siguiente

Init(): este método es invocado por el browser inmediatamente después de la carga del applet, y en el hay que colocar las inicialización de variables que vayan a permancer durante todo el ciclo de vida, ya que no vuelve a ser llamado posteriormente.

Start(): este método es llamado tanto la primera vez que se carga el applet como todas aquellas veces en que el browser vuelva a presentar el documento HTML que hace referencia del applet, después de haber presentado otros diferentes.

Paint(Graphics contexto): el browser invoca este método cada vez que es necesario el repintado del area de trabajo del applet. El argumento, un objeto de tipo graphics, ofrece métodos que permiten dibujar en esta area.

Update(Graphics contexto): el método update () definido en la clase applet se limita al rellenar el area de trabajo con el color de fondo por defecto y después llama a paint() con el mismo argumento. Para evitar parpadeos debido a este relleno, es conveniente redefinir ambos métodos, incluir todo el código de repintado en update() y hacer el método paint() no haga mas que llamar a update().

Stop(): este método es llamado por el browser cada vez que el usuario del mismo abandona el documento HTML que hace referencia al applet. Aquí puede ser interesante parar todos los threads que se dediquen a visualizar información en pantalla y a otros trabajos relacionados.

Destroy(): este método es llamado cuando por cualquier circunstancia va a concluir definitivamente la ejecución del applet y esta va a ser descargada de la memoria. Se debe incluir aquí el código de limpieza necesario para que la ejecución termine ordenadamente. Cierre de sockets, de url.etc.

5.2. Interfaces Gráficas AWT (SWT) Swing

5.2.1. El awt

El awt es un componente de interfaz de usuario, se ha trabajado básicamente en características comunes a todos los sistemas de ventanas más populares como Windows, Macintosh, Solaris, Linux, entre otros. Aunque no se ha aprovechado al 100% las características de este, nos permite un acercamiento a las características que estos sistemas prestan y un manejo más intuitivo para el usuario final.

5.2.1.1 Componentes del AWT

Dentro de los componentes más comunes del awt se encuentran los siguientes:

Label

La clase label es una clase derivada de canvas que sabe, además dibujar un texto dentro de las fronteras delimitadas por su representación gráfica. Se puede especificar tanto el texto como el alineamiento dentro del rectángulo por medio de los parámetros del constructor. Para esto último existen 3 constantes, label.LEFT, label.RIGHT, label.CENTER, que sitúan el texto, respectivamente, ajustado a la izquierda, derecha o centrado.

Button

Este componente gráfico que permite invocar una cierta acción cuando el usuario pulsa el botón del ratón manteniendo el cursor del mismo encima del botón. La representación gráfica varía según el sistema de ventanas sobre el que se trabaje.

Los botones llevan asociada una etiqueta, que se pasa como un objeto del tipo string, y cuyo texto va necesariamente centrado dentro de la presentación gráfica del botón. La etiqueta se le asigna al botón en el momento de su creación como un parámetro del constructor.

Checkbox y checkboxgroup

Los componentes de este tipo checkbox están compuestos por un pequeño recuadro, que puede contener o no una marca, y una etiqueta identificativa. Habitualmente se emplea para selecciones de tipo sí/no o activo/inactivo. El usuario puede cambiar de un estado a otro haciendo click con el ratón encima del recuadro de marca.

Choice

La clase choice define un tipo de componente grafico que permite presentar una lista de selecciones desplegable. Por defecto, se presenta en un recuadro una de las opciones. Un botón a la derecha de este recuadro permite desplegar la lista completa. Una vez escogida una de las opciones con el raton o el teclado, la lista desaparece y la nueva opción se presenta en el recuadro del texto.

Lista

Esta clase define un tipo de componente grafico que permite presentar a un tiempo varias opciones, con la posibilidad de seleccionar una o varias de ellas, según el tipo de la lista. Se corresponde casi exactamente con las list boxes de Windows. El que la lista sea de selección simple o multiple viene dado por el segundo argumento del constructor que toma, respectivamente, los valores false o true. El primer argumento determina cuantos elementos van a ser visibles al mismo tiempo.

TextArea

Este componente permite editar varias líneas de texto llano. El constructor de textarea permite especificar tanto el texto inicial que se va a introducir en el componente como el número de filas y columnas que va a admitir este. Una vez creado el componentes, el usuario puede editar el contenido como desee. Al igual que con textfield, este contenido se puede recuperar con el método toString(), heredado de la clase component.

