Tema 6 Abstracción y generalización

Interfaces

1. Defínanse las interfaces “Alimentarse”, “Carrera” y “Natacion”.

PAQUETE: clases.interfaces INTERFAZ: Alimentarse		
TIPO	MIEMBROS	
int	GRAMOS_POR_KILO	1000
float	KILOS_POR_GRAMO	0.001F
Object	comer()	
Object	comer(float kg)	
Object	beber()	
Object	beber(float litros)	

PAQUETE: clases.interfaces INTERFAZ: Carrera		
TIPO	MIEMBROS	
int	METROS_POR_KILOMETRO	1000
float	KILOMETROS_POR_METRO	0.001F
Object	correr()	
Object	correr(float metros)	

PAQUETE: clases.interfaces.deportes INTERFAZ: Natacion		
TIPO	MIEMBROS	
Object	nadar()	
Object	nadar(float metros)	

2. Defínase una interfaz de nombre “Acumulable” en “clases/interfaces” con los siguientes miembros:

a) PI: Constante de clase de tipo real con el valor de 3.1416
b) Método de objeto acumular() →Object
c) Método de objeto acumular(Object valor) →Object
d) Método de objeto esNulo() → boolean

3. Defínase una interfaz de nombre “Calculo” en “clases/interfaces” con los siguientes miembros:

a) Método de objeto sumar(Object valor) →Object
b) Método de objeto restar(Object valor) →Object

4. Defínase una interfaz de nombre "Operable" en "clases/interfaces" con los siguientes miembros:

PAQUETE: clases.interfaces INTERFAZ: Operable		
TIPO	MIEMBROS	
Object	sumar(Object a, Object b)	
Object	restar(Object a, Object b)	
Object	multiplicar(Object a, Object b)	

5. Defínase la interfaz "ICoche" con los siguientes miembros:

PAQUETE: clases.interfaces INTERFAZ: ICoche		
TIPO	MIEMBROS	
Enumerado	Colores	
String	getMatricula()	
Object	setMatricula(String)	
Colores	getColor()	
Object	setColor(Colores color)	
double	getVelocidad()	
Object	setVelocidad(double velocidad)	
Object	acelerar()	
Object	frenar()	
Object	clone()	

- Colores → Enumerado con los valores "blanco", "negro", "azul", "verde", "amarillo", "rosa", "naranja" y "rojo"

Herencia de interfaces

6. Defínase la interfaz "Triatlon".

PAQUETE: clases.interfaces.deportes INTERFAZ: Triatlon EXTIENDE: Carrera, Natacion		
TIPO	MIEMBROS	
Object	pedalear()	
Object	pedalear(float km)	

7. Defínase una interfaz de nombre "Aritmetica" en "clases/interfaces" que herede de "Acumulable" y de "Calculo" con los siguientes miembros:

a) Método de objeto multiplicar(Object valor) →Object

Implementación de interfaces

8. Considérese las implementaciones de las interfaces "Alimentarse" y "Carrera" en la clase Persona del siguiente modo:

- Object correr(float km) → Disminuirá el peso de la persona en el número de kilómetros recorridos multiplicado por 0.05. Se lanzará un error si el número de kilómetros recorridos fuera menor o igual que 0. Se devuelve el objeto que invoca el método.
- Object correr() → Se invoca el método anterior con el valor de 1 kilometro.
- Object comer(float kg) → Aumentará el peso de la persona en el número de kilogramos comidos multiplicado por 0.3. Se lanzará un error si el número de kilogramos comidos fuera menor o igual que 0. Se devuelve el objeto que invoca el método.
- Object comer() → Se invoca el método anterior con el valor de 1 kilogramo.
- Object beber(float kg) → Aumentará el peso de la persona en el número de litros bebidos multiplicado por 0.1. Se lanzará un error si el número de litros bebidos fuera menor o igual que 0. Se devuelve el objeto que invoca el método.
- Object beber() → Se invoca el método anterior con el valor de 1 litro.

Hágase la siguiente aplicación:

a) Crear una persona inicializando su nombre.
b) Establecer a 55 kg el peso de dicha persona.
c) La persona se come 500 gramos.
d) Se muestra el nuevo peso de la persona.
e) La persona bebe 0,75 litros.
f) Se muestra el nuevo peso de la persona.
g) La persona corre 3 kilómetros.
h) Se muestra el nuevo peso de la persona.

