

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

Laboratorio: Diseño y modelado de clases.

Herencia en programación es una característica de optimización de la programación orientada a objetos, en donde una clase padre consta de métodos y atributos que puede compartir con una clase hijo. Básicamente esto sirve para no duplicar métodos ni atributos. Antes de iniciar a programar, debemos diseñar el programa que queremos realizar, y esto lo hacemos a través de diagramas UML(Unified Modeling Language), y entre los diferentes métodos de diagramas que existen en UML, existe un diagrama para realizar las clases del desarrollo.

Un Diagrama de Clases obtiene las clases de los objetos y la manera de asociar las mismas por medio de relaciones. Entre las relaciones mas comunes existen las de Asociación, Herencia, Agregación, Composición y Dependencia.

1. Ejercicio 1

1.1. Procedimiento de diseño de clases.

Se analiza profundamente la documentación del problema y se crea una clase padre abstracta con los atributos y métodos que se puedan tener en común entre las clases. Luego se analizan los métodos y atributos que caracterizan diferencias para crear las clases hijo.

En el caso del Ejercicio 1, para crear la clase padre se tiene en común un ID único, de la misma manera se ha agregado el nombre de un producto para poder saber cual producto es el de cada ID. En las clases hijo se tiene un producto físico que contiene el peso, alto, ancho, largo y coste de transporte del producto, además del nombre y del ID generados en la clase padre y heredados a la clase hijo. En el caso del producto virtual se tiene solamente el numero de la licencia virtual, ya que en el enunciado se muestra que virtualmente se cuenta solo con licencias, además de la herencia del ID y el nombre del producto. Apuntes de clase. [1].

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

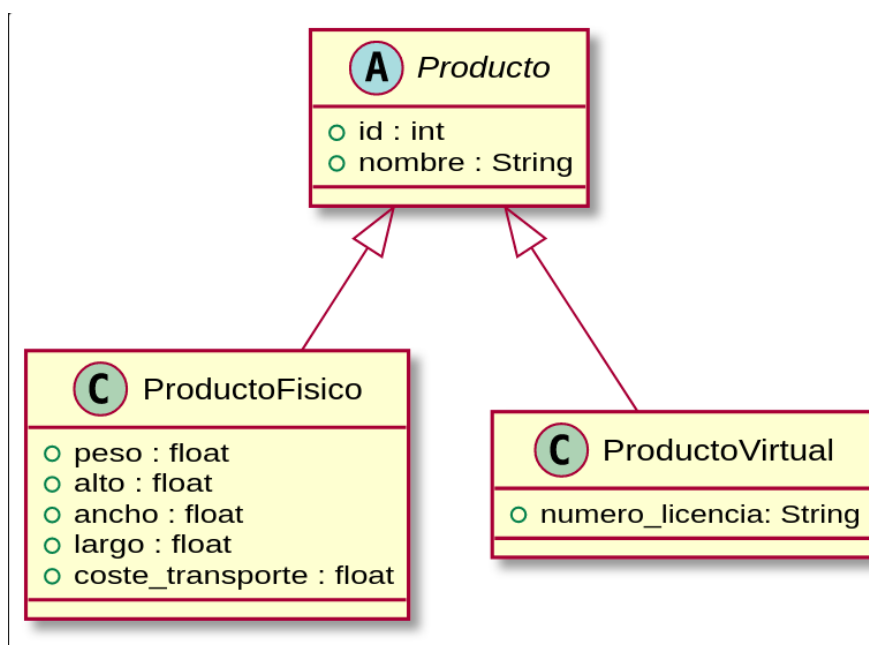


Figura 1: Diagrama de clases de ejercicio 1

1.2. Diagrama de clases.

El diagrama de clases de la figura 1 fue generado a partir de una herramienta *Open Source* llamada *PlantUML*. El código para generar el diagrama de clases es el siguiente:

```

1      @startuml
2          abstract class Producto{
3              +id : int
4              +nombre : String      }
5
6          class ProductoFisico{
7              +peso : float
8              +alto : float
9              +ancho : float
10             +largo : float
11             +coste_transporte : float      }
12
13         class ProductoVirtual{
  
```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

14         +numero_licencia: String    }
15
16     Producto <|-- ProductoFisico
17     Producto <|-- ProductoVirtual
18 @enduml

```

Listing 1: Código con sintaxis PlantUML para generar grafico de diagrama de clases del ejercicio 1.

1.3. Creando código Java a través de diagrama de clases.

Existen varios plugins para Netbeans, Eclipse y demás entornos de desarrollo para crear las clases de Java a través de los diagramas UML, sin embargo para fines demostrativos, realizamos las clases a mano en función de lo que pide el laboratorio.

La clase padre se llama *Producto*, a partir de la clase padre, se crean 2 clases hijo, que son *ProductoFisico* y *ProductoVirtual*. El código correspondiente a las clases se encuentra a continuación.

