Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Soluciones de la actividad 1. Ejercicios sobre sistemas dinámicos.

Es importante que tenga en cuenta que estas soluciones ni son la mejor opción ni la única, se han proporcionado como guía y para mostrar algunos ejemplos de implementación con elementos de Java.

Por un lado se ha considerado lo exigido al alumno, y por otro la propuesta que mejora lo pedido.

Ejercicios del 1 al 3

En estas soluciones encontrará :

1. **Modelado (si se pide). Consideraciones sobre el modelado de acuerdo con lo que se extrae del enunciado. Se mostrará en una tabla.**
    a. **Modelo mínimo exigido en UML**
2. **Definición de entidades e implementación de relaciones en Java .**
    a. **Especificación de las entidades indicadas en la tabla (como se representan en Java)**
    b. **Implementación de las relaciones identificadas en la tabla.**

Esto se obtendría mediante la exportación del modelo de ArgoUML a Java. *Es importante tener en cuenta que se deben revisar por si la exportación no es del todo correcta.

**Entidad puede ser una clase, un atributo o un método.*

3. **Especificación y descripción de la implementación de las funciones que se piden.**
    a. **En principio, los constructores, los *getters* y los *setters* se dan por supuestos como necesarios para crear los objetos e inicializarlos, y para acceder y modificar los atributos (sobre todo los privados y protegidos), por tanto no suelen aparecer en el modelo pero se pueden poner si se considera necesario.**

b. **Las funciones que se piden deben de aparecer en el modelo, con los parámetros y el tipo de retorno necesarios.**

4. **Código Java exigido por la exportación desde ArgoUML con las correcciones oportunas para adaptarlo (en los ejercicios 1 y 2) y lo implementado en el resto de los ejercicios:**
   a. **Clases en Java (de acuerdo con el modelo presentado al ser exportado)**
   b. **Atributos y funciones (de acuerdo con el modelo presentado al ser exportado). Es importante la visibilidad que se ha considerado en el modelo.**
   c. **Implementación de las relaciones de acuerdo con como aparecen en el modelo.**
   d. **Implementación de las funciones que se pidan.**

Al final de las soluciones de los ejercicios del 1 al 3 encontrará:

1. **Propuesta de modelo del docente que le ayudará a ampliar sus conocimientos.**
2. **Propuesta de implementación Proyecto ejecutable (donde se pedía).**

Esta solución no es ni la mejor ni la única, se proporciona para ayudar a incrementar los conocimientos y para mostrar el uso de estructuras dinámicas en java, excepciones y otros recurso.

Ejercicios 4 y 5:

Se presenta un modelo y la descripción de la implementación. Además se adjunta código completo.

En estos ejercicios se debe tener en cuenta que lo pedido es menos de lo que se presenta y en el ejercicio 5 no se ha pedido proyecto ejecutable.

Lo importante era un esbozo de implementación en el que se muestre el uso del polimorfismo como se indica en las soluciones

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Además, en las soluciones encontrará:

 Pistas de donde encontrar información sobre lo que se aplica en la solución en los temas y en las presentaciones de clase.

 Enlaces a tutoriales o libros en los que aparece información sobre lo usado**.**

Tabla de contenido

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Solución ejercicio 1 de la actividad 1.

EJERCICIO 1 (Diseño de clases I):

Una empresa ha decidido informatizar el sistema de control de stock de sus productos. La empresa tiene productos físicos y productos virtuales (licencias de software) que se identifican por un ID único. Los productos físicos tienen unos costes de transporte asociados, y unas características físicas (peso, alto, ancho, largo) mientras que los virtuales no.

Realizar el modelado de clases en UML y exportar las clases desde ArgoUML.

Se debe entregar:

- Parte de la memoria correspondiente a esta sección en la que se explique en todo detalle el procedimiento seguido para diseñar las clases. Debe de aparecer también el diseño final de la clase en UML (captura de pantalla).
- Parte de la memoria con la explicación sobre cómo se ha hecho la exportación de clases a .java.
- Archivo con la clase en ArgoUML.

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Solución Ejercicio 1

En este ejercicio se pide realizar el modelado en UML y exportar las clases, es decir, se deben obtener las clases a partir del modelo de ArgoUML. Hay que tener en cuenta que a veces ArgoUML no realiza las exportaciones correctamente o usa tipos obsoletos. Se pedía mostrar el proceso de exportación a .java. Siempre ***que se muestra un proceso se debe mostrar el resultado****. Había dos formas: adjuntando los archivos Java o introducir ese código en la memoria.*

1. Modelado

Una buena práctica para llevar a cabo el modelado es extraer del enunciado nombres, sintagmas nominales, verbos, adjetivos del enunciado y frases en general que pueden representar una funcionalidad o entidades y sus características.

En la presentación del Tema 1 de clase se encuentran algunas recomendaciones como esta

https://campusvirtual.unir.net/access/lessonbuilder/item/5426977/group/PER1582-439-8206-1582/Documentación/MAC_MIMC_Sesion01_Tema01.pdf

A continuación se presentará en la tabla, los términos y las frases seleccionadas del enunciado y la descripción de lo que se extrae, el tipo de entidades correspondientes que se pueden identificar en el modelo y las posibles relaciones que las entidades indicadas pueden tener con otras.

Término	*Descripción*	*Tipo de entidad*	*Características de la entidad*	
Clases		***Nombre de la clase***	***Relación***	***Cardinalidad***
empresa *sistema de control de stocks de sus* ***productos***	**De este nombre (empresa) y del sintagma nominal se extrae que la empresa cuenta con una serie de productos de las que controlará su stock* **Los productos se pueden ir añadiendo o eliminando en cualquier momento*	*Clase Empresa*	***Relación de agregación****: La "empresa" es contenedora de productos* **No se considera composición ya que los productos se podrán añadir y quitar durante la vida de la empresa*	***0 a muchos (0..*)*** *ó uno a muchos (1..*) si se considera que cuando se da de alta la empresa tiene que haber al menos un producto*
productos físicos	**un producto físico cuenta con una serie de características que difieren de un producto a otro.* **la palabra físico no denota un atributo, es un marcador de tipo o de la naturaleza de los productos.* **Comparte atributos con otros tipos de productos (por eso existirá un jerarquía)*	*Clase ProductoFisico*	***Relación de herencia:*** *Subclase de* ***Producto***	***No tiene***

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Término	*Descripción*	*Tipo de entidad*	*Características de la entidad*	
Clases		***Nombre de la clase***	***Relación***	***Cardinalidad***
productos virtuales	**un producto virtual cuenta con una serie de características que difieren de un producto a otro (no se especifica ninguna pero se pueden deducir). *la palabra virtual no denota un atributo, es un marcador de tipo o de la naturaleza de los productos. *Comparte atributos con otros tipos de productos (por eso existirá una jerarquía)*	*Clase ProductoVirtual*	***Relación de herencia**: Subclase de **Producto***	***No tiene***
producto	*Esta clase se crea para contener las características comunes a los productos*	*Clase Producto*	***Relación de herencia: Clase base de la jerarquía de productos***	***No tiene***
Atributos		***Nombre***	***Clase a la que pertenece***	***Tipo***
ID unico	*Es una característica común de los productos*	*id*	*Producto (contiene las características comunes a los productos)*	*int o String*
costes de transporte	*Es una característica de los productos físicos*	*costeTransporte*	*ProductoFisico*	*float*
características físicas	*Se puede considerar como una entidad formada por un conjunto de características*	*Clase ConjuntoCaracteristicasFisicas*	*ProductoFisico*	*Clase con 4 atributos de tipo float*
	Se puede diferenciar el peso de las dimensiones	*peso*	*ProductoFisico*	*float (peso)*
		Clase ConjuntoDimensiones		*Clase con 3 atributos float*
	Todas las características por separado	*peso*	*ProductoFisico*	*float*
		alto		*float*
		ancho		*float*
		largo		*float*
Métodos		***Nombre***	***Clase a la que pertenece***	***Tipo***
No se pide ninguno				

a. Modelo mínimo

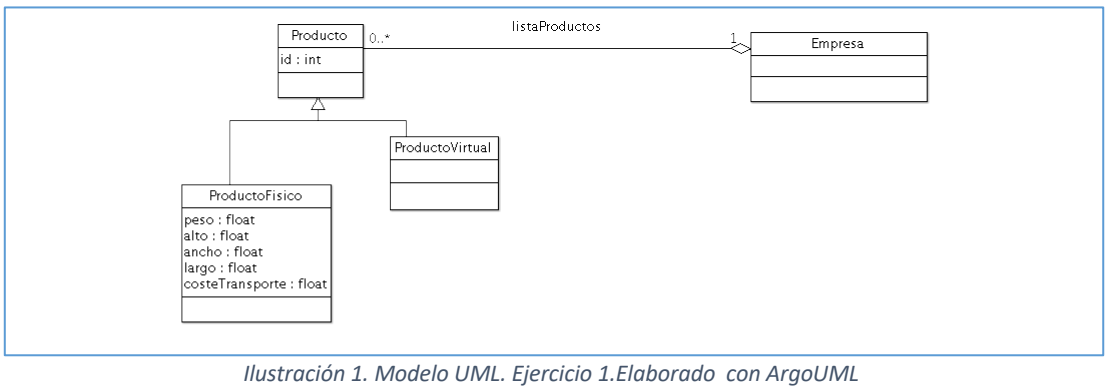


Ilustración 1. Modelo UML. Ejercicio 1.Elaborado con ArgoUML

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

2. Definición de entidades e implementación de relaciones en Java .

a. Especificación de las entidades indicadas en la tabla y sus atributos (como se representan en Java)