5.2.1.2 Swing

Swing existe desde la JDK 1.1 (como un componente agregado). Antes de la existencia de Swing, las interfaces gráficas con el usuario se realizaban a través de AWT (Abstract Window Toolkit). Normalmente, para toda componente AWT existe una componente Swing que la reemplaza, por ejemplo, la clase Button de AWT es remplazada por la clase JButton de Swing.

Las componentes de Swing utilizan la infraestructura de AWT, incluyendo el modelo de eventos AWT, el cual rige cómo una componente reacciona a eventos tales como, eventos de teclado, mouse, etc... Es por esto, que la mayoría de los programas Swing, dentro de los compontes más usuales encontramos.

JLabel

Empezemos por uno de los elementos gráficos más sencillos, el JLabel, pero ahora utilicemos una aplicación gráfica utilizando extendí JFrame, veamos como quedaría una aplicación muy sencilla:

```
import javax.swing.*;

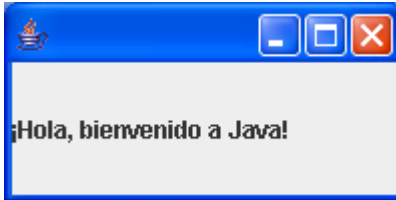
public class AplicacionSwing1 extends JFrame {

    public static void main( String argv[] ) {
        AplicacionSwing1 app = new AplicacionSwing1();
    }

    public AplicacionSwing1() {
        JLabel hola = new JLabel( "¡Hola, bienvenido a Java!" );

        getContentPane().add( hola,"Center" );
        setSize( 200,100);
        setVisible( true );
    }
}
```

La ejecución de esta aplicación sería:



En la pantalla se ve una figura muy similar a un applet, con la diferencia de que podemos ejecutarlo sin un archivo HTML asociado a él.

Este tipo de aplicativos (no applet), requiere que utilicemos una instrucción main para crear el objeto a ejecutar.

```
public static void main( String argv[] ) {
    AplicacionSwing1 app = new AplicacionSwing1();
}

public AplicacionSwing1() {
    JLabel hola = new JLabel( "¡Hola, bienvenido a Java!" );

    getContentPane().add( hola,"Center" );
    setSize( 200,100);
}
```

```
setVisible( true );  
}
```

El JLabel es la clase que nos permite tener más opciones que el Label normal del paquete awt.

Constructor Summary	
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(Icon image)	Creates a JLabel instance with the specified image.
JLabel(Icon image, int horizontalAlignment)	Creates a JLabel instance with the specified image and horizontal alignment.
JLabel(String text)	Creates a JLabel instance with the specified text.
JLabel(String text, Icon icon, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.
JLabel(String text, int horizontalAlignment)	Creates a JLabel instance with the specified text and horizontal alignment.

Una variación de esta aplicación es utilizar el constructor con alineación:

```
import javax.swing.*;
```

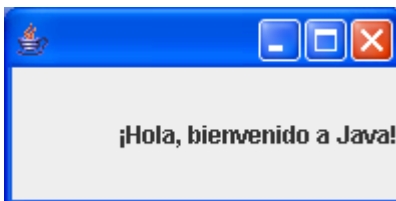
```
public class AplicacionSwing1 extends JFrame {
```

```
    public static void main( String argv[] ) {  
        AplicacionSwing1 app = new AplicacionSwing1();  
    }
```

```
    public AplicacionSwing1() {  
        JLabel hola = new JLabel( "¡Hola, bienvenido a Java!",JLabel.RIGHT);
```

```
        getContentPane().add( hola,"Center" );  
        setSize( 200,100);  
        setVisible( true );  
    }
```

```
}
```



Ahora si queremos utilizar la misma aplicación, pero con un icono, podemos tener la siguiente aplicación gráfica:

```
import java.awt.*;
import javax.swing.*;

public class AplicacionSwing2 extends JFrame {

    public static void main( String argv[] ) {
        AplicacionSwing2 app = new AplicacionSwing2();
        app.setSize( 300,150 );
        app.setVisible( true );
    }

    public AplicacionSwing2() {
        setLayout(new GridLayout(1,1,5,5));
        Icon icon = new ImageIcon("ejercicio.gif");
        JLabel hola = new JLabel( icon, JLabel.CENTER);
        add( hola );
    }
}
```

La cual despliega el siguiente Frame:



Donde el icono "ejercicio.gif" aparece centrado.

