Universidad Tecnológica de Pereira

Mision TIC 2022

El futuro digital es de todos

MinTIC

Unidad 2

04 POO en Java y Pilares de POO

Hechos QUE CONECTAN

Retomando el Ejercicio 1

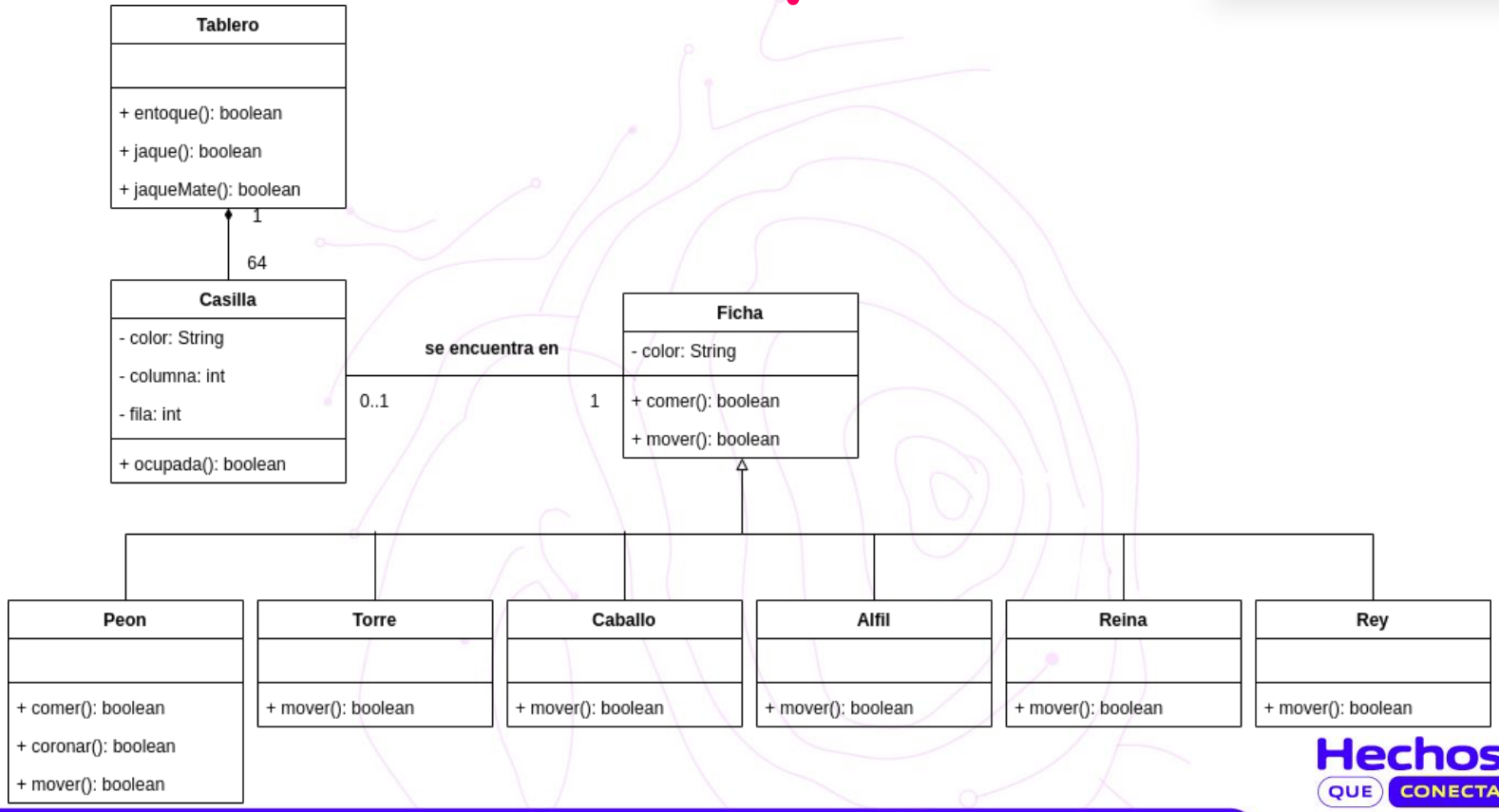
Realizar el diagrama de clases para el juego del Ajedrez

El ajedrez es un juego entre dos contrincantes en el que cada uno dispone al inicio de 16 piezas móviles que se colocan sobre un tablero, dividido en 64 casillas o escaques.

Se juega sobre un tablero cuadriculado de 8×8 casillas (llamadas escaques),2 alternadas en colores blanco y negro, que constituyen las 64 posibles posiciones de las piezas para el desarrollo del juego. Al principio del juego cada jugador tiene dieciséis piezas: un rey, una dama, dos alfiles, dos caballos, dos torres y ocho peones. Se trata de un juego de estrategia en el que el objetivo es «derrocar» al rey del oponente. Esto se hace amenazando la casilla que ocupa el rey con alguna de las piezas propias sin que el otro jugador pueda proteger a su rey interponiendo una pieza entre su rey y la pieza que lo amenaza, mover su rey a un escaque libre o capturar a la pieza que lo está amenazando, lo que trae como resultado el jaque mate y el fin de la partida.