Entidad	*Especificación*
Clase Empresa	`class Empresa`
	Atributos: *No se pedía ninguno, algunos posibles nombre, dirección..*
Clase Producto	`class Producto` *(clase base de la jerarquía)*

	Atributos
	`protected int id`
	`protected String id`
	** es protegido porque está en la clase base y así es visible en la clase derivada*
	** se inicializa en el constructor*

Clase ProductoFisico	`class ProductoFisico extends Producto` *(subclase de* `Producto`*)*

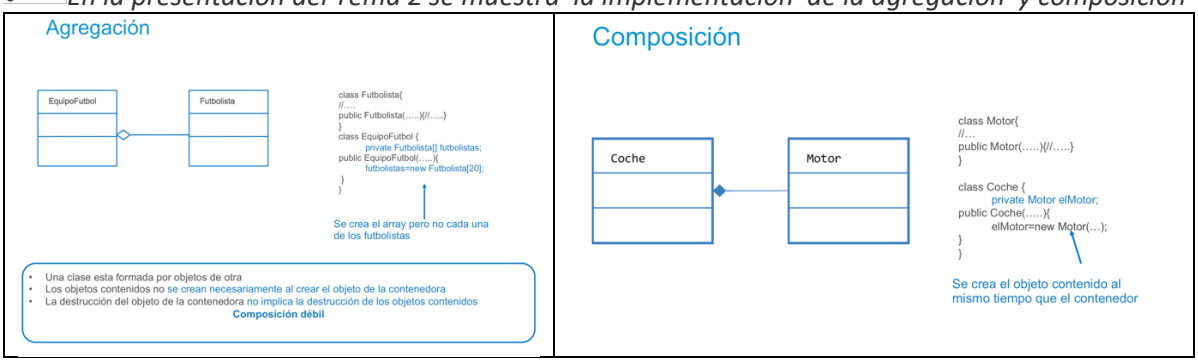
	Atributos
	`private float peso`
	`private float alto`
	`private float ancho`
	`private float largo`
	`private float costeTransporte`
	**los atributos son privados a no ser que se necesite darles visibilidad*
	** se inicializan en el constructor*

Clase ProductoVirtual	`class ProductoVirtual extends Producto` *(subclase de* `Producto`*)*
	Atributos: *No se pedía ninguno, alguno posible version*

b. Implementación de las relaciones identificadas en la tabla.

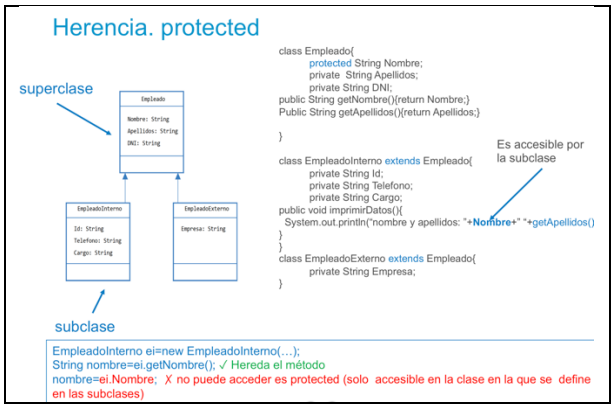
Entidad	*Especificación*
Clase Empresa	***Relación de agregación*** `Producto[] listaProductos` *(si el id de Producto es* `int`*)* **Se agregan los productos de la clase base (se podrá usar polimorfismo en el uso de funciones comunes y tratar los productos de manera uniforme. Para las funciones propias de cada clase derivada siempre se puede hacer una conversión de tipo.* *Por ejemplo, si se accede al segundo producto de la lista y se quiere usar una función de la clase y este es físico see debe hacer una conversión* `((ProductoFisico)ListaProductos[2])` **¡Problema! (numero limitado de productos)* **En el constructor se crea el array pero sin productos (o con uno si se ha puesto cardinalidad de 1..*)*
Clase Producto	***Relaciones*** *No tiene, salvo que esta agregado en* `Empresa` *pero la relación de* ***agregación es unidireccional*** *y no se implementa en la clase de los objetos contenidos*
Clase ProductoFisico	***Relación de herencia (****Es subclase de Producto)* `class ProductoFisico extends Producto`
Clase ProductoVirtual	***Relación de herencia (****Es subclase de Producto)* `class ProductoVirtual extends Producto`

En la presentación del Tema 2 se muestra la implementación de la agregación y composición



Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

En la presentación del Tema 2 se muestra la implementación de la herencia y los atributos protegidos



Donde encontrarlo:
https://campusvirtual.unir.net/access/lessonbuilder/item/5426980/group/PER1582-439-8206-1582/Documentación/MAC_MIMC_Sesion02_Tema02.pdf
Tema 2 . Relaciones entre clases

3. Especificación y descripción de la implementación de las funciones

a. Constructores, getters y setters

Entidad	*Especificación*
`class Empresa`	***Constructor (se debe implementar para poder mostrar la implementación de la agregación*** `Empresa(int numProductos)` **Al usar un array se necesita saber el número total de productos *Se crea el array* `listaProductos=new Producto[numProductos]`

Para el resto de las clases no se pide ninguna función

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

4. Código Java exigido por la exportación desde ArgoUML con las correcciones oportunas

Código Java:

```
public class Empresa {

   /**
    *
    * @element-type Producto
    */
  private Producto[]  listaProductos;

  //constructor
  public Empresa(int numProductos) {
          /**
           * implementa la relacion de agregacion,
           * genera el array  con tamaño indicado y no se podrá cambiar
           */
          listaProductos=new Producto[numProductos];
  }

}
public class Producto {
  /**
         * protegido para que  sea visible en las clases derivadas
  */
  protected int id;


}
public class ProductoFisico extends Producto {

  private float peso;

  private float alto;

  private float ancho;

  private float largo;

  private float costeTransporte;

}

public class ProductoVirtual extends Producto {

}
```

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Solución ejercicio 2 de la actividad 1.

EJERCICIO 2 (Diseño de clases II):

Se desea informatizar el proceso de compra online de la empresa anterior.

Los usuarios pueden registrarse en la web, y realizar pedidos tanto de productos físicos como virtuales.

Un mismo usuario no podrá comprar más de una unidad de un producto virtual.

Un pedido se compone de una serie de líneas de pedido, y una dirección de envío y otra de facturación.

Realizar el modelado de clases en UML y exportar las clases desde ArgoUML.

Se entrega:

- Parte de la memoria correspondiente a esta sección en la que se explique en todo detalle el procedimiento seguido para diseñar las clases. Debe de aparecer también el diseño final de la clase en UML (captura de pantalla).
- Parte de la memoria con la explicación sobre cómo se ha hecho la exportación de clases a .java.
- Archivo con la clase en ArgoUML.

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Solución Ejercicio 2

En este ejercicio se pide realizar el modelado en UML y exportar las clases, es decir, se deben obtener las clases a partir del modelo de ArgoUML. Hay que tener en cuenta que a veces ArgoUML no realiza las exportaciones correctamente o usa tipos obsoletos. Se pedía mostrar el proceso de exportación a .java. Siempre ***que se muestra un proceso se debe mostrar el resultado****. Había dos formas: adjuntando los archivos Java o introducir ese código en la memoria.*

En la solución se muestra lo que se ha añadido a la solución del ejercicio 1. Se añadirán nuevas clases y nuevos atributos y métodos a las nuevas o a las que ya existen.

1. Modelado

Término	*Descripción*	*Tipo de entidad*	*Características de la entidad*	
Clases		***Nombre de la clase***	***Relación***	***Cardinalidad***
usuarios	*Los usuarios o clientes pueden realizar acciones y tienen características como la dirección, por ejemplo, luego es una entidad del modelo*	*Clase Cliente*	***Relación de composición*** *El cliente tiene una dirección que se crea desde el principio*	***1*** *Un cliente tiene una dirección de facturación*
			Relación de asociación *Los clientes contarán con una lista de pedidos (relación bidireccional)*	***0..**** *Un cliente puede tener asociados 0 o más pedidos*
pedidos	*Un pedido contiene líneas de pedido y otras características*	*Clase Pedido*	***Relación de asociación*** *El pedido tiene un cliente asociado (la otra parte de la relación anterior)*	***1*** *un pedido corresponde a un único cliente*
			Relación de agregación *Un pedido está formado por una serie de líneas de pedido (una por producto comprado)*	***1..**** *una o más líneas de pedido en cada pedido*
líneas de pedido	*una línea de pedido es un elemento de un pedido y contiene el producto y la cantidad que se compra (podrían considerarse más atributos pero no se pedía)*	*Clase LineaPedido*	***Relación de agregación*** *Una línea tiene un producto agregado.*	***1***
dirección de envío dirección de facturación	*las direcciones están compuestas por varios atributos que se pueden agrupar en una entidad*	*Clase Direccion*		
empresa sistema de control de stocks de sus ***productos***	**Se debe considerar que ahora la empresa debe contener y gestionar clientes y pedidos*	*Clase Empresa*	***Relación de agregación*** *Contiene una lista de clientes*	***0..****
			Relación de agregación *Contiene una lista de pedidos*	***0..****

