UAEM | Universidad Autónoma del Estado de México

Ingeniería en Computación

"2015, Año del Bicentenario Luctuoso de José María Morelos y Pavón"

Unidad de Aprendizaje:

Programación Avanzada

Unidad de Competencia I:

Programación modular y recursiva

M. en C. Edith Cristina Herrera Luna

Agosto 2015

Propósito de la Unidad de Aprendizaje

- Servir de enlace entre el aprendizaje de los paradigmas estructurado y orientado a objetos, a través de la programación modular. Presentar al alumno técnicas de programación avanzada como la recursividad.

- Proporcionar las habilidades necesarias para evaluar la complejidad de un algoritmo de ordenamiento o de búsqueda, así como estrategias para resolver problemas de alta complejidad, mediante técnicas de diseño avanzadas.

Programación Avanzada
INTRODUCCIÓN

- Una vez adquiridas las habilidades de programación básicas bajo el paradigma estructurado, el alumno debe conocer otros paradigmas como la programación modular y recursiva.

- Junto con estos paradigmas el alumno se adentra en cuestiones de algorítmica, con temas de análisis y diseño de algoritmos: funcionamiento y orden de complejidad de los métodos de ordenamiento y búsqueda, técnicas de diseño como algoritmos voraces, algoritmos divide y vencerás, programación dinámica, algoritmos vuelta atrás, algoritmos ramifica y poda.

- Esta formación permitirá al futuro ingeniero enfrentarse a retos de programación de alta complejidad con la certeza de poder no solo dar una solución a un problema dado sino de dar la solución optima y ser capaz de evaluar que tan buena es la solución dada.

PROGRAMACIÓN MODULAR Y RECURSIVA

Unidad de Competencia I

OBJETIVO:

- Identificar, desarrollar y programar algoritmos bajo los paradigmas de programación modular y recursiva.

Programación modular y recursiva
CONTENIDO

A. ¿Qué es un paradigma de programación?

B. Diferentes paradigmas de programación

C. Programación Modular
   1. ¿Qué es?
   2. Ejemplo
   3. Características y Ventajas
   4. Programación Estructurada: Estructuras de Control

D. Programación Recursiva
   1. ¿Qué es?
   2. Ejemplo

¿Qué es un paradigma de programación?

C. U. UAEM ZUMPANGO / ICO / PA / ECHL

¿Qué es un paradigma de programación?

- **PARADIGMA**: (Griego: *paradeigma*) Usado como sinónimo de *ejemplo*, *modelo* o patrón. Un significado contemporaneo:

  "Es el conjunto de prácticas o teorías que definen una disciplina científica, luego de haber sido, y siendo aún puestas, a numerosas pruebas y análisis a través del tiempo, y por ello aún se mantienen vigentes".

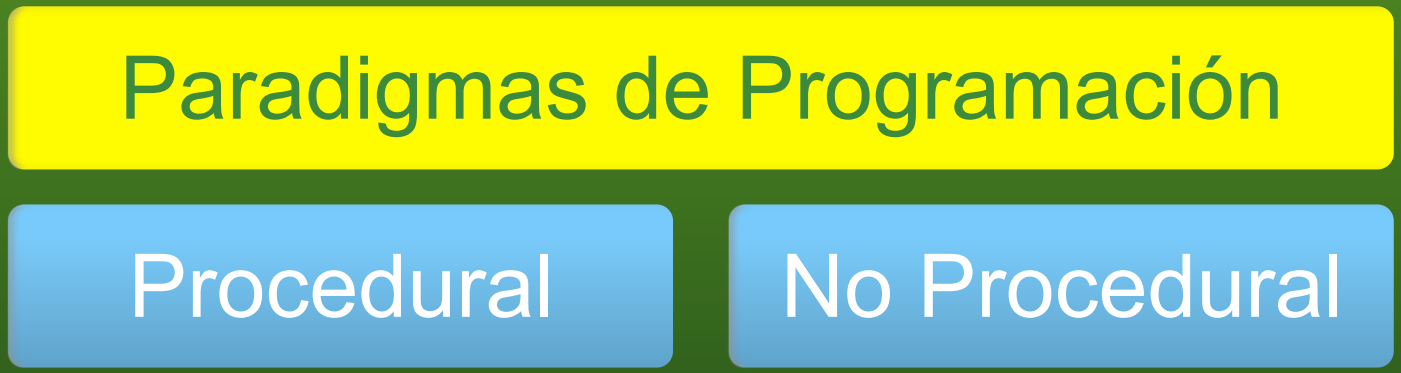
Un **paradigma de programación** representa un enfoque particular o filosofía para diseñar soluciones. Los paradigmas difieren unos de otros, en los conceptos y la forma de abstraer los elementos involucrados en un problema, así como en los pasos que integran su solución del problema, en otras palabras, el cómputo.