```

1     public abstract class Producto {
2         private String ID;
3         private String Nombre;
4
5         public String get_id(){
6             return ID;
7         }
8
9         public String get_nombre(){
10            return Nombre;
11        }
12    }
13
14    public class ProductoFisico extends Producto {
15        private float peso;
16        private float alto;

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

17     private float ancho;
18     private float largo;
19     private float coste_transporte;
20
21     public ProductoFisico(float peso, float alto, float ancho, float ...
        largo, float costo){
22         this.peso = peso;
23         this.alto = alto;
24         this.ancho = ancho;
25         this.largo = largo;
26         this.coste_transporte = costo;
27     }
28 }
29
30 public class ProductoVirtual {
31     private String numero_licencia;
32
33     public ProductoVirtual(String numero_licencia){
34         this.numero_licencia = numero_licencia;
35     }
36 }

```

Listing 2: Código de Java del ejercicio 1

2. Ejercicio 2

2.1. Procedimiento de diseño de clases

El procedimiento de diseño de clases ha sido el mismo que en el ejemplo 1, solamente se han considerado componentes que son necesarios para desarrollar compras en una web, como el stock de los productos, la dirección de entrega, facturación, etc. Todo esto se ha considerado de la manera mas óptima posible sin extenderse demasiado en el diagrama a modo de ejercicio, ya que en un ambiente en la vida real seguramente las clases necesitan muchos mas campos posiblemente distribuidos de

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

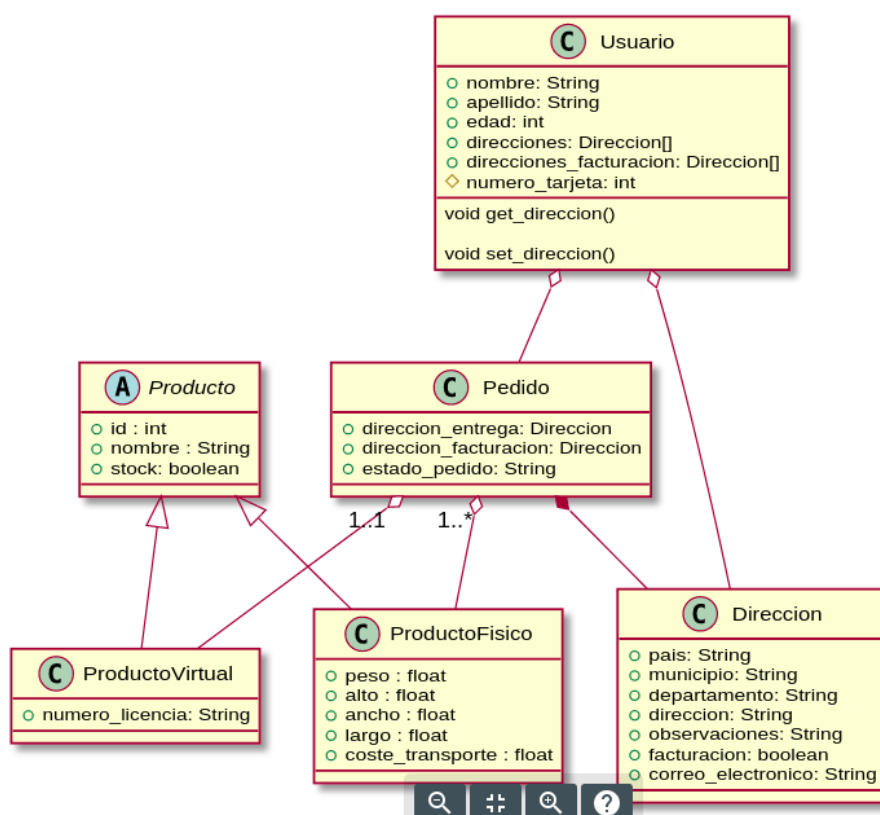


Figura 2: Diagrama de clases de ejercicio 2

otra manera, pero como ya se ha mencionado, esto es en calidad de ejercicio de la actividad de laboratorio.

Se ha considerado solamente una clase Abstracta que hace referencia a la clase producto. y es la única clase padre de todo el desarrollo. Sus clases hijas son ProductoVirtual y ProductoFisico. Luego, todas las clases tienen diferentes combinaciones entre agregaciones y composiciones en base a las cuales se le dará utilidad. La manera en la que se relacionan todas las clases se desarrolla junto al diagrama de clases. Apuntes de clase. [1].

2.2. Diagrama de clases

El diagrama de clases de la figura 2 fue generado a partir de una herramienta *Open Source* llamada *PlantUML*. El código para generar el diagrama de clases es el siguiente:

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

1      @startuml
2          abstract class Producto{
3              +id : int
4              +nombre : String
5              +stock: boolean}
6
7          class ProductoFisico{
8              +peso : float
9              +alto : float
10             +ancho : float
11             +largo : float
12             +coste_transporte : float}
13
14         class ProductoVirtual{
15             +numero_licencia: String}
16
17         class Usuario{
18             +nombre: String
19             +apellido: String
20             +edad: int
21             +direcciones: Direccion[]
22             +direcciones_facturacion: Direccion[]
23             #numero_tarjeta: int
24             ' Getter
25             void get_direccion()
26             ' Setter
27             void set_direccion() }
28
29         class Direccion{
30             +pais: String
31             +municipio: String
32             +departamento: String
33             +direccion: String

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

34         +observaciones: String
35         +facturacion: boolean
36         +correo_electronico: String}
37
38     class Pedido{
39         +direccion_entrega: Direccion
40         +direccion_facturacion: Direccion
41         +estado_pedido: String}
42
43     Producto <|-- ProductoFisico
44     Producto <|-- ProductoVirtual
45     Usuario o-- Direccion
46     Pedido *-- Direccion
47     Pedido "1..*" o-- ProductoFisico
48     Pedido "1..1" o-- ProductoVirtual
49     Usuario o-- Pedido
50 @enduml

```

Listing 3: Código con sintaxis PlantUML para generar gráfico de diagrama de clases del ejercicio 2.