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Término	*Descripción*	*Tipo de entidad*	*Características de la entidad*	
Atributos		***Nombre***	***Clase a la que pertenece***	***Tipo***
unidades que se compran	*Es una característica de la línea de pedido que indica que cantidad de un producto se va a comprar (en los virtuales siempre es 1)*	*cantidad*	*ProductoFisico, ProductoVirtual(max. 1)*	*int*
Atributos de cliente	*Un cliente tiene una serie de atributos.*	*nombre apellidos dnil email clave idCliente*	*Cliente*	*Todos String menos el idCliente que es int*
Atributos de dirección	*Un dirección está compuesta por varios atributos.*	*calle numero codigoPostal población pais*	*Dirección*	*Todos String*
Atributos pedido	*Contiene el Úmero de pedido*	*numPedido*	*Pedido*	*int*
Atributos línea pedido	*Contiene el id del producto comprado y la cantidad que se compra*	*cantidad*	*LineaPedido*	*int*
Métodos		***Nombre***	***Clase a la que pertenece***	***Parámetros y retorno***
registrarse en la web	*Se necesita un método en la clase Empresa que de alta un cliente, con los parámetros necesarios para dar de alta a un cliente.*	*registrarCliente*	*Empresa*	***Parámetros*** *nombre, apellidos, dni, email, clave, calle, numero, codigoPostal, población, país* ***Tipo de retorno*** *String (idCliente)*
realizar pedidos	*Esto requiere un método en la clase Empresa que permita generar un pedido vacío para empezar la compra*	*hacerPedido*	*Empresa*	***Parámetros*** *idCliente* ***Tipo de retorno*** *int (idPedido)*
añadir dirección de envío	*Esto requiere un método que añada una dirección de envío en el pedido*	*aniadirDirEnvio*	*Pedido*	***Parámetros*** *calle, numero, codigoPostal, población, país* ***Tipo de retorno*** *void*
Un mismo usuario no podrá comprar más de una unidad de un producto virtual	*Esto requiere dos métodos, uno que controle en el momento de la compra del producto virtual **que no se pide más de una unidad** y otro que en el pedido compruebe si ya se ha comprado ese producto y lo inserte creando una línea de pedido*	*comprar (también debería estar en los otros productos pero implementada de forma distintas, luego puede ser una función sobrecargada)*	*ProductoVirtual, ProductoFisico, Producto*	***Parámetros*** *pedido, cantidad* ***Tipo de retorno*** *void*
		insertarProducto	*Pedido*	***Parámetros*** *pedido, cantidad* ***Tipo de retorno*** *void*

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

a. Modelo mínimo

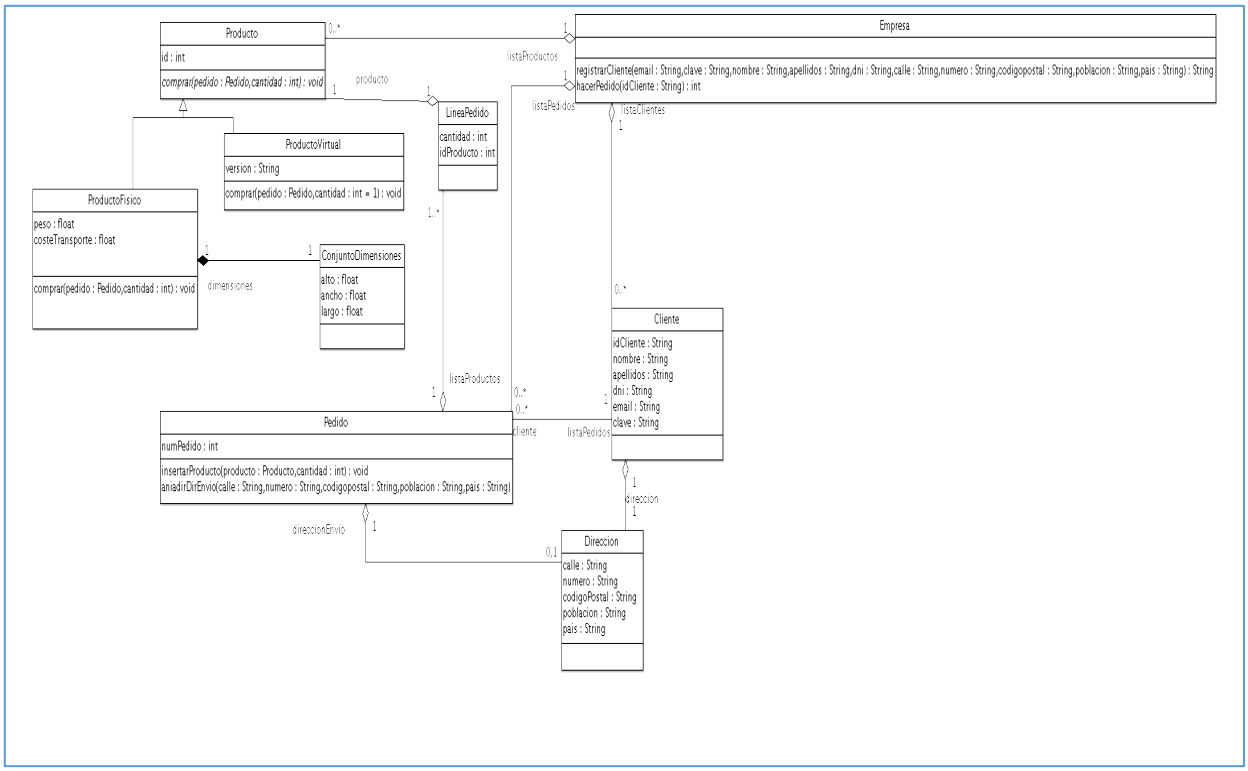


Ilustración 2. Modelo UML. Ejercicio 2.Elaborado con ArgoUML

2. Definición de entidades e implementación de relaciones en Java .

a. Especificación de las entidades indicadas en la tabla y sus atributos (como se representan en Java)

Entidad	*Especificación*
Clase Cliente	`class Cliente`
	Atributos: `int idCliente, String nombre, String apellidos, String dni, String email, String clave`
Clase Dirección	`class Direccion`
	Atributos: `String calle, String numero, String codigoPostal, String Poblacion, String pais(son todos publicos)`
Clase Pedido	`class Pedido`
	Atributos: `int numPedido`
Clase LineaPedido	`class LineaPedido`
	Atributos: `int cantidad, int idProducto`

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

b. Implementación de las relaciones identificadas en la tabla.

Entidad	*Especificación*
Clase Empresa	***Relación de agregación clientes*** `Cliente[] listaClientes` *(si el id de Cliente es* `int`*)* **Se agregan los clientes* **¡Problema! (numero limitado de clientes)* **En el constructor se crea el array pero sin clientes*
	Relación de agregación pedidos `Pedido[] listaPedidos` *(si numPedido de Pedido es* `int`*)* **Se agregan los pedidos* **¡Problema! (numero limitado de pedidos)* **En el constructor se crea el array pero sin pedidos*
Clase Pedido	***Relación de agregación líneas de Pedido*** `LineaPedido[] listaProductos` **Se agregan las líneas de pedido* **¡Problema! (numero limitado de productos en un pedido)* **En el constructor se crea el array pero sin líneas*
	Relación de asociación con cliente `Cliente cliente` **Se asocia en el constructor porque desde el momento en el que se crea el pedido vacío se asocia al cliente*
Clase Cliente	***Relación de agregación direccion*** `Direccion direccion` **Se crea en el constructor*
	Relación de asociación con pedido `Pedido[] listaPedidos` **Se asocia en el constructor porque desde el momento en el que se crea el pedido vacío se asocia al cliente* **Se agregan los pedidos* **¡Problema! (numero limitado de pedidos)* **En el constructor se crea el array pero sin pedidos*
Clase LineaPedido	***La relación de agregación con Producto se ha implementado mediante el atributo*** `int idProducto`***.***

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

3. Especificación y descripción de la implementación de las funciones

a. Constructores, getters y setters

Entidad	*Especificación*

`class Empresa`	***Constructor (se debe implementar para poder mostrar la implementación de la agregación)***
	`Empresa(int numProductos, int numPedidos, int numClientes)`
	**El problema de usar arrays es que se debe de conocer a priori cuantos elementos se van a crear, se puede pasar como parámetro o decidir en el constructor cuantos.*
	**Al usar un array se necesita saber el número total de productos*
	**Se crean los arrays*
	`listaProductos=new Producto[numProductos]`
	`listaPedidos=new Pedido[numPedidos]`
	`listaClientes=new Cliente[numClientes]`

	registrarCliente (solo se especifica no se pedía implementación) `int registrarCliente(String email, String clave, String nombre, String apellidos, String dni, String calle, String numero, String codigopostal, String poblacion, String pais)`
	hacerPedido (solo se especifica no se pedía implementación) `int hacerPedido(int idCliente)`

`class Pedido`	***Constructor (se debe implementar para poder mostrar la implementación de la agregación y la asociación)***
	`Pedido(Cliente _cliente, int numPedido)`
	**El problema de usar arrays es que se debe de conocer a priori cuantos elementos se van a crear, se puede pasar como parámetro o decidir en el constructor cuantos.*
	**Al usar un array se necesita saber el número total de productos, se crea*
	`listaProductos=new LineaPedido[num]`
	**Se asocia el cliente*
	`cliente=_cliente;`

