UNIVERSIDAD NACIONAL DE LA MATANZA
DEPARTAMENTO DE INGENIERÍA E INVESTIGACIONES TECNOLÓGICAS

PROGRAMACIÓN AVANZADA

GUÍA DE TRABAJOS PRÁCTICOS

Integrantes de la Cátedra

Jefe De Cátedra :	Mg. Verónica Aubin
Jefes de TTPP:	Ing. Leonardo Blautzik Ing. José Luis Cabrera
Ayudantes de 1ra:	Ing. Lucas Videla
Ayudantes de 2da:	Julio Crispino Ing. Lucas Ponce de León Federico Gasior

INDICE

OBJETIVOS

Objetivos Generales:

1. Capacidad de aplicar los conocimientos en la práctica.
2. Capacidad para organizar y planificar el trabajo.
3. Capacidad para analizar, plantear y resolver problemas.
4. Capacidad de trabajo en equipo.
5. Compromiso para realizar el trabajo con Calidad.
6. Aceptar el uso de INTERNET como un medio habitual para la realizar consultas, bajar librerías y software.

Objetivos Específicos:

1. Proporcionar una comprensión sólida de los conceptos fundamentales del modelo de objetos.
2. Lograr que el alumno tenga una visión abstracta y concreta de cada concepto, es decir, que además de entender el concepto en sí, sepa de sus posibles aplicaciones y de su implantación computacional.
3. Mejorar la técnica de diseño de algoritmos y su implantación como programas.
4. Aceptar a las metodologías iterativas, incremental y en espiral, para el desarrollo del software como parte de la cultura del buen programador.
5. Aceptar la técnica de Desarrollo Dirigido por Tests (Test Driven Development o TDD) para

el desarrollo sel software

6. Aceptar algunos principios de la programación XP (programación Extrema) como parte de la cultura del buen programador.
7. Aceptar algunos principios de PSP (Personal Software Process) como parte de la cultura del buen programador.
8. Reconocer e incluir a la Verificación y Validación de Software como parte fundamental en la actividad del programador.
9. Saber realizar buenas estimaciones para el desarrollo del trabajo en base a las métricas personales y grupales para el desarrollo del software.
10. Resolver problemas del tipo Nivel 1, 2 y parcialmente del nivel 3 de la OIA (Olimpíadas Informáticas Argentinas)
11. Dominio avanzado de algún lenguaje que soporte la programación orientada a objetos
12. Conocimientos básicos de la programación funcional y lógica

PROGRAMA ANALÍTICO. CONTENIDOS TEÓRICOS Y PRÁCTICOS:

Unidad I Una Metodología del desarrollo de software (3 clases)

Metodologías iterativas, incremental, espiralada. Procesos Ágiles: Programación extrema (XP) y Scrum. Métricas del software. Principios de la programación XP; Conceptos de PSP (Personal Software Process); Desarrollo Dirigido por Tests (Test Driven Development o TDD). Una metodología propuesta por la cátedra para el desarrollo de software.
La preparación del lote de prueba. Documentación del Lote de Pruebas. El programa probador. El programa verificador del Input. Ventajas de preparar la prueba antes de comenzar la programación. Pruebas del software: de caja negra, de caja blanca, inspecciones. Ventajas de la inspección del código fuente.

Unidad II Paradigmas y metodologías de programación (1 clase)

Paradigmas de programación: Imperativo, Declarativo, Funcional y lógico. Estilos de programación en la programación Imperativa: Estructurada por procedimientos, modular, con objetos y orientada a objetos. Apoyo de un lenguaje para cada estilo de programación. Programación orientada a objetos y lenguajes orientados a objetos.

Unidad III El Lenguajes de Programación JAVA (7 clases)

Tipos de datos abstractos. Estructuras de control. Recursividad. Eventos. Excepciones. Concurrencia. Estructuras de Datos en JAVA. Datos y métodos. Métodos static. Constructores. Las clases complejo, vectorMath y matrizMath. Implantación de las clases correspondiente a los TDA básicos: Clase lista, pila y cola.

Unidad IV Herencia Y Polimorfismo (2 clase)

Jerarquías de clases: Generalización y Especialización (“es un“). Composición y Agregación. (“tiene un“ y ”es parte de“). Miembros protegidos de la clase base. Clases derivadas. Herencia

simple y múltiple. Accesibilidad. Clases abstractas. Métodos virtuales. Aplicación de métodos polimórficos. Diagrama de clases usando UML.

Unidad V Complejidad Computacional (4 clases)

Medición del tiempo de ejecución de un algoritmo. Orden de un algoritmo. Velocidad de crecimiento del tiempo de ejecución. Función O. Efectos al aumentar la velocidad del computador y/o al aumentar el tamaño del problema. Medición del tiempo de ejecución. Comparaciones de los distintos algoritmos desarrollados en los TTPP.

Unidad VI Algoritmos (6 clases)

Algoritmos de ordenamientos y sus implantaciones. Comparaciones de los distintos algoritmos en función de sus operaciones básicas. Estabilidad y sensibilidad al input de un algoritmo de ordenamiento. Utilización de estructuras estáticas y dinámicas. Algoritmos de búsqueda. Algoritmos de Grafos: Dijkstra, Floyd, Warshall, Prim, Kruskal y coloreo de grafos. Resolución de problemas reales y ficticios. Concurrencia y Paralelismo. Algoritmos concurrentes, distribuidos y paralelos.

Unidad VII Taller Programación Básica en Java con WebSphere/Rational (16 clases)

Conceptos básicos de Java. Conceptos y características de J2ME, J2SE, J2EE. Arquitectura de los distintos frameworks. Implementación de un cliente pesado en Java. Implementación de un JavaBean. Acceso a datos.

Unidad VIII Programación Lógica (4 clases)

Lógica de predicados. argumentos, interpretación, reglas, preguntas, variables, ámbito de una variable, corte. Estructura de un programa Prolog. Listas y secuencias finitas. Aplicaciones en grafos. Resolución de caminos mínimos. Equivalencia entre el Prolog y los operadores del TDA conjunto. Equivalencia entre Prolog y consultas SQL.

Unidad IX. Programación Funcional (1 clases)

Introducción a la programación funcional. Haskell & Hugs. Tipos y clases. Funciones y operadores.

BIBLIOGRAFÍA

Título	Autor(es)	Editorial	Año Edición	Ejemplares disponibles en UNLaM
Core Java Volumen I – Fundamentos 7ma Edición	Horstmann Cornell	Pearson Prentice Hill	2005	5
Java, cómo programar. 7ma Edición	Deitel -Deitel	Pearson Prentice Hill. Séptima edición	2008	2 de la 2da edicion y 7 de la 4ta edicion
Manual de Java	Naughton P	McGraw –Hill	2002	1
Estructuras de datos y algoritmos.	Aho, Hopcroft,j Ullman.	Addison-Wesley Iberoamericana	1983	5
Algoritmos en C++	Sedgewick	Addison-Wesley	1995	5
Introducción a la Programación en Java	D. Arnow- G. Weiss.	Addison Wesley	2001	1
Orientación a Objetos con Java y UML	FONTELA CARLOS	NUEVA LIBRERIA	2003	-
Scrum y XP desde las trincheras Como hacemos Scrum	Henrik Kniberg	*http://www.proyectalis.com/wp-content/uploads/2008/02/scrum-y-xp-desde-las-trincheras.pdf*	2007	-
PROLOG	GiannesiniKanouiPaseroCaneghem	Addison Wesley Iberoamericana	1989	1
Canal YouTube Programación Lógica	Gustavo Dejean	http://www.youtube.com/playlist?list=PLAA2F26B4E3985B9E		-
Thinking in Java, 4th. edition	Bruce Eckel	Prentice Hall http://www.saeedsh.com/resources/Thinking%20in%20Java%204th%20Ed.pdf	2006	-
Estructuras de datos y algoritmos	Mark Allen Weiss	Addison Wesley Iberoamericana	1995	1