JButton, JToggleButton, JCheckBox, JRadioButton

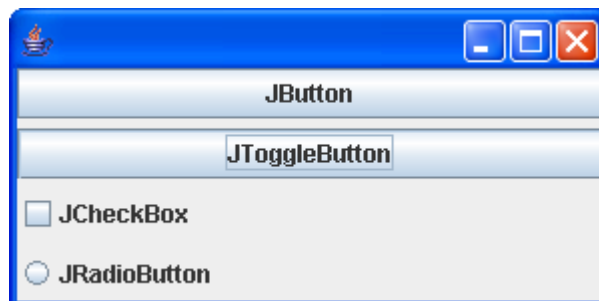
El JButton al igual que el Button le permite al usuario dar acciones para la ejecución de algún grupo de instrucciones. Veamos una aplicación que tiene varios tipos de botones:

```
import java.awt.*;
import javax.swing.*;

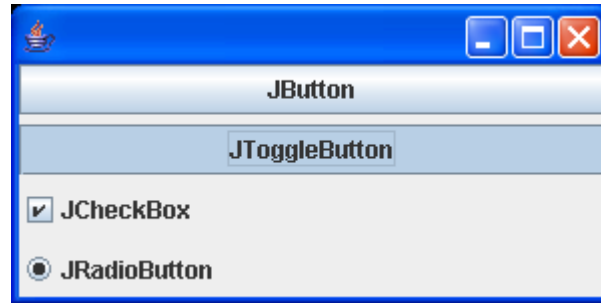
public class AplicacionSwing3 extends JFrame {

    public static void main( String argv[] ) {
        AplicacionSwing3 app = new AplicacionSwing3();
        app.setSize( 300,150 );
        app.setVisible( true );
    }

    public AplicacionSwing3() {
        setLayout(new GridLayout(4,1,5,5));
        JButton boton1 = new JButton("JButton");
        JToggleButton boton2 = new JToggleButton("JToggleButton");
        JCheckBox boton3 = new JCheckBox("JCheckBox");
        JRadioButton boton4 = new JRadioButton("JRadioButton");
        add(boton1);
        add(boton2);
        add(boton3);
        add(boton4);
    }
}
```



Al seleccionar el JToggleButton se sombre, el JCheckBox se selecciona con una marca \surd y el JRadioButton se selecciona con un punto adentro.



Constructor Summary

JButton()

Creates a button with no set text or icon.

JButton(Action a)

Creates a button where properties are taken from the Action supplied.

JButton(Icon icon)

Creates a button with an icon.

JButton(String text)

Creates a button with text.

JButton(String text, Icon icon)

Creates a button with initial text and an icon.

Utilizando el constructor que tiene el icono como parámetro la aplicación quedaría así:

```
import java.awt.*;
import javax.swing.*;

public class AplicacionSwing3 extends JFrame {

    public static void main( String argv[] ) {
        AplicacionSwing3 app = new AplicacionSwing3();
        app.setSize( 300,150 );
        app.setVisible( true );
    }

    public AplicacionSwing3() {
        setLayout(new GridLayout(4,1,5,5));
```

```
Icon icon = new ImageIcon("star0.gif");
JButton boton1 = new JButton("JButton", icon);
JToggleButton boton2 = new JToggleButton("JToggleButton");
JCheckBox boton3 = new JCheckBox("JCheckBox");
JRadioButton boton4 = new JRadioButton("JRadioButton");
add(boton1);
add(boton2);
add(boton3);
add(boton4);
}
}
```

Y la ejecución de la aplicación se vería así:



La manera en la que se realizan las operaciones para el botón en esta aplicación gráfica es igual que con el applet, hay que darle a cada botón la facilidad de que sea utilizado con el `addActionListener` e implementar también esta clase y utilizar el `actionPerformed` para ejecutar las instrucciones, estas con el fin de que entienda que el usuario selecciono con el raton y haga la tarea programada para ello.