(PersonaInterfaz)

9. Impleméntese la interfaz de "Acumulable" en la clase "Rectangulo" del siguiente modo:

a) esNulo() → Devuelve verdadero si el rectángulo tiene como ancho y alto el mínimo posible (2).
b) acumular() → Incrementa tanto el ancho como el alto del rectángulo en una unidad.
c) acumular(Object) → Si el objeto a acumular no es un rectángulo se lanza una excepción de usuario interna de nombre "IllegalArgumentException"

con el mensaje "El parámetro no es de tipo rectángulo"; de ser un rectángulo, se incrementa el rectángulo que invoca al método en el ancho y alto del rectángulo pasado como parámetro.

Hágase una aplicación que realice lo siguiente:

a) Inicialícese un rectángulo de 3 x 4.
b) Inicialícese un cuadrado de lado 4.
c) Muéstrese la dimensión (saltando de línea) y dibújese el rectángulo.
d) Acumúlese al rectángulo uno tanto al ancho como al alto.
e) Muéstrese la dimensión (saltando de línea) y dibújese el rectángulo.
f) Acumúlese al rectángulo el cuadrado.
g) Muéstrese la dimensión (saltando de línea) y dibújese el rectángulo.

(RectanguloInterfaz)

10. Impleméntese la interfaz de "Acumulable" y "Calculo" en la clase "Caja" del siguiente modo:

a) Los métodos de la interfaz "Acumulable" definidos en "Rectángulo" no se sobrescribirán.
b) sumar(Object) → Si el objeto a acumular no es una caja se lanza una excepción de usuario interna de nombre "IllegalArgumentException" con el mensaje "El parámetro no es de tipo caja"; de ser una caja, se sumará al ancho/alto/largo de la caja que invoca el método, el ancho/alto/largo de la caja pasado como parámetro.
c) restar (Object) → Si el objeto a acumular no es una caja se lanza una excepción de usuario interna de nombre "IllegalArgumentoException" con el mensaje "El parámetro no es de tipo caja"; de ser una caja, se restará al ancho/alto/largo de la caja que invoca el método, el ancho/alto/largo de la caja pasado como parámetro. En el caso de que alguna de las medidas fuera inferior al mínimo posible (2), se dejará como valor el del mínimo posible.

Hágase la siguiente aplicación:

a) Inicialícese una caja "c" de 12 x 5 x 8
d) Inicialícese una caja "c1" de 17 x 43 x 45
e) Inicialícese una caja "c2" de 20 x 3 x 45
f) Muéstrese la dimensión de "c", "c1" y "c2" (véase la salida)
g) Obténgase "c1 + c" y muéstrese la dimensión del nuevo "c1" (véase la salida)
h) Obténgase "c2 - c" y muéstrese la dimensión del nuevo "c2" (véase la salida)

(CajaInterfaz)

11. Impleméntese la interfaz de "Aritmética" en la clase "Cuadrado" del siguiente modo:

a) Los métodos de la interfaz "Acumulable" definidos en "Rectángulo" no se sobrescribirán.
b) sumar(Object) → Si el objeto a acumular no es un cuadrado se lanza una excepción de usuario interna de nombre "IllegalArgumentException" con el mensaje "El parámetro no es de tipo cuadrado"; de ser un cuadrado, se sumará al lado del cuadrado que invoca el método, el lado del cuadrado pasado como parámetro.
c) restar(Object) → Si el objeto a acumular no es un cuadrado se lanza una excepción de usuario interna de nombre "IllegalArgumentException" con el mensaje "El parámetro no es de tipo cuadrado"; de ser un cuadrado, se restará al lado del cuadrado que invoca el método, el lado del cuadrado pasado como parámetro. En el caso de que el nuevo lado fuera inferior al mínimo posible (2), se dejará como valor el del mínimo posible.
d) multiplicar(Object) → Si el objeto a acumular no es un cuadrado se lanza una excepción de usuario interna de nombre "IllegalArgumentException" con el mensaje "El parámetro no es de tipo cuadrado"; de ser un cuadrado, se multiplicará al lado del cuadrado que invoca el método, el lado del cuadrado pasado como parámetro.