USO DE COMPUTADORAS

SOFTWARE A UTILIZAR

- Java SUN, J2SE6.x o superior
- Eclipse
- Swi-prolog
- Haskell

CALENDARIO DE ACTIVIDADES *(no incluye Taller)*

Nº de Clase	Semana de Clase	Unidad Temática o Actividad

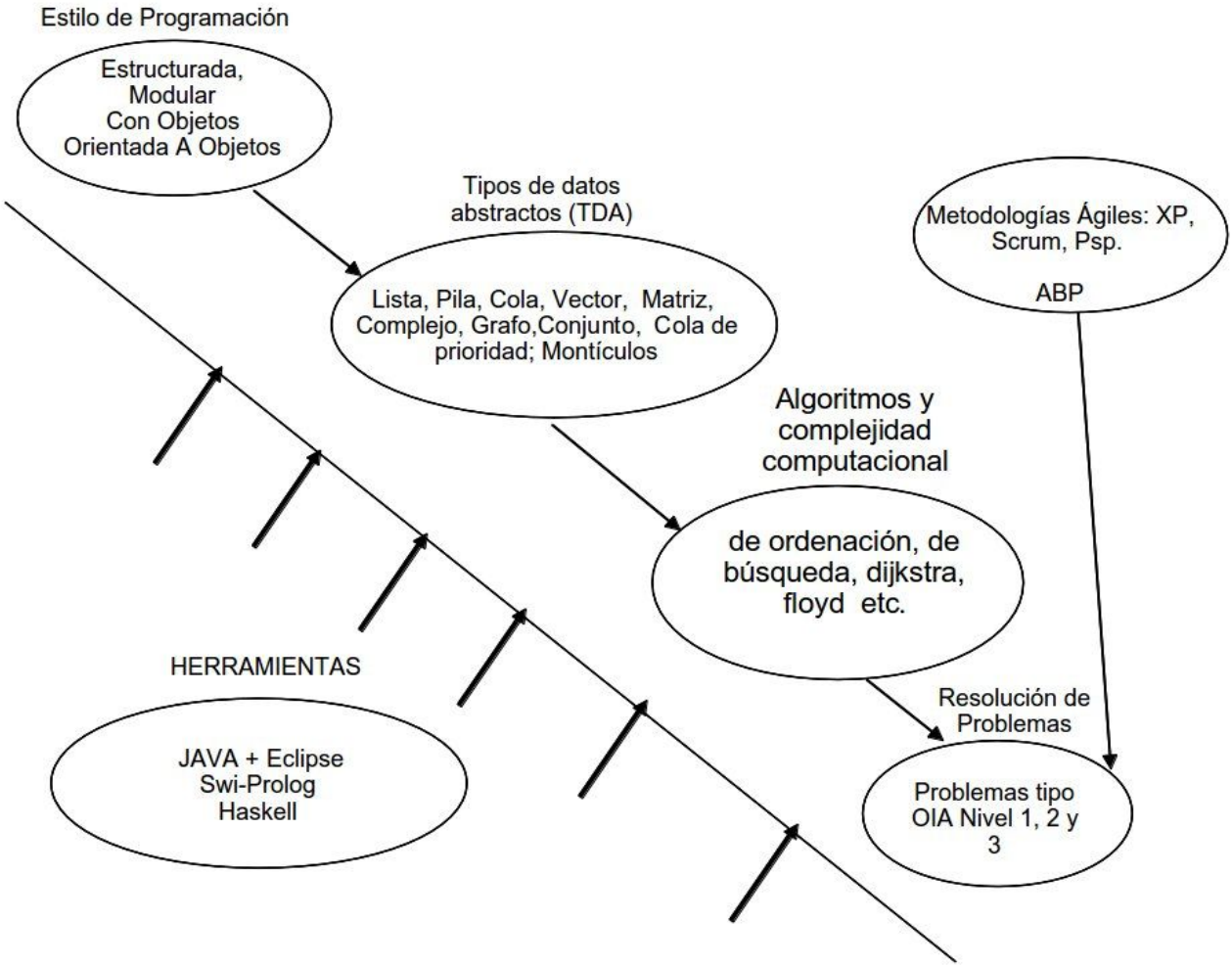
1	1	I
2	1	I - II
4	2	I y III - resolución de ejercicios de guía I
5	2	III - guía nro 1 y 2
7	**3**	III
8	3	V
10	4	V
11	4	V
13	5	V
14	5	IV- III
16	6	IV - III
17	6	III Consultas y práctica
19	7	VIII- resolución de ejercicios de guía X
20	7	VIII- resolución de ejercicios de guía X
22	8	VIII- resolución de ejercicios de guía X
23	8	VIII- resolución de ejercicios de guía X
25	9	1° Parcial
26	9	Corrección 1° Parcial
28	10	III
29	10	VI
31	11	VI
32	11	VI
34	12	VI
35	12	VI
37	13	VI Consultas y práctica.
38	13	**2do. PARCIAL**
40	14	Corrección 2° Parcial
41	14	IX
43	15	Consultas y práctica
44	15	**Recuperatorio**
46	16	Entrega deTP pendiente y resultados.
48	16	**Último día de clase:** Corrección entrega de resultados. Publicación de notas.
	17	Atención pre-exámenes a alumnos
	18	Exámenes finales
	19	Revisión de Exámenes
	20	Reuniones de cátedra - Articulación de contenidos
	21	Atención pre exámenes a alumnos
	22	Exámenes finales
	23	Exámenes finales

24	Evaluación cursada 2er cuatrimestre - Nuevas herramientas Didácticas
25	Reuniones de catedra - Articulación Contenidos
26	Conformación grupos de trabajo revisión temas teórico y prácticos

GUIAS DE TP (TODAS) (ver en archivo anexo ítem guía de ejercicios)

PRÁCTICA I Prueba de software
PRÁCTICA II Tipos de datos abstractos fundamentales (TDA)
PRÁCTICA III Herencia y Polimorfismo
PRÁCTICA IV Resolución de problemas Nivel 1
PRÁCTICA V Resolución de problemas Nivel 2
Práctica VI Complejidad computacional
Práctica VII Algoritmos de ordenamiento y búsqueda
PRÁCTICA VIII Grafos
PRÁCTICA V Resolución de problemas Nivel 3
PRÁCTICA IX Prolog
PRÁCTICA X Haskell

MAPA CONCEPTUAL PRINCIPAL DE LA CÁTEDRA DE PROGRAMACIÓN AVANZADA



DESCRIPCIÓN DE LA ACTIVIDAD CURRICULAR

MODALIDAD DE ENSEÑANZA EMPLEADA.

1. Exposición oral por parte del profesor
2. Resolución de casos reales o imaginarios
3. Consultas (personalizadas y/o públicas).
4. Utilización del software disponible
5. Utilización de INTERNET como un medio habitual para la realización de consultas, bajar librerías, software y como medio de ayuda y comunicación con los proveedores de software.
6. Competencias de resolución de algoritmos.
7. Uso de Recursos Didácticos: video tutoriales propios, guía de ejercicios, ejemplos impresos, URLs Institucional y públicos, mails.

REGLAMENTO DE PROMOCIÓN

El alumno deberá realizar 5 TTPP especiales. Cada uno de ellos deberá enviarlo por e-mail con anticipación al comienzo de la clase. Además deberá presentar en cada entrega dos carátulas iguales (uno para el tutor y otro para el alumno). El envío debe incluir al menos un .doc con el siguiente detalle: índice, un resumen, desarrollo, apéndices, gráficos (si corresponde) y conclusiones. Además debe contener (si corresponde) el lote de prueba y su respectiva documentación. El formato de las paginas del .doc será A-4. El envío se debe realizar en un único archivo compactado .rar. El docente siempre responderá con un "recibido" a cada envío.

En cada TP se respetaran las normas Standard utilizadas en Java. Cada class debe contener su propio test. El proyecto debe contener un lote de pruebas y/o programa probador según corresponda.

Parciales

Se tomarán dos parciales según figura en el cronograma. Los parciales corregidos son entregados en mano a los alumnos, teniendo la posibilidad de realizar preguntas sobre las correcciones efectuadas. Previamente a la entrega de los parciales corregidos, se explica brevemente la solución a los problemas planteados haciendo hincapié en los puntos donde se observaron los errores mas comunes.

Recuperatorios y condiciones para rendirlos

Según el reglamento interno del departamento, habrá un único recuperatorio.

Régimen de trabajos prácticos especiales

Los trabajos prácticos especiales son parte fundamental de la cátedra. En ellos, el alumno, además de llevar a la práctica los puntos teóricos vistos en clase, también debe comprobar su verdadera aplicabilidad en caso real.
La no presentación del TP especial en la fecha propuesta significa su desaprobación.
La realización de cada TP especial, será grupal, pero su evaluación será en forma individual.
Para aprobar cada TP , el alumno, deberá responder un coloquio sobre el mismo, donde deberá defender sus conceptos principales y efectuar las modificaciones necesarias si se le cambian las condiciones de contexto.
El alumno que desapruebe el TP tendrá una nueva y única fecha de presentación, excepcionalmente podrá haber una 3ra fecha de presentación.

Regularidad

Para regularizar la materia el alumno deberá aprobar los dos parciales, tener los 5 TTPP especiales aprobados, el 75% de presentismo y el TP especial del Taller de Java aprobado. El régimen de promociones está completamente sujeto al reglamento de la UNLM y es de público conocimiento.

GUÍA DE TTPP

PRÁCTICA I (prueba del software)

Para cada uno de los siguientes problemas, se pide:

a) Crear un **lote de pruebas** que asegure, con buenas probabilidades de éxito, el correcto funcionamiento del programa que intenta resolverlo.
b) Documentar, en un archivo aparte, cada uno de los casos que compone su lote de pruebas, indicando cual sería el output esperado para cada uno de ellos y si corresponde cuales son los datos de entrada y qué es lo que se intenta probar en cada caso. En caso de que los datos de entrada sean numerosos, simplemente indique el nombre del archivo de entrada.
c) Cuando se justifique, codifique un **programa Probador**; es decir, un programa que dado dos archivos (el archivo de entrada y el de salida de otro programa) nos diga si la salida es correcta (se corresponde con le archivo de entrada).

Problema 1: **"Vendedoras Premiadas"**
Problema 2: **"Letras extremas"**
Problema 3: **“Construyendo una casa en un pedregal”**

PRÁCTICA IIa. Tipos de datos abstractos fundamentales (TDA)

Para cada uno de los siguientes TDA, defina en Java la class correspondiente con los métodos asociados. Cada class debe tener un método miembro main, donde debe implementarse el test para cada método.

Complejo:
Implementar los métodos que permitan realizar las siguientes operaciones entre objetos de la clase: sumar, restar, multiplicar, comparar (equals), clone(), obtener el modulo (modulo()), toString().

