Apunte TDA, Clases y Objetos

INTRODUCCION

> Definición: Un tipo de dato abstracto (TDA) o Tipo abstracto de datos (TAD) es un conjunto de datos que conforman un concepto y una colección de operaciones definidas sobre el mismo.

El concepto de Abstracción de Datos se refiere a reconocer los elementos involucrados en los programas y todas las operaciones asociadas a ellos, las cuales abarcan su creación, acceso y manipulación. La programación orientada a objetos tiene sus bases en el concepto de abstracción de datos.

El estado y la funcionalidad de un objeto están definidos en una clase, por lo tanto un objeto es una instancia de la Clase en la cual se define su estructura y comportamiento. Cada objeto puede ser accedido o modificado a través de las operaciones definidas por su clase. Además de ejecutar acciones generales, podemos hacer lo siguiente:

- crear nuevos objetos a partir de valores iniciales.
- copiar objetos (clonar)
- modificar algo de los objetos
- comprobar la igualdad.
- Obtener información sobre el objeto .

El agrupamiento de datos y las operaciones que se aplican sobre ellos, forman un agregado, el ocultamiento de los detalles del agregado, se conoce como encapsulación.

Los programadores deben ser capaces de reutilizar objetos en lugar de implementarlos repetidamente. Cuando tenemos una implementación del objeto exacto que necesitamos usar, la reutilización es simple. El problema esta en utilizar un objeto disponible cuando no es exactamente el que se necesita, sino solamente uno muy parecido, en esos el uso de herencia permite extender la funcionalidad de una clase.

Un objeto es una instancia de una clase, donde los componentes pueden ser funciones o datos, denominados respectivamente métodos y atributos, y, se puede restringir la visibilidad de dichos componentes.

Las operaciones manipulan el comportamiento de una clase, y se acceda a ellas mediante el operador punto, al igual que a los atributos de un objeto.

Los objetos tienen un estado interno que se puede manipular aplicando el operador punto para seleccionar un método.

1. ABSTRACCION DE DATOS

Un sistema de software es usualmente complejo, por lo tanto para simplificar su complejidad, muchas veces es necesario descomponerlo en partes manejables, donde cada parte pueda ser representada de alguna manera, abstrayendo los aspectos esenciales del sistema. Para describir estos modelos se utiliza el lenguaje UML el cual se basa en objetos. Es seleccionar partes de un todo complejo, ignorando el resto, esto es filtrar aspectos relevantes y obtener soluciones más generales, se considera la resolución de un problema sin tener en cuenta los detalles por debajo de cierto nivel.

La abstracción de datos es considerar (abstraer) sólo las cosas importantes, dejando de lado los detalles. Por ejemplo, si estamos hablando de un Alumno, no necesitamos explicar nada, todos sabemos que van a tener datos personales, tipo nombre y apellido, dni, también van a contar de un legajo, y algunos datos mas. Podemos integrar todas las características en un sólo concepto concepto. Al programa le basta con utilizar el término “Alumno”, el cual será un Tipo de Dato Abstracto formado por una estructura de datos (de uno o más) tipos de datos en conjunto con un grupo de operaciones.

Un modelo orientado a objetos maneja los conceptos de abstracción, encapsulación, modularidad y jerarquía

- Abstracción: es encontrar las características esenciales de algo
- Encapsulación: Es ocultar la implementación de la abstracción
- Modularidad: Es poder subdividir la aplicación en módulos cohesivos y débilmente acoplados, que realicen cada módulo una única funcionalidad
- Jerarquía: Poder organizar de las abstracciones

Un TDA ‘Abstract Data Type’ o ADT en ingles, define un concepto nuevo que puede manejarse con independencia de la estructura de datos para representarlo. Es una generalización de los tipos de datos básicos y de las operaciones primitivas. Los atributos internos que representan el estado de un objeto no deberían ser manipulados directamente por el usuario de la clase, sino solamente a través de los métodos. Entonces, un TDA es un grupo de datos que cumplen cierta condicion + un conjunto de operaciones que representan su comportamiento. *TDA = Representacion (estructuras de datos) + Operaciones (metodos)*

Cuando se está diseñando una clase, es importante ocultar los detalles internos de cómo está compuesto (de los tipos de datos que tienen cada una de los atributos), por ejemplo en un Alumno, el dni se puede tomar como un número entero, como un número entero largo o como un String, ese tipo de detalles entra en la estructura interna del TDA.