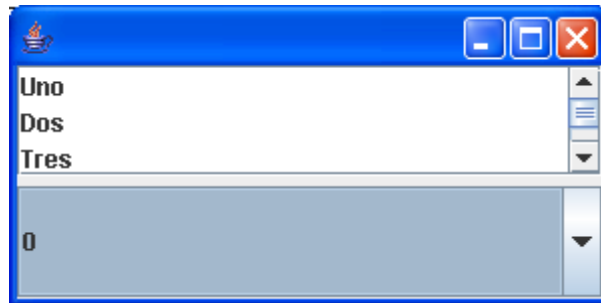
JList, JCombo

Las listas y combos en Swing funcionan del mismo modo que lo hacían en el AWT, aunque tienen incrementada la funcionalidad a través de algunas funciones de conveniencia que se han incorporado. Por ejemplo, **JList** tiene un constructor al que se puede pasar un array de objetos `String` para que los presente, un ejemplo es la siguiente aplicación:

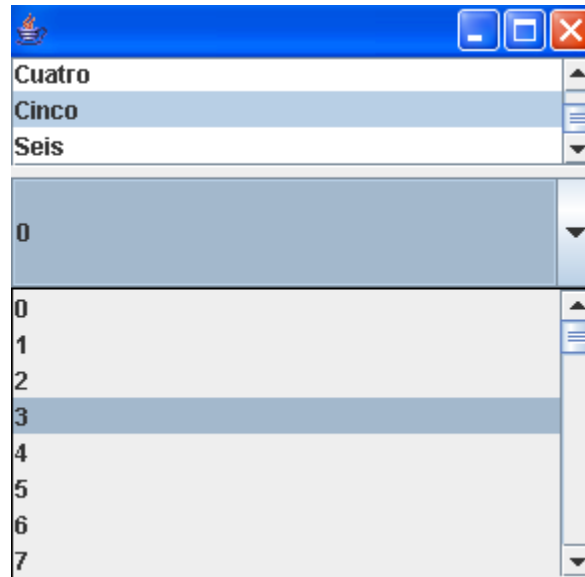
```
import java.awt.*;
import javax.swing.*;

public class AplicacionSwing4 extends JFrame {
```

```
private static String datos[] = {  
    "Uno", "Dos", "Tres",  
    "Cuatro", "Cinco", "Seis",  
};  
public static void main( String argv[] ) {  
    AplicacionSwing4 app = new AplicacionSwing4();  
    app.setSize( 300,150 );  
    app.setVisible( true );  
  
}  
  
public AplicacionSwing4() {  
    setLayout(new GridLayout(2,1,5,5));  
    JList lista = new JList( datos );  
    add( new JScrollPane( lista ) );  
    JComboBox combo = new JComboBox();  
    for( int i=0; i < 100; i++ )  
        combo.addItem( Integer.toString( i ) );  
    add( combo );  
}  
}
```



Donde al seleccionar la barra deslizador del objeto JList podemos seleccionar alguno de los elementos de la lista, igual podemos hacer con el objeto de la clase Combo:



Aplicaciones con Java Swing

En Java Swing se puede utilizar la clase `JMenuBar` para poder crear barras de menús y dar opciones a ejecutar, haciendo así más amplia la gama de opciones de la aplicación construida.

Constructor Summary

[`JMenuBar\(\)`](#)

Creates a new menu bar.

Method Summary

<code>JMenu</code>	<code>add(JMenu c)</code> Appends the specified menu to the end of the menu bar.
<code>void</code>	<code>addNotify()</code> Overrides <code>JComponent.addNotify</code> to register this menu bar with the current keyboard manager.
<code>AccessibleContext</code>	<code>getAccessibleContext()</code> Gets the <code>AccessibleContext</code> associated with this <code>JMenuBar</code> .
<code>Component</code>	<code>getComponent()</code> Implemented to be a <code>MenuElement</code> .
<code>Component</code>	<code>getComponentAtIndex(int i)</code> Deprecated. <i>replaced by <code>getComponent(int i)</code></i>
<code>int</code>	<code>getComponentIndex(Component c)</code> Returns the index of the specified component.
<code>JMenu</code>	<code>getHelpMenu()</code> Gets the help menu for the menu bar.

El método add es comúnmente utilizado para añadir elementos del objeto JMenu, que sería cada menú a aparecer. El JMenu tiene los siguientes constructores y algunos de los métodos utilizados:

Constructor Summary

[JMenu](#)()

Constructs a new JMenu with no text.