Sobrecargue las funciones de manera que se pueda trabajar con objetos no complejos, por ejemplo, sumar un complejo con un número entero.

Observe que z * ž da un número Real, llamado módulo.

Crear una clase OrdenadoraDeComplejos que ordene los números complejos según los siguientes criterios: por su módulo, por la parte real, por la parte imaginaria.

VectorMath:

Implementar los métodos que permitan realizar las siguientes operaciones entre objetos de la clase:
Nota: Ver en descargas "Suma de vectores con exception"

- suma de vectores
- resta de vectores
- producto de un vectorMath por otro vectorMath
- producto de un vectorMath por una MatrizMath
- producto de un vectorMath por un Real
- normaUno(); normaDos; normaInfinito()
- equals()
- clone()

Ejemplo de un VectorMath en archivo de entrada.
En la primera linea se indica la dimensión del vector, luego, de a uno por línea se indican las componentes del vector.

myVector.in
6 1.23 2 -7 6 9 2.3E3

MatrizMath:
Implementar los métodos que permitan realizar las siguientes operaciones entre objetos de
la clase:

- suma de matrices
- resta de matrices
- producto de matrices
- producto de un matriz por un vector
- producto de un matriz por una constante tipo float
- matrizInversa() (Implementar el cálculo del error cometido.)
- determinante()
- normaUno(), normaDos(); normaInfinito();
- equals().

Ejemplo de una MatrizMath en archivo de entrada.
En la primera linea se indican las filas y columnas de la matriz, luego, de a uno por línea se indican para cada posición de la matriz, las componentes.

Si la matriz fuera:	En un archivo se representaría como:
2 4 5 9 -2 -1	3 2 0 0 2 0 1 4 1 0 5 1 1 9 2 0 -2 2 1 -1

SEL (Sistemas de ecuaciones lineales)

- constructores
- static bool test()
- resolver ()
- mostrarResultado()
- calcularErrorSolucion()

Pila <<Interface>>:

- empty( )// es vacía
- push(dato) //apilar
- pop( )//desapilar
- peek( )//devuelve una referencia al elemento de la cima
- vaciar( )

PilaEstática implementando la interfaz Pila

PilaDinamica implementando la interfaz Pila

Cola<<Interface>>

- empty( )//vacía
- offer(dato) ;//añadir elemento
- poll( );//Quitar elemento
- peek();una referencia al primer elemento, pero no elimina
- vaciar ( );

ColaEstatica implementando la interfaz Cola

ColaDinamica implementando la interfaz Cola

Lista /list:

- push_back() Inserta un elemento al final

- pop_back() Borra un elemento al final
- push_front() Inserta un elemento al comienzo
- pop_front()Inserta un elemento al final
- remove() Elimina un elemento de un valor determinado.
- reverse() //invierte el orden de los elementos en la lista
- insert(posición, dato)//insertar
- erase() //Eliminar por posición
- empty() //vacía
- buscar (dato);
- buscar (posición);
- vaciar ( );

PRÁCTICA IIb. TDD sobre TDA

Sobre cada ejercicio de la práctica anterior, implementar la alternativa codificando mediante la metodología de Test-driven development.

PRÁCTICA III - Herencia y polimorfismo

1. Desarrollar en Java las clases Punto2D y Punto3D, de manera tal que Punto3D sea una clase derivada de Punto2D. Implemente en cada una el método equals() de manera tal de obtener por pantalla las salidas correctas a las siguientes instrucciones:

```
a) Punto2D p2D = new Punto2D(0,0);
b) Punto3D p3D = new Punto3D(0,0,1);
c) Punto3D a3D = null;
d) out.println(p2D.equals(p3D));//false
e) out.println(p3D.equals(p2D));// false
f) out.println(p2D.equals(a3D));//false
g) out.println(p3D.equals(a3D));//false
h) a3D = new Punto3D(0,0,0);
i) out.println(p2D.equals(a3D));//false
j) out.println(a3D.equals(p2D));//false
```

//Importante: la salida no debe tener exceptions

2. Desarrollar la class PilaHL como una class derivada de la class Lista (ya implementada en la práctica II) y que implemente la interfaz Pila.

3. Desarrollar la class ColaHL como una class derivada de la class Lista (ya implementada en la práctica II) y que implemente la interfaz Cola.

4. Desarrollar la class PilaCL usando la relación contiene (composición) a la class Lista (ya implementada en la práctica II) y que implemente la interfaz Pila.

5. Desarrollar la class ColaCL usando la relación contiene (composición) a la class

Lista (ya implementada en la práctica II) y que implemente la interfaz Cola.

6. Extraer conclusiones a partir de comparar las implementaciones 2 y 4.

7. Ídem para 3 y 5.

8. Desarrollar en Java la jerarquía de clases formada por la class Figura (abstracta) y dos clases derivadas Rectángulo y Círculo. Se deberá tener en cuenta lo siguiente:

    a. La class Figura contiene en su declaración las posición x, y del centro de la figura.
    b. La class Rectangulo tiene ancho y alto como variables miembro.
    c. La class Círculo contiene los atributos centro y radio.
    d. Ambas class implementan las interfaces Rotable y Dibujable

```
public interface Dibujable{
      public void dibujar();
}

public interface Rotable {
      public void rotar();
}
```

    e. Implementar los métodos equals y área para la jerarquía de clases.

9. Desarrollar en Java la jerarquía de clases formada por la class Empleado y la class derivadas Jefe.

    a. Atributos de Empleado: Legajo, Nombre y salario.
    b. Atributo de Jefe: Sector.
    c. Implemente los métodos equals

10. Indicar si son Verdaderas o Falsas las siguientes afirmaciones. Justifique:

- **a.** Cuando una clase contiene un método abstracto tiene que declararse abstracta.
- **b.** Todos los métodos de una clase abstracta tienen que ser abstractos.
- **c**. Las clases abstractas no pueden tener métodos privados (no se podrían implementar) ni tampoco estáticos.
- **d.** Una clase abstracta tiene que derivarse obligatoriamente.
- **e.** Se puede hacer un new de una clase abstracta.
- **f.** La principal diferencia entre interface y abstract es que un interface proporciona un mecanismo de encapsulación de los protocolos de los métodos sin forzar al usuario a utilizar la herencia

11. Dado el siguiente código indicar cuál es la salida.

```
class BaseA
 {
      void mostrar() {
```

```
        out.println("clase BaseA");
                }
}
class DerivadaB extends BaseA
{
        void mostrar()
        {   out.println("clase DerivadaB");
        }
}
//Si ejecutamos:
BaseA a1 = new BaseA();
a1.mostrar();
DerivadaB b1 = new DerivadaB();
b1.mostrar();
BaseA a2 = new DerivadaB();
a2.mostrar();
```

PRÁCTICA IV - Resolución de problemas Nivel 1

Para cada uno de los problemas propuestos a continuación:

a. Realizar una ficha con las **estimaciones** y **métricas**.
b. Generar un **lote de pruebas** que asegure, con buenas probabilidades de éxito, el correcto funcionamiento del programa que intentará resolver.
c. Documentar, en un archivo aparte, cada uno de los casos que compone su lote de
pruebas.
d. Codificar un **programa** que lo resuelva.

Problema 1: "Secuencias Máximas (Nivel 1 OIA 2004)"
Problema 2: "Palíndromo (Nivel 1 OIA 1998)"
Problema 3: "Depósitos subterráneos (Nivel 1 OIA 2008)"

PRÁCTICA V Resolución de problemas Nivel 2

Para cada uno de los problemas propuestos a continuación:

a. Realizar una ficha con las **estimaciones** y **métricas.**
b. Generar un **lote de pruebas** que asegure, con buenas probabilidades de éxito, el correcto funcionamiento del programa que intentará resolver.
c. Documentar, en un archivo aparte, cada uno de los casos que compone su lote de
pruebas.

d. Realizar el **Diagrama de Clases**.
e. Codificar un **programa** que lo resuelva.

Problema 1: "A correr que es muy saludable!!" (Nivel 2 OIA 2010)
Problema 2: "Construyendo una casa en un pedregal" (Nivel 2 OIA 2007)
Problema 3: "Alta en el cielo"(Nivel 2 OIA Selección 2007)"

PRÁCTICA VI - Complejidad computacional

1) Para cada una de las funciones siguientes indique su orden de crecimiento con la notación O(f(n)) y ordene por velocidad de crecimiento (en sentido creciente)

   a) $6 + n^2$; $4 + n + 2\log(n)$; $n^2 + 3 * 2^n$; $5n^3$
   b) $5n$; $n^2 + 5 * 2^n$; 9; $7*n$
   c) 2; $\log(n) + 2n$; $5n + 2^n$; n

2) Cuales de las siguientes afirmaciones son verdaderas. Justifique.

   a) $n \in O(1)$
   b) $500 \in O(n)$
   c) $4n \in O(4^n)$
   d) $n^2 + 1/5n^3 \in O(n^3)$
   e) $n*(n+2)^2/3 \in O(n^2)$
   f) $2 \log_2 n \in O(\log_4 n)$
   g) $3^n \in O(2^n)$

3) Para cada uno de los siguientes segmentos de algoritmos analice sus complejidades usando notación O grande

   a)

```
Leer(a)
n ← a*a
c ← 0
MIENTRAS (a>1) HACER
      a ← a/2
      PARA i=1,n HACER
      c ← c*2
Escribir(c)
```

   b)

```
k ← n
PARA i=1,n HACER
      b ← i
      PARA k=k,b (paso=-1) HACER
         SI a(k-1)>a(k) ENTONCES
               Aux ← a(k-1)
               a(k-1) ← a(k)
```

```
a(k) ← Aux
```

4) Suponiendo que se sabe que el tiempo de ejecución de un algoritmo pertenece a O(NlogN) y que el de otro algoritmo pertenece a $O(n^2)$, se pide:

a) Que se puede decir sobre el rendimiento relativo de estos algoritmos ?

b) Suponiendo que ambos algoritmos resuelvan el mismo problema, enumere en cuales caso implementaría uno y en cuales implementaría el otro.

c) en el caso de que ambos algoritmos sean de ordenación interna, enumere situaciones donde sería valido usar el algoritmo cuadrático.