Hágase una aplicación en la que se realice lo siguiente:
a) Se inicialice 3 cuadrados de lado 6 y un cuarto cuadrado de lado 2.
b) Al primer cuadrado se le suma el cuarto.
c) Al segundo cuadrado se le resta el cuarto
d) Al tercer cuadrado se le multiplica por el cuarto.
e) Se muestra la dimensión (saltando de línea) y se dibujan los tres primeros cuadrados.

(CuadradoInterfaz)

Variables de tipo interfaz

12. Hágase una aplicación en la que se realice lo siguiente:
a) Defínase una variable de tipo la interfaz "Acumulable".
b) Asígnese a dicha variable un rectángulo de 2x3.
c) Acumúlese a tal rectángulo una unidad tanto a su ancho como a su alto.
d) Dibújese dicho rectángulo.

(VariableInterfaz)

13. Hágase una aplicación en la que se realice lo siguiente:
a) Defínase dos variables de tipo Rectangulo de nombres "r1" y "r2" con el valor de un rectángulo de 6x3 y un cuadro con valor por defecto.
b) Defínase una variable de tipo un array de 4 elementos de tipo interfaz "Acumulable" inicializada con los siguientes valores: "r1", un rectángulo de 2x4, un cuadrado de 3x3 y un rectángulo por defecto.

c) Acumúlese a cada uno de los rectángulos contenidos en el array el cuadrado "r2".
d) Muéstrese la dimensión de cada uno de los rectángulos contenidos en el array.

(VariableArrayInterfaz)

Dependencias entre clases

14. Defínanse en la clase Persona un atributo de tipo ICoche con sus métodos get y set. Se modificará el método "ToString()" para que se muestre la conversión a cadena de ICoche.

Clases abstractas

15. Defínase la siguiente clase:

PAQUETE: clases CLASE ABSTRACTA: Operaciones			
ACCESO	TIPO	MIEMBROS	
A publico	Object	sumar(Object, Object)	
A publico	Object	restar(Object, Object)	
A publico	Object	multiplicar(Object, Object)	

16. Defínanse la siguiente clase abstracta "Animal"

PAQUETE: clases CLASE ABSTRACTA: Animal			
ACCESO	TIPO	MIEMBROS	
constante publica	int	NUMERO_DE_PATAS	
privado	String	nombre	"Desconocido"
privado	float	peso	0.3F
publico	String	getNombre()	
publico	float	getPeso()	
publico	Animal	setNombre(String nombre)	
publico	Animal	setPeso(float peso)	
protegido		Animal(String nombre, float peso)	
protegido		Animal(float peso, String nombre)	
protegido		Animal(float peso)	
protegido		Animal(String nombre)	
protegido		Animal()	
abstracto publico	Animal	comer()	
abstracto publico	Animal	comer(int)	
abstracto protected	int	setNumeroDePatas()	
REEMPLAZAMIENTO DE METODOS			

publico	String	toString()	

- NUMERO_DE_PATAS → Almacena el número de patas que tiene el animal
- nombre → Nombre del animal
- peso → Peso del animal
- getNombre() → Devuelve el nombre del animal
- getPeso() → Devuelve el peso del animal
- setNombre(String nombre) → Establece el valor del atributo "nombre". Se lanza un error si no es una palabra en mayúsculas. Se devuelve el objeto que invoca el método.
- setPeso(float peso) → Establece el valor del atributo "peso". Se lanza un error si no es un valor positivo. Se devuelve el objeto que invoca el método.
- Animal(String nombre, float peso) y Animal(float peso, String nombre)→ Crena un objeto "Animal" de nombre "nombre" y peso "peso"
- Animal(String nombre) → Crena un objeto "Animal" de nombre "nombre"
- Animal(float peso) → Crena un objeto "Animal" con el peso "peso"
- Animal() → Crea un objeto "Animal" con los valores por defecto para cada atributo.
- toString() → Devuelve una cadena con la descripción de cada atributo y su valor en líneas distintas.

17. Defínase una clase abstracta de nombre "Figura" en "clases/figuras" que implementa las interfaces "Acumulable" y "Comparable" con los siguientes miembros:

a) VERTICES → Constante de objeto de tipo entero. No puede tomar valores menores que 3
b) Un excepción de usuario con el nombre "VerticesInvalidosException".
c) int setVERTICES(int) → Método protegido de objeto que válida el valor del número de vértices que tiene la figura. Si se intenta asignar un valor incorrecto se lanzará la excepción "VerticesInvalidosException" con el mensaje "El numero de vertices debe ser superior a 2". De ser válido el número de vértices, se devuelve dicho valor.
d) Figura(int) → Constructor protegido para inicializar el número de vértices.
e) int perimetro() → Método abstracto (público)
f) int area() → Método abstracto (público)
g) Sobrescribir todos los métodos de las dos interfaces declarándolos como métodos abstractos (públicos)
h) Sobrescribir el método "toString".