	Insertar producto (se especifica y se esboza el control de los productos virtuales)
	`void insertarProducto(Producto producto, int cantidad)`
	****Se controla que si es un producto virtual y ya está ese producto en el pedido no se añade, la cantidad será 1 porque se controla al comprar un producto virtual)***
	`{`
	`//si producto es virtual`
	`//entonces si el producto ya esta en el pedido se indica que el producto ya ha`
	`//sido comprado`
	`}`

	Insertar dirección de envío (solo se especifica no se pedía implementación) `void aniadirDirEnvio(String calle, String numero, String codigopostal, String poblacion, String pais)`

`class Producto, ProductoFisico, ProductoVirtual`	***Comprar producto (se especifica y se esboza el control de los productos virtuales, no se ha penalizado el que no exista este método)***
	`void comprar(int cantidad)`
	****en la implementación de producto virtual se control la cantidad sea 1***
	`{`
	`//si cantidad!=1 por ejemplo se puede indicar que solo se va a agregar una`
	`//unidad al pedido`
	`}`

`class Cliente`	***Constructor (se debe implementar para poder mostrar la implementación de la agregación y la asociación)***
	`Cliente(int idCliente, String nombre,String apellidos,String dni,String email,String clave,String calle,String numero,String codigoPostal, String poblacion, String pais)`
	Para **implementar la asociación con pedidos, al usar un array se necesita saber el número total de productos, se crea*
	`listaPedidos=new Pedido[num];`
	**Agregación dirección*
	`direccion=new Direccion(calle,numero,codigoPostal,poblacion,pais);`

`class LineaPedido`	*No se pedían funciones de esta*

Para el resto de las clases no se pide ninguna función

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

4. Código Java exigido por la exportación desde ArgoUML con las correcciones oportunas

Código Java:

```
public class Empresa {
  private Producto[]  listaProductos;
  private Cliente[]  listaClientes;
  private Pedido[]  listaPedidos;
  //constructor
  public Empresa(int numProductos, int numPedidos, int numClientes) {
          /*
           * implementa la relacion de agregacion,
           * genera el array  con tamaño indicado por el parámetro o podría decidirse este
           * número en el constructor y no se podrá cambiar. NO es buena solución por la
           * limitacion
           */
          listaProductos=new Producto[numProductos];
          listaPedidos=new Pedido[numPedidos]
          listaClientes=new Cliente[numClientes]
  }

  public int registrarCliente(String email, String clave, String nombre, String apellidos, String
      dni, String calle, String numero, String codigopostal, String poblacion, String pais){

    //no se pide implementacion
  }
  public int hacerPedido(int idCliente){

//no se pide implementacion
  }
}
*****************************************************************************************
public class Producto {
  /**
         * protegido para que  sea visible en las clases derivadas
  */
  protected int id;

  abstract public void comprar(Pedido pedido,int cantidad);
}
*****************************************************************************************

public class ProductoFisico extends Producto {

  private float peso;
  private float alto;
  private float ancho;
  private float largo;
  private float costeTransporte;

public void comprar(Pedido pedido,int cantidad){
        //no se pide implementacion
}
}

*****************************************************************************************

public class ProductoVirtual extends Producto {

public void comprar(Pedido pedido,int cantidad)
{
//si cantidad!=1 por ejemplo se puede indicar que  solo se va a agregar una
//unidad al  pedido
}
}
*****************************************************************************************
```

```
public class Cliente {
  private int idCliente;
  private String nombre;
  private String apellidos;
  private String dni;
  private String _mail;
  private String clave;
//relaciones de asociación con pedidos y de agregación con dirección
  private  Pedido[] listaPedidos;
  private Direccion direccion;
//constructor que  implementa las relaciones
public Cliente(int idCliente, String _nombre, String _apellidos, String _dni,
 String _email, String _clave, String calle,String numero,String codigoPostal,
 String poblacion, String pais){
        idCliente=_idCliente;
        nombre=_nombre;
        apellidos=_apellidos;
        dni=_dni;
        email=_email;
        clave=_clave;
      //relaciones
        listaPedidos=new Pedido[10];
        direccion=new Direccion(calle,numero,codigoPostal,poblacion,pais);
    }
}
*****************************************************************************************
public class Direccion {
  public String calle;
  public String numero;
  public String codigoPostal;
  public String poblacion;
  public String pais;
}
*****************************************************************************************
```

```
public class LineaPedido {

  private int cantidad;

relación de agregación con producto

  private int idProducto;

}

*****************************************************************************************
public class Pedido {

  private int numPedido;

  //relación de asociación con  cliente

  private Cliente cliente;

 //relación de agregación con dirección

 private Direccion direccionEnvio=null;

//relación de  agregación con línea de pedidos

private LineaPedido[]  listaProductos;

public Pedido(Cliente _cliente, int _numPedido){

      numPedido=_numPedido;

     //relación de  asociación cardinalidad 1

      cliente=_cliente;

    //relación de agregación

      listaProductos=new LineaPedido[10];

   //la dir de envío se agregará cuando haga falta

  }

  public void insertarProducto(Producto producto, int cantidad) {
        //si producto es virtual
        //entonces si el producto ya esta en el pedido se indica que el producto  ya  ha
        //sido comprado
}

  public void aniadirDirEnvio(String calle, String numero, String codigopostal,

   String poblacion, String pais) {

      //se añade la dirección de envío

      _direccionEnvio=new Direccion(calle,numero,codigopostal,poblacion,pais);

  }

}

}
```

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

EJERCICIO 3 (Diseño de clases II):

Se requiere realizar un programa informático que gestione el almacén de la empresa anterior.

El programa debe tener 2 funcionalidades principales:

1- Introducir stock de producto. Se piden los datos de los mismos si procede: (id de artículo, tipo: virtual/físico, ancho, alto, largo, peso, unidades stock)

2- quitar stock de producto. Se ha realizado una venta y se necesita retirar unidades del almacén. Se pide id de artículo y cantidad a retirar (no puede ser mayor que el stock disponible)

Realizar la implementación en Eclipse.

A tener en cuenta:

- La implementación de las clases debe de utilizar la herencia.
- Se debe de implementar un programa informático que pida por pantalla los datos y luego los muestre.

Se entrega:

- Parte de la memoria correspondiente a esta sección en la que se explique en todo detalle el procedimiento seguido para diseñar las clases en Eclipse (con el código comentado es suficiente). Se debe de explicar la estrategia seguida para crear el programa y añadir el código comentando para qué sirve cada parte.
- Archivos ejecutables con el proyecto exportado de Eclipse con todos los archivos asociados.

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Solución Ejercicio 3

En este ejercicio se pide un programa en Java que gestione el almacén de los productos de la empresa con dos funcionalidades, añadir stock y quitar stock.

En la solución se muestra lo que se ha añadido a la solución del ejercicio 1 sin tener en cuenta el ejercicio 2, es decir, no había que implementar las funciones del ejercicio 2, solo las de control de stock y las que se necesite para ello. No se añadirán nuevas clases , pero si nuevos atributos y métodos. Se considera que el almacen es la lista de productos que tiene la empresa. Se podría haber planteado un clase Almacen que contuviese la lista de producto y estuviese agregada en la empresa. En principio, no se exige pero está bien si se hace. Se va a cambiar el nombre del atributo de la lista por Almacen para ser más explicito con lo que se pide.

La E/S se hará por consola. No se pedían Ventanas ni eventos.

1. Modelado

Término	*Descripción*	*Tipo de entidad*	*Características de la entidad*	
Clases		***Nombre de la clase***	***Relación***	***Cardinalidad***
empresa *sistema de control de stocks de sus* ***productos***	**Se debe considerar que ahora la empresa debe contener y gestionar clientes y pedidos*	*Clase Empresa*	***Relación de agregación*** *Contiene una lista de clientes*	***0..****
			Relación de agregación *Contiene una lista de pedidos*	***0..****
Atributos		***Nombre***	***Clase a la que pertenece***	***Tipo***
stock	*Cantidad de unidades que hay existencia para un producto (se considera que solo tiene sentido en los productos físicos)* *Si se ha considerado en los dos se ha dado por válido*	*stock*	*ProductoFisico*	*int*
Métodos		***Nombre***	***Clase a la que pertenece***	***Parámetros y retorno***
introducir Stock deProducto	*Esta función no es realmente un setter, luego no se debe usar ese nombre.* *Se indica la cantidad de unidades que se quiere insertar de un producto.* *Si ese producto no existe se puede o añadir uno nuevo o lo que es más lógico indicar que el producto con ese id no existe*	*introducirStockProducto*	*Empresa*	***Parámetros*** *idProducto, int cantidad* ***Tipo de retorno*** *void*

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Término	*Descripción*	*Tipo de entidad*	*Características de la entidad*	
quitar stock de producto	*Esta función será invocada cuando se hace una comprar para disminuir el stock del producto cuando se hace una compra*	*quitarStockProducto*	*Empresa*	***Parámetros*** *idProducto, cantidad*
				Tipo de retorno *void*

a. Modelo mínimo

No se pedía el modelo en UML

2. Definición de entidades e implementación de relaciones en Java .

a. Especificación de las entidades indicadas en la tabla y sus atributos (como se representan en Java)

Entidad	*Especificación*
Clase ProductoFisico	`class ProductoFisico`
	Atributos: `int stock`

b. Implementación de las relaciones identificadas en la tabla.