Diferentes paradigmas de programación

C. U. UAEM ZUMPANGO / ICO / PA / ECHL

Diferentes paradigmas de programación

- Los paradigmas de programación se pueden clasificar como *Procedurales* y *No Procedurales.*
- Sin embargo, algunos paradigmas y lenguajes tienen características que los sitúan en mas de un grupo.



Diferentes paradigmas de programación

Algunas características de la programación procedural son:

- Conocida también como programación algorítmica, de procedimientos o convencional, se basa en la implementación de una algoritmo.
- Generalmente no son del tipo de Inteligencia Artificial.
- Procede en forma secuencial (No en sentido estricto)
- El programador debe especificar con exactitud cómo debe codificarse la solución de un problema

Diferentes paradigmas de programación

Algunos paradigmas de programación son:

- Programación Estructurada:
  - Divide el código en bloques o estructuras que pueden comunicarse entre si o no, se realizan secuencias de instrucciones
- Programación Modular
- Programación Recursiva
- Programación Orientada a Objetos:
  - Abstrae elementos dando una visión del mundo real para representarlos usando objetos, métodos, atributos y clases; junto con técnicas como abstracción, encapsulamiento, polimorfismo y herencia.

Programación Modular

C. U. UAEM ZUMPANGO / ICO / PA / ECHL

¿Qué es la programación modular?

- En general, los programas para la vida cotidiana son extensos y complejos, su programación requiere un buen diseño, desarrollo y mantenimiento.
- La **abstracción** es un proceso mental fundamental para el desarrollo de un buen programa
- Un esquema que resume el proceso de diseño de programas podría ser:
  1. Descomposición del problema en bloques o módulos de sólida cohesión interna (*Diseño modular*)
  2. Programación de cada módulo mediante métodos estructurados (*Programación estructurada*)
  3. Integración de módulos mediante procedimientos descendentes (Top-Down) o ascendentes (Bottom-Up)

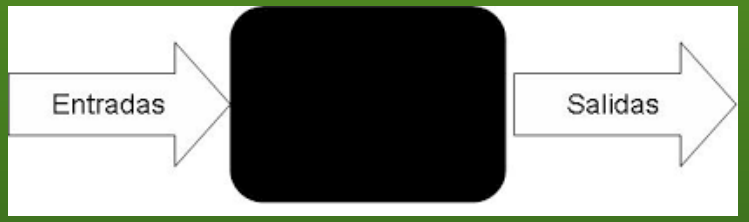
¿Qué es la programación modular?

- Este tipo de programación consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo mas sencillo, legible y manejable. Cada modulo se desarrolla y analiza de manera independiente, y generalmente un modulo mantiene un control y/o comunicación con todos los demás.

DIVIDE Y VENCERÁS

Modulo de un programa

- **Modulo**: es un conjunto de instrucciones contiguas y lógicamente encadenadas con un propósito específico; tienen un identificador con el cual se hace referencia a ellos y pueden ser invocados por el programa.



- Un modulo puede ser como una caja negra la cual se comunica con otros módulos únicamente por sus datos de entrada y salida.

Modulo de un programa

- Los módulos se comunican por medio de parámetros (datos).
- Pueden tener un dato de retorno en función de sus parámetros de entrada.
- Al crear un módulo se considera su funcionalidad, rendimiento y diseño (máxima cohesión – mínimo acoplamiento)

Ejemplo de programación modular

Características y Ventajas

- Permite programar, depurar, probar y mantener el programa por módulos. Se toman partes individuales mas pequeñas.
- Es mas fácil dar mantenimiento a una parte del sistema que a todo.
- Disminuye la complejidad del sistema y el costo de desarrollo.
- Se pueden detectar errores y/o deficiencias y repararlos sin que se afecte a otras tareas.