(FiguraAbstracto)

Herencia de clases abstractas

18. Deívese de la clase “Animal” las clases “Cuadrupedo” y “Perro”

PAQUETE: clases CLASE ABSTRACTA: Cuadrupedo EXTIENDE: Animal			
ACCESO	TIPO	MIEMBROS	
privado	float	altura	0,1F
protegido		Cuadrupedol(String nombre, float peso, float altura)	
protegido		Cuadrupedol(String nombre, float peso)	
protegido		Cuadrupedo()	
abstracto publico	Cuadrupedo	comer()	
abstracto publico	Cuadrupedo	comer(int kg)	
REEMPLAZAMIENTO DE METODOS			
publico	String	toString	
publico	int	setNumeroDePatas()	

- altura → Almacena la altura del cuadrupedo
- Cuadrupedo(String nombre, float peso, float altura) → Crea un cuadrúpedo de nombre “nombre”, peso “peso” y altura “altura”
- Cuadrupedo(String nombre, float peso) → Crea un cuadrúpedo de nombre “nombre” y peso “peso”
- Cuadrupedo() → Crea un objeto “Cuadrupedo” con los valores por defecto para cada atributo.
- toString() → Devuelve una cadena con la descripción de cada atributo y su valor en líneas distintas.

PAQUETE: clases CLASE: Perro EXTIENDE: Cuadrupedo			
ACCESO	TIPO	MIEMBROS	
constantes estática publica	float	INCREMENTO_PESO_POR_KILO	0,1F
publico	int	getPesoEnGramos()	
publico		Perro(float peso, String nombre)	
publico		Perro(String nombre, float peso)	
publico	Perro	comer()	
publico	Perro	comer(int kg)	
REEMPLAZAMIENTO DE METODOS			
publico	String	toString	
publico	int	setNumeroDePatas()	

- getPesoEnGramos() → Devuelve el peso del “Perro” en gramos

- Perro(String nombre, float peso), Perro(String nombre, float peso) → Crea un perro de nombre "nombre" y peso "peso"
- comer(int kg) → Incrementa el peso del animal en el valor de kg* INCREMENTO_PESO_POR_KILO. Se devuelve un error con el mensaje "La cantidad no puede ser negativa" si el valor de "kg" es menor que cero. Se devuelve el objeto que invoca el método.
- comer() → Incrementa el peso del animal al comer 1kg. Se invoca el método anterior. Se devuelve el objeto que invoca el método.
- setNumeroDePatas() → Se devuelve 4
- toString() → Devuelve una cadena con la descripción de cada atributo y su valor en líneas distintas.

Hágase la siguiente aplicación: Crear un perro con su nombre y su peso, muestra su peso antes y después de comerse 1kg de comida, y lo convierte a cadena.

(PerroAbstracto)

19. Derívese la clase "Rectángulo" de la clase "Figura" realizando las siguientes modificaciones:

a) Quítese la importación e implementación de las interfaces "Acumulable" y "Comparable".
b) Invóquese el constructor de la clase "Figura" en los correspondientes constructores de la clase "Rectangulo"
c) Muéstrese además de todos los atributos de la clase "Rectángulo" el atributo de la clase "Figura" (se mostrará como una constante de clase de la clase "Rectangulo")
d) Modifíquese correctamente las clases "Cuadrado" y "Caja" (en está última, el número de vértices es de 8)

Hágase una aplicación en la que se inicialice un rectángulo de 5 x 4, un cuadrado de lado 7 y una caja de 6x3x4, y se muestren todas las características de cada uno de los objetos creados.

(FiguraAbstracto)

Clases anónimas que implementan una interfaz

20. Hágase una aplicación en la que se cree una clase anónima que implemente la interfaz "Operable" de forma que sirva para sumar, restar y multiplicar números enteros. Se lanzarán excepciones de usuario si los objetos introducidos no fueran de tipo entero. Léanse dos enteros, y obténgase la suma.