El diagrama de clases de la figura 2 tiene una clase abstracta como clase padre llamada producto. Esta clase contiene el ID del producto y nombre del ejercicio 1, pero también fue añadido el stock del producto para poder hacer compras en una web. De esta clase heredan 2 clases mas, que son la de producto físico y producto virtual, tal y como se muestra en el ejercicio 1, solo que en este caso son agregaciones de una clase llamada pedido, la cual contiene todos los productos físicos que se agreguen al pedido, un producto virtual en caso de necesitarse. Al ser agregaciones puede que el pedido se mantenga en blanco en algún momento, siendo el coste total de \$0.00, mientras que si fueran composiciones, deberíamos de tener al menos una unidad, sin embargo no sabemos si el cliente requerirá solamente una unidad de un producto físico o una unidad de un producto virtual, así que es preferente dejarlos como agregaciones.

La clase pedido tiene una composición de la clase dirección, ya que un pedido obligatoriamente debe tener una dirección. En este caso por simplicidad del diagrama fue agregada la dirección de correo

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

electrónico dentro de la misma clase de direcciones, ya que un producto virtual necesita de una dirección de correo electrónico, mientras que un producto físico necesita una dirección física.

La clase usuario tiene una agregación de la clase Dirección, y no se considera solamente la composición de la dirección hacia el pedido. Esto es debido a que un usuario puede tener una dirección totalmente diferente a la dirección de entrega del pedido, tanto en dirección física como en dirección de correo electrónico (Suponiendo que un pedido puede ser una entrega de regalo). En este caso por simplicidad se agrego la factura como un booleano, solamente para determinar si la factura se desea en papel o factura electrónica con los datos del usuario.

La clase usuario cuenta con los datos necesarios para poder hacer un pedido, al igual que una agregación de la clase Dirección que es totalmente independiente a la dirección de entrega. Esto como datos personales en caso de quejas o reportes. Apuntes de clase. [1].

2.3. Creando código Java a través de diagrama de clases.

```

1 public abstract class Producto {
2     private String ID;
3     private String Nombre;
4     private boolean Stock;
5
6     public Producto(String ID, String Nombre, boolean Stock){
7         this.ID = ID;
8         this.Nombre = Nombre;
9         this.Stock = Stock;
10    }
11
12    public String get_id(){
13        return ID;
14    }
15
16    public String get_nombre(){

```


Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

17         return Nombre;
18     }
19
20     public boolean get_stock(){
21         return stock;
22     }
23
24     public void set_stock(value){
25         this.stock = value;
26     }
27 }
28
29 public class ProductoVirtual extends Producto{
30     private String numero_licencia;
31
32     public ProductoVirtual(String ID, String Nombre, boolean Stock, ...
        String numero_licencia){
33         super(ID, Nombre, Stock);
34         this.numero_licencia = numero_licencia;
35     }
36 }
37
38 public class ProductoFisico extends Producto {
39     private float peso;
40     private float alto;
41     private float ancho;
42     private float largo;
43     private float coste_transporte;
44
45     public ProductoFisico(String ID, String Nombre, boolean Stock, float ...
        peso, float alto, float ancho, float largo, float costo){
46         super(ID, Nombre, Stock);
47         this.peso = peso;
48         this.alto = alto;

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

49         this.ancho = ancho;
50         this.largo = largo;
51         this.coste_transporte = costo;
52     }
53 }
54
55 public class Direccion {
56     private String Pais;
57     private String Municipio;
58     private String Departamento;
59     private String Direccion;
60     private String Observaciones;
61     private String Email;
62     private boolean Facturacion;
63
64     public Direccion() {}
65
66     public Direccion(String Email, boolean Fact){
67         this.Email = Email;
68         this.Facturacion = Fact;
69     }
70
71     public Direccion(String Pais, String Mun, String Depto, String Dir, ...
72         String Obs, boolean Fact){
73         this.Pais = Pais;
74         this.Municipio = Mun;
75         this.Direccion = Dir;
76         this.Observaciones = Obs;
77         this.Facturacion = Fact;
78     }
79 }
80
81 public class Pedido {

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

82     private final Direccion Dir_Entrega;
83     private ProductoFisico ProductoFisico;
84     private ProductoVirtual ProductoVirtual;
85     private String estado_pedido;
86
87     public Pedido(String estado_pedido){
88         this.Dir_Entrega = new Direccion();
89         this.estado_pedido = estado_pedido;
90     }
91 }
92
93 public class Usuario {
94     private String Nombre;
95     private String Apellido;
96     private int Edad;
97     private int Numero_Tarjeta;
98     private Direccion[] Domicilios;
99     private Pedido Pedidos;
100
101     public Usuario(){}
102 }

```

Listing 4: Código de Java del ejercicio 2

3. Ejercicio 3

3.1. Procedimiento de diseño de clases

El procedimiento para el diagrama de clases del ejemplo 3 tiene algunas variaciones respecto al ejemplo 2, dado que en este caso se agrega una clase llamada inventario y se crean agregaciones de *ProductoFisico* y *Productivirtual*, ya que en este caso se debe pasar por inventario antes de gestionar una compra.

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

Un usuario tiene una agregación de un pedido, por lo cual para que el usuario exista, no es necesario tener al menos un pedido. El pedido tiene composiciones de dirección y de inventario, ya que ambas clases son necesarias para poder generar un pedido, mientras que el inventario tiene agregaciones de producto físico y producto virtual, ya que no es necesario tener siempre productos en el inventario para que estos existan. La figura 3 muestra el diagrama realizado para resolver el ejercicio 3. Apuntes de clase. [1].