Características y Ventajas

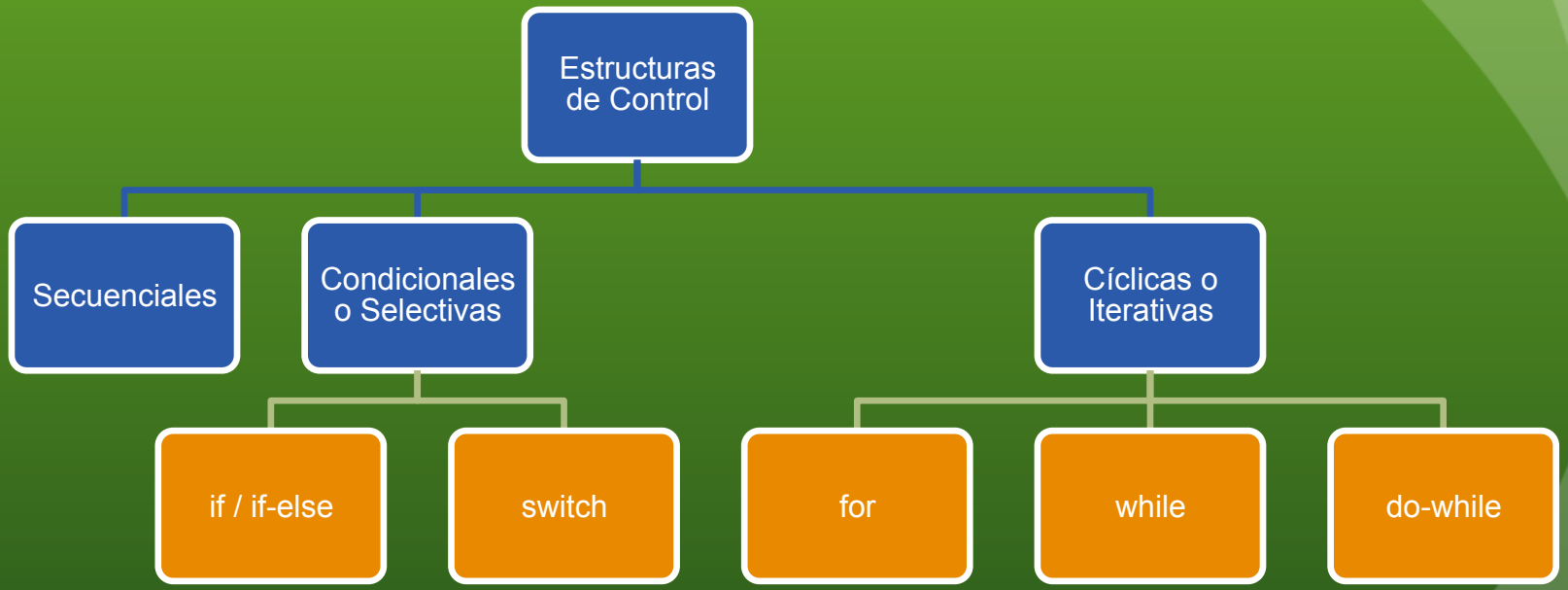
- Se puede tratar un modulo entre varios uno o programadores de acuerdo a la complejidad del mismo.
- Se puede reutilizar módulos o secciones de ellos invocándolos en lugar de definir nuevos.
- Facilita las ampliaciones y modificaciones del programa, incorporando nuevos módulos.

Programación Estructurada: Estructuras de Control

- Para programar cada modulo se aplican diferentes técnicas de acuerdo al propósito del mismo.
- La programación estructurada puede ser un complemento de la programación modular.
- Se basa en el uso de estructuras de control y un diseño descendente (top-down)
- Diseño descendente: Consiste en crear un diseño jerárquico tipo árbol en el que los niveles cercanos a la raíz se refieren al problema a grandes rasgos, mientras se desciende por la estructura, cada nivel se tiene mas detalle hasta llegar a las hojas del árbol que representan operaciones simples del modulo.

Programación Estructurada: Estructuras de Control

- Conjunto de instrucciones o sentencias que permiten controlar el flujo de información en un programa.