(InterfazAnonima)

Clases anónimas que derivan de una clase abstracta

21. Hágase una aplicación en la que se cree una clase anónima que implemente la clase abstracta "Operaciones" de forma que sirva para sumar y restar rectángulos.

a) La suma de dos rectángulos es otro rectángulo con valores por defecto de ancho/alto el máximo de los anchos/altos de los rectángulos introducidos.

b) La resta de dos rectángulos es otro rectángulo con valores por defecto de ancho/alto el mínimo de los anchos/altos de los rectángulos introducidos.

c) El producto de dos rectángulos será otro rectángulo cuyo ancho y alto es la semisuma de los anchos y altos de los rectángulos dados.

d) Se lanzarán excepciones de usuario si los objetos introducidos no fueran de tipo rectángulo.

e) Hágase una aplicación que cree dos rectángulos con ancho y alto valores aleatorios entre 2 y el máximo posible; se mostrará la dimensión de los dos rectángulos y se dibujarán; se obtendrá la suma, la resta y el producto de tales rectángulos y se mostrarán sus dimensiones y se dibujarán.

(AbstractaAnonima)

Clases anónimas que derivan de una clase

Se considera la siguiente clase:

```
package poo;

public class Operar {
	public Object sumar(Object a, Object b)
	{
		return null;
	}

	public Object restar(Object a, Object b)
	{
		return null;
	}
}
```

22. Hágase una aplicación en la que se cree una clase anónima que derive de la clase "Operar" de forma que sirva para sumar y restar objetos de tipo persona.

a) La suma de dos personas es una persona con valores por defecto cuya edad es la semisuma truncada de las edades de las personas introducidas.

b) La resta de dos personas es una persona con valores por defecto cuya edad es la diferencia de las edades introducidas con signo positivo.

c) Se lanzarán excepciones de usuario si los objetos introducidos no fueran de tipo "Persona"

En dicha aplicación se declaran dos personas; se establece como edad de cada persona un valor aleatorio entre 20 y 80, ambos inclusive; se muestran las dos personas; se obtiene la suma y resta de las dos personas y se muestran.

(DerivadaAnonima)

Métodos de comparación con clases anónimas

23. Defínase en la clase "Rectángulo" la constante estática para comparar dos rectángulos por el valor de su perímetro a partir de una clase anónima de la clase "Comparator". Hágase una aplicación en la que se generen aleatoriamente 10 rectángulos con ancho y alto aleatorio entre 2 y 10. Compárese el primero u el último por el valor de su perímetro. Ordénense por el perímetro y muéstrese el perímetro de cada rectángulo junto con su dibujo. (RectanguloComparator)

Interfaces genéricas

24. Defínase la siguiente interfaz genérica

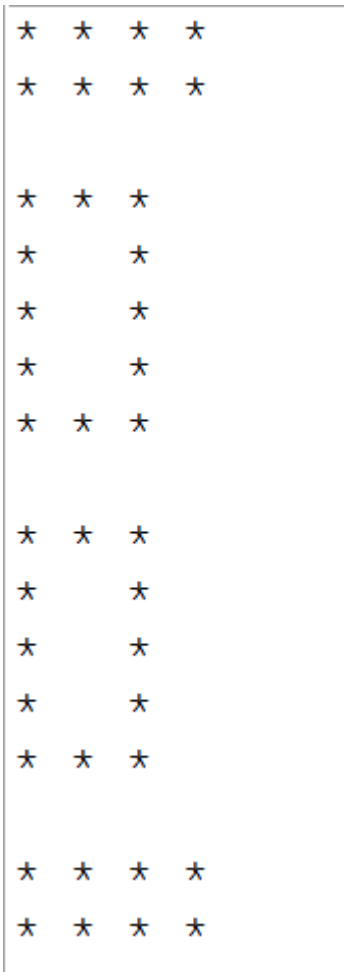
PAQUETE: clases.interfaces.genericas INTERFAZ: Copiable<T>	
TIPO	MIEMBROS
T	copiar(T o)

Impleméntese dicha interfaz en una clase de modo que se devuelva una copia de valor pasado como argumento del método copiar.