```

1 @startuml
2     abstract class Producto{
3         +id : int
4         +nombre : String
5         -stock: boolean
6         -conteo_producto: int
7         void get_stock()
8         void set_stock()
9         void set_conteo()
10        void get_conteo()    }
11
12    class ProductoFisico{
13        +peso : float
14        +alto : float
15        +ancho : float
16        +largo : float
17        +coste_transporte : float    }
18
19    class ProductoVirtual{
20        +numero_licencia: String    }
21
22    class Usuario{
23        +nombre: String
24        +apellido: String
25        +edad: int

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

26         +direcciones: Direccion[]
27         +direcciones_facturacion: Direccion[]
28         #numero_tarjeta: int
29         ' Getter
30         void get_direccion()
31         ' Setter
32         void set_direccion()      }
33
34     class Direccion{
35         +pais: String
36         +municipio: String
37         +departamento: String
38         +direccion: String
39         +observaciones: String
40         +facturacion: boolean
41         +correo_electronico: String      }
42
43     class Pedido{
44         +direccion_entrega: Direccion
45         +direccion_facturacion: Direccion
46         +estado_pedido: String      }
47
48     class Inventario{
49         +productos_fisicos[ProductoFisico]
50         +productos_virtuales[ProductoVirtual]
51
52         void agregar_stock()
53         void quitar_stock(int id)      }
54
55     Producto <|-- ProductoFisico
56     Producto <|-- ProductoVirtual
57     Usuario o-- Direccion
58     Pedido *-- Direccion
59     Usuario o-- Pedido

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

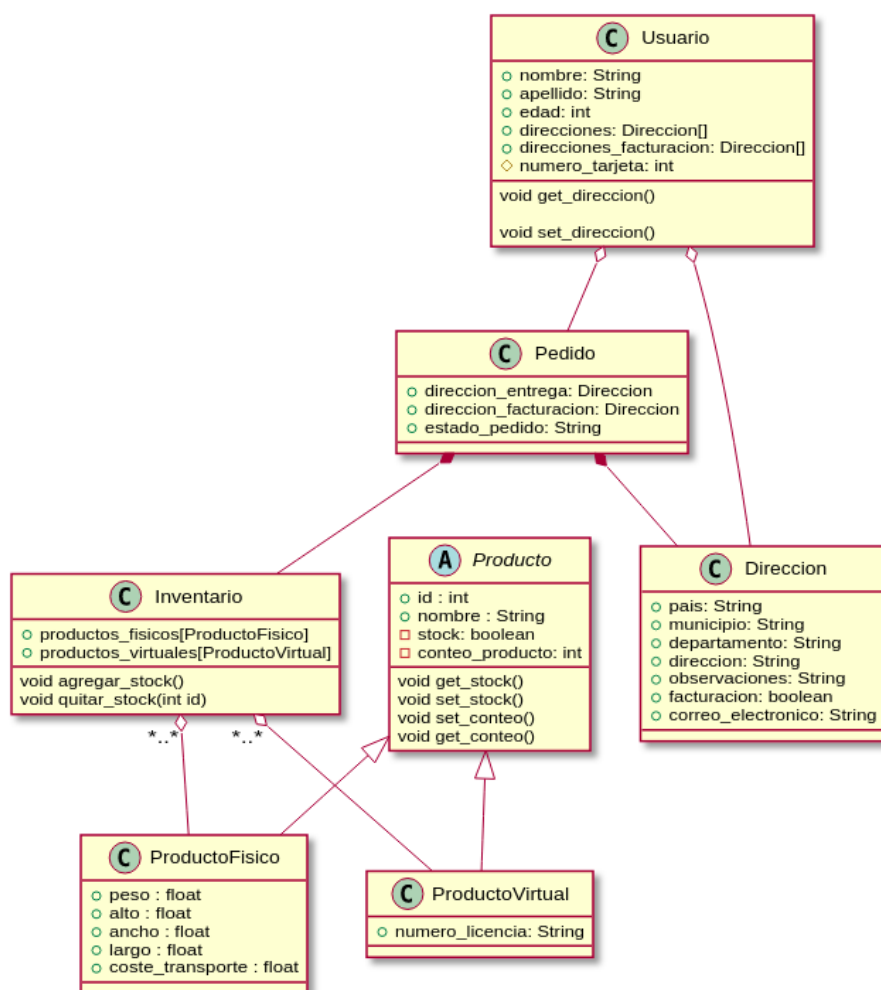


Figura 3: Diagrama de clases de ejercicio 3

```

60     Inventario "*" o-- ProductoFisico
61     Inventario "*" o-- ProductoVirtual
62     Pedido *-- Inventario
63 @enduml