Las operaciones de la interfaz son el único mecanismo de acceso a la estructura de datos del TDA y se implementan mediante métodos.

Los datos del TDA pueden ser accedidos mediante operaciones definidas llamada interfaz. Algunas de las funcionalidades de los métodos describen sus características, otros la inicialización de una instancia, otros cómo se realiza las comprobaciones de igualdad o cómo se lleva a cabo la salida.

El ciclo de vida de un TDA es: definición, especificación, implementación y uso.

En la ***definición*** se define el tipo de la instancia, si es mutable (puede ser modificado después de creado) o inmutable (no se puede modificar luego de la creación), sobre qué dominio está y que posible valores tomará el tipo de datos.
En la ***especificación*** se determinan los datos o valores conforman el tipo de dato, también se especifican las operaciones sintácticamente (reglas para hacer referencia a una operación) y semánticamente (significado o consecuencia de cada operación).
La ***implementación;*** es donde se escribe el código que implemente el tipo de dato en un lenguaje dado (Java, Pascal, etc), para luego probar que funciona bien, para luego probarlo ***utilizándolo*** en cualquier programas,

Por ejemplo, para definir un TDA, es necesario responder lo siguiente:
¿Qué partes lo componen? Ejemplo, si estoy pensando en un TDA fecha seguramente lo compondrán tres partes ((día, mes, año). Luego se debe definir los tipos de valores que puede tomar, ejemplo, son valores enteros, el mes puede variar entre 1 y 12,

Otro ejemplo puede ser el TDA racional,
En la especificación se mencioan qué operaciones pueden existir, esto es:

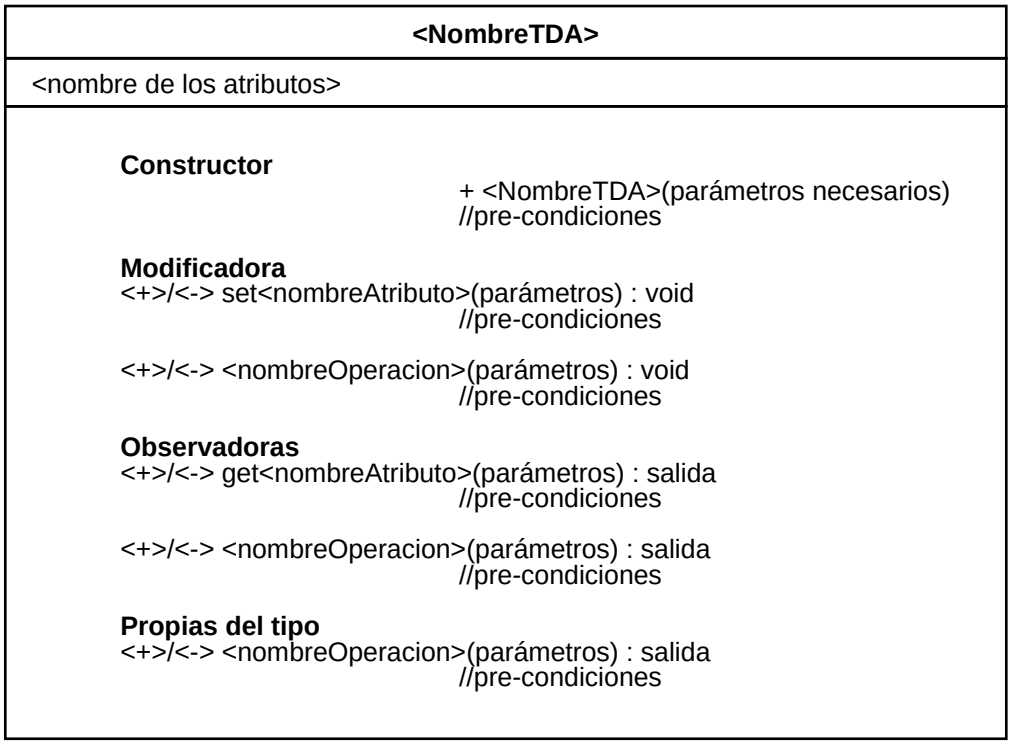
- ¿Cómo se puede crear un racional?
- ¿Cómo obtener el valor actual?
- ¿Cómo modificar su valor actual?
- ¿Qué operaciones son típicas sobre números racionales?