Leer resto de información: https://es.wikipedia.org/wiki/Ajedrez

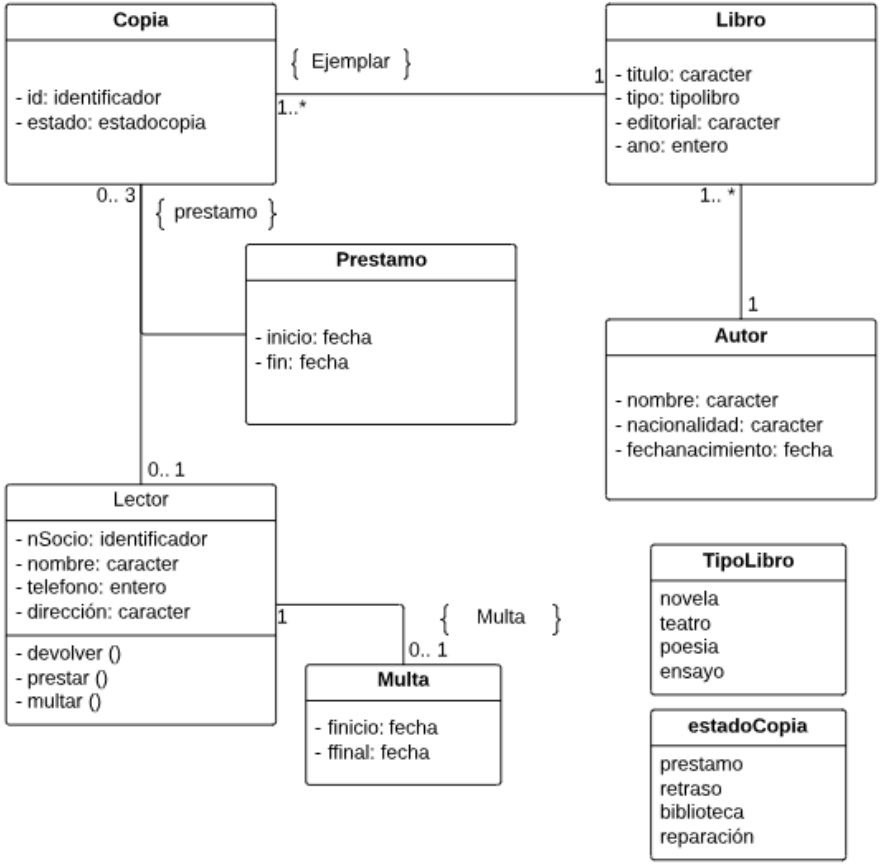
Solución Ajedrez



Ejercicio 2

- Una biblioteca tiene copias de libros. Estos últimos se caracterizan por su nombre, tipo (novela, teatro, poesía, ensayo), editorial, año y autor.
- Los autores se caracterizan por su nombre, nacionalidad y fecha de nacimiento.
- Cada copia tiene un identificador, y puede estar en la biblioteca, prestada, con retraso o en reparación.
- Los lectores pueden tener un máximo de 3 libros en préstamo.
- Cada libro se presta un máximo de 30 días, por cada día de retraso, se impone una "multa" de dos días sin posibilidad de coger un nuevo libro.
- Realiza un diagrama de clases para realizar el préstamo y devolución de libros.

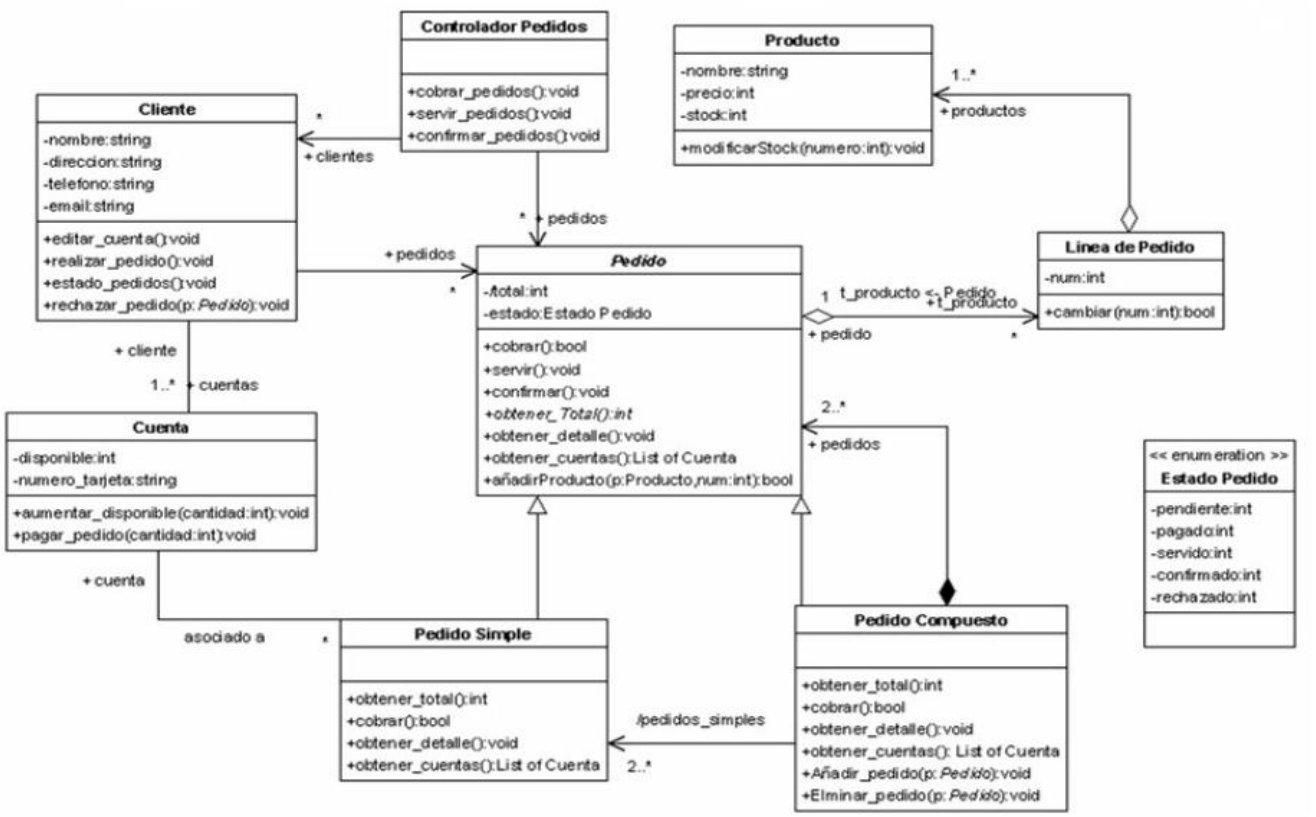
Solución Biblioteca



Ejercicio 3

- Manejar clientes (se guarda su nombre, dirección, teléfono y e-mail). Un cliente puede tener una o varias cuentas para el pago de los pedidos. Cada cuenta está asociada a una tarjeta de crédito, y tiene una cierta cantidad disponible de dinero, que el cliente debe aumentar periódicamente para poder realizar nuevos pedidos.
- Un cliente puede empezar a realizar un pedido sólo si tiene alguna cuenta con dinero disponible. Al realizar un pedido, un cliente puede agruparlos en pedidos simples o compuestos. Los pedidos simples están asociados a una sola cuenta de pago y (por restricciones en la distribución) contienen un máximo de 20 unidades del mismo o distinto tipo de producto.
- Existe una clase (de la cual debe haber una única instancia en la aplicación) responsable del cobro, orden de distribución y confirmación de los pedidos. El cobro de los pedidos se hace una vez al día, y el proceso consiste en comprobar todos los pedidos pendientes de cobro, y cobrarlos de la cuenta de pago correspondiente.