[JMenu](#)([Action](#) a)

Constructs a menu whose properties are taken from the [Action](#) supplied.

[JMenu](#)([String](#) s)

Constructs a new JMenu with the supplied string as its text.

[JMenu](#)([String](#) s, boolean b)

Constructs a new JMenu with the supplied string as its text and specified as a tear-off menu or not.

Method Summary

JMenuItem	add (Action a) Creates a new menu item attached to the specified Action object and appends it to the end of this menu.
Component	add (Component c) Appends a component to the end of this menu.
Component	add (Component c, int index) Adds the specified component to this container at the given position.
JMenuItem	add (JMenuItem menuItem) Appends a menu item to the end of this menu.

Constructor Summary

[`JTextPane\(\)`](#)

Creates a new JTextPane.

[`JTextPane\(StyledDocument doc\)`](#)

Creates a new JTextPane, with a specified document model.

Method Summary

<code>Style</code>	<code>addStyle(String nm, Style parent)</code> Adds a new style into the logical style hierarchy.
protected <code>EditorKit</code>	<code>createDefaultEditorKit()</code> Creates the EditorKit to use by default.
<code>AttributeSet</code>	<code>getCharacterAttributes()</code> Fetches the character attributes in effect at the current location of the caret, or null.
<code>MutableAttributeSet</code>	<code>getInputAttributes()</code> Gets the input attributes for the pane.
<code>Style</code>	<code>getLogicalStyle()</code> Fetches the logical style assigned to the paragraph represented by the current position of the caret, or null.
<code>AttributeSet</code>	<code>getParagraphAttributes()</code> Fetches the current paragraph attributes in effect at the location of the caret, or null if none.

Un ejemplo simple que aplica los conceptos anteriores es:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;

public class AplicacionSwing5 extends JPanel implements ActionListener {
    private Style estiloMorado,estiloGris,estiloCeleste,estiloRojo,estiloAzul;
    private JTextPane texto;

    public AplicacionSwing5() {
        setLayout( new BorderLayout() );
        JMenuBar menu = new JMenuBar();
        JMenu estilo = new JMenu( "Estilo" );
        menu.add( estilo );

        JMenuItem mi = new JMenuItem( "Morado" );
        estilo.add( mi );
        mi.addActionListener(this);
```

```
mi = new JMenuItem( "Gris" );
estilo.add( mi );
mi.addActionListener(this);
mi = new JMenuItem( "Celeste" );
estilo.add( mi );
mi.addActionListener(this);
mi = new JMenuItem( "Rojo" );
estilo.add( mi );
mi.addActionListener(this);
    mi = new JMenuItem( "Azul" );
estilo.add( mi );
mi.addActionListener( this );
add( menu,BorderLayout.NORTH );

StyleContext sc = new StyleContext();
estiloMorado = sc.addStyle( null,null );
StyleConstants.setForeground( estiloMorado,Color.magenta );
estiloGris = sc.addStyle( null,null );
StyleConstants.setForeground( estiloGris,Color.gray );
StyleConstants.setFontSize( estiloGris,24 );
estiloCeleste = sc.addStyle( null,null );
StyleConstants.setForeground( estiloCeleste,Color.cyan );
estiloRojo = sc.addStyle( null,null );
StyleConstants.setForeground( estiloRojo,Color.red );
estiloAzul = sc.addStyle( null,null );
StyleConstants.setForeground( estiloAzul,Color.blue );

DefaultStyledDocument doc = new DefaultStyledDocument(sc);

JTextPane texto = new JTextPane(doc);
add( texto,BorderLayout.CENTER );
}

public void actionPerformed( ActionEvent e ) {
    Style estilo = null;
    String color = (String) e.getActionCommand();

    if( color.equals( "Morado" ) ) {
        estilo = estiloMorado;
    } else if( color.equals( "Celeste" ) ) {
```

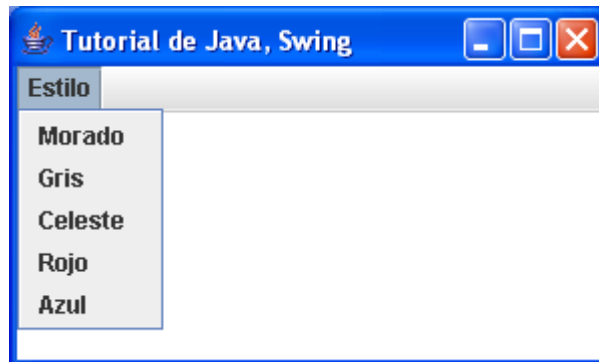
```
        estilo = estiloCeleste;
    } else if( color.equals( "Gris" ) ) {
        estilo = estiloGris;
    } else if( color.equals( "Rojo" ) ) {
        estilo = estiloRojo;
    } else if( color.equals( "Azul" ) ) {
        estilo = estiloAzul;
    }
    texto.setCharacterAttributes( estilo,false );
}