5) Sea la siguiente tabla:

T(n) Tiempo de ejecución proporcional a:	Tamaño máximo del problema para 10^3seg	Tamaño máximo del problema para 10^4seg	Incremento
100 n	10		
$5n^2$	14		
$n^3 / 2$	12		
2^n	10		

Se pide: a) completar los datos que faltan.

b) saque alguna conclusión observando los respectivos incrementos.

6)

O(n)	1	100 veces más rápido	1000 veces más rápido
n	N_1		
n^2	N_2		
n^3	N_3		
n^5	N_4		
2^n	N_5		
3^n	N_6		

Donde N_i es el tamaño máximo del problema que se puede resolver en una computadora dada. Se

pide completar la tabla precedente en función de N_i, suponiendo que se corre el mismo algoritmo en una máquina 100 veces más veloz y la última columna en una máquina 1000 veces más veloz.

7)

	N_i / seg	N_i / minuto	N_i / hs.
n	1000		
n log n	140		
n^2	31		
n^3	10		
2^n	9		

Dada la tabla precedente se pide calcular el N_i para un minuto y para una hora.

8) Ídem al ejercicio 5 pero se pide calcular el Ni para una máquina 10 veces más rápida.

	N_i / seg	10 veces más rápido
n	1000	
n log n	140	
n^2	31	
n^3	10	
2^n	9	

9) Para cada uno de los algoritmos vistos en la práctica anterior compare el gráfico obtenido con el O(n) correspondiente. (Analice teniendo en cuenta el mejor caso, el caso promedio y el peor caso).

10) Calcule la complejidad del algoritmo “calcular el factorial de n” con recursividad y sin recursividad. Realice mediciones de tiempo.

11) Realice una ficha técnica de su entorno de programación, indicando las características del mismo y cuanto tarda en realizar las siguientes operaciones:

a) 100.000.000 de sumas.

b) 100.000.000 de multiplicaciones

c) 100.000.000 de divisiones

d) 100.000.000 de comparaciones

e) 100.000.000 de asignaciones

12) Determine si existe diferencia de tiempo, en su entorno de programación, en la ejecución de un algoritmo si cambia los ciclos For por While o do While.

13)

O(f(n))	n=10	n=20	n=30	n=40	n=50	n=60
n						
n^2						
n^3						
n^5						
2^n						
3^n						
n!						

Suponga que en la tabla anterior, f(n) se puede igualar a la cantidad de operaciones que realiza un algoritmo. Sabiendo que en una computadora determinada se realizan 100.000.000 de operaciones por segundo complete la tabla dada. Analice el resultado.

Qué sucede si logramos una máquina un millón de veces más rápida?

14) Si se sabe que el algoritmo de selección tardó 10" en ordenar 10000 mentos, cuántos elementos podría ordenar en el triple de tiempo?, cuánto tardaría en ordenar el triple de elementos?, cuánto tardaría en ordenar el triple de elementos en una máquina 3 veces mas rápida?

15) Suponga que debe ordenar un vector por el método de Selección. Su entrada es de 2000 elementos y en su PC obtuvo un tiempo de 10 segundos. Si debiera ordenar un vector de 5000

entradas, cuál sería el tiempo estimado de ejecución? Cuál sería el tamaño máximo de vector que podría ordenar en 40 segundos?

Resuelva este problema reemplazando el método de ordenamiento por cada uno de los métodos vistos.

16) Analice la complejidad computacional del algoritmo de búsqueda binaria. Compárelo con el de búsqueda lineal.
17) Se tiene un arreglo desordenado cuyo registro se compone de 10% clave y 90% datos, si se sabe que en una máquina determinada el algoritmo de burbujeo tardó 10 segundos en ordenar 4000 elementos, cuánto tardaría en ordenar 8000 elementos?, Cuánto tardaría el algoritmo de inserción en ordenar ambos vectores en la misma máquina?
18) El algoritmo de selección ordenó un vector de 10000 enteros en 5 seg. Cuál será el tamaño del vector que espera ordenar en 20 seg. ? Justifique. Qué sucede si ordena ambos vectores en una máquina 5 veces más rápida?
19) Utilizando el algoritmo de inserción se ordenó en un tiempo T1, N1 elementos. En un tiempo de dos T1 se ordenó un vector de dos N1 elementos. Explique las condiciones bajo las cuales pudieron hacerse las pruebas.
20) Utilizando el algoritmo de QuickSort se ordenó un vector de 1 millón de elementos en un tiempo T1. Cuántos elementos espera ordenar en el doble de tiempo?

PRÁCTICA VI - Algoritmos de ordenamiento y búsqueda

1. Implemente los tres algoritmos elementales de ordenación**. (Selección, inserción y burbujeo**)

   a) Cuál de los tres métodos elementales de ordenación es el más rápido en una estructura de datos que ya está ordenada?

   b) Cuál de los tres métodos elementales es el más rápido para ordenar una estructura de datos que se encuentra en orden inverso?

   c) Ídem para una estructura desordenada al azar.

2. Implementar los siguientes algoritmos: **Shell, Cuenta de Distribución General y Particular**, **Quicksort y fusión.**

3. Cuáles de los algoritmos vistos son estables?

4. Cuáles de los 8 algoritmos vistos son sensibles a la entrada?

5. Grafique el tiempo de ejecución de cada algoritmo en función de la cantidad de elementos a ordenar y el tamaño de cada elemento. (Utilice una planilla de cálculo). Debe superar los 250000 elementos para los algoritmos no cuadráticos.
6. Implemente los algoritmos de búsqueda secuencial y binaria.
7. Qué criterios utilizaría para evaluar un algoritmo de ordenamiento? Descríbalos brevemente.
8. Escriba el pseudocódigo de cada algoritmo visto en clase.
9. Codifique una función para cada algoritmo visto en clase.
10. En qué casos utilizaría el algoritmo de Fusión y en qué casos QuickSort? Compare estos algoritmos.
11. Ordenar por cada algoritmo la siguiente cadena, mostrando cada paso.

ORDENAMIENTOS

12. Muestre los sucesivos cambios en la siguiente estructura al ordenar por Quicksort sin mejoras el siguiente vector:

ABADABA

13. Explique que mejoras se pueden aplicar en el algoritmo de Quicksort.
14. Se tiene un vector con 200000 registros de 5000by, de los cuales 4 forman la clave, y se desconoce el orden previo. Qué algoritmo utilizaría para ordenarlo? Por qué?
15. Qué criterios utilizaría para elegir entre usar el algoritmo de Inserción y QuickSort?
16. Enumere los motivos que pueden existir para elegir Inserción en vez de Shell.
17. Cuál es a su juicio el mejor algoritmo de ordenación? Justifique.

18. Un programador ejecuta dos algoritmos de ordenamiento: Quicksort y Selección. Al comparar los tiempos de ejecución para la misma entrada exclama: No puede ser, tardan lo mismo! Explíquele que pudo pasar y si es posible, solucione el problema.

19. Tengo una estructura a ordenar, la característica de la entrada se dice que es tenuemente perturbada; es decir que cada dato no está en su lugar definitivo pero sí cerca de él. Qué algoritmo utilizaría? Justifique.

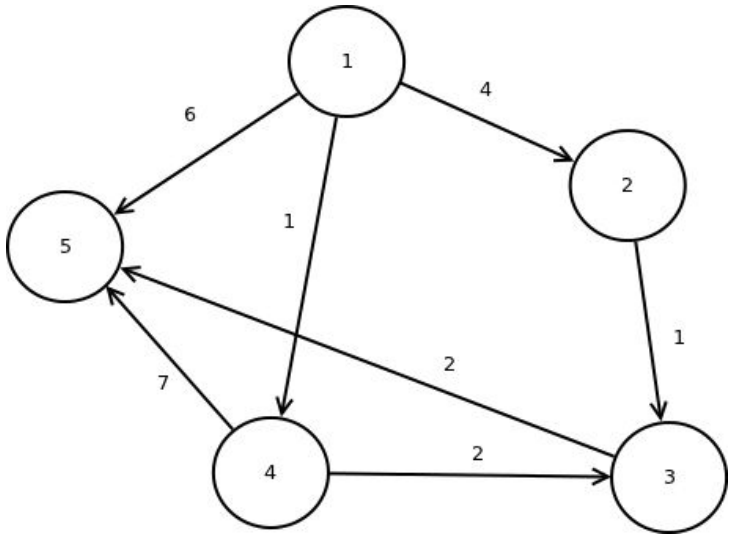
20. Cuáles de los algoritmos de ordenación aseguran el menor tiempo de ejecución? Cuál es su orden?