Para la definición y especificación de un TDA se utiliza un Lenguaje de Modelado unificado, llamado UML
UML es un lenguaje visual para especificar, construir y documentar sistemas" (OMG - Object Management Group) - UML significa Unified Modeling Languaje. Unificado, por el aporte de muchos metodos y notaciones, independiente de implementaciones, plataformas y lenguajes.

2. ESPECIFICAR UN TDA MEDIANTE UML

Un TDA está compuesto de tres secciones, la primera sección contiene el nombre del TDA, la segunda sección muestra la estructura interna del TDA(atributos), y la tercera sección muestra el comportamiento (operaciones).
En general como pueden tener variado comportamiento, el mismo se puede clasificar de acuerdo a su naturaleza en:

- Operaciones de creación: donde se crea una variable del tipo vacía, con algún valor en particular, con valores pasados por parámetro
- Operaciones observadoras: sirven para pbtener el valor de la variable (atributo) o de una parte
- Operaciones modificadoras: modifican el valor de la variable, esta exsisten en el caso que sea mutable, o sea que pueda ser modificada,
- Operaciones propias del tipo: que tiene que ver con la naturaleza de la clase, por ejemplo en una clase Racional podrìan ser las operaciones matemáticas.



Otro ejemplo podría ser la definición y especificación del TDA Racional, en el mismo se pueden destacar la visibilidad de los atributos numerador y denominador (private) así como la visibilidad de las operaciones (public).

Racional
<-> Entero numerador <-> Entero denominador
Constructor + Racional(numerador, denominador); **Modificadoras** <+> setNumerador(Entero) : Racional <+> setDenominador(Entero) : Racional // pre: parámetro≠0 **Observadoras** <+> getDenominador() : Entero <+> getNumerador() : Entero <+> aCadena():String <+> aFlotante(): float <+> clonar(): Racional **Propias del tipo** <+> sumar(Racional) : Racional <+> restar(Racional) : Racional <+> multiplicar (Racional) : Racional <+> dividir (Racional) : Racional // pre: parámetro≠0 <+> esIgual (Racional) : Boolean <+> esMenor (Racional) : Boolean <+> simplificar():void //pre-condiciones

Véase como ejemplo la especificación de la operación dividir del TDA Racional.

dividir: (Racional) : Racional

Devuelve el cociente de la división entre el número que invoca a la operación (dividendo) y el número que recibe como parámetro (divisor).

Precondición: parámetro distinto de cero.

3. TDA en Java

La especificación UML es INDEPENDIENTE del lenguaje de implementación. Para implementar, primero hay que elegir el lenguaje, Nosotros usaremos Java, donde para implementar el tipo de dato se utilizan las clases y para implementar las operaciones se utilizan los métodos. Luego que es especificado el TDA correspondiente, se realiza la implementación del mismo, esto es se pasa a un lenguaje de programación.

Las Clases son tipos definidos por el usuario, las mismas permiten definir como se componen y comportan los objetos que se van a crear de esa clase. También permiten definir métodos para actuar sobre los objetos de esa clase. Las clases son declaraciones o abstracciones de objetos, por lo tanto, una clase es la definición de un objeto.