Programación Estructurada: Estructuras de Control

```
if ( condición / expresión ) {
        //Caso verdadero
} else {
        //Caso falso
}
```

```
switch ( opción ) {
        case 1:
        break;
        case 2:
        case 3:
        break;
        default:
}
```

Programación Estructurada: Estructuras de Control

```
for ( inicialización ; condición / expresión ; incremento / decremento )
{

        //Acciones a realizar si cumplen la condición

}
```

```
while (condición / expresión) {

        //Acciones a realizar si

        //cumplen la condición

}
```

```
do {

        //Acciones a realizar si

        //cumplen la condición

} while (condición / expresión)  ;
```

Práctica

- Programar el sistema de calificaciones de alumnos usando programación modular y programación estructurada.

Programación Recursiva

C. U. UAEM ZUMPANGO / ICO / PA / ECHL

¿Qué es Recursión?

- Es una técnica de programación que consiste en resolver un problema por medio de una o más divisiones de sí mismo más simples.

- Consiste en una función en la que su definición contiene una invocación a si misma, de tal forma que se resuelve una parte del problema hasta llegar a un punto de parada.

¿Qué es Recursión?

Un algoritmo recursivo consta de dos partes:

- Caso Base: Es la resolución del problema de manera directa
- Caso Recursivo: Caso en el que el problema se divide en versiones más pequeñas de si mismo.

Observación: Se pueden tener

varios casos base y recursivos.

¿Cómo diseñar un algoritmo Recursivo?

1. Reconocer el caso base y proporcionar una solución para él.
2. Diseñar una estrategia para dividir el problema en versiones más pequeñas del mismo considerando avanzar hacia el caso base.
3. Combine las soluciones de los problemas más pequeños para obtener la solución del problema original.

Ejemplo 1

```
writeVertical(3):
3
writeVertical(12):
1
2
writeVertical(123):
1
2
3
```

```
1  public class RecursionDemo1
2  {
3      public static void main(String[] args)
4      {
5          System.out.println("writeVertical(3):");
6          writeVertical(3);

7          System.out.println("writeVertical(12):");
8          writeVertical(12);

9          System.out.println("writeVertical(123):");
10         writeVertical(123);
11     }

12     public static void writeVertical(int n)
13     {
14         if (n < 10)
15         {
16             System.out.println(n);
17         }
18         else //n is two or more digits long:
19         {
20             writeVertical(n/10);
21             System.out.println(n%10);
22         }
23     }
24 }
```

Caso Base

Caso Recursivo

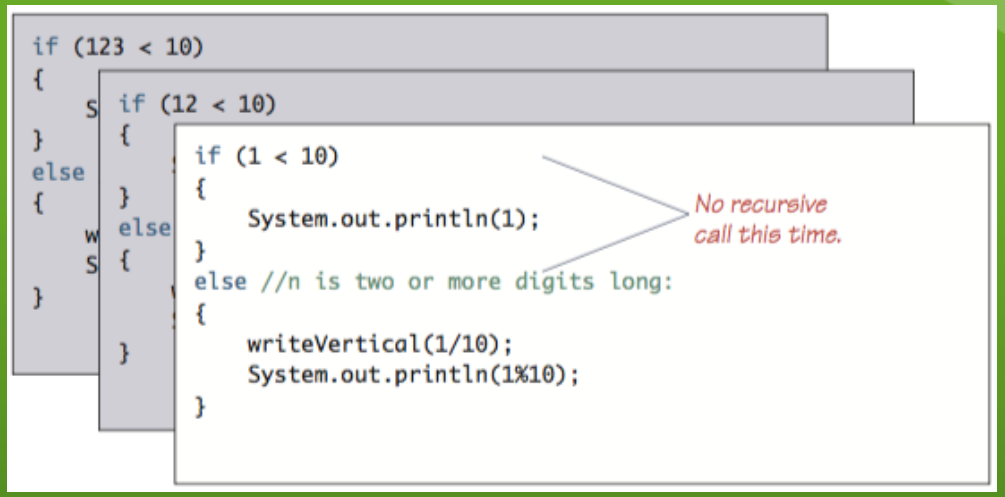
```
if (123 < 10)
{
    System.out.println(123);
}
else //n is two or more digits long:
{
    writeVertical(123/10);
    System.out.println(123%10);
}
```