21. Tengo que ordenar una lista enlazada, cuáles de los algoritmos vistos en clase serían los candidatos a ser utilizados en una primera instancia? Si además se sabe que la entrada de los datos es aleatoria, y que la lista es grande; con cuál se quedaría?. Justifique.

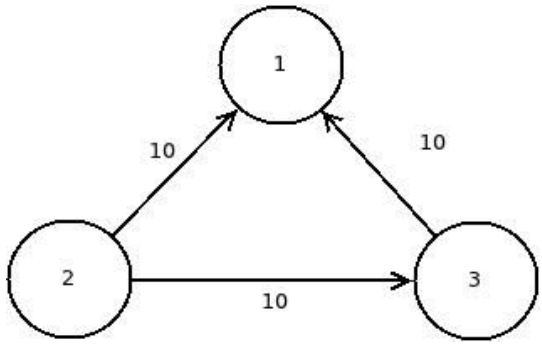
22. Escriba un programa para encontrar los k elementos más pequeños de un arreglo de longitud n. ¿Cuál es la complejidad de tiempo del programa?

PRÁCTICA VIII – Grafos

1. Escriba en pseudo código el algoritmo de Dijkstra



2. Desarrollar paso a paso por Warshall y exclusivamente en forma gráfica el siguiente grafo



3. En un archivo llamado distancias.in se encuentran guardadas las distancias por caminos entre distintas localidades. Todos los caminos son de doble mano. Se desea una aplicación que solicite por pantalla un origen y un destino; y a continuación proporcione la ruta más corta entre ambas localidades y su distancia. Por ejemplo, si el contenido del archivo es el siguiente:

   Buenos Aires Rosario 400
   Rosario Santa Fe 150
   Buenos Aires San Pedro 200
   San Pedro Rosario 200

   Esta podría ser una sesión de consulta:
   input:
   Origen: Buenos Aires
   Destino: Santa Fe

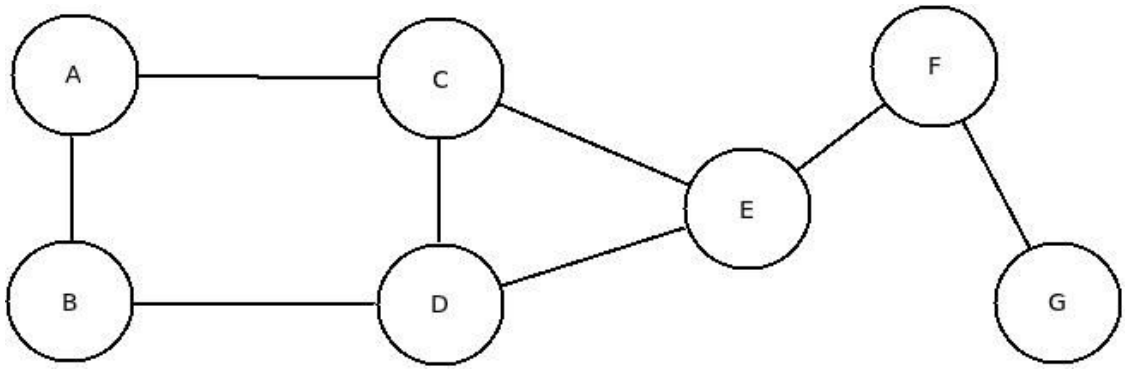
Output: La ruta más corta es de 550km: Buenos Aires - Rosario - Santa Fe

La aplicación debe gestionar correctamente las siguientes situaciones:
a) Alguna de las localidades especificadas puede no aparecer en el archivo, ya sea por un error del usuario o simplemente porque no conste en el.
b) Puede que, aun existiendo ambas, no exista una ruta debido a que por una circunstancia excepcional ambas se encuentren incomunicadas entre sí.
c. Indicar en que algoritmo está basada su aplicación

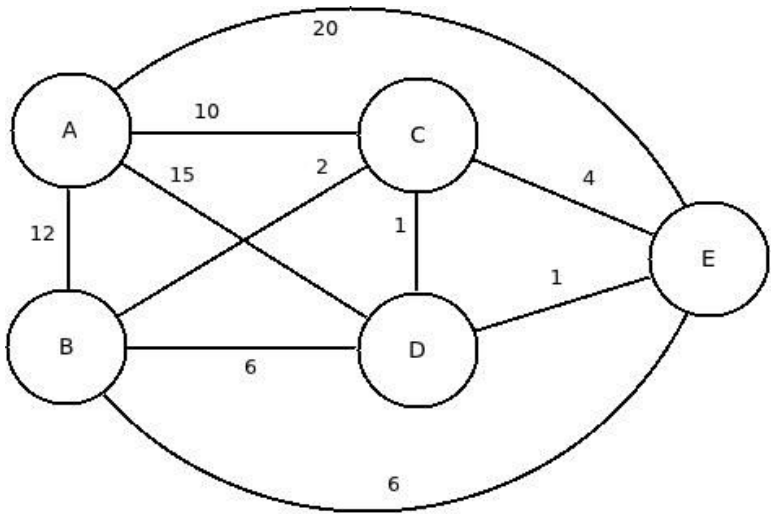
4. Trace un grafo dirigido que corresponda a la siguiente relación: x está relacionada con y si x-y es divisible entre 3 (considere que x e y son los enteros entre 1 y 12). Para el grafo obtenido:

    a. Indicar si el grafo contiene ciclos y cuáles son.
    b. Mostrar todas las trayectorias simples de 1 a 12.

5. Ej.3¿Cuál sería la representación del siguiente grafo por medio de estos tres elementos: una matriz de adyacencia, una lista de adyacencia y una lista de arcos?
6.



7. Dado el siguiente grafo mostrar paso a paso que hace el algoritmo de:
    a. Prim
    b. Kruskall



PRÁCTICA IX Resolución de problemas Nivel 3

Para cada uno de los problemas propuestos a continuación:

a. Realizar una ficha con las **estimaciones** y **métricas.**
b. Generar un **lote de pruebas** que asegure, con buenas probabilidades de éxito, el correcto funcionamiento del programa que intentará resolver.
c. **Documentar**, en un archivo aparte, cada uno de los casos que compone su lote de pruebas.
d. Realizar el **Diagrama de Clases**.
e. Codificar un **programa** que lo resuelva.

Problema 1: Batiendo al Minotauro (OIA 2007)
Problema 2: Usando la red de subterráneos (OIA 2007)
Problema 3: Rescatando a la princesa (OIA 2008)

PRÁCTICA X- PROLOG

1. Las siguientes clausulas corresponden al programa "menú" de un restaurante. El restaurante ofrece **menús completos** compuestos por una entrada, un plato principal y un postre. El plato principal puede ser carne o pescado.

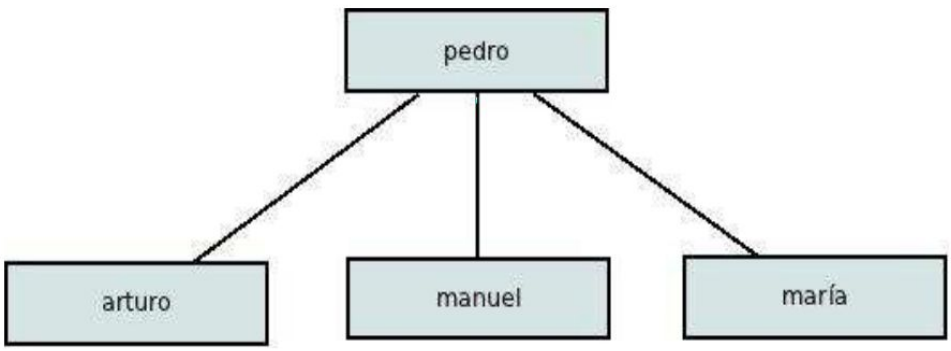
```
%clauses

entrada(paella).
entrada(gazpacho).
entrada(consomé).

carne(filete_de_cerdo).
carne(pollo_asado).

pescado(trucha).
pescado(bacalao).

postre(flan).
postre(helado).
postre(pastel).

%fin clauses
```

Implementar las reglas necesarias para formular las siguientes consultas en Prolog:

a) Cuáles son las comidas que ofrece el restaurante?
b) Cuáles son las comidas que tienen Consomé en las entradas?
c) Cuáles son las comidas que no contienen flan como postre?

2. Completar el programa "menú" de manera que una comida esté formada también por la elección de una bebida, a elegir entre vino, cerveza o agua mineral.

3. El árbol genealógico siguiente:



se describe con el programa Prolog:

```
hombre(pedro) .
```

```
hombre(manuel) .
hombre(arturo) .
mujer(maría).
padre(pedro, manuel).
padre(pedro, arturo).
padre(pedro, maría) .
```

A partir de estas afirmaciones, formular las reglas generales de:

```
niño(X,Y)//expresa que X es hijo o hija de Y.
hijo(X,Y)//expresa que X es un hijo varón de Y.
hija(x,y)//expresa que X es una hija de Y.
hermano-o-hermana(X,Y)//expresa que X es hermano o hermana de Y.
hermano(X,Y)//expresa que X es un hermano de Y.
hermana(X,Y)//expresa que X es una hermana de Y.
```

Nota: Un individuo no puede ser hermano ni hermana de sí mismo.