Solución pedidos



El futuro digital es de todos

MinTIC

Universidad Tecnológica de Pereira

Mision TIC 2022

Programación Orientada a Objetos en Java

Hechos QUE CONECTAN

Clases en Java

```
<modificador> class <Nombre Clase> {
   <Atributos>
   <Constructores>
   <Métodos>
}
```

Clases en Java

Modificador de clase: Determina la visibilidad o uso de la clase.

- public: visible para todas las clases
- final: Última definición de clase. No podrá tener subclases.
- <sin modificador>: Solo visible por clases del paquete.

```
<modificador> class <Nombre Clase> {
    <Atributos>
    <Constructores>
    <Métodos>
}

public final class MiPrimerClase {

}
```

Clases en Java

Nombre de clase

- Piensa en un nombre apropiado para su clase. *No se limite* a llamar su clase XYZ, sigla o cualquier nombre al azar.
- Los nombres de clase deben *comenzar con una letra mayúscula.*
- El nombre del archivo de la clase debe tener el mismo nombre de la clase pública.

```
<modificador> class <Nombre Clase> {
    <Atributos>
    <Constructores>
    <Métodos>
}

public final class MiPrimerClase {

}
```

Clases en Java

Atributos

```
<modificador> <tipo> <nombre> [= <valor x defecto>];
```

Modificadores:

- **public**: Pueden ser modificados por un objeto externo.
- **private**: Sólo son accesibles dentro de la clase. **(Recomendado)**
- **protected**: Público para paquete e hijas, privado para los demás.
- **final**: Solo puede tener 1 valor (constante).
- **static**: Compartido por todos los objetos. Si un objeto cambia el valor, todos veran el cambio. También es llamada ***variable de clase***.
- **‹sin modificador›**: Solo visible por clases del paquete.

```
<modificador> class <Nombre Clase> {
  <Atributos>
  <Constructores>
  <Métodos>
}
```

```
public final class MiPrimerClase {
  private static final Double PI = 3.1416;
  private Integer contador = 0;
}
```

Clases en Java

Atributos

- Declarar todas las variables de instancia en la parte superior de la declaración de clase.
- Declare una variable por cada línea.
- Las variables de instancia, igual que cualquier otra variable, deben *comenzar con letra minúscula.*
- Las constantes (*final*) deben ser nombradas en *mayúscula sostenida*.
- Utilice un tipo de dato adecuado para cada variable. Se recomienda el uso de *clases wrapper* en lugar de tipos básicos.
- Declarar las variables de instancia como *private* de modo que sólo los métodos de clase puedan acceder a ellos directamente.

```
<modificador> class <Nombre Clase> {
   <Atributos>
   <Constructores>
   <Métodos>
}


public final class MiPrimerClase {
   private static final double PI = 3.1416;
   private Integer contador = 0;
}
```

Clases en Java

Métodos

```
<modificador> <tipo retorno> <nombre>
(<parametro>*) {
    <sentencia>*
}
```

Modificadores:

- public:Pueden ser llamados por un objeto externo.
- private: Sólo son accesibles dentro de la clase.
  protected: Público para paquete e hijas, privado para los demás.
- final: Última definición. No podrá ser reescrita (overwrite)
- static: Pueden ser llamados sin necesidad de crear instancia de la clase. Solo puede usar atributos static.
- <sin modificador>: Solo visible por clases del paquete.

```
<modificador> class <Nombre Clase> {
  <Atributos>
  <Constructores>
  <Métodos>
}

public final class MiPrimerClase {
  private static final double PI = 3.1416;
  private Integer contador = 0;

  public void incrementarContador(Integer cantidad) {
    contador += cantidad;
  }

}
```

Clases en Java

Métodos

```
<modificador> <tipo retorno> <nombre> (<parametro>*) {
  <sentencia>*
}
```

Tipo de retorno: puede ser cualquier tipo de datos, incluyendo ***void*** en caso de no devolver un valor.

Nombre de método:

- Los nombres reflejan los comportamientos, por lo tanto deben empezar con una acción (**verbo**).
- Los nombres de métodos deben ***comenzar con letra minúscula***.

Parámetros: es una lista de declaraciones de variables (separados por coma) que determina la información que necesita para su ejecución.

```
<modificador> class <Nombre Clase> {
  <Atributos>
  <Constructores>
  <Métodos>
}

public final class MiPrimerClase {
  private static final double PI = 3.1416;
  private Integer contador = 0;

  public void incrementarContador(Integer cantidad) {
    contador += cantidad;
  }

}
```

Clases en Java

Métodos de acceso y mutadores (setter y getter)

- Métodos de acceso
  - Se utiliza para leer los valores de nuestras variables de clase.
  - Normalmente se llama: set<NombreVariable>
  - También devuelve un valor de tipo de dato de la variable.
- Métodos mutadores
  - Se utiliza para escribir o cambiar los valores de nuestras variables de clase.
  - Normalmente se llama: get<NombreVariable>

```
<modificador> class <Nombre Clase> {
  <Atributos>
  <Constructores>
  <Métodos>
}
```

```
public final class MiPrimerClase {
  private Integer contador = 0;

  public Integer getContador() {
    return contador;
  }

  public void setContador(Integer valor) {
    contador = valor;
  }

}
```

Hechos QUE CONECTAN

Clases en Java

Constructores

```
<modificador> <nombre clase> (<parametro>*) {
  <sentencia>*
}
```

- Son unos ***métodos especiales*** que permiten la creación de objetos de la clase.
- El nombre de los constructores deben ser igual al nombre de la clase y no retornan ningún valor.
- Solo se puede llamar utilizando el operador **new** duranta la instanciación de la clase.
- El constructor por defecto es aquel que se define sin parámetros. Si no se define ningún constructor en la clase, el lenguaje crea uno por defecto.

```
<modificador> class <Nombre Clase> {
  <Atributos>
  <Constructores>
  <Métodos>
}
```

```
public final class MiPrimerClase {

  public MiPrimerClase(){
  }

  public MiPrimerClase(Integer cont) {
    contador = cont;
  }

}
```