public static void main( String argv[] ) {
    JFrame app = new JFrame( "Tutorial de Java, Swing" );

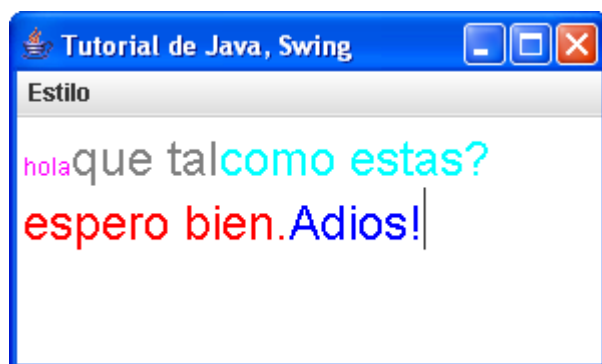
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent evt ){
            System.exit( 0 );
        }
    });
    app.getContentPane().add( new AplicacionSwing5(),BorderLayout.CENTER );
    app.setSize( 300,180 );

    app.setVisible( true );
}
}
```

La ejecución de la aplicación sería:



Y una vez utilizadas las opciones del menú:



JTable

JTable es la clase en principio, se creó para constituir un interfaz ligado a bases de datos a través del Java Database Connectivity (JDBC), y así evitar la complejidad inherente al manejo de los datos, proporcionando mucha flexibilidad al programador.

Hay suficientes características como para montar desde una hoja de cálculo básica hasta información para escribir un texto completo.

Constructor Summary	
JTable ()	Constructs a default JTable that is initialized with a default data model, a default column model, and a default selection model.
JTable (int numRows, int numColumns)	Constructs a JTable with numRows and numColumns of empty cells using DefaultTableModel.
JTable (Object [] [] rowData, Object [] columnNames)	Constructs a JTable to display the values in the two dimensional array, rowData, with column names, columnNames.
JTable (TableModel dm)	Constructs a JTable that is initialized with dm as the data model, a default column model, and a default selection model.
JTable (TableModel dm, TableColumnModel cm)	Constructs a JTable that is initialized with dm as the data model, cm as the column model, and a default selection model.
JTable (TableModel dm, TableColumnModel cm, ListSelectionModel sm)	Constructs a JTable that is initialized with dm as the data model, cm as the column model, and sm as the selection model.
JTable (Vector rowData, Vector columnNames)	Constructs a JTable to display the values in the Vector of Vectors, rowData, with column names, columnNames.

Una manera sencilla de usar la JTable es tomando ayuda de la TableModel, que nos ayuda a modelar la tabla, y por otro lado la clase JScrollPane permite añadirle un deslizador a ambas partes de la tabla tanto horizontal como vertical, un ejemplo sencillo de cómo se manejaría esto sería crear el objeto dataModel como se ve a continuación (se crea definiendo una clase Interna dentro de la clase misma “AbstractTableModel”).

```
TableModel dataModel = new AbstractTableModel() {  
    public int getColumnCount() {  
        return 10;  
    }  
    public int getRowCount() {  
        return 10;  
    }  
    public Object getValueAt(int row, int col) {  
        return new Integer(row*col);  
    }  
};  
JTable table = new JTable(dataModel);  
JScrollPane scrollpane = new JScrollPane(table);
```

Un ejemplo de aplicación gráfica que genera una JTable es el siguiente:

```
import java.awt.*;  
import java.awt.event.*;  
import java.util.*;  
import javax.swing.*;  
import javax.swing.table.*;
```