Clases genéricas

25. Hágase una aplicación en la que se intercambien rectángulos.

a) Defínase un objeto Intercambiar<Rectangulo> inicializado con un rectángulo de 4 x 2 y un segundo rectángulo de 3 x 5
b) Dibújese el primer rectángulo obteniéndolo a partir del objeto Intercambiar.
c) Dibújese el segundo rectángulo obteniéndolo a partir del objeto Intercambiar.
d) Intercámbiese los dos rectángulos
e) Repítanse los pasos b) y c)



(IntercambiarRectangulos)

26. Defínase una clase genérica de nombre "Par" en "clases/genéricas" para dos tipos de datos <T,S> con los siguientes miembros:

a) clave → Atributo privado de objeto variable de tipo T con el valor inicial de null. Representa la clave del par de valores.
b) valor → Atributo privado de objeto variable de tipo S con el valor inicial de null. Representa el valor del par de valores.
c) T getClave() → Método público de objeto que devuelve el atributo clave.
d) T getValor() → Método público de objeto que devuelve el atributo valor.
e) Par setClave(T) → Método público de objeto que establece como nuevo valor del atributo "clave" el del parámetro. Si el parámetro fuera null se lanzará una excepción interna de usuario del tipo "ObjetoNuloException" con el mansaje "La clave no puede ser nula".
f) Par setValor(S) → Método público de objeto que establece como nuevo valor del atributo "valor" el del parámetro. Si el parámetro fuera null se lanzará una excepción interna de usuario del tipo "ObjetoNuloException" con el mansaje "El valor no puede ser nulo".
g) Par(T,S) → Constructor público que permite inicializar la clave y el valor.

(ParObjetos)

27. Hágase una aplicación en la que se definan objetos de la clase genérica "Par":

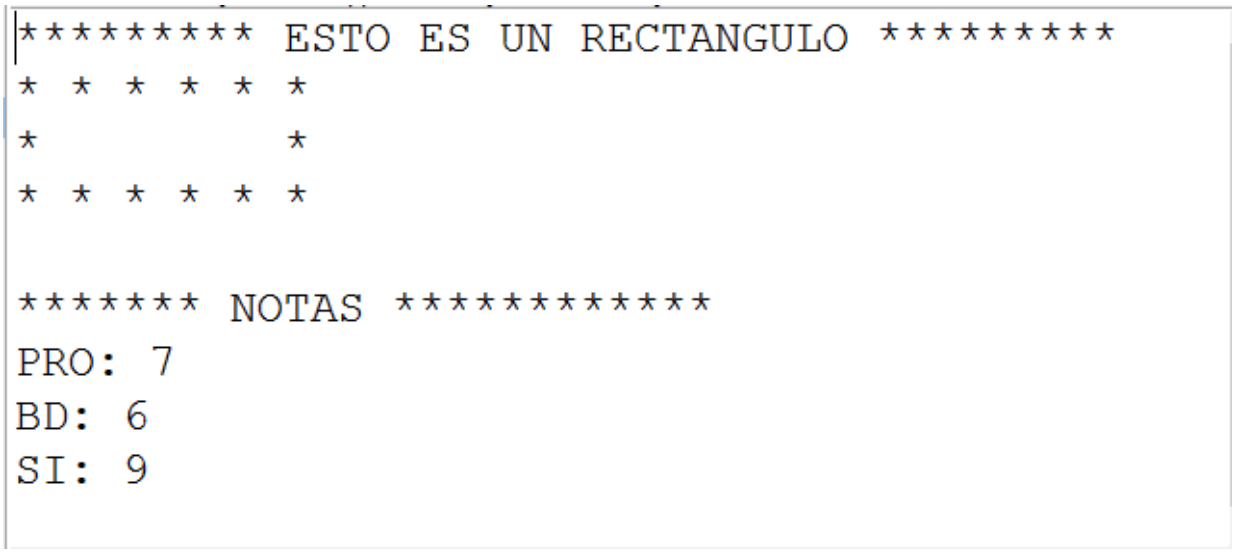
a) Defínase un objeto´"p1" de tipo Par<String,Rectangulo> e inicialícese con la clave "********* ESTO ES UN RECTANGULO *********\n" y valor de un rectángulo de 6x3.

b) Defínase un vector "notas" de objetos con 3 objetos de tipo Par<String,Integer> con los siguientes pares de clave y valor: "PRO" y 7; "BD" y 6; "SI" y 9.

c) Muéstrese la clave de "p1" y dibújese el valor de "p1"

d) Muéstrese un mensaje del tipo "******** NOTAS **********" saltando de línea.

e) Recórrase las "notas" y muéstrese cada Par<String,Integer> en líneas distintas con el formato <clave>: <valor>.