Clases en Java

Palabra reservada **this**

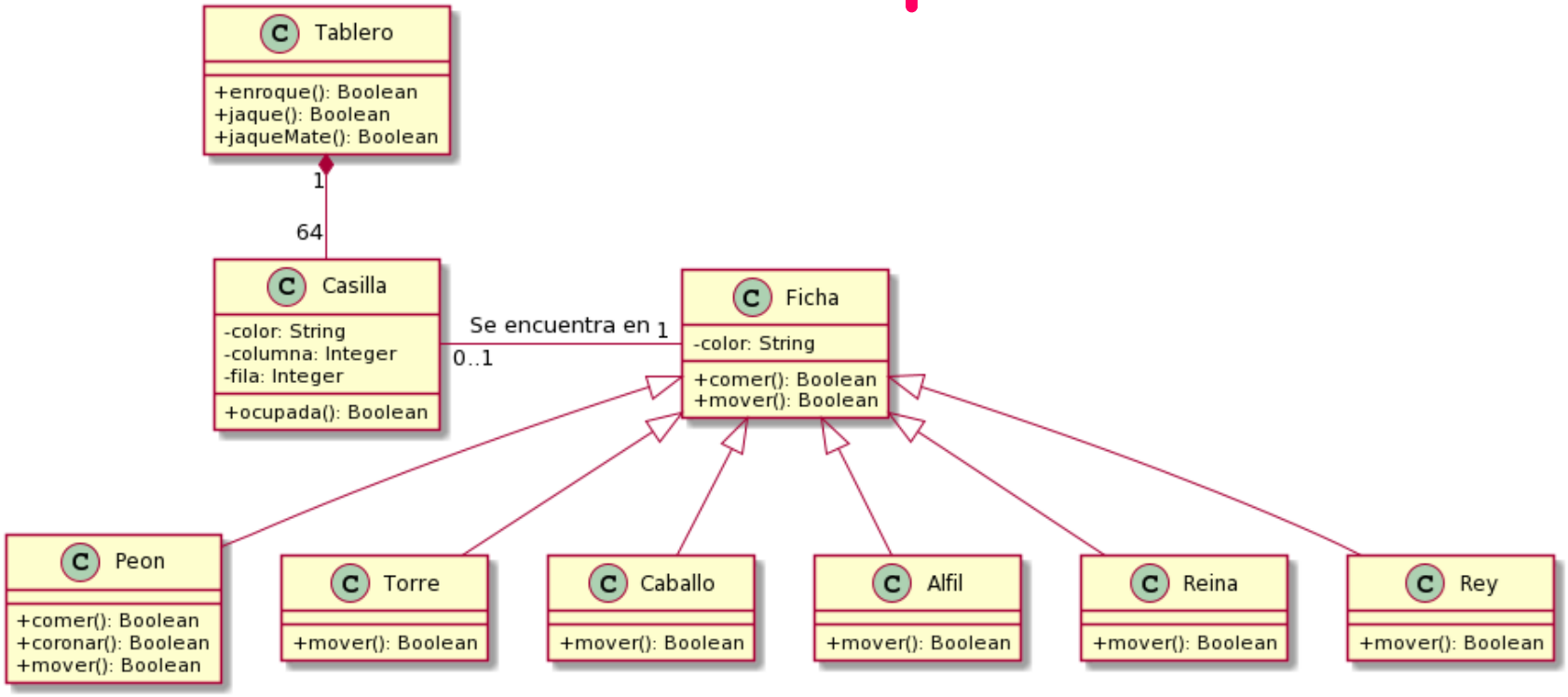
- Desambiguar los atributos locales de variables locales.
- Para referirse al objeto que invocó el método no estático.
- para referirse a otros constructores de la clase.

```
public class EjemploThis {

  private Integer dato;

  public EjemploThis() {
    this(100);
  }

  public EjemploThis(Integer dato) {
    this.setDato(dato);
  }

  public void setDato(Integer dato) {
    this.dato = dato;
  }

  public Integer getDato() {
    return this.dato;
  }

}
```

Vamos al código

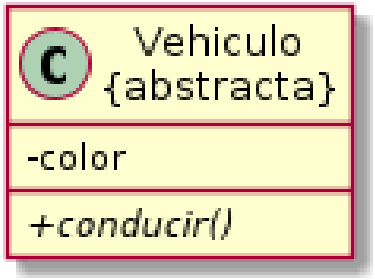
- Crear un proyecto Maven para los ejercicios de la clase 4
  - Ctrl + Shift + P
  - *› Java: Create Java Project...*
  - *Maven*
  - *maven-archetype-quickstart, ‹versión más reciente›*
  - group Id: co.edu.utp.misiontic2022.c2
  - artefact Id: clase04
- Crear las clases necesarias para modelar el ejercicio del Ajedrez
- La clase con la función main() solo se usa para hacer el llamado a las funciones que se quieren probar.

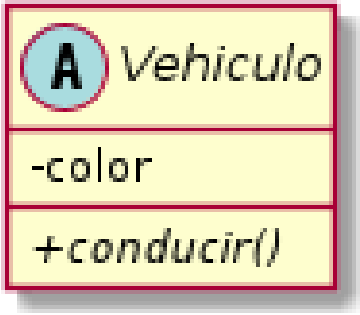
Solución Ajedrez



Clase Abstracta

- Una clase abstracta es aquella que no tiene objetos.
- Sólo se utiliza para heredar a partir de ella, es decir, en una clase abstracta se describen los atributos y las operaciones comunes para otras clases.
- Se especifica de forma explícita poniendo {abstracta} dentro del compartimento del nombre de la clase y debajo de su nombre, o decorando el nombre en *cursiva*. Si tiene algún método abstracto, también va en *cursiva*.
- Una clase abstracta tiene operaciones abstractas. Una operación abstracta es aquella que no tiene método de implementación en la superclase donde está definida.





Clases abstractas en Java

- Las clases abstractas son un tipo de clase especial que define que la clase no está completamente definida y por lo tanto no puede ser instanciada.
- Debe ser marcada con el modificador abstract y esto permite que dentro de la definición de la clase puedan existir métodos abstractos (que no esté definida su implementación).