Entidad	*Especificación*
Clase Empresa	***Relación de agregación productos (se cambia el nombre por almacen)*** `Producto[] almacen` **Se agregan los productos* **¡Problema! (numero limitado de productos)* **En el constructor se crea el array pero sin productos*

3. Especificación y descripción de la implementación de las funciones

Entidad	*Especificación*

`class Empresa`	***Constructor (se debe implementar para poder mostrar la implementación de la agregación)***
	`Empresa(int numProductos, int numPedidos, int numClientes)`
	**El problema de usar arrays es que se debe de conocer a priori cuantos elementos se van a crear, se puede pasar como parámetro o decidir en el constructor cuantos.*
	**Al usar un array se necesita saber el número total de productos*
	**Se crean los arrays*
	`almacen=new Producto[numProductos]`
	`listaPedidos=new Pedido[numPedidos]`
	`listaClientes=new Cliente[numClientes]`

	aniadirStockProducto `void aniadirStockProducto(int idProducto, int cantidad)` **Encuentra el producto correspondiente y si es físico incrementa su stock, se puede considerar también que el stock es común para los dos*
	quitarStockProducto `void quitarStockProducto(int idProducto, int cantidad)` **Encuentra el producto correspondiente y si es físico decrementa su stock*

Para el resto de las clases no se pide ninguna función, se han implementado algunas funciones adicionales para añadir productos y visualización.

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

4. Código Java exigido por la exportación desde ArgoUML con las correcciones oportunas

Código Java:

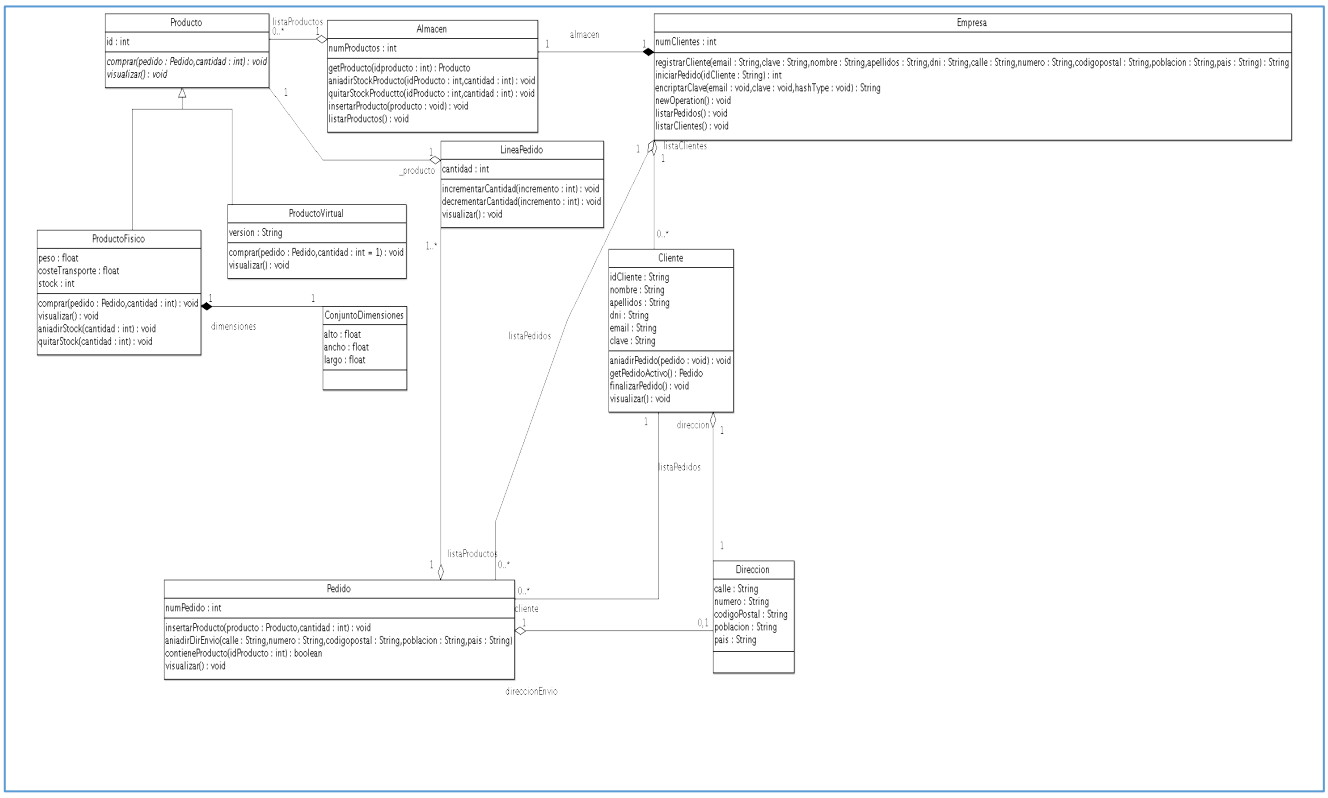
Se adjunta el Proyecto de Eclipse con la solución en el aula virtual

https://campusvirtual.unir.net/access/lessonbuilder/item/5453239/group/PER1582-439-8206-1582/Documentación/Ejercicio3-1.zip

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Solución propuesta completa con los elementos de los ejercicios 1,2 y 3

Modelo



Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Implementación del modelo

Entidad	*Especificación*
Clase Empresa	`class Empresa`
	Atributos: `private int numClientes` **sirve para control del numero de clientes que se han dado de alta en toda la historia de la empresa y generar claves distintas*
	Relaciones de agregación de almacen, cliente y pedido 1. `Almacen almacen` **se crea en el contructor aunque esté vacío* 2. `HashMap<String, Cliente> listaClientes` *(si el idCliente es* `String`*)* ** permite número ilimitado de clientes, crece dinámicamente* **indexa por el identificador* **se crea en el constructor pero sin clientes* 3. un `ArrayList<Pedido>` *(si el id de Pedido es int y se asignan de forma consecutiva, nunca se borran los pedidos aunque se anulen, esto permite asegurar que el id no cambia)* ** permite número ilimitado de pedidos, crece dinámicamente* **se crea en el constructor pero sin productos*
	Getters. *Se implementan funciones que devuelven el Cliente correspondiente a un identificador y otras.* ***Metodos*** 1. `public String registrarCliente(String email, String clave, String nombre, String apellidos, String dni, String calle, String numero, String codigopostal, String poblacion, String pais)` **Añade un nuevo cliente y lo inicializa con su constructor (la clave introducida por el cliente se debe encriptar), devuelve el identificador del cliente*
	`Cliente nuevoCliente=new Cliente(idCliente, nombre, apellidos, dni,email, claveEncriptada, calle, numero, codigopostal, poblacion, pais);` `listaClientes.put(idCliente, nuevoCliente);`
	2. `public int iniciarPedido(String idCliente)` **Añade un nuevo pedido con el número correspondiente y se los asocia al cliente, retorna el numero de pedido*

	`Cliente cliente=listaClientes.get(idCliente);`
	
	`Pedido pedido=new Pedido(cliente,idPedido)`
	`listaPedidos.add(pedido);`
	`cliente.aniadirPedido(pedido);`
	`return idPedido`

	3. `private String encriptarClave(String email, String clave, String hashType)` **encriptala clave facilitada por el cliente eligiendo tipo de hash* **es privada porque solo se usa en la empresa y no se le da visibilidad* 4. `public void listarPedidos()` 5. `public void listarClientes()` **listan los pedido y clientes*
Clase Almacen	`class Almacen`
	Atributos: `private int numProductos` **sirve para control del numero de productos que se han dado de alta en toda la historia de la empresa y generar claves distintas (hay otras formas mejores)*
	Relaciones de agregación de almacen, cliente y pedido 1. `HashMap<Integer, Producto> listaProductos` *(el id del producto es* `int`*)* ** permite número ilimitado de Productos, crece dinámicamente* **indexa por el identificador (aunque se eliminen productos, la posición en un array o ArrayList cambiaría y por tanto su índice, pero en este el identificador no porque es la clave)* **se crea en el constructor pero sin productos*