(ParObjetos)

28. Defínase una clase genérica de nombre "Vector" (Array) en "clases/genéricas" para un solo tipo de datos <T> con los siguientes miembros:

a) elementos → Atributo privado de objeto variable de tipo T[] con el valor inicial de null. Representa todos los elementos del vector.

b) CLASE → Constante pública de tipo Class que almacena el tipo del atributo "elementos"

c) T getElementos() → Método privado de objeto que devuelve el valor del atributo "elementos"

d) Vector setElementos(T[]) → Método público de objeto que establece como nuevo valor del atributo "elementos" el del parámetro.

e) Vector() → Constructor que inicializa la constante CLASE e inicializa el atributo "elementos" con 0 elementos.

f) int size() → Método público de objeto que devuelve el tamaño del array "elementos"

g) boolean isEmpty() → Método público de objeto que devuelve verdadero si el array de elementos tiene 0 elementos; y falso en caso contrario.

h) T getElemento(int) →Método público de objeto que devuelve el elemento que está en la posición del parámetro. Si el array está vacío o el valor del parámetro es menor que 0 o superior o igual que el tamaño del array, se devolverá null.
i) T getUltimo() → Método público de objeto que devuelve el último elemento del vector. Si el vector estuviera vacío, se devolverá null.
j) T getPrimero() → Método público de objeto que devuelve el primer elemento del vector. Si el vector estuviera vacío, se devolverá null.

METODOS PARA AÑADIR

k) Vector add(int,T) → Método que añade en una posición (valor del primer parámetro) un elemento (valor del segundo parámetro). Si la posición no estuviera en el rango de 0 a tamaño del array, ambos incluídas, o el elemento fuera null, entonces se lanza una excepción del tipo "IllegalArgumentException" con el mensaje "Posición invalida o elemento nulo".
l) Vector add(T) → Método público de objeto que añade por el final del vector de "elementos" el elemento del parámetro. Se invocará el método anterior.
m) Vector addNew(int, T) → Método público de objeto que añade en una posición (valor del primer parámetro) un nuevo elemento (valor del segundo parámetro). Si el elemento a insertar ya existiese, entonces se lanzará una excepción del tipo "IllegalArgumentException" con el mensaje "Ya existe el elemento"; si no existiese, se invoca el método "add(int,T)".
n) Vector addNew(T) → Método público de objeto que añade por el final un nuevo elemento. Se invocará el método anterior.

METODOS PARA ELIMINAR

o) Vector remove(int) → Método público de objeto que elimina el elemento que ocupa la posición dada en el parámetro. Si la posición estuviera fuera del rango de 0 a tamaño del array menos uno, entonces no se elimina ningún elemento.
p) Vector removeFirst(T) → Método público de objeto que elimina el primer elemento del array que es "igual" al elemento pasado como parámetro.
q) Vector removeLast(T) → Método público de objeto que elimina el último elemento del array que es "igual" al elemento pasado como parámetro.
r) Vector removeAll(T) → Método público de objeto que borra todos los elementos que son "iguales" al elemento pasado como parámetro.