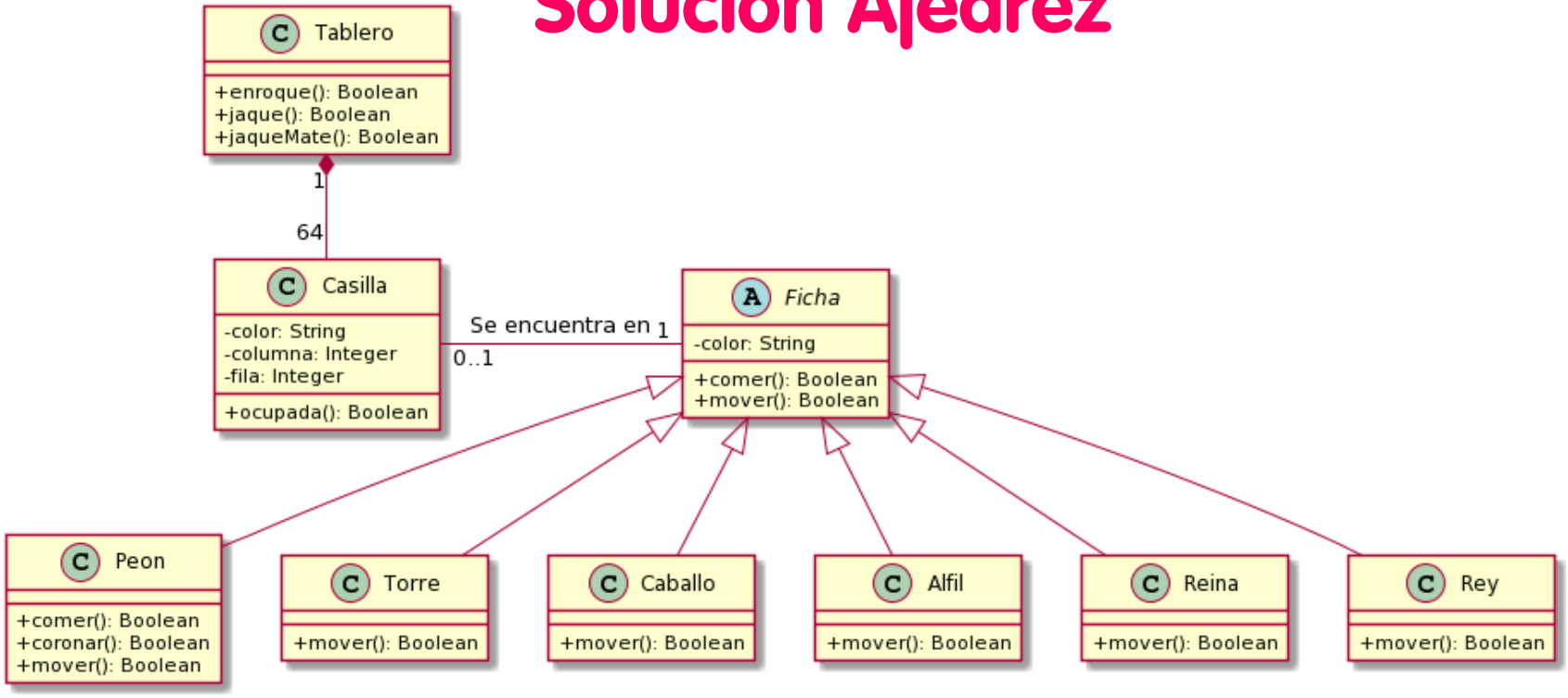
```
<modificador> abstract class <Nombre Clase> {
  <Atributos>
  <Constructores>
  <Métodos>
}
```

```
public abstract class ClaseAbstracta {

  public abstract implementar();

}
```

Vamos al código

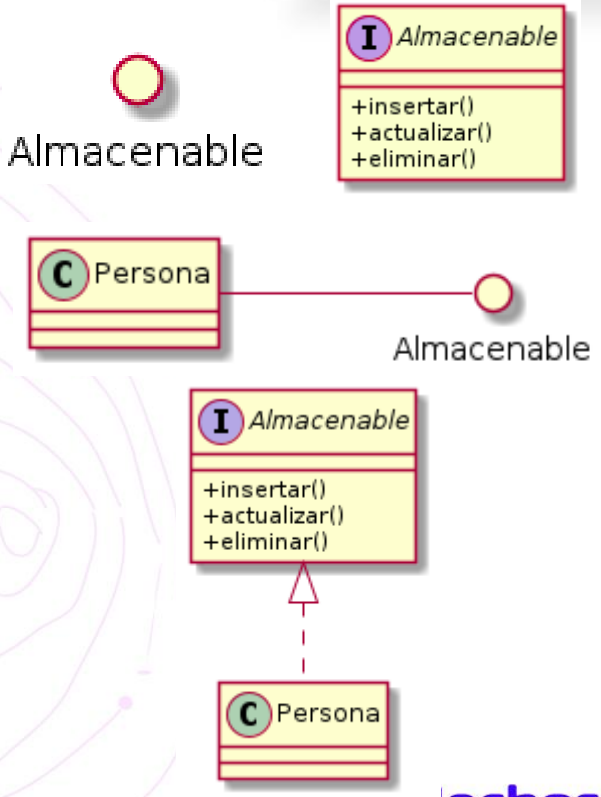
- Modifica el proyecto de la clase04 ya que la clase Ficha es abstracta y hacemos al método mover() abstracto también.

Solución Ajedrez



Interface

- Describe sólo operaciones (métodos) abstractas, que especifican un comportamiento que una clase puede elegir soportar implementando la interfaz. No incluyen atributos ni ninguna implementación.
- Al igual que las clases, las interfaces pueden participar en relaciones de asociación, dependencia y generalización.
- Las interfaces se representan gráficamente por medio de un círculo pequeño con un nombre situado debajo de él.



Interfaces en Java

```
<modificador> interface <Nombre Clase> {
   <Métodos>
}
```

Interfaces en Java

Interfaces

Permite que las clases, independiente de su ubicación en la jerarquía de clases, implementar comportamientos comunes.

Modificador de clase: Determina la visibilidad o uso de la clase.

- public: visible para todas las clases
- <sin modificador>: Solo visible por clases del paquete.

```
<modificador> interface <Nombre Clase> {
   <Métodos>
}
```

```
public interface Impresora {
   public void imprimir(String texto);
   public int getVelocidad();
}
```

Interfaces en Java

Interfaces

- TODOS los métodos de las interface no tienen cuerpo.
- Toda clase que cumpla con el contrato definido por la interface, debe dar la implementación o convertirse en una clase abstracta.
- Desde Java 8, existe el concepto de default method donde podemos dar una implementación por defecto a un método definido en la interface.

```
<modificador> interface <Nombre Clase> {
  <Métodos>
}

public interface Impresora {

  public void imprimir(String texto);

  public int getVelocidad();

  public default boolean esMasRapida(Impresora i) {
    if (i.getVelocidad() > this.getVelocidad()) {
      return false;
    } else {
      return true;
    }
  }

}
```

Interfaces en Java

```
<modificador> interface <Nombre Clase> {
  <Métodos>
}

public interface Impresora {

  public void imprimir(String texto);

  public int getVelocidad();

  public default boolean esMasRapida(Impresora i) {
    if (i.getVelocidad() > this.getVelocidad()) {
      return false;
    } else {
      return true;
    }
  }

}
```

```
public class ImpresoraTinta implements Impresora {
  private int velocidad;

  public ImpresoraTinta(int velocidad) {
    this.velocidad = velocidad;
  }

  @Override
  public int getVelocidad() {
    return velocidad;
  }

  @Override
  public void imprimir(String texto) {
    System.out.println("la impresora de tinta imprime:"
        + texto);
  }

}
```

¿Cómo decidir si conviene una clase abstracta o una interface?