Como un ejemplo, tomemos la clase Mascota, si se baja a implementación en java, esta clases se grabara en un archivo llamado Mascota.java. Las clases se prueban previo a ser utilizadas dentro de una aplicación por lo que en general nos vamos a encontrar con el código de prueba de cada clase ADT generado.
En el código java siguiente, mostramos la declaración de una clase Mascota en java, la cual estará en el archivo llamado Mascota.java. Veamos las dos secciones que forman parte de este TDA, una parte de su estructura con sus atributos privados y una parte de operaciones, con sus métodos públicos. Luego tendremos

una clase para probar dicho TDA, que la llamaremos TestMascota y su función es probar las operaciones implementadas en el TDA creado.
El formato que tiene una clase en Java se visualiza a continuación. Nótese que tiene una parte para la estructura interna y una parte paa las operaciones.

Para usar el TDA crearemos instancias de la clase (Objeto).

```
public class Mascota
/**
 Clase para datos de mascotas.
*/

// Estructura – variables de instancia
private String nombre;
private int edad,
private double peso;

// operaciones - interfaz
public double getPeso(){
   return peso;    // es lo mismo que poner this.peso
};

public void setPeso(double nuevoPeso){
  peso = nuevoPeso;
};

.....
```

Los atributos internos que representan el estado de un objeto (en nuestro caso de una mascota) no deberían ser manipulados directamente por el usuario de la clase (nadie podría acceder al peso o nombre de la mascota para modificarlo/consultarlo), sino solamente a través de métodos establecidos.
Esta idea se puede reforzar ocultando componentes al usuario. En general, todos los componentes de datos deberían ser privadas. (tener la palabra clave *private* delante).
La sección pública representa la porción visible al usuario del objeto. Puesto que esperamos ocultar la estructura de los datos, solamente debería aparecer los métodos y constantes como sección pública.

Para probar la funcionalidad de un TDA se utilizan una clase, llamada por ejemplo TestMascota. Veamos un ejemplo de probar el TDA en dicha clase.

```
public class TestMascota

// Estructura
public static void main (String[ ] args){
    //Para declarar y crear un objeto miMascota de la clase Mascota.

    Mascota miMascota = new Mascota();

    //utilizan una operacion de Mascota para modificar el atributo
    miMascota.setPeso(35);

    //accede al peso de la mascota mediante la operación getPeso()
    System.out.println(" El peso de mi mascota es de "+ miMascota.getPeso());
    };
};      //finaliza la clase para probar el TDA Mascota
```

La clase tiene como componentes atributos (datos) y métodos (funciones). Los méetodos pueden actuar sobre los atributos y pueden llamar a otros métodos.
El indicador de visibilidad *public* significa que la componente es accesible desde afuera a través del operador punto. El modicador de visibilidad *private* significa que el componente solamente es accesible a través de méetodos de la clase. Si no hay modicador de visibilidad, tenemos acceso de paquete, el cual se estudia mas adelante. Hay un modificador conocido como *protected* que tiene en cuenta la herencia entre clases, y se verá en detalle más adelante.

Para ejemplificar para qué sirve la visibilidad, mostraremos qué sentencia no puede ser realizada en TestMascota

```
Mascota miMascota = new Mascota();

//OJO!!!, no se puede acceder al atributo peso desde afuera de la clase
miMascota.peso = 50;       //MAL
System.out.println(" El peso de mi mascota es de "+ miMascota.peso );   //MAL
```

4. CLASES Y OBJETOS

Clase es una definición de un tipo de objeto, es una plantilla para construir objetos de un tipo dado, por lo cual especifica el tipo de cada ítem de datos (parte) que tiene esa clase. Cada clase determina los datos que tendrán los objetos y los métodos para manipularlos.

Cada objeto tiene los mismos ítems, pero distintos valores para los mismos ìtems. Todos los objetos de la misma clase responden a los mismos métodos. Los Objetos son variables o instancias de una clase. Cada objeto tiene: datos + métodos. Los ítems de datos también se denominan campos, atributos o variables de instancia.

El método main estará en una sola clase que será la encargada de crear las instancias de los objetos necesarios y comenzar la ejecución. Para probar cada clase que definamos, implementaremos su clase Test (sólo la clase Test tendrá el método main).