```

Listing 5: Código para generar gráfico de diagrama de clases del ejercicio 3.

```

1 public abstract class Producto {
2     private String ID;
3     private String Nombre;

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

4     private boolean Stock;
5
6     public Producto(String ID, String Nombre, boolean Stock){
7         this.ID = ID;
8         this.Nombre = Nombre;
9         this.Stock = Stock;
10    }
11
12    public String get_id(){
13        return ID;
14    }
15
16    public String get_nombre(){
17        return Nombre;
18    }
19
20    public boolean get_stock(){
21        return Stock;
22    }
23 }
24
25 public class ProductoVirtual extends Producto{
26     private String numero_licencia;
27
28     public ProductoVirtual(String ID, String Nombre, boolean Stock, ...
29         String numero_licencia){
30         super(ID, Nombre, Stock);
31         this.numero_licencia = numero_licencia;
32     }
33
34 public class ProductoFisico extends Producto {
35     private float peso;
36     private float alto;

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

37     private float ancho;
38     private float largo;
39     private float coste_transporte;
40
41     public ProductoFisico(String ID, String Nombre, boolean Stock, float ...
        peso, float alto, float ancho, float largo, float costo){
42         super(ID, Nombre, Stock);
43         this.peso = peso;
44         this.alto = alto;
45         this.ancho = ancho;
46         this.largo = largo;
47         this.coste_transporte = costo;
48     }
49 }
50
51 public class Direccion {
52     private String Pais;
53     private String Municipio;
54     private String Departamento;
55     private String Direccion;
56     private String Observaciones;
57     private String Email;
58     private boolean Facturacion;
59
60     public Direccion(){
61
62     }
63
64     public Direccion(String Email, boolean Fact){
65         this.Email = Email;
66         this.Facturacion = Fact;
67     }
68
69     public Direccion(String Pais, String Mun, String Depto, String Dir, ...

```


Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

        String Obs, boolean Fact){
70         this.Pais = Pais;
71         this.Municipio = Mun;
72         this.Departamento = Depto;
73         this.Direccion = Dir;
74         this.Observaciones = Obs;
75         this.Facturacion = Fact;
76     }
77 }
78
79 public class Inventario {
80     private ProductoFisico[] ProductoFisico;
81     private ProductoVirtual[] ProductoVirtual;
82
83     public Inventario(){
84     }
85 }
86
87 public class Pedido {
88     private final Direccion Dir_Entrega;
89     private final Inventario Inventario;
90     private String estado_pedido;
91
92     public Pedido(String estado_pedido){
93         this.Dir_Entrega = new Direccion();
94         this.Inventario = new Inventario();
95         this.estado_pedido = estado_pedido;
96     }
97 }
98
99 public class Usuario {
100     private String Nombre;
101     private String Apellido;
102     private int Edad;

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

103     private int Numero_Tarjeta;
104     private Direccion[] Domicilios;
105     private Pedido Pedidos;
106
107     public Usuario() {
108     }
109 }

```

Listing 6: Código de Java del ejercicio 3

4. Ejercicio 4

4.1. Procedimiento de diseño de clases y polimorfismo

La clase *Vehiculo* es en este caso la clase padre, por lo tanto es una clase abstracta, y contiene como atributo el color del vehículo. Esta clase tiene método constructor en el que se agrega el color ingresado en los parámetros de la función, hacia el atributo de la clase. El método *get_color()* retorna el valor que tiene el atributo *color*, mientras que el método *calcular_consumo()* únicamente esta definido, porque sera diferente para cada una de las clases hijo, este método sera obligatorio implementarlo en las clases hijo que extiendan la clase abstracta. Es en esta parte donde funciona el polimorfismo.

```

1 package ejercicio_4;
2 public abstract class Vehiculo {
3     private String color;
4
5     public Vehiculo(String color) {
6         this.color = color;
7     }
8
9     public abstract float calcular_consumo();

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

10
11     public String get_color() {
12         return color;
13     }
14 }

```

Listing 7: Código de la clase Vehículo, ejercicio 4.

La clase *Coche* hereda de la clase abstracta *Vehículo* mencionada anteriormente, y de la misma manera tiene un método constructor en donde se ingresa el color. Es necesario destacar que el color en este caso, no es un atributo de la clase *coche*, sino de la clase *Vehículo*, por lo cual se utiliza la función *super()* para enviar el parámetro al método constructor de la clase padre. El método *calcular_consumo()* hace referencia al cálculo del consumo del vehículo.

```

1 package ejercicio_4;
2 public class Coche extends Vehiculo{
3     private float potencia;
4
5     public Coche(String color, float potencia) {
6         super(color);
7         this.potencia = potencia;
8     }
9
10    public float calcular_consumo() {
11        return (float) (this.potencia * 4.0 * 0.12);
12    }
13 }

```

Listing 8: Código de la clase Coche, ejercicio 4.

La clase *Moto*, de igual manera, hereda de la clase *Vehículo*, y también tiene un método constructor en donde se ingresa el color que, por medio de una función *super()* lo asigna directamente al constructor de la clase *vehículo*. También tiene un método *calcular_consumo()* en donde se calcula

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

el consumo de potencia según la cantidad de ruedas que tiene este vehículo.

```

1 package ejercicio_4;
2 public class Moto extends Vehiculo{
3     private float potencia;
4
5     public Moto(String color, float potencia) {
6         super(color);
7         this.potencia = potencia;
8     }
9
10    public float calcular_consumo() {
11        return (float) (this.potencia * 2.0 * 0.12);
12    }
13 }

```

Listing 9: Código de la clase Moto, ejercicio 4.

La clase *Main()* es la clase principal, en donde se crean los objetos de clase *Moto* y objetos de clase *Coche*. En esta clase mostramos en pantalla el consumo de cada vehículo según la cantidad de ruedas que este tiene utilizando polimorfismo.