4. Una agencia de viajes propone a sus clientes viajes de una o dos semanas a Roma, Londres o Túnez.

El catálogo de la agencia contiene, para cada destino, el precio del transporte (con independencia de la duración) y el precio de una semana de estancia que varía según el destino y el nivel de comodidad elegidos: hotel, hostal o camping.

Escribir el conjunto de declaraciones que describen este catálogo (se muestra a continuación).

```
%clauses
transporte(roma,20).
transporte(londres,30).
transporte(tunez,10).

alojamiento(roma,hotel,50).
alojamiento(roma,hostal,30).
alojamiento(roma,camping,10).
alojamiento(londres,hotel,60).
alojamiento(londres,hostal,40).
alojamiento(londres,camping,20).
alojamiento(tunez,hotel,40).
alojamiento(tunez,hostal,20).
alojamiento(tunez,camping,5).
%finclauses
```

Expresar la relación `viaje(C,S,H,P)` que se interpreta por: el viaje a la ciudad *C* durante *S* semanas con estancia en H cuesta *P* pesos.

Completar con `viajeeconomico(C,S,H,P,Pmax)` que expresa que el costo P es menor quePmax pesos.

Se utilizarán las reglas secundarias:

```
multiplicar (P,X,Y):- P is X*Y.
sumar (S,X,Y):- S is X+Y.
menor (X,Y):-  X<Y
```

interpretables respectivamente por: P= X*Y; S= X+Y ; X<Y.

5. Una agencia matrimonial tiene un fichero de candidatos al matrimonio organizado según las declaraciones siguientes:

```
hombre(N,A,C,E).
mujer(N,A,C,E).
```

donde *n* es el nombre de un hombre o una mujer, *a* su altura (alta, media, baja), *c* el color de su cabello (rubio, castaño, pelirrojo, negro) y *e* su edad (joven,adulta,vieja).

```
gusta(N,M,L,S).
```

que indica que a la persona *n* le gusta el género de música *m* (clásica, pop, jazz), el género de literatura *L* (aventura, ciencia-ficción, policíaca), y practica el deporte *s* (tenis, natación, jogging).

```
busca(N,A,C,E).
```

que expresa que la persona *n* busca una pareja de altura *a,* con cabello color *c* y edad *e.* Se considera que dos personas *x* e *y* de sexos diferentes son adecuadas si *x* conviene a *y* e *y* conviene a *x.* Se dice que x conviene a *y* si *x* conviene físicamente a *y* (la altura, edad y color de cabello de *y* son los que busca *x*) y si, además, los gustos de *x* e *y* en música, literatura y deporte coinciden.

6. Con el programa "menú", dado en clase, formular en castellano las siguientes consultas Prolog y decir cuál es el resultado de la ejecución:

```
?- comida(E,P,D),!.
?comida(E,P,D),pescado (P),!.
?comida(E,P,D),!,pescado(P).
```

Analizar el comportamiento del operador ! (operador de corte o " cut").

7. Modificar el programa del ejercicio 1 (menú) para poder consultar cual es el menú completo que tiene menor cantidad de calorías.

8. Basado en el ejemplo de paises.pl visto en clase, complete la base de conocimientos

```
pais_superficie(P,A)
```

con todos los países de latinoamérica y codifique las reglas prolog que permitan encontrar el país de mayor superficie.

9. Listado anual de comisiones.

Dado el listado de vendedores y ventas semestrales se desea obtener el listado anual de comisiones. Las comisiones se liquidan de la siguiente manera:

20% del total vendido en el año para aquellos vendedores que hayan tenido ventas en ambos semestres y cada una de ellas supera los $ 20000.
10% del total vendido en el año para aquellos vendedores que hayan tenido ventas en ambos semestres, pero no superan los $ 20000 en alguno de estos.
5% del total vendido para los vendedores que no registran ventas en algún semestre

se dispone de los siguientes datos:

```
ventas1erSem(vendedor, importe).
.
.
ventas2doSem(vendedor, importe).
```

Nota: No todos los vendedores venden en ambos semestres, todos los importes son mayores que cero. En caso de no registrarse ventas en algún semestre, no figura la regla correspondiente para ese vendedor.

10. Recursividad en prolog:

1. Codifique en prolog las reglas necesarias para obtener el término N en la serie de Gauss
2. Codifique en prolog las reglas necesarias para obtener el término N en la serie de Fibonacci
3. Codifique en prolog las reglas necesarias para obtener el factorial de un número natural N.
4. Codifique en prolog las reglas necesarias para obtener el producto de dos numeros X e Y, aplicando sumas sucesivas.
5. Codifique en prolog las reglas necesarias para obtener la potencia N de un numero X aplicando multiplicaciones sucesivas.
6. Codifique en prolog las reglas necesarias para obtener la el cociente entre dos números a partir de restas sucesivas.
7. Definir la relación **mcd** (+X,+Y,?Z) que se verifique si Z es el máximo común divisor entre X e Y.
   Por ejemplo:

   ?d(10,15,X).
   X = 5

11. Realizar un programa PROLOG que permita encontrar a los antecesores de una persona.

PRÁCTICA X- HASKELL

1. Dado dos números enteros A y B, implemente una función que retorne la división entera ambos por el método de las restas sucesivas
2. Dado dos números enteros A y B encuentre el MCD (máximo común divisor) entre ambos.
3. Escribir una función para hallar la potencia de un número.
4. Definir una función menor que devuelve el menor de sus dos argumentos enteros.
5. Definir una función maxDeTres que devuelve el máximo de sus argumentos enteros
6. Implemente una función recursiva para calcular el factorial de un número

ANEXO 1

ASPECTOS BÁSICOS DE LA PROGRAMACIÓN OO[1]

Para resumir los principios básicos de la programación orientada a objetos:

ESTRUCTURA DEL PROGRAMA
- Un programa es un conjunto de objetos (o tareas) que se comunican entre sí.
- Los objetos son principalmente unidades estructurales y procesos o tareas.

OBJETOS
- Un objeto agrupa datos y procedimientos.
- Un objeto recuerda información (datos).
- Un objeto procesa información (procedimientos).
- Cada objeto tienen un ciclo vital.

CLASES
- Las clases son esquemas para los objetos.
- Los objetos se crean a partir de especificaciones de clases.
- Las clases son tipos de datos más las tareas que manejan.

COMUNICACIÓN
- Los objetos envían y reciben mensajes.
- Los mensajes llevan información.
- Los mensajes llevan procesamiento.
- Los mensajes son llamadas de procedimiento.

PROCESAMIENTO
- Los objetos contienen procedimientos.
- Los procedimientos contenidos en el interior de los objetos son métodos.
- Los métodos procesan información.
- Los métodos son código ejecutable.

[1] de la bibliografía de la cátedra : Greg Voss, 1994. Programación orientada a objetos. MCGraw-Hill

HERENCIA, POLIMORFISMO Y OOP

La herencia y el polimorfismo son las claves de la programación orientada a objetos. En la lista que sigue se resumen las características básicas de herencia y polimorfismo que se presenta en la bibliografía adjunta.

CLASIFICACIÓN

- La clasificación impone un orden.
- Los objetos se clasifican de acuerdo con las propiedades compartidas.
- Los objetos pueden tener distintas instrumentaciones y aún así compartir propiedades.
- Las propiedades de los objetos se resaltan o ignoran dependiendo del uso que se pretenda darles.
- Los objetos con propiedades semejantes pertenecen a la misma clase.
- Las propiedades de un objeto se refiere tanto a su estructura (si la tiene) como a sus funciones. Se podrá dar preferencia a una, a otra o a ambas a la vez.

HERENCIA

Programación jerárquica

- Las jerarquías de tipo expresan propiedades comunes de objetos.
- Las jerarquías de tipo expresan las diferencias de caso especial entre objetos similares.
- Las jerarquías de tipo son jerarquías de clase.
- Las clases ancestro definen propiedades comunes.
- Las clases descendientes definen diferencias de caso especial.

Clase base

- Un ancestro común en una jerarquía de clase es una clase base.
- Las clases base definen un protocolo de comunicación común para los descendientes.
- Los objetos de una clase base común responden a los mismos mensajes.

Herencia

La herencia es un mecanismo que permite la definición de una nueva clase a partir de otra clase ya existente. La herencia está fuertemente ligada al concepto de reutilización de código.
Existen dos tipos de herencia: Simple y Múltiple.

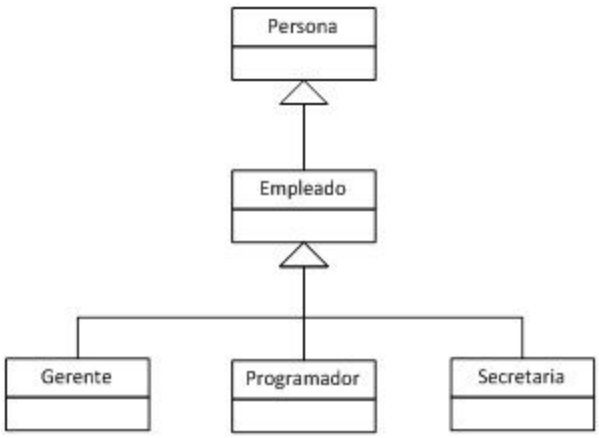
- Simple: Indica que se pueden definir nuevas clases a partir de una clase inicial.
- Múltiple: Se pueden definir nuevas clases a partir de dos ó más clases iniciales.