METODOS DE ORDENACIÓN

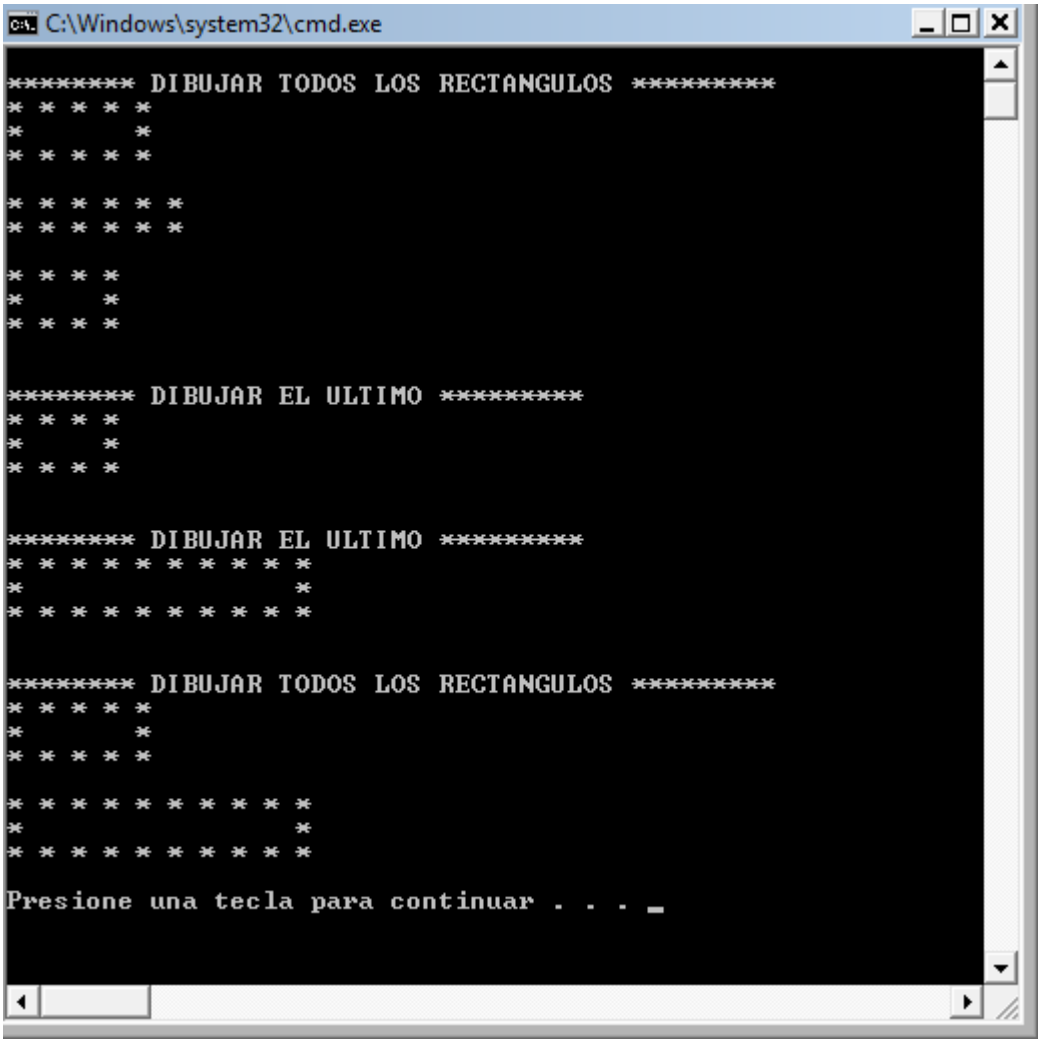
s) Vector ordenar(Comparator<T>) → Método que ordena el array de elementos según el criterio de comparación dado.

(VectorRectangulos)

29. Hágase una aplicación en la que se almacene un vector de rectángulos:

a) Inicialícese un vector de rectángulos con el constructor implícito.
b) Añádase al vector de forma sucesiva los siguientes rectángulos: uno de 5x3; otro de 6x2 y otro de 4x3.

c) Muéstrese por consola el mensaje "******* DIBUJAR TODOS LOS RECTANGULOS **********", saltando de línea
d) Recórrase el vector y dibújese todos los rectángulos almacenados.
e) Muéstrese por consola el mensaje "******* DIBUJAR EL ULTIMO **********", saltando de línea.
f) Dibújese el último rectángulo almacenado en el vector.
g) Actualícese el ancho del último rectángulo insertado a 10.
h) Repítase los pasos e) y f)
i) Repítase los pasos c) y d)



(VectorRectangulos)

30. Hágase una aplicación que permita leer rectángulos, que se almacenan en un Vector de rectángulos, hasta introducir un rectángulo que sea nulo (este último también se almacenará). A continuación, se dibujarán todos los rectángulos cuyo ancho y alto sea distinto al introducido en la aplicación.

(AplicacionVectorRectangulos)

Herencias de clases genéricas

31. Defínase una clase que encapsule un solo atributo con administradores de gestión de eventos y validación.

PAQUETE: clases CLASE: Property<T>			
ACCESO	TIPO	MIEMBROS	
constante publica	PropertyChangeSupport	cambios	null
publico	int	getPesoEnGramos()	
publico		Perro(float peso, String nombre)	
publico		Perro(String nombre, float peso)	
publico	Perro	comer()	
publico	Perro	comer(int kg)	
REEMPLAZAMIENTO DE METODOS			
publico	String	toString	
publico	int	setNumeroDePatas()	