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

	Getters. *Se implementan funciones que devuelven el número de productos, el producto correspondiente a un número).* ***Metodos*** 1. `public void aniadirStockProducto(int idProducto, int cantidad)` 2. `public void quitarStockProducto(int idProducto, int cantidad)` **cambian el stock del producto elegido* **!importante!:* *instanceof: comprueba de que tipo es el objeto referenciado, esto es fundamental cuando se usan jerarquías manejadas con referencias a la base* *Conversión del producto: se usa la conversión a producto físico porque la función de aniadir o quitar sctock solo se ha implementado en ellos* 3. `public void insertarProducto(Producto producto)` **inserta un producto nuevo (debería comprobar si ya existe)* 4. `public void listarProductos()` **imprimer los productos del almacen*
Clase Cliente	`class Cliente`
	Atributos: `private String idClient3;private String nombre;private String apellidos;private String dni;private String email;private String clave; private int pedidoActivo=null;` **controla el pedido en el que insertar las compras (hay formas mejores de hacerlo).*
	Relación asociacion con pedidos 1. `ArrayList<Pedido> listaPedidos` ** permite número ilimitado de pedidos, crece y decrece dinámicamente* **se crea en el constructor pero sin pedidos.* ***Relación agregación con dirección*** 2. `Direccion direccion` ** no se inicializa en el constructor, se hará si se pide*
	Getters. *Se implementan funciones que devuelven valores de atributos.* ***Metodos*** 1. `public void aniadirPedido(Pedido pedido)` **asocia un pedido nuevo al cliente, vacío y lo pone como activo* 2. `public void finalizarPedido()` **pone a null pedido activo (debería hacer más cosas en el sistema, pero no es ahora relevante)* 3. `public void visualizar()` **imprime los datos del cliente*
Clase Direccion	`class Direccion`
	Atributos: `String calle, String numero, String codigoPostal, String Poblacion, String pais(son todos publicos)` **se implementan todos públicos para simplificar, hay que tener en cuenta que la dirección como tal se oculta en las clases que la agregan.*
	Relaciones
	Getters. *No son necesarios al ser atributos públicos* ***Metodos*** `public void visualizar()` **imprime los datos de la dirección*
Clase Pedido	`class Pedido`
	Atributos: `final private int numPedido.` **Se pone como final porque no se debe cambiar nunca*
	Relación agregación con línea de pedidos 1. `arraylist<LineaPedido> listaProductos` ** permite número ilimitado de líneas, es decir productos, crece y decrece dinámicamente* **se crea en el constructor pero sin productos* ***Relación agregación con dirección de envío*** 2. `Direccion direccion` ** no se inicializa en el constructor, se hará si se pide* ***Relación asociación con Cliente*** 3.`Cliente cliente` ** se inicializa en el constructor* ** también se podría implementar con una atributo de tipo String que incluya el idCliente*

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

	Getters. *Se implementan funciones que devuelven valores de atributos.*
	Metodos
	`1. private boolean contieneProducto (int idProducto)`
	`2. private LineaPedido getLinea(int idProducto)`
	**sin funciones privadas que seusan internamente en la implementación de otras*
	`3. public void insertarProducto(Producto producto , int cantidad)`
	**Añade un nuevo Producto al Pedido, es invocado por el método comprar de los productos.*
	**Comprueba si el producto está en el pedido y si es Fisico incrementa la cantidad que ya había y si es virtual eleva una excepción YaestaEnPedido , la excepción es capturada para que la función termine bien. Lo que se hace con la excepción en escribir un mensaje y terminar la función (el flujo de ejecución del programa continuará de forma normal. Esto cumple con el requisito de que no se puede comprar más de una vez un producto virtual.*
	**Si el producto no estaba, crea una línea de pedido con el producto y su cantidad. y lo inicializa con su constructor (la clave introducida por el cliente se debe encriptar), devuelve el identificador del cliente*
	`4. public void aniadirDirEnvio(String calle, String numero, String codigopostal, String poblacion, String pais)`
	**añade la dirección de envío si se requiere*

Clase LineaPedido	`class LineaPedido`
	Atributos: `int cantidad, int idProducto`
	Relaciones: La relación de agregación se ha implementado mediante el atributo int idProducto.

	Getters. *Se implementan funciones que devuelven valores de atributos.*
	Metodos
	`1. public void incrementarCantidad(int incremento)`
	`2. public void decrementarCantidad(int incremento)`
	**modifican las unidades de producto compradas*
	`3. public void visualizar()`
	**imprime la línea de pedido*

Clase Producto	`class Producto` → *es la base de la jerarquía*
	Atributos: *se implementan el comun como* `protected` `protected int id`
	Relaciones: *No tiene*

	Getters. *Se implementan funciones que devuelven valores de atributos.*
	Metodos
	`1. abstract public void comprar(Pedido pedido, int cantidad)`
	`2. abstract public void visualizar()`
	**son funciones abstractas que serán sobrecargadas en las clases derivadas para implementar la operación de compra y la visualización*
	**son funciones polimórficas*

Clase ConjuntoDimensiones	`class ConjuntoDimensiones`
	Atributos: *se implementan los especificados en el modelo como* `public` *(si es privado también es correcto), facilita el uso ya que el objeto va a ser encapsulado en ProductoFisico como private.* `Float _alto` →*se inicializa en el constructor* `Float _ancho` →*se inicializa en el constructor* `Float _largo` →*se inicializa en el constructor*
	Relaciones: *No tiene*
	Métodos: *solo se implementa el constructor*
Clase ProductoFisico	`class ProductoFisico extends Producto` (***subclase de*** `Producto`)

	Atributos: *se implementan los especificados en el modelo como* `private`
	`final float peso (no se puede modificar)`
	`float costeTransporte`
	`int stock`
	**se inicializan en el constructor.*
	se considera que el peso no cambia (las constantes en Java se declaran como **final)*

	Relaciones:
	1. ***Herencia****. Es subclase de Producto*
	`class ProductoFisico extends Producto`
	2. ***Composición.*** *El producto tiene unas dimensiones desde el momento que se inserta y no cambiarán, se supone que el cambio de dimensiones es cambio de producto.*
	`ConjuntoDimensiones _dimensiones`
	**se crea un objeto en el constructor, es obligatorio por ser composición.*
	se considera también **final*

	Getters y Setters. *Se implementan funciones que devuelven valores de atributos y otras que modifican los atributos que se pueden modificar (los que no son final).*
	Metodos
	`1. public void comprar(Pedido pedido, int cantidad)`
	**Esta función comprueba si hay existencias para la cantidad que se pide, si no lanza la excepción* `CantidadExcedeExistencias`*, que se captura y recupera la función. Emite los mensajes y vuele a solicitar una cantidad nueva para comprar.*
	`2. public void visualizar()`
	** visualiza los datos del producto fisico*

Clase ProductoVirtual	`class ProductoVirtual` (***subclase de*** `Producto`)
	Atributos: *se propone el siguiente atributo como* `private String version` **se inicializa en el constructor (no se pedía)*
	Relaciones: *1.* ***Herencia****. Es subclase de Producto* `class ProductoVirtual extends Producto`

	Getters y Setters. *Se implementan funciones que devuelven valores de atributos y otras que modifican los atributos que se pueden modificar (los que no son final).*
	Metodos
	`1. public void comprar(Pedido pedido, int cantidad)`
	**Esta función comprueba si se ha puesto una cantidad mayor que 1 y si es así se indica que solo se asignará una unidad del producto y añade el producto al pedido*
	`2. public void visualizar()`
	** visualiza los datos del producto virtual*

En la presentación del Tema 2 se muestra la implementación de la agregación y composición

Creación de una excepción personalizada

Objetos excepción
- Crear una clase que tenga como clase base a Exception (directa o indirectamente).
- Implementar los constructores (al menos uno sin argumentos y otro que reciba un string).
- Añadir los métodos que se estimen necesarios.

```
class MiExcepcion extends Exception{
      public MiExcepcion(){super();}
      public MiExcepcion(String s){super(s);}
}
```

Bloque try

```
try{
        objeto.invocación_método_lanza_excepción();

}catch (Exception e){
     sentencias que se ejecutan si se lanza una excepción en el bloque try
}
```

```
try{
        throw excepción;

}catch (Exception e){
     sentencias que se ejecutan si se lanza una excepción en el bloque try
}
```

Donde encontrarlo:
https://campusvirtual.unir.net/access/lessonbuilder/item/5447884/group/PER1582-439-8206-1582/Documentación/MAC__MIMC__Sesion06_Tema04.Excepciones.pdf
Tema 4 . Excepciones

Clases y ejecutable

Se adjunta en el campus virtual

https://campusvirtual.unir.net/access/lessonbuilder/item/5453236/group/PER1582-439-8206-1582/Documentación/Ejercicio3propuesto.zip

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

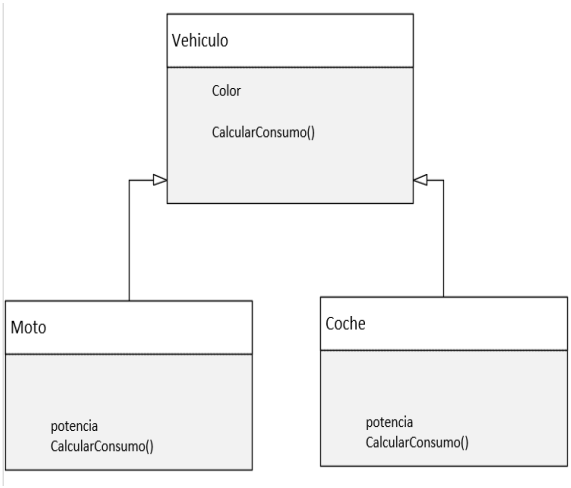
Solución ejercicio 4 de la actividad 1.

EJERCICIO 4 (Utilización del polimorfismo):

Se requiere realizar un programa informático que según el diagrama de clases adjunto haga uso del polimorfismo.

Nota: El cálculo del consumo se realiza como:

potencia * nº de ruedas del vehículo *0,12



Realizar la implementación en Eclipse.

A tener en cuenta:

- El programa puede pedir por pantalla los parámetros necesarios.
- El programa debe de mostrar por pantalla alguna figura.
- Se valorará especialmente el uso del polimorfismo.

Se entrega:

- Parte de la memoria correspondiente a esta sección en la que se explique en todo detalle el procedimiento seguido para diseñar las clases en Eclipse (con el código comentado es suficiente). Se debe de explicar cómo y dónde se ha utilizado el polimorfismo.
- Archivos ejecutables con el proyecto exportado de Eclipse con todos los archivos asociados.