Java sólo permite herencia simple.

El concepto de herencia conduce a una estructura jerárquica de clases o parecida a una estructura de árbol. Cada clase tiene sólo una clase padre. La clase padre de cualquier clase es conocida como superclase. La clase hija de una superclase es llamada subclase.

Nos encontramos ante herencia cuando entre dos clases podemos encontrar la relación “es un” entre ambas.
Características:

- Reutilización de código.
- Se puede aplicar el concepto de sobrecarga y en particular el de sobre-escritura de métodos.
- Utilización de ***super*** en su implementación.
- Se representa mediante la palabra reservada ***extends*** en su definición.




```
public class Persona
{
      ...

      public Persona(){}
}

public class Empleado extends Persona
{
      ...

      public Empleado(){super();}
}

public class Gerente extends Empleado
{
      ...
      public Gerente(){super();}
}
```

POLIMORFISMO Y NEXOS DINÁMICOS

La comunicación regula la estructura orientada a objetos

- La programación orientada a objetos propicia la especificación formal del protocolo de comunicación.
- Los objetos se comunican a través de interfaces públicas.
- Los lenguajes de programación orientados a objetos detectan la violación del protocolo de interfaz.

Nexos dinámicos

- Los nexos dinámicos despachan llamadas generales para métodos específicos.
- Los nexos dinámicos seleccionan métodos específicos con base en el tipo de objetos.
- Los nexos dinámicos resuelven las llamadas en el momento del procesamiento.
- Los nexos dinámicos eliminan la información de tipo mantenida por el programador.
- Los nexos dinámicos eliminan los interruptores de despacho de método manejados por el programador.

Colección y recipientes polimórficos

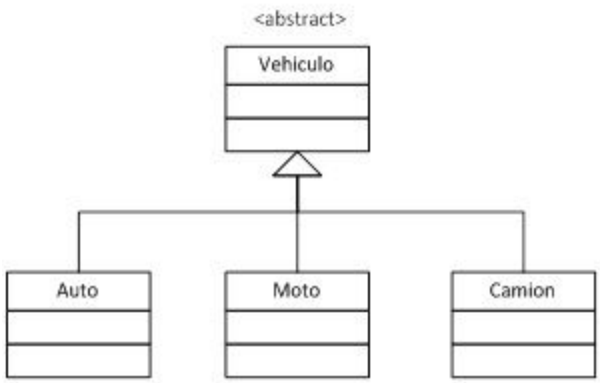
- Las colecciones y los recipientes contienen objetos sin nombre.
- Los objetos con una clase base común pueden colocarse en la misma colección.
- Al despacho de método lo maneja el receptor en vez del emisor.
- El polimorfismo generaliza la comunicación y simplifica el mantenimiento.

Polimorfismo

Se denomina polimorfismo a la habilidad de una variable por referencia de cambiar su comportamiento en función de que instancia de objeto posee. Dicho de otra forma, el polimorfismo consiste en conseguir que un objeto de una clase se comporte como un objeto de cualquiera de sus subclases, dependiendo de la forma de llamar a los métodos de dicha clase o subclase.

Clase Abstracta

Una clase abstracta es una que sirve para definir un concepto que no tiene sentido por sí mismo, sino como agrupador de otros conceptos relacionados. Por lo tanto, sirve para definir subclases. (Una plantilla). Como consecuencia, no puede instanciarse.
Por definición una clase abstracta debe tener por lo menos un método abstracto.
Un método abstracto es aquel sólo tenemos su defnición en la clase asbtracta y que su implementación se realiza en las subclases que la heredan.
Una clase abstracta puede tener además métodos no abstractos (Constructores, etc…)
Se hace uso del concepto de polimorfismo y herencia.

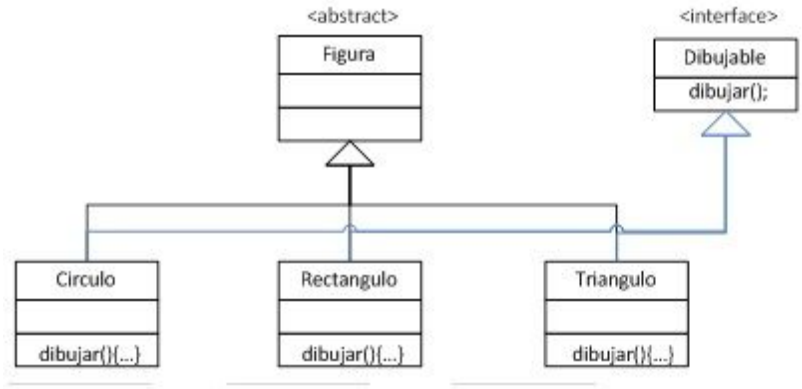



```
public abstract class Vehiculo
{
      ...
      public Vehiculo(){}

      public abstract int getCantRuedas();
}

public class Auto extends Vehiculo
{
      ...
      public Auto(){super();}
      public int getCantRuedas(){return 4;}
```

```
}
```

Interface

Una interface es una clase abstracta pura, esto significa que todos sus métodos son abstractos.
Sirve para definir un comportamiento.
Puede contener definiciones de constantes.
Aquellas clases que la implementan deben implementar sus métodos.
No puede instanciarse.
Se puede implementar más de una interface.
Hace uso del concepto de polimorfismo.




```
public interface Dibujable
{
      public void dibujar();
}

public abstract class Figura
{
      ...
      public double area();

public class Circulo extends Figura implements Dibujable
{
      ...
      public double area(){...}
      public void dibujar(){...}
}

public class Rectangulo extends Figura implements Dibujable
{
      ...
      public double area(){...}
      public void dibujar(){...}
}
```

APOYO DEL LENGUAJE A TIPOS DEFINIDOS POR EL USUARIO

Los tipos definidos por el usuario son la clave para la abstracción de datos. La abstracción de datos permite que sistemas complejos se separen en componentes activos de procesamiento, los cuales ocultan el detalle de la instrumentación y minimizan las conexiones entre unos y otros. Las conexiones se minimizan en un sistema complejo al definir interfaces restrictivas para los componentes a fin de asegurar que la instrumentación de un componente nunca dependa de la instrumentación interna de otro componente.

Permitir que los usuarios definan nuevos tipos de objetos para crear componentes complejos y tratar las instancias de esos tipos como objetos que se manipulan a través de la transferencia de mensajes se denomina programación *basada en objetos*. Los tipos definidos por el usuario son la característica clave del lenguaje necesaria para dar apoyo a la programación basada en objetos.

Sin embargo, para la programación *orientada a objetos* se requiere otra herramienta más. La programación orientada a objetos da más estructura organizacional a los sistemas complejos, al identificar las características comunes en un conjunto de objetos y permitir que estos aspectos comunes se expresen en un lenguaje de programación.

Al agrupar las clases en una jerarquía y clasificar los tipos de acuerdo con las propiedades compartidas se logra manejar en conjunto lo común de los tipos de objetos.

APOYO DEL LENGUAJE PARA LA PROGRAMACIÓN ORIENTADA A OBJETOS

- Además de tipos abstractos de datos, la programación orientada a objetos requiere la habilidad para organizar los objetos de acuerdo con propiedades y protocolos de comunicación compartidos.

- La comunicación polimórfica es posible si una clase base común a todas las clases de la jerarquía define el protocolo básico de comunicación al declarar un conjunto de métodos comunes como métodos virtuales.

- Las características del lenguaje que se requieren para dar apoyo a la clasificación y la comunicación polimórfica son la herencia y los nexos dinámicos. Tanto C++, como Pascal orientado a objetos, Smalltalk y Actor dan apoyo a la herencia y a los nexos dinámicos.

- En Smaltalk y Actor, todos los métodos son virtuales (dinámicos) por omisión. En C++ y en Pascal, los métodos son estáticos por omisión.

- Los nexos dinámicos entre mensajes y métodos pueden especificarse al declarar los métodos como virtuales en las clases base que definen el protocolo de comunicación tanto en C++ como en Pascal orientado a objetos.

- Por lo general, la programación orientada a objetos implica colecciones de objetos provenientes de una clase dada, tales como ventanas, formas, controles, flujos o archivos.

- Los poderosos sistemas de manejo por ventana dependen en mucho de la herencia y del paso polimórfico de mensajes para simplificar la interfaz de programación.

- La principal diferencia entre las bibliotecas de clase para la programación orientada a objetos y las bibliotecas tradicionales que proporcionan C y Pascal es que las bibliotecas orientadas a objetos dan apoyo al paso polimórfico de mensajes y permiten al usuario construír nuevas clases de caso especial basadas en las clases que se proporcionan en la biblioteca de clase.

- El acceso al código fuente para la biblioteca no es importantes, porque los nexos dinámicos en una biblioteca bien diseñada dan al usuario control total sobre el grado de especialización que requieren los nuevos tipos de objetos.

- Los marcos de aplicación son tipos especiales de bibliotecas polimórficas que se utilizan para construir interfaces de programación destinadas a programas de aplicación. Los marcos de aplicación evolucionaron a partir de los ambientes de programación orientada a objetos y de las bibliotecas de clase en general. En particular, los ambientes de programación de Smaltalk y Actor son importantes porque inspiraron la idea de los marcos de aplicación que ahora se están adaptando a C++ y a Pascal orientado a objetos.