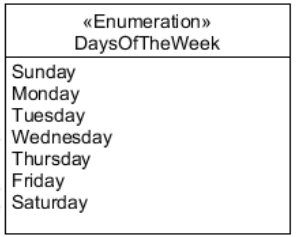
Debe usarse una clase abstracta cuando se está modelando una jerarquía de clases y una interfaz cuando se pretende homogeneizar nombre entre objetos que no están emparentados


#1
Dilemas de un desarrollador
CLASE ABSTRACTA
VS
INTERFAZ

Enumerados en Java

```
<modificador> enum <Nombre Clase> {
   <Valores>
}
```

Enumerados en Java



Java Enum

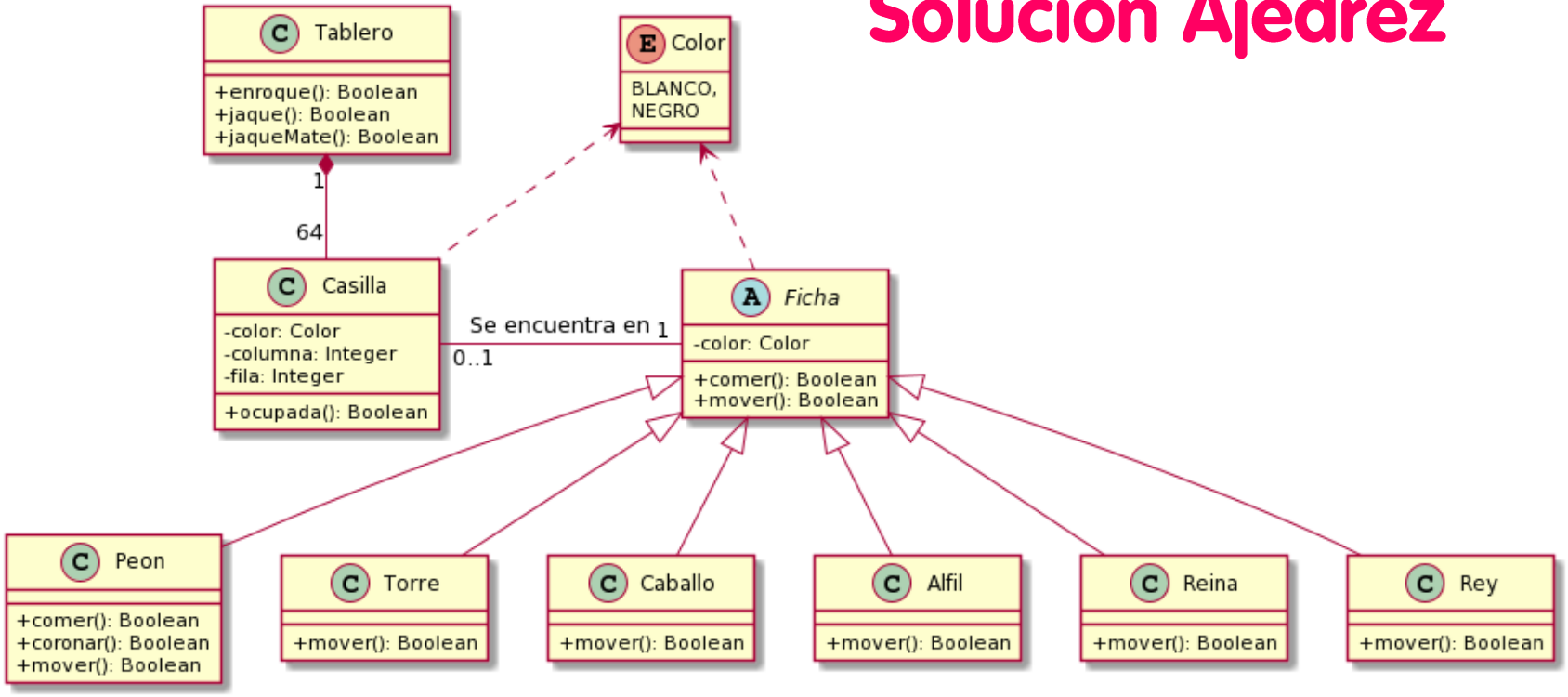
- Son tipos de datos especiales que le permiten a una variable comportarse como un conjunto de constantes predefinidas.
- También se usan para restringir el contenido de una variable.
- El valor de la variable tiene que ser uno del conjunto de valores que fue previamente definida para ella.

```
<modificador> enum <Nombre Clase> {

   <Métodos>

}

public enum DiaSemana {

      LUNES,
      MARTES,
      MIERCOLES,
      JUEVES,
      VIERNES,
      SABADO,
      DOMINGO

}
```

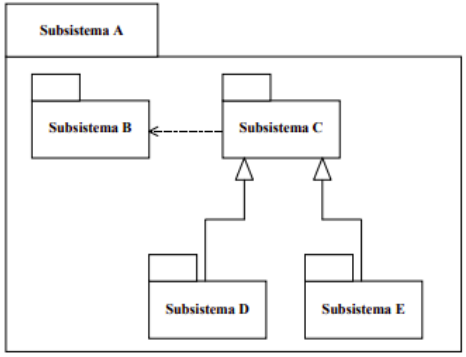
Vamos al código

- Modifica el proyecto de la clase04 ya que podemos crear una clase Enumeradora para limitar el color.

Solución Ajedrez



Paquetes



- Un paquete es un mecanismo para agrupar clases u otros elementos de otro tipo de diagramas en modelos más grandes, donde dichos elementos pueden ser enlazados.
- Esta característica proporciona un mecanismo conveniente para la gestión de una gran cantidad de clases e interfaces y evitar posible conflicto de nombres.
- Los paquetes, desde el punto de vista del sistema operativo, son carpetas que contienen los archivos que contienen las clases e interfaces.
- Muchas metodologías de POO usan el término subsistema para describir un paquete.