```

1 package ejercicio_4;
2 public class Main {
3     public static void main(String[] args) {
4         Coche toyota = new Coche("rojo", (float) 20);
5         Moto toyota_moto = new Moto("rojo", (float) 20);
6
7         System.out.println("Consumo para Coche:");
8         System.out.println(toyota.calcular_consumo());
9         System.out.println("Consumo para Moto:");
10        System.out.println(toyota_moto.calcular_consumo());
11    }

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

12 }

Listing 10: Código de la clase Main, ejercicio 4.

El polimorfismo se refiere a *muchas formas* entre métodos cuando hay una jerarquía de clases asociadas por herencia, ya que un método de la clase padre puede ser útil para una clase hijo pero para otra no, sin embargo, con polimorfismo podemos adecuar los métodos para la necesidad de cada clase.

5. Ejercicio 5

5.1. Procedimiento de diseño de clases

En el caso del ejercicio 5 se definió una jerarquía de clases, Figura 5, teniendo 2 clases abstractas que encapsulan ciertas características que tienen en común las figuras. La clase abstracta de mayor jerarquía y de la cual heredan las demás es la clase *Figura*, tiene definidos los atributos color y nombre, ya que cualquier figura que definamos después tiene que tener el atributo color, en esta clase también se definieron los métodos abstractos *area* y *perimetro*, los cuales tendrán que ser implementados de forma obligatoria en las clases que hereden de figura. Los polígonos están definidos como una figura geométrica con un número finito de lados rectos, teniendo en cuenta este concepto se definió la clase abstracta *Poligono* que encapsula muchas características en común de figuras geométricas categorizadas como polígonos, para efectos de este trabajo la única característica o atributo que se definió para polígono es el número de lados. las clases *Rectangulo* y *TrianguloEquilatero* heredan de la clase polígono, tiene sus propios atributos característicos de la figura geométrica. La clase *Circulo* hereda de la clase abstracta *Figura* y tiene el atributo radio.

5.2. Implementación de ejercicio 5

La clase abstracta *Figura* tiene los atributos color y nombre, también la definición de los métodos *area* y *nombre* los cuales serán implementados en las clases hijo.

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

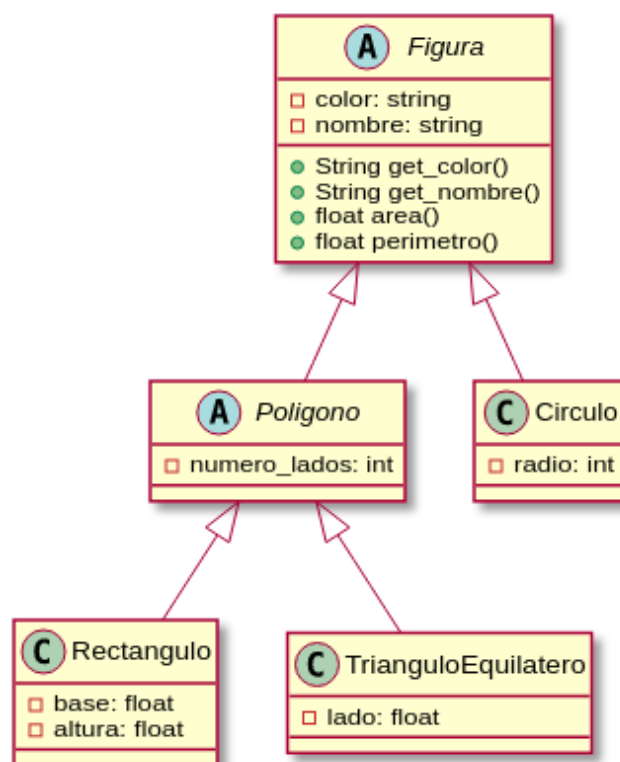


Figura 4: Diagrama de clases de ejercicio 5

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

1      package ejercicio_5;
2
3      public abstract class Figura {
4          private String color;
5          private String nombre;
6
7          public Figura(String color, String nombre) {
8              this.color = color;
9              this.nombre = nombre;
10         }
11
12         public abstract float area();
13         public abstract float perimetro();
14
15         public String get_color() {
16             return color;
17         }
18
19         public String get_nombre() {
20             return nombre;
21         }
22
23     }

```

Listing 11: Código de la clase *Figura*, ejercicio 5.

En la clase abstracta *Poligono*, no es necesario implementar los métodos de la clase *Figura*, pero si es obligatorio implementarlos en las clases que heredan de *Poligono*, ya que cada polígono tiene formulas diferentes para calcular el área.

```

1
2      package ejercicio_5;
3

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

4      public abstract class Poligono extends Figura{
5          private int numero_lados;
6
7          public Poligono(String color, String nombre, int ...
              numero_lados) {
8              super(color, nombre);
9              this.numero_lados = numero_lados;
10         }
11
12         public int get_numero_lados() {
13             return numero_lados;
14         }
15
16     }

```

Listing 12: Código de la clase Poligono, ejercicio 5.

Las clases *Rectangulo* y *TrianguloEquilatero* heredan de Poligono, y esta a su vez hereda de *Figura*, por lo que es necesario implementar en estas clases los métodos área y perímetro, podemos ver que en cada una de las clases es diferente la implementación ya que cada figura tiene diferente fórmula para el cálculo del área.