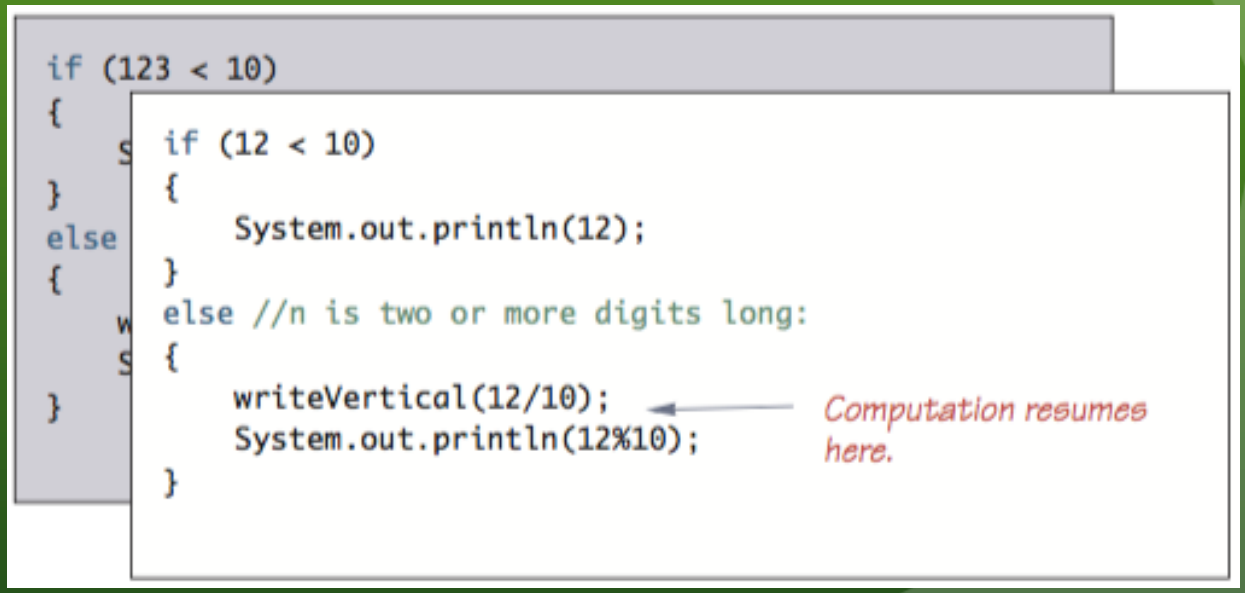
Computation will stop here until the recursive call returns.

```
if (123 < 10)
{
    System
}
else //n i
{
    writeV
    System
}
```