Importando clases y paquetes

- Para poder utilizar las clases fuera del paquete que se está trabajando, se tiene que importar las clases o el paquete de dichas clases.
- Por defecto, todos los programas de Java importan el paquete `java.lang.*`, es por esto que se pueden utilizar las clases String y System dentro de un programa sin necesidad de importar este paquete.
- La sintaxis de importación de una clase es:

```
import <nombre paquete>.<nombre clase>;
```

- La sintaxis para importar todas las clases de un paquete es:

```
import <nombre paquete>;
```

Creando paquetes

- Para crear nuestro propio paquete, debemos crear la carpeta en la estructura de directorios.
- las clases que se encuentran en un paquete, deben tener una instrucción que diga a qué paquete perteneces:

```
package <nombre paquete>;
```

- Los paquetes también se pueden anidar. En este caso, el intérprete de java espera que la estructura de directorios que contiene las clases, coincida con la jerarquía de paquetes.
- El nombre del paquete, cuando se encuentra anidado, se pondrá un punto "." en cada nivel de la estructura.

Ej: `javax.swing, java.utils, oracle.jdbc.driver`

Objetos en Java

- Para crear objetos en Java, las creamos como variables de tipo de datos Clase, usando la palabra reservada new y llamando el constructor deseado.

```
String cadena = "Hola Mundo";
MiPrimerPrograma primer = new MiPrimerPrograma(10);
```

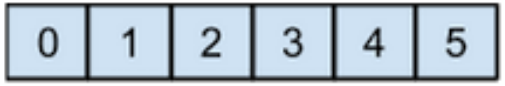
- En caso de Herencia de clases o implementación de interfaces, se pueden crear objetos usando variables de tipo SuperClase o Interface

```
ClaseAbstracta clase = new ClaseConcreta();
Impresora impresoraTinta = new ImpresoraTinta();
```

- Desde Java 10 podemos crear variables de objetos sin definir la clase. El lenguaje infiere la clase a partir del constructor.

```
var segundo = new MiPrimerPrograma(15);
```

Arrays en Java



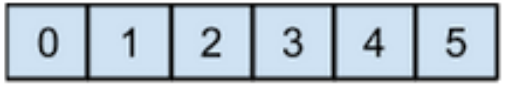
- Una array o arreglo es una colección de variables del mismo tipo, a la que se hace referencia por un nombre común.
- En Java, los arrays son objetos que pueden tener una o más dimensiones, aunque el array unidimensional es el más común.
- Para declarar un array en Java

```
tipo[] nombre_array;

int[] intArray;
```

- Para crear un array en Java, debemos usar a palabra new y el tamaño del array.

```
nombre_array = new tipo[tamaño];

intArray = new int[5];
```

Arrays en Java



- También es posible inicializar el array con una lista de valores

```
int[] intArray = { 1,2,3,4,5,6,7,8,9,10 };
```

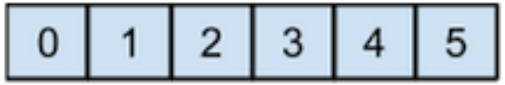
- Para asignar un valor a un array, lo asignamos con el nombre de la variable y el índice entre corchetes. Los índices inician en 0 y termina en tamaño - 1.

```
var intArray = new int[2];
intArray[0]  = 1;
intArray[1]  = 2;
```

- Para obtener el tamaño del array, usamos el atributo length. Para obtener el valor usamos el nombre del array y su índice.

```
for (int i = 0; i < intArray.length; i++) {
     System.out.println("Elemento en el índice " + i + " : "+ intArray[i]);
}
```

Arrays en Java



- Una matriz bidimensional puede tener varias filas, y en cada fila no tiene por qué haber el mismo número de elementos o columnas.
- Podemos verla como un array que en su contenido tiene otro array que no es necesario que tengan el mismo tamaño.

```
int[][] matrizCuadrada = new int[3][3];
int[][] matrizIrregular = new int[3];
matrizIrregular[0] = new int[3];
matrizIrregular[1] = new int[20];
matrizIrregular[2] = new int[1];
```

- Para obtener el valor usamos el nombre del array y sus índices.

```
matrizCuadrada[2][1]
matrizIrregular[1][15]
```

El futuro digital es de todos

MinTIC

Universidad Tecnológica de Pereira

Mision TIC 2022

Pilares de la Programación Orientada a Objetos

Hechos QUE CONECTAN

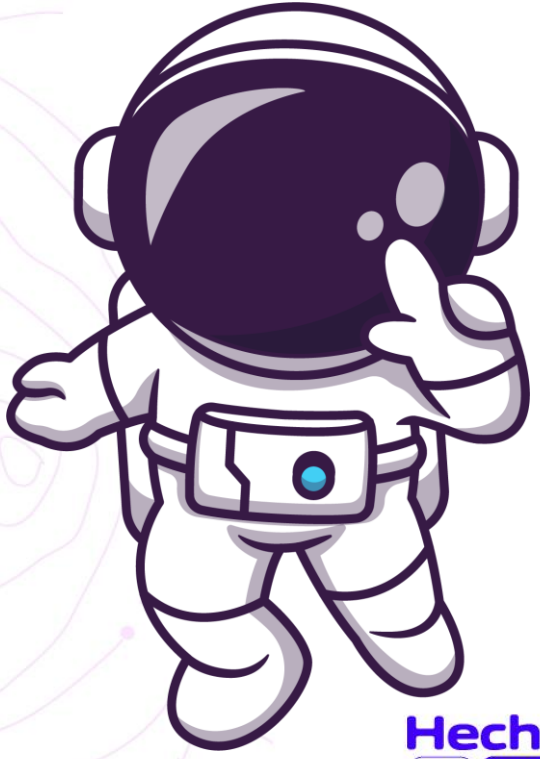

PROGRAMACIÓN ORIENTADA A OBJETOS

Pilares de P.O.O.

La P.O.O. tiene varios pilares para asegurar la simplicidad de código y su reutilización. Estos son:

- Abstracción
- Encapsulamiento
- Herencia
- Polimorfismo

Las dos primeras están relacionadas con la búsqueda de simplificar el código y las dos siguientes con la reutilización.



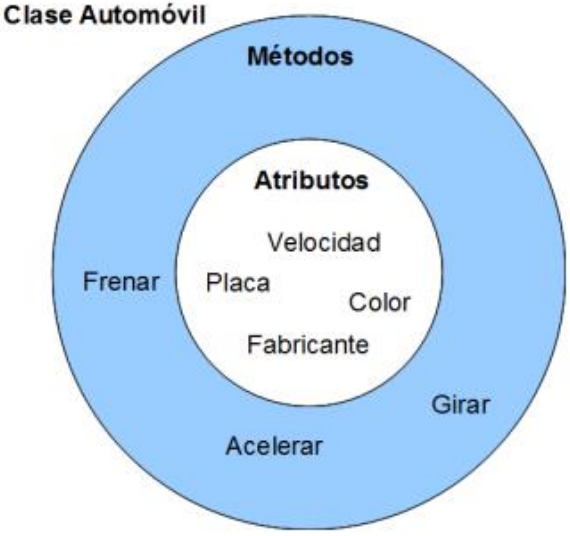
Abstracción

- La abstracción consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan.
- El término se refiere al énfasis en el ¿que hace? más que en el ¿cómo lo hace? (característica de caja negra).
- Le permite identificar las características y comportamientos de un objeto y con los cuales se construirá la clase. Esto quiere decir que a través de este pilar o fundamento es posible reconocer los atributos y métodos de un objeto.