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

-

Solución Ejercicio 4

La solución proporcionada es más extensa de lo que se pedía.

Para la calificación se ha considerado:

- Implementación de las relaciones de herencia
- Menú mínimo para la entrada de datos
- Implementación de la función polimórfica
- Uso de la función mediante referencias a la clase base que contuviesen objetos de las derivadas

Modelo

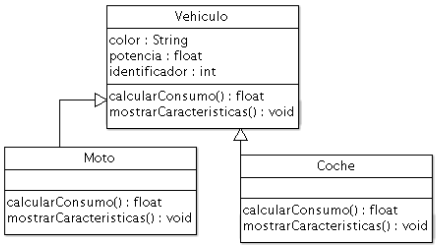


Ilustración 3. Modelo ejercicio 4

Implementación del modelo

Entidad	*Especificación*
Clase Vehículo	`class Vehiculo` *(clase base de la jerarquía)*
	Atributos: `private Color color` **se ha creado un tipo enumerado color (esto no es absolutamente necesario y no se ha considerado en la calificación)*
	`public enum Color {ROJO,AZUL,VERDE,BLANCO};`

	`private float potencia`
	`private int identificador`
	**se ha creado un identificador del vehículo para su representación (esto no es absolutamente necesario y no se ha considerado en la calificación)*

	Relaciones: es clase base
	Metodos 1. `abstract public float calcularConsumo()` 2. `abstract public void mostrarCaracteristicas()` **son funciones abstractas que serán sobrecargadas en las clases derivadas para implementar el calculo del consumo y la visualización* ****son funciones polimórficas***
	`class Coche`

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Clase Coche	***Atributos:***
	Relaciones de herencia `class Coche extends Vehiculo`

	Metodos
	`1. public float calcularConsumo()`
	**calcula el consumo del coche*
	`2. public void mostrarCaracteristicas()`
	**muestra el identificador, las características y una figura*

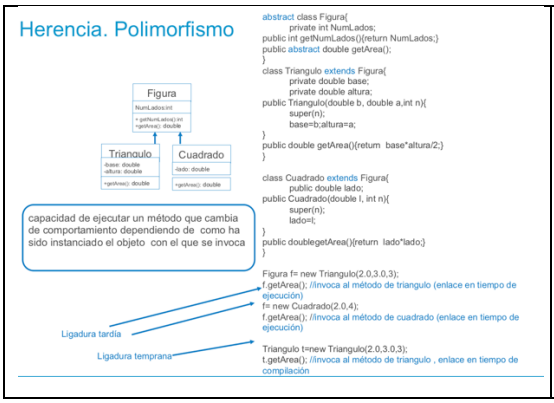
Clase Moto	`class Moto`
	Atributos:
	Relaciones de herencia `class Moto extends Vehiculo`

	Metodos
	`1. public float calcularConsumo()`
	**calcula el consumo del coche*
	`2. public void mostrarCaracteristicas()`
	**muestra el identificador, las características y una figura*

Uso del polimorfismo

Para que exista polimorfismo se debe invocar a una función sobrecargada de la jerarquía con una referencia a la clase base que pueda contener objetos de las clases derivadas. De esta forma se produce una ***ligadura dinámica o tardía*** *entre la función y el objeto para el que se ejecuta. Esta ligadura se realiza en tiempo de ejecución porque en compilación no se conoce el objeto al que va a apuntar la referencia a la clase base. Si se conoce el objeto con el que se llama en tiempo de compilación se produce la* ***ligadura estática o temprana****. Se ejecuta el método del tipo de objeto ligado*

Luego el polimorfismo se implementa en las funciones que invocan a las funciones polimórficas o en el programa principal.



Sobrecargar un método es implementar varios métodos con el mismo nombre pero diferentes parámetros. No se pueden redefinir métodos final ni métodos de clase (`static`).

La **ligadura** (conexión de un método con su llamada) se puede hacer en tiempo de ejecución.

- **Ligadura temprana** (métodos normales, sobrecargados, final).
  - En tiempo de compilación y enlace, el compilador necesita saber la referencia sobre la que se aplica el método.
- **Ligadura tardía** (polimorfismo).
  - La referencia al objeto que llama al método se hace en tiempo de ejecución.
  - No se sabe inicialmente el tipo de objeto que utiliza la llamada.

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Donde encontrarlo:
https://campusvirtual.unir.net/access/lessonbuilder/item/5426980/group/PER1582-439-8206-1582/Documentación/MAC_MIMC_Sesion02_Tema02.pdf
Tema 2. Relaciones entre clases. Página 7.

Polimorfismo en este ejercicio

Se ve en las funciones de la clase que contiene al programa principal

```
private static Vehiculo[] flota=new Vehiculo[5];
//se crea una flota de vehículos, es un array de referencias a la clase base
//no se conoce los tipos de vehiculo que se van a meter en el array hasta la ejecución
     private static int contadorVehiculos=0;
     static Scanner flujoEntrada=new Scanner(System.in);
     public static void insertarVehiculo(int id,int tipoVehiculo,Color color, int potencia){
         //según sea el tipo de vehículo creará una moto o coche y lo guarda en el array
         //luego tanto los coches como las motos se crean en tiempo de ejecución
          if (tipoVehiculo==0){
             flota[contadorVehiculos++]=new Moto(id,color,potencia);//referenciado por vehículo
          }
          else
             flota[contadorVehiculos++]=new Coche(id,color,potencia);//referenciado por vehículo

     }
     public static void configurarVehiculo(int id) throws IOException{
        System.out.println("Indique el vehículo que quiere configurar: 0(moto)/1(Coche)");
        int vehiculo=flujoEntrada.nextInt();
        flujoEntrada.nextLine();
        System.out.println("Indique el color del vehículo: ROJO, AZUL, VERDE, BLANCO");
        String color=flujoEntrada.next();
        flujoEntrada.nextLine();
        System.out.println("Introduzca la potencia vehículo");
        int potencia=flujoEntrada.nextInt();
        insertarVehiculo(id,vehiculo,Color.valueOf(color),potencia);
        flujoEntrada.nextLine();

    }
     public static void mostrarFlota(){
         //justo aquí es donde se implementa el polimorfismo
         //se invoca a la función sobrecargada mediante referencias a la clase base
         //pero los objetos contenido no son de la clase base pero no se conocen hasta la ejecucióv
         //flota[i] se liga de forma tardía o dinámica al método mostrarCaracterísticas
         //dependiendo de lo que contenga flota[i] se ejecuta la función de una clase o de otra
          for (int i=0;i<5;i++) flota[i].mostrarCaracteristicas();
     }

    /**
     * @param args the command line arguments
     */

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

        System.out.println("Introduzca la flota de vehiculos");
```

```
for (int i=0;i<5;i++)configurarVehiculo(i);
System.out.println("La flota  de vehiculos disponibles es:");
System.out.println("**************************************");
mostrarFlota();
 //si se hiciera esto, sería ligadura temprana o estática  y  no  habría polimorfismo
Coche coche=new Coche(…..);
 coche.mostrarCaracteristicas();
```

Clases y ejecutable

Se adjunta en el campus virtual

https://campusvirtual.unir.net/access/lessonbuilder/item/5453230/group/PER1582-439-8206-1582/Documentación/Ejercicio4.zip

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Solución ejercicio 5 de la actividad 1.

EJERCICIO 5 (Composición y agregación):

Se pretende definir un sistema para dibujar figuras geométricas en JAVA, de distintos tipos como polígonos o figuras circulares por medio de sus vértices, centros y magnitudes necesarias para su representación. Se debe contar con tablero de figuras. En este se pueden representar distintas figuras que no puedan cambiar de posición pero si de color. Además se deben calcular sus áreas y perímetros o longitudes e imprimir el resultado para cada figura.

Realizar el diseño en UML.

Se entrega:

- Parte de la memoria correspondiente a esta sección en la que se explique en todo detalle el procedimiento seguido para diseñar las clases y dónde y porqué se utiliza composición y/o agregación. Debe de aparecer también el diseño final de la clases en UML (captura de pantalla)
- Realice la especificación de las clases haciendo hincapié en la definición de sus atributos, los prototipos de los métodos, mostrar los métodos polimórficos, si es que existen y la implementación de las relaciones de agregación y composición que se identifiquen. No es necesario implementar los métodos completamente y por tanto no se generará un ejecutable.