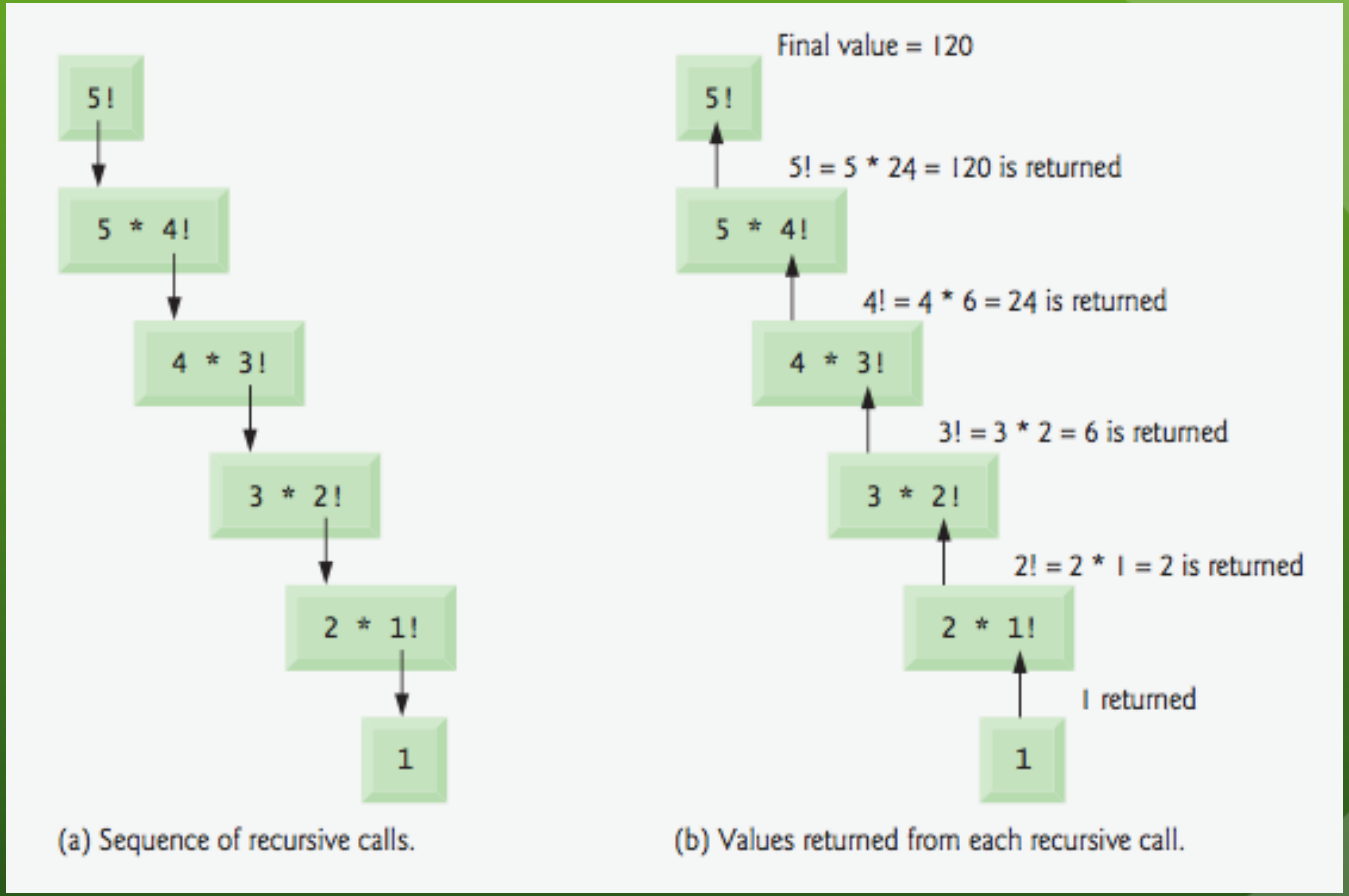
```
if (12 < 10)
{
    System.out.println(12);
}
else //n is two or more digits long:
{
    writeVertical(12/10);
    System.out.println(12%10);
}
```

Computation will stop here until the recursive call returns.


if (123 < 10)
if (12 < 10)
if (1 < 10)
{
System.out.println(1);
}
else //n is two or more digits long:
{
writeVertical(1/10);
System.out.println(1%10);
}
No recursive
call this time.


if (123 < 10)
if (12 < 10)
{
System.out.println(12);
}
else //n is two or more digits long:
{
writeVertical(12/10);
System.out.println(12%10);
}
Computation resumes
here.

Ejemplo 2: Factorial



```java
public class FactorialCalculator
{
   // recursive method factorial
   public long factorial( long number )
   {
      if ( number <= 1 ) // test for base case
         return 1; // base cases: 0! = 1 and 1! = 1
      else // recursion step
         return number * factorial( number - 1 );
   } // end method factorial

   // output factorials for values 0-10
   public void displayFactorials()
   {
      // calculate the factorials of 0 through 10
      for ( int counter = 0; counter <= 10; counter++ )
         System.out.printf( "%d! = %d\n", counter, factorial( counter ) );
   } // end method displayFactorials
} // end class FactorialCalculator
```

Ejemplo 3: Potencia

```
21
22      public static double potencia ( int base, int exp ){
23          if( exp == 0)
24              return 1;
25          else {
26              if( exp > 0 )   //positivo
27                  return (base  * potencia( base, exp-1) );
28              else    //negativo
29                  return ( 1.0 / potencia (base, -exp) );
30          }
31      }
32
33  }//clase
```

```
1   import java.util.Scanner;
2
3   public class PotenciaR{
4
5       public static void main( String[] args)
6       {
7           int b, e;
8           double p;
9           String frase;
10          Scanner lee = new Scanner(System.in);
11
12          System.out.println("\nPotencia de dos numeros");
13          System.out.println("Ingresa la base: ");
14          b = lee.nextInt();
15          System.out.println("Ingresa el exponente: ");
16          e = lee.nextInt();
17          p = potencia(b,e);
18          System.out.println("Potencia: "+p+"\n");
19
20      }
```