```

1      package ejercicio_5;
2
3      public class Rectangulo extends Poligono{
4          private float base;
5          private float altura;
6
7
8          public Rectangulo(String color, String nombre, float base, float ...
              altura) {
9              super(color, nombre, 4);
10             this.base = base;

```


Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

11         this.altura = altura;
12     }
13
14     @Override
15     public float area() {
16         return base*altura;
17     }
18
19     @Override
20     public float perimetro() {
21         return (2*base) + (2*altura);
22     }
23
24 }
```

Listing 13: Código de la clase Rectangulo, ejercicio 5.

```

1     package ejercicio_5;
2
3     import java.lang.Math;
4
5     public class TrianguloEquilatero extends Poligono{
6         private int lado;
7
8         public TrianguloEquilatero(String color, String nombre, int lado) {
9             super(color, nombre, 3);
10            this.lado = lado;
11        }
12
13        @Override
14        public float area() {
15            return (float) (Math.pow(lado,2) * (Math.sqrt(3)/4));
16        }

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

17
18     @Override
19     public float perimetro() {
20         return this.get_numero_lados() * lado;
21     }

```

Listing 14: Código de la clase TrianguloEquilatero, ejercicio 5.

```

1     package ejercicio_5;
2
3     public class Circulo extends Figura{
4         private float radio;
5
6         public Circulo(String color, String nombre, float radio) {
7             super(color, nombre);
8             this.radio = radio;
9         }
10
11        public float get_radio() {
12            return radio;
13        }
14
15        @Override
16        public float area() {
17            return (float) (Math.PI * Math.pow(radio, 2));
18        }
19
20        @Override
21        public float perimetro() {
22            return (float) (Math.PI * 2 * radio) ;
23        }
24
25    }

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

Listing 15: Código de la clase Circulo, ejercicio 5.

5.2.1. Evaluación de la implementación

Se implementó también una clase donde se definen diferentes figuras y se calcula el área y perímetro de las mismas, de esta forma podemos evaluar el funcionamiento de la implementación. Podemos ver que todas las figuras reciben como parámetros el color y el nombre los cuales son requeridos por la clase padre *Figura*.

```

1 public class TestFiguras {
2     public static void main(String[] args) {
3
4         Rectangulo cuadrado = new Rectangulo("verde", "Cuadrado", 2, 2);
5         Rectangulo rectangulo = new Rectangulo("verde", "Rectangulo", 2, 4);
6         TrianguloEquilatero triangulo = new TrianguloEquilatero("azul", ...
            "Triangulo equilatero", 4);
7         Circulo circulo = new Circulo("rojo", "Circulo", 5);
8
9         System.out.println("==== Poligonos ====");
10        System.out.println("Nombre Figura: "+ cuadrado.get_nombre());
11        System.out.println(
12            ">> Color: " + cuadrado.get_color() + " | Area: " + ...
            cuadrado.area() +
13            " | Perimetro: " + cuadrado.perimetro() + " | Numero ...
            Lados: " + cuadrado.get_numero_lados());
14
15        System.out.println("Nombre Figura: "+ rectangulo.get_nombre());
16        System.out.println(
17            ">> Color: " + rectangulo.get_color() + " | Area: " + ...
            rectangulo.area() +
18            " | Perimetro: " + rectangulo.perimetro() + " | Numero ...

```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

19         Lados: " + cuadrado.get_numero_lados());
20
21     System.out.println("Nombre Figura: "+ triangulo.get_nombre());
22     System.out.println(
23         ">> Color: " + triangulo.get_color() + " | Area: " + ...
24         triangulo.area() +
25         " | Perimetro: " + triangulo.perimetro() + " | Numero ...
26         Lados: " + triangulo.get_numero_lados());
27
28     System.out.println("");
29     System.out.println("==== Circulos ====");
30     System.out.println("Nombre Figura: "+ circulo.get_nombre());
31     System.out.println(
32         ">> Color: " + circulo.get_color() + " | Area: " + ...
33         circulo.area() +
34         " | Circunferencia: " + circulo.perimetro() + " | Radio: ...
35         " + circulo.get_radio());
36
37 }
38
39 }
```