```
class AplicacionSwing6 extends JPanel {  
    private JTable tabla;  
    private JScrollPane panelScroll;  
    private String tituloColumna[];  
    private String datoColumna[][];  
  
    public AplicacionSwing6() {  
        setLayout( new BorderLayout() );  
        // se crean las columnas con su titulo por separado  
        tituloColumna = new String[8];  
  
        for( int i=0; i < 8; i++ ) {  
            tituloColumna[i] = "Col: "+i;  
        }  
        datoColumna = new String[100][8];  
        // se cargan los valores en cada celda para lo que queremos que aparezca  
        for( int i=0; i < 100; i++ ) {  
            for( int j=0; j < 8; j++ ) {
```

```
        datoColumna[i][j] = "" + j + "," + i;
    }
}
// se crea una instancia del componente Swing
tabla = new JTable( datoColumna,tituloColumna );
// se cambia la presentación a la tabla
tabla.setShowHorizontalLines( false );
tabla.setRowSelectionAllowed( true );
tabla.setColumnSelectionAllowed( true );
// se cambia el color de la zona seleccionada (rojo/blanco)
tabla.setSelectionForeground( Color.white );
tabla.setSelectionBackground( Color.red );
// la tabla se añade a un objeto de la clase JScrollPane
// para que tenga la facilidad de barras deslizadoras
panelScroll = new JScrollPane( tabla );
add( panelScroll, BorderLayout.CENTER );
}
public static void main( String args[] ) {
    JFrame ventana = new JFrame( "Utilizando JTable" );
    ventana.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent evt ){
            System.exit( 0 );
        }
    });
    ventana.getContentPane().add( new AplicacionSwing6(),BorderLayout.CENTER );
    ventana.setSize( 300,180 );

    ventana.setVisible( true );
}
}
```

La aplicación se vería de la siguiente manera:



Col: 0	Col: 1	Col: 2	Col: 3	Col: 4	Col: 5	Col: 6	Col: 7
0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7

EJERCICIO DE AUTOEVALUACIÓN

1. General un applet que contenga los elementos mas comunes del awt, diseñando un formulario de matricula de una universidad
2. Crear una aplicación que maneje las ordenes de un restaurante por mesa, usar los componentes Swing para esta tarea

Prueba Final

1. Crear un aplicativo que contenga un menú, que permita crear una lista de opciones con color, numero y figura geométrica, al seleccionar la ultima opción deberá mostrar la figura, tantas veces como se haya definido y el color que se especifico.

Actividad

Diseñar mediante Swing, una calculadora similar a la de Windows (simple), que permita realizar las operaciones básicas de esta.

6. FUENTES

6.1. Libros

HOLZNER, Steven. (2000): La biblia de java 2. Anaya multimedia

CEBALLOS, Francisco Javier. (2006): Java 2 Interfaces graficas y aplicaciones para internet. Alfaomega Ra-Ma.

JOYANES AGUILAR, Luis; FERNANDEZ AZUELA, Matilde. (2001): Java 2 Manual del programador. Ra-Ma.

CEBALLOS, Francisco Javier. (2005): Java 2 Curso de Programación. Ra-Ma.

ECKEL, Bruce. (2000): Pensando en Java. Prentice-Hall.

VILLALOBOS, Jorge; CASALLAS, Ruby. (2006): Fundamentos de Programación. Prentice Hall.

DEAN, John; DEAN, Raymond. (S.F): Introducción a la programación en java. McGraw Hill

FERNANDEZ, Carmen. (S.F): Java Basico. Starbook.

Páginas web

(S.A). (S.F): www.sun.com.co

(S.A). (S.F): <http://profesores.fi-b.unam.mx>

(S.A). (S.F): www.java.com

(S.A). (S.F): cupi2.uniandes.edu.co

(S.A). (S.F): www.programacion.com/java

(S.A). (S.F): www.monografias.com

(S.A). (S.F): <http://www.javahispano.org/>

6.1.1. Presencial

El proceso de participación dentro de la materia lenguaje de programación I (java), es netamente práctica y participativa, acompañado de investigación complementaria a los temas tratados.

6.1.2. Distancia

El manejo de distancia no dista mucho del método presencial, dadas las características del área, es un ambiente teórico práctica, permitiendo la interacción permanente del aprendiz y de la complementación de los temas en mediante investigación y práctica de los temas.

6.1.3. Evaluación

El tema evaluativo se conforma de un 80% del tema en forma práctica con los temas tratados e investigados y 20% teórico con los conceptos de la materia demanda y que son fundamentales para una comprensión optima de los temas.