ANEXO 2

La clase String de Java[2]

Métodos explicados:

int length() // proporciona la longitud de la cadena.

int compareTo(String hola) //devuelve cero si las cadenas son iguales; negativo si es < y positivo si es > ; siempre por orden alfabético

boolean startsWith(String)// devuelve **true** o **false**, según que el String comienza o no con dicho prefijo.

int indexOf(char)// devuelve la posición de la primera ocurrencia del carácter dentro del String

int indexOf (String) //devuelve la primera ocurrencia de un substring dentro del string

boolean equals("hola2") // devuelve **true** cuando los dos strings son iguales o **false** si son distintos

String substring(posición inicial)/// extrae un substring desde una posición determinada hasta el final del string.

String substring(posición inicial, posición final)/// extrae un substring especificando la posición de comienzo y la del final.
String str=String.valueOf(valor)// convierte un número en String.

String: StringApp.java

Dentro de un objeto de la clases *String* o *StringBuffer*, Java crea un array de caracteres de una forma similar a como lo hace el lenguaje C++. A este array se accede a través de las funciones miembro de la clase.
Los strings u objetos de la clase *String* se pueden crear explícitamente o implícitamente. Para crear un string implícitamente basta poner una cadena de caracteres entre comillas dobles. Por ejemplo, cuando se escribe

```
System.out.println("El primer programa");
```

Java crea un objeto de la clase *String* automáticamente.
Para crear un string explícitamente escribimos

```
String str=new String("El primer programa");
```

También se puede escribir, alternativamente

```
String str="El primer programa";
```

Para crear un string nulo se puede hacer de estas dos formas

[2]Material extraído de: http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/string.htm

```
String str="";
String str=new String();
```

Un string nulo es aquél que no contiene caracteres, pero es un objeto de la clase *String*. Sin embargo,

```
String str;
```

está declarando un objeto *str* de la clase *String*, pero aún no se ha creado ningún objeto de esta clase.

Cómo se obtiene información acerca del string

Una vez creado un objeto de la clase *String* podemos obtener información relevante acerca del objeto a través de las funciones miembro.
Para obtener la longitud, número de caracteres que guarda un string se llama a la función miembro *length*.

```
String str="El primer programa";
int longitud=str.length();
```

Podemos conocer si un string comienza con un determinado prefijo, llamando al método *startsWith*, que devuelve **true** o **false**, según que el string comience o no por dicho prefijo

```
String str="El primer programa";
boolean resultado=str.startsWith("El");
```

En este ejemplo la variable resultado tomará el valor **true**.
De modo similar, podemos saber si un string finaliza con un conjunto dado de caracteres, mediante la función miembro *endsWith*.

```
String str="El primer programa";
boolean resultado=str.endsWith("programa");
```

Si se quiere obtener la posición de la primera ocurrencia de la letra p, se usa la función *indexOf*.

```
String str="El primer programa";
int pos=str.indexOf('p');
```

Para obtener las sucesivas posiciones de la letra p, se llama a otra versión de la misma función

```
pos=str.indexOf('p', pos+1);
```

El segundo argumento le dice a la función *indexOf* que empiece a buscar la primera ocurrencia de la letra p a partir de la posición *pos+1*.
Otra versión de *indexOf* busca la primera ocurrencia de un substring dentro del string.

```
String str="El primer programa";
int pos=str.indexOf("pro");
```

Vemos que una clase puede definir varias funciones miembro con el mismo nombre pero que tienen distinto número de parámetros o de distinto tipo.

Comparación de strings

equals: EqualsApp.java
La comparación de strings nos da la oportunidad de distinguir entre el operador lógico == y la función miembro *equals* de la clase *String*. En el siguiente código

```
String str1="El lenguaje Java";
String str2=new String("El lenguaje Java");
if(str1==str2){
        System.out.println("Los mismos objetos");
}else{
        System.out.println("Distintos objetos");
}
if(str1.equals(str2)){
        System.out.println("El mismo contenido");
}else{
        System.out.println("Distinto contenido");
}
```

Esta porción de código devolverá que *str1* y *str2* son dstintos objetos pero con el mismo contenido. *str1* y *str2* ocupan posiciones distintas en memoria pero guradan los mismos datos.
Cambiemos la segunda sentencia y escribamos

```
String str1="El lenguaje Java";
String str2=str1;
System.out.prinln("Son el mimso objeto "+(str1==str2);
```

Los objetos *str1* y *str2* guardan la misma referencia al objeto de la clase *String* creado. La expresión *(str1==str2*) devolverá **true**.
Así pues, el método *equals* compara un string con un objeto cualquiera que puede ser otro string, y devuelve **true** cuando dos strings son iguales o **false** si son distintos.

```
String str="El lenguaje Java";
boolean resultado=str.equals("El lenguaje Java");
```

La variable *resultado* tomará el valor **true.**
La función miembro *comapareTo* devuelve un entero menor que cero si el objeto string es menor (en orden alfabético) que el string dado, cero si son iguales, y mayor que cero si el objeto string es mayor que el string dado.

```
String str="Tomás";
int resultado=str.compareTo("Alberto");
```

La variable entera *resultado* tomará un valor mayor que cero, ya que Tomás está después de Alberto en orden alfabético.

```
String str="Alberto";
int resultado=str.compareTo("Tomás");
```

La variable entera *resultado* tomará un valor menor que cero, ya que Alberto está antes que Tomás en orden alfabético.
NOTA: nunca utilice == para comparar dos cadenas

Extraer un substring de un string

En muchas ocasiones es necesario extraer una porción o substring de un string dado. Para este propósito hay una función miembro de la clase *String* denominada *substring*.
Para extraer un substring desde una posición determinada hasta el final del string escribimos

```
String str="El lenguaje Java";
String subStr=str.substring(12);
```

Se obtendrá el substring "Java".
Una segunda versión de la función miembro *substring*, nos permite extraer un substring especificando la posición de comienzo y la el final.

```
String str="El lenguaje Java";
String subStr=str.substring(3, 11);
```

Se obtendrá el substring "lenguaje". Recuérdese, que las posiciones se empiezan a contar desde cero.

Convertir un número a string

Para convertir un número en string se emplea la función miembro estática *valueOf* (más adelante explicaremos este tipo de funciones).

```
int valor=10;
String str=String.valueOf(valor);
```

La clase *String* proporciona versiones de *valueOf* para convertir los datos primitivos: **int**, **long**, **float**, **double**.
Esta función se emplea mucho cuando programamos applets, por ejemplo, cuando queremos mostrar el resultado de un cálculo en el área de trabajo de la ventana o en un control de edición.

Convertir un string en número

Cuando introducimos caracteres en un control de edición a veces es inevitable que aparezcan espacios ya sea al comienzo o al final. Para eliminar estos espacios tenemos la función miembro *trim*

```
String str="  12 ";
String str1=str.trim();
```

Para convertir un string en número entero, primero quitamos los espacios en blanco al principio y al final y luego, llamamos a la función miembro estática *parseInt* de la clase *Integer* (clase envolvente que describe los números enteros)

```
String str="  12 ";
int numero=Integer.parseInt(str.trim());
```

Para convertir un string en número decimal (**double**) se requieren dos pasos: convertir el string en un objeto de la clase envolvente *Double*, mediante la función miembro estática *valueOf*, y a

continuación convertir el objeto de la clase *Double* en un tipo primitivo **double** mediante la función *doubleValue*

```
String str="12.35 ";
double numero=Double.valueOf(str).doubleValue();
```

Se puede hacer el mismo procedimiento para convertir un string a número entero

```
String str="12";
int numero=Integer.valueOf(str).intValue();
```

La clase StringBuffer

En la sección dedicada a los operadores hemos visto que es posible concatenar cadenas de caracteres, es, decir, objetos de la clase *String*. Ahora bien, los objetos de la clase *String* son constantes lo cual significa que por defecto, solamente se pueden crear y leer pero no se pueden modificar.
Imaginemos una función miembro a la cual se le pasa un array de cadenas de caracteres. Los arrays se estudiarán en la siguiente página.

```
public class CrearMensaje{
	public String getMensaje(String[] palabras){
		String mensaje="";
		for(int i=0; i<palabras.length; i++){
			mensaje+=" "+palabras[i];
		}
		return mensaje;
	}
//...
}
```

Cada vez que se añade una nueva palabra, se reserva una nueva porción de memoria y se desecha la vieja porción de memoria que es más pequeña (una palabra menos) para que sea liberada por elrecolector de basura (grabage collector). Si el bucle se realiza 1000 veces, habrá 1000 porciones de memoria que el recolector de basura ha de identificar y liberar.
Para evitar este trabajo extra al recolector de basura, se puede emplear la clase *StringBuffer* que nos permite crear objetos dinámicos, que pueden modificarse.

```
public class CrearMensaje{
	public String getMensaje(String[] palabras){
		StringBuffer mensaje=new StringBuffer();
		for(int i=0; i<palabras.length; i++){
			mensaje.append("  ");
			mensaje.append(palabras[i]);
		}
		return mensaje.toString();
	}
//...
}
```

La función *append* incrementa la memoria reservada para el objeto *mensaje* con una palabra más sin crear nueva memoria, cada vez que se ejecuta el bucle. La función toString, que veremos más adelante, convierte un objeto en una cadena de caracteres.