Programas usando recursión

- Escribe un método recursivo que retorne la suma entera de los dígitos de una cadena.
  - *Entrada*: 34hola6mundo *Salida*: 13
- Escribe una función que repita cada cadena de un carácter.
  - *Entrada*: holaMundo *Salida*: hhoollaaMMuunnddoo
- Calcular el máximo común divisor (MCD) de dos números
  - *MCD(x, y)* ➔ *y si y es divisor de x*
  - *MCD(x, y)* ➔ *MCD(y, x % y) si y no es divisor de x*

Programas usando recursión

- Identificar si una frase es un palíndromo o identificar si un número es capicúa
- Implementar búsqueda binaria
- Escribe un método que imprima la suma de los cuadrados de los primeros *n* números. $1^2 + 2^2 + 3^2 + .. + n^2$
- Calcula de manera recursiva el n-ésimo numero Fibonacci.

Referencias:

- Base, S.; Van Gelder, A.(2002). “Algoritmos Computacionales: Introducción al análisis y diseño” Ed. Addison Wesley
- Bovet, D. P.; Crescenci, P. (2006). “*Introduction to the theory of complexity*” Ed. Creative Commons
- Brassard, G.; Bratley, P. (1997). "Fundamentos de Algoritmia", Ed. Prentice Hall.
- Cairó, Osvaldo y Guardati, Silvia. (2006). *Estructuras de datos* (3a. Edición). McGraw-Hill.
- Lee R. Teng. S. Chang R. y Tsai Y. (2007). *Introducción al diseño de algoritmos*. McGraw- Hill
- Drozdeck, Adam. (2007). *Estructuras de datos y algoritmos en Java* (2a Edición). Thomson.

Referencias:

- Joyanes, Luis. M. Fernández, L: Sánchez, I. Zahonero. (2005). *Estructuras de datos en C.* McGraw-Hill. Schaum.
- Koffman, Elliot y Wolfgang, Paul. (2008). *Estructura de datos con C++*. Objetos, abstracciones y diseño. McGraw-Hill.
- Savitch, Walter. *Absolute Java*, Segunda Edición,

GRACIAS

Continua Unidad de Competencia II:
Algoritmos de Ordenamiento y Búsqueda

C. U. UAEM ZUMPANGO / ICO / PA / ECHL

Guía para el Profesor

- Las primeras diapositivas muestran el propósito, justificación y objetivos de la unidad de aprendizaje. Se presentan para que el alumno identifique dichos elementos.
- El contenido, conforme a la unidad de aprendizaje, maneja los temas de un menor a mayor grado de dificultad.
- Se trata de manera fluida el uso de Estructuras de Control, pues el alumno ya sabe manejarlas, solo se hace énfasis en su sintaxis pues en las siguientes unidades de competencia se hará estudio de ellas al analizar su eficiencia y orden de complejidad.
- Se incorpora un diagrama como practica para el tema de programación modular, se recomienda que entre todo el grupo se cree un solo sistema, conformando equipos de máximo 4 integrantes y cada equipo programara un modulo del sistema. Considere el análisis y diseño del sistema con todo el grupo.

Guía para el Profesor

- Para los temas de programación Recursiva, se exponen tres ejemplos. El primero muestra un método recursivo que imprime una cadena de manera vertical, las imágenes permiten analizar el programa “paso a paso”. El segundo ejemplo hace uso del retorno de un valor en la función recursiva a diferencia del ejercicio anterior. Y el tercer ejercicio muestra un ejemplo con dos casos recursivos y uno base.

- Aunque se sugieren varios programas para prácticas, se recomienda que se realicen varios programas basados en los ejemplos con pequeñas modificaciones para que el “pensamiento recursivo” sea captado.

- Es importante hacer pruebas de escritorio y/o dibujos de los procedimientos porque el alumno esta acostumbrado a un enfoque iterativo y cambar su visión a programación recursiva necesita de mucha práctica.

- Observar que no se implementen soluciones iterativas, indicar que el uso de ciclos que llamen varias veces al a función no es recursión. **No solo tener la invocación de si mismo hace a un método recursivo.**