Automóvil
- placa: String - color: String - fabricante: String - velocidad: Double
+ acelerar() + girar() + frenar()

Encapsulamiento

- Permite el ocultamiento de la complejidad del código, pertenece a la parte privada de la clase y que no puede ser vista desde ningún otro programa.
- El encapsulamiento está relacionado con el acceso a un código desde el código de otra clase; sin embargo en términos generales, esta representación gráfica es conveniente para comprender el concepto de encapsulamiento.
- Solo se debe visibilizar desde fuera del objeto solo lo que necesita ser visible.

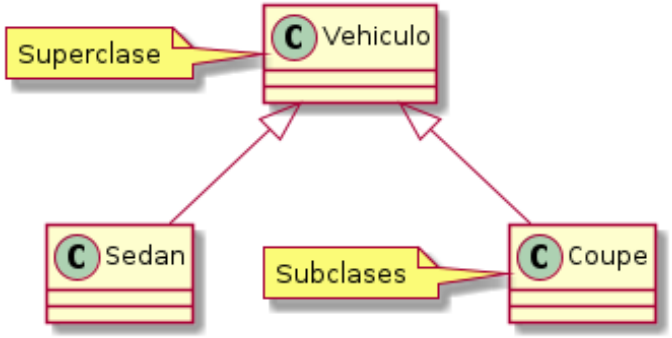


Encapsulamiento en Java

```java
public class Persona {

  private String nombre;
  private String apellido;
  private Integer codigo;

  public Persona() {
  }

  public Persona(String nombre, String apellido) {
   this.nombre = nombre;
   this.apellido = apellido;
  }

  public void setNombre(String nombre) {
    this.nombre = nombre;
  }

  public String getNombre() {
    return this.nombre;
  }

  ...
```

```java
  ...

  public void setApellido(String apellido) {
    this.apellido = apellido;
  }

  public String getApellido() {
    return this.apellido;
  }

  public void setCodigo(Integer codigo) {
    this.codigo = codigo;
  }

  public Integer getCodigo() {
    return this.codigo;
  }

}
```

Herencia



- En Java, todas las clases, incluyendo las clases que componen el API de Java, son una subclase de la superclase Object.
- Superclase: Clase que se encuentra sobre una clase específica en la jerarquía de clases.
- Subclase: cualquier clase debajo de una clase específica en la jerarquía de clases.
- Una vez que un comportamiento (método) se define en una superclase, el comportamiento se hereda automáticamente a todas las subclases.
- Por lo tanto, usted puede codificar un método de una sola vez y que puede ser utilizado por todas las subclases.
- La subclase sólo tiene que implementar los métodos que difieren entre ella y el padre.

Herencia en Java

```
public class Persona {

  protected String nombre;
  protected String direccion;

  public Persona(){
    System.out.println("Constructor
Persona");
    nombre = "";
    direccion = "";
  }

  . . .

}
```

```
public class Estudiante extends Persona {

  public Estudiante() {
    System.out.println("Constructor Estudiante");
  }
  . . .

}

public class Profesor extends Persona {

  public Profesor() {
    System.out.println("Constructor Profesor");
  }
  . . .

}
```

Herencia en Java

Palabra reservada **super**

- ***Sirve para referirse a atributos o métodos de la superclase.***
- Para llamar a constructores de la superclase

```
class SuperClase {

 protected Integer entero;

 public void mostrarEntero() {
  System.out.println("entero = " + entero);
 }

}

public class SubClase extends SuperClase {

 private Integer entero;

 public void mostrarEntero() {
  System.out.println("valor padre = " + super.entero);
  System.out.println("valor = " + entero);
 }

 public void mostrarSuperEntero() {
  super.mostrarEntero();
 }

}
```

Herencia en Java

Palabra reservada **super**

- Sirve para referirse a atributos o métodos de la superclase.
- *Para llamar a constructores de la superclase*

```
public abstract class Persona {

 private String nombre;
 private String apellido;

 public Persona(String nombre, String apellido) {
  this.nombre = nombre;
  this.apellido = apellido;
 }

}

public class Estudiante extends Persona {

 private Integer codigo;

 public Estudiante(String nombre, String apellido,
                   Integer codigo) {
  super(nombre, apellido);
  this.codigo = codigo;
 }

}
```

Polimorfismo

1. Polimorfismo es la capacidad de un objeto de adquirir varias formas. El uso más común se da cuando se utiliza la referencia de una clase padre, para referirse al objeto de la clase hijo.
2. A través de esta característica es posible definir varios métodos o comportamientos de un objeto bajo un mismo nombre, de forma tal que es posible modificar los parámetros del método, o reescribir su funcionamiento, o incrementar más funcionalidades a un método. También suele definirse como "sobrecarga de parámetros".

Polimorfismo en Java (1)

```
class Animal {
  public void makeSound() {
    System.out.println("Grr...");
  }

}

class Cat extends Animal {
  @Override
  public void makeSound() {
    System.out.println("Meow");
  }

}

class Dog extends Animal {
  @Override
  public void makeSound() {
    System.out.println("Woof");
  }

}
```

```
public class Main {

  public static void main(String[] args) {
    Animal a = new Dog();
    Animal b = new Cat();

    a.makeSound(); // "Woof"

    b.makeSound(); // "Meow"

  }

}
```

Polimorfismo en Java (2)

```
class Sobrecarga {

  void demo(Integer a) {
    System.out.println ("a: " + a);
  }

  void demo(Integer a, Integer b) {
    System.out.println ("a and b: "+ a +","+ b);
  }

  Double demo(Double a) {
    System.out.println("Double a: " + a);
    return a*a;
  }

}
```

```
public class Main {

  public static void main(String[] args) {
    var objeto = new Sobrecarga();

    Double resultado;

    objeto.demo(10); // "a: 10"

    objeto.demo(10, 20); // "a and b: 10,20"

    result = objeto.demo(5.5); // "Double a: 5.5"

    System.out.println("O/P : " + resultado);

  }

}
```

Para la próxima sesión...

- Leer material llamado Convenciones de código Java