Listing 16: Código para evaluar la implementación, ejercicio 5.

```

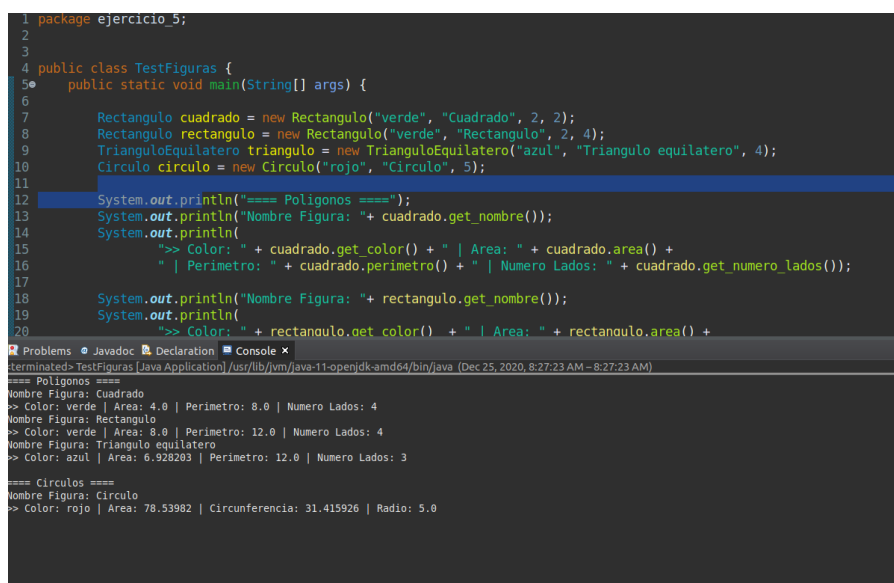
1  ==== Poligonos ====
2  Nombre Figura: Cuadrado
3  >> Color: verde | Area: 4.0 | Perimetro: 8.0 | Numero Lados: 4
4  Nombre Figura: Rectangulo
5  >> Color: verde | Area: 8.0 | Perimetro: 12.0 | Numero Lados: 4
6  Nombre Figura: Triangulo equilatero
7  >> Color: azul | Area: 6.928203 | Perimetro: 12.0 | Numero Lados: 3
8
```

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

```

1 package ejercicio_5;
2
3
4 public class TestFiguras {
5     public static void main(String[] args) {
6
7         Rectangulo cuadrado = new Rectangulo("verde", "Cuadrado", 2, 2);
8         Rectangulo rectangulo = new Rectangulo("verde", "Rectangulo", 2, 4);
9         TrianguloEquilatero triangulo = new TrianguloEquilatero("azul", "Triangulo equilatero", 4);
10        Circulo circulo = new Circulo("rojo", "Circulo", 5);
11
12        System.out.println("==== Poligonos ====");
13        System.out.println("Nombre Figura: "+ cuadrado.get_nombre());
14        System.out.println(
15            ">> Color: " + cuadrado.get_color() + " | Area: " + cuadrado.area() +
16            " | Perimetro: " + cuadrado.perimetro() + " | Numero Lados: " + cuadrado.get_numero_lados());
17
18        System.out.println("Nombre Figura: "+ rectangulo.get_nombre());
19        System.out.println(
20            ">> Color: " + rectangulo.get_color() + " | Area: " + rectangulo.area() +

```



```

==== Poligonos ====
Nombre Figura: Cuadrado
>> Color: verde | Area: 4.0 | Perimetro: 8.0 | Numero Lados: 4
Nombre Figura: Rectangulo
>> Color: verde | Area: 8.0 | Perimetro: 12.0 | Numero Lados: 4
Nombre Figura: Triangulo equilatero
>> Color: azul | Area: 6.928203 | Perimetro: 12.0 | Numero Lados: 3

==== Circulos ====
Nombre Figura: Circulo
>> Color: rojo | Area: 78.53982 | Circunferencia: 31.415926 | Radio: 5.0

```

Figura 5: Probando implementacion ejercicio 5

```

9  ==== Circulos ====
10 Nombre Figura: Circulo
11 >> Color: rojo | Area: 78.53982 | Circunferencia: 31.415926 | Radio: 5.0

```

Listing 17: Resultado de ejecución, ejercicio 5.

6. anexos

PlantUML es un software con el que se puede generar diagramas UML a través de una sintaxis específica, tal y como se muestra en la figura 6

Referencias

- [1] Maria L. Diez Platas. Apuntes de clase de métodos avanzados de programación científica y computación, 2020.

Asignatura	Datos de los alumnos	Fecha
Métodos Avanzados de Programación Científica.	Balsells Orellana, Jorge Augusto	25/12/2020
	Díaz Saborio, Erick Wilfredo	

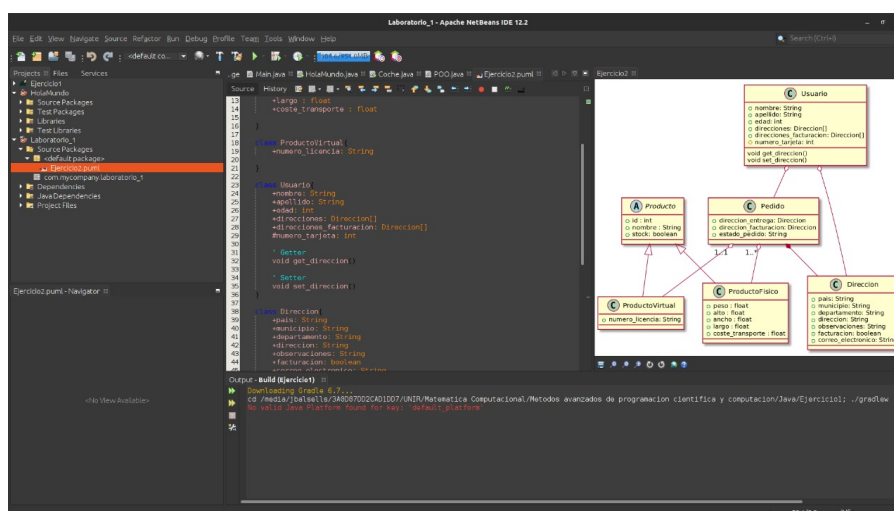


Figura 6: PlantUML generando diagrama de clases en NetBeans.