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Solución Ejercicio 5

Modelo

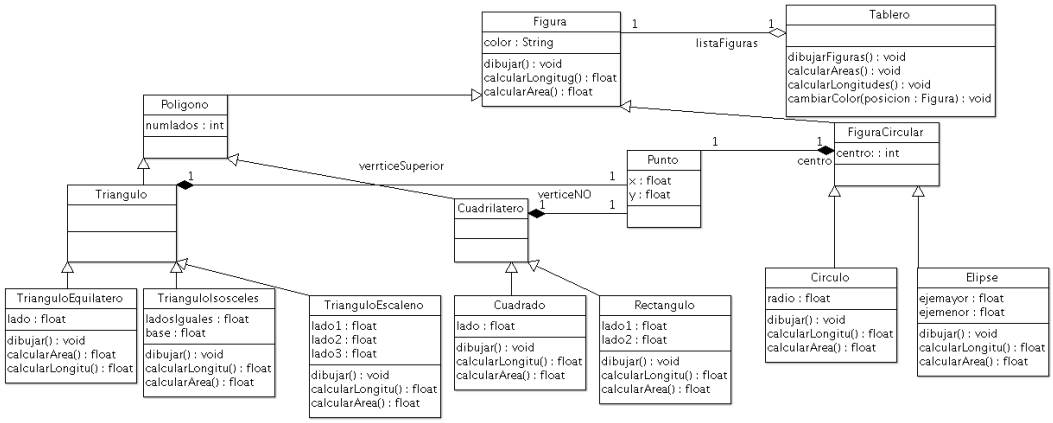


Ilustración 4. Modelo Ejercicio 5

El modelo presentado es mucho más extenso de lo que se pedía. *La agregación de punto puede o estar en el diseño que se haya planteado y el tablero puede que tampoco este y sea sustituido por un contenedor en el* `main` *siempre que sean arrays u otras estructuras que mantengan referencias de la clase base con objetos de clases derivadas contenidos en ellas, porque no habría polimorfismo.*

A continuación se muestra otro modelo en el que se han incorporado elemento para mostrar gráficamente las figuras. No se usan ventanas ni la biblioteca `Graphics`*. El ejemplo nuevamente está orientado al polimorfismo.*

Se ha considerado para la calificación únicamente la necesidad de la existencia de una jerarquía mínima de figuras y la invocación de alguna de las funciones polimórficas con referencia a la clase base.

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

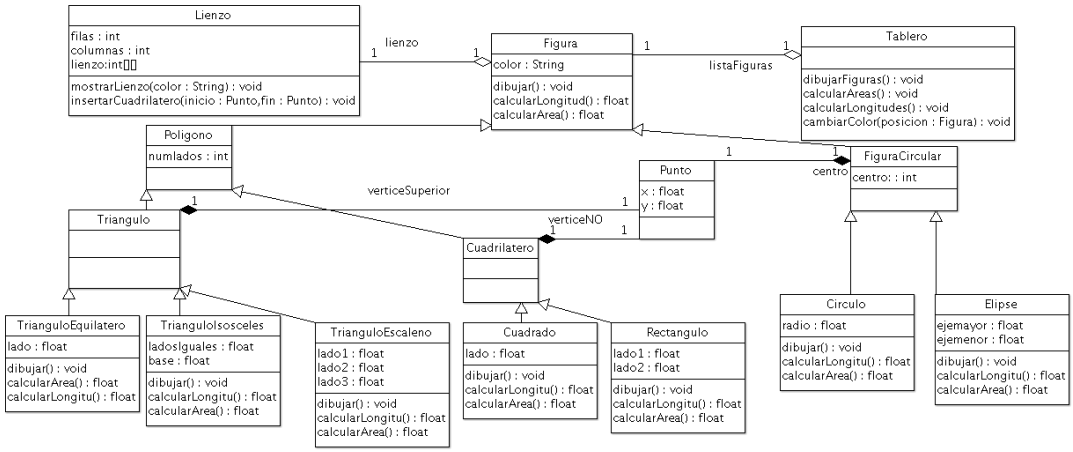


Ilustración 5. Modelo ejercicio 5 con lienzos de dibujo

A cada figura se le asociará un lienzo cuando vaya a ser dibujada. En la jerarquía de figuras hay muchas clases abstractas intermedias que o bien tiene atributos comunes a sus subclases o se construyen para agrupación conceptual.

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Implementación del modelo

Entidad	*Especificación*
Clase Tablero	`class Tablero` (

	Atributos:
	`private int numFiguras`
	`private int numElementos`
	**será el numero máximo de figurar(lienzos) que se dibujará en el tablero*

	Relaciones: ***Agregación deFiguras*** `private Figura[] figuras;` **se crea en el constructor aunque vacío.* ****las figuras se manejan a través de referencias a la base con lo que se usará el polimorfismo cuando se invoque a un método polimórfico (sobrecargado)***
	Metodos 1. `public void insertarfigura(Figura figura)` **inserta figuras distintas en el tablero* 2. `public void dibujarFiguras()` **implementa el polimorfismo porque invoca a un método polimórfico con referencias a la clase base(ligadura dinámica o tardía)* 3. `public void calcularAreas()` **implementa el polimorfismo porque invoca a un método polimórfico con referencias a la clase base(ligadura dinámica o tardía)* 4. `public void calcularLongitudes()` **implementa el polimorfismo porque invoca a un método polimórfico con referencias a la clase base(ligadura dinámica o tardía)* 5. `public void cambiarColor(int posicionFigura, String color)` **cambia el color de una figura del tablero*
Clase Figura	`abstract class Figura` *(clase base de la jerarquía)*
	Atributos: `private String color` **se podría haber creado un enumerado*
	Relaciones: ***Agregación de lienzo*** **se construye si se invoca al método* `dibujar`
	Metodos 6. `abstract public void dibujar()` 7. `abstract public double calcularLongitud()` 8. `abstract public double calcularArea()` **son funciones abstractas que serán sobrecargadas en las clases derivadas para implementar el dibujo y el calculo de las dimensiones indicadas* ****son funciones polimórficas*** ***Setters y getters.*** *Se implementan para modificación o acceso a atributos privados*
Clase Punto	`class Punto`

	Atributos:
	`private int x`
	`private int y`
	**se han considerado las coordenadas como* `int` *porque se va a dibujar un lienzo como una matriz. En la realidad las coordenadas deberían ser* `double` *o* `float`

	Relaciones
	Metodos getters
Clase Lienzo	`class Lienzo`

	Atributos:
	`private final int filas = 30;`
	`private final int columnas = 60;`
	`private int[][] lienzo;`
	**número de filas y columnas del lienzo (constantes final)*

	Relaciones

	Metodos 1. `public void mostrarLienzo(String color)` **dependiendo de lo que haya en el array de lienzo, muestra carácter(-)para marcar los bordes del lienzo, blanco si no tiene nada en ese lugar y la letra del color de la figura en el área de la figura* 2. `public void insertarCuadrilatero(Punto inicio, Punto fin)` **calcula lo que hay en cada punto del lienzo* 3. `public void calcularAreas()` **implementa el polimorfismo porque invoca a un método polimórfico con referencias a la clase base(ligadura dinámica o tardía)* 4. `public void calcularLongitudes()` **implementa el polimorfismo porque invoca a un método polimórfico con referencias a la clase base(ligadura dinámica o tardía)* 5. `public void cambiarColor(int posicionFigura, String color)` ****cambia el color de una figura del tablero***
Clase Cuadrilatero	`abstract class Cuadrilatero extends Figura`
	Atributos: `protected Punto verticeNO` **hereda numlados y Color de las clases superiores*
	Relación de agregación Punto `protected Punto verticeNO` **punto de partida para dibujar el cuadrilatero*
	Metodos *No implementa los métodos abstractos*
Clase Rectangulo,	`class Rectangulo extends Cuadrilatero`

	Atributos:
	`private int ancho`
	`private int alto`
	**se han considerado las magnitudes como* `int` *porque se va a dibujar un lienzo como una matriz y se necesitan calcular desplazamientos enteros. En la realidad las coordenadas deberían ser* `double` *o* `float`
	****hereda el vértice Noroeste de la clase Poligono (abstracta) a partir del que se comienza a dibujar la figura y el numero de lado(no se usa), además del color de la Figura***

	Relaciones de herencia `class Rectangulo extends Cuadrilatero`
	Metodos 1. `public void dibujar()` **crea el lienzo, calcula el punto Sureste, inserta el rectángulo en el lienzo y lo muestra. Imporme el área y la longitud.* 2. `public double calcularLongitud()` 3. `public double calcularArea()` **calculo de las dimensiones indicadas* ****son funciones polimórficas*** *Setters y getters.* ***Se implementan para modificación o acceso a atributos privados***
Clase Cuadrado	`class Cuadrado extends Cuadrilatero`

	Atributos:
	`private int lado`
	`private int alto`
	**se han considerado las magnitudes como* `int` *porque se va a dibujar un lienzo como una matriz y se necesitan calcular desplazamientos enteros. En la realidad las coordenadas deberían ser* `double` *o* `float`
	****hereda el vértice Noroeste de la clase Poligono (abstracta) a partir del que se comienza a dibujar la figura y el numero de lado(no se usa), además del color de la Figura***

	Relaciones de herencia `class Cuadrado extends Cuadrilatero`
	Metodos 1. `public void dibujar()` **crea el lienzo, calcula el punto Sureste, inserta el cuadrado en el lienzo y lo muestra. Imporme el área y la longitud.* 2. `public double calcularLongitud()` 3. `public double calcularArea()` **calculo de las dimensiones indicadas* ****son funciones polimórficas*** ***Setters y getters.*** *Se implementan para modificación o acceso a atributos privados*

Las demás clases se implementan de formas similares teniendo en cuenta sus características.

En la clase principal se han definido funciones para toma de datos.

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	

Clases y ejecutable

Se adjunta en el campus virtual

https://campusvirtual.unir.net/access/lessonbuilder/item/5453233/group/PER1582-439-8206-1582/Documentación/Ejercicio5.zip

Asignatura	Datos del alumno	Fecha
Métodos Avanzados de Programación Científica y Computación	Apellidos:	
	Nombre:	