Cada definición de clase va en un archivo separado, se usa el mismo nombre para la clase y para el archivo (más la extensión “.java”)

Buena práctica de programación: Ej. CocheCarrera.java para la clase CocheCarrera. Colocaremos todas las clases que necesitamos para ejecutar un programa en la misma carpeta.

Una vez definida la clase, crearemos y usaremos objetos de ese tipo en algún programa (es decir, desde otra clase). Por lo cual la clase que desea usar un objeto deberá:

1. declarar una variable del tipo (clase) deseada

2. crear el objeto (instanciación)
3. invocar el o los métodos necesarios de acuerdo a la funcionalidad deseada

Encapsular información: Los objetos incluyen ítems de datos métodos para accionar sobre los datos
Ocultar Información: Protege los datos internos de un objeto, no permite acceso directo.

Para implementar un TDA en Java usaremos una clase, recordando:
1. Usar el modificador private para todos los atributos
2. No dar al usuario el archivo de la clase
3. Al usuario le damos sólo la interfaz: un archivo con el nombre de la clase, la descripción de los métodos y sus encabezados

Sobre la interfaz: El encabezado brinda los nombres y parámetros de cada método, indica al usuario cómo usar la clase y sus métodos, y eso es todo lo que el usuario necesita conocer

Buena práctica de programación:
- Tener un área de declaración de variables en cada método), que inicialice las variables cuando las declaren,
- Que no se declare variables dentro de un bucle (esto es debido a que toma tiempo durante la ejecución, crear y destruir variables, por lo que es mejor hacerlo sólo una vez antes del bucle).
- Si el compilador requiere inicializar una variable y no se cuenta con un valor inicial, se inicializa en null.
- Podemos usar == y != para ver si una variable clase es igual a null, porque null es usado como una dirección (la dirección nula). Si invocamos un método usando una variable inicializada en null, obtendremos un mensaje de error que dice "Null Pointer Exception".

5. CREACION DE OBJETOS

Para crear un objeto se utiliza la palabra clave new
Sintaxis:

```
nombClase nombInstancia = new nombClase; //crea un objeto de nomClase
```

O también en momentos separados:

```
nombClase nombInstancia;
nombInstancia = new nombClase;
```

Para Invocar un método se se usa el operador punto

```
nombInstancia.metodo()  //nomInstancia es el objeto llamador
```

clase String

```
String nombUsuario;
System.out.println(nombUsuario.length());
```

Clase Mascota

```
Mascota miMascota;
System.out.println(miMascota.toString());
```

6. ACCESOS A VARIABLES DE INSTANCIA Y METODOS

Para forzar la ocultación de información de los objetos siempre vamos a preferir que las variables de instancia sean PRIVATE. El acceso a las variables se realizará mediante métodos especiales. Estos métodos ayudarán a controlar el acceso a las variables de instancia y mantener la coherencia.

Para consultar o modificar el valor del atributo definimos pares de métodos:

set (inicializar, asignar un nuevo valor al atributo)

get (obtener el valor actual del atributo)

Entonces si hay una variable de instancia o atributo `private String nombre;`

definimos dos “métodos de acceso” especiales:

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

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

Si se desea que el usuario no pueda modificar el atributo, pero si verlo, puede existir sólo el método get. Si es una variable que quiere mantenerse totalmente oculta a los posibles usuarios de la clase, no habrá ningún método set ni get para ella (sólo será visible y modificable dentro de la clase), todo dependerá del problema a modelar.

Si la variable de instancia es public se puede acceder desde otra clase, si la variable de instancia es private NO SE PUEDE acceder desde otra clase. PERO....

> Nunca usaremos variables de instancia públicas porque van en contra del concepto de encapsulación. Usaremos siempre variables de instancia privadas y ofreceremos métodos para manipularlas

```
public class TestMascota
{
  public static main (String arg){
     Mascota miMasc = new Mascota("Cata");
     miMasc.nombre = "Catalina";          //ERROR, nombre es private
     System.out.print(miMasc.nombre);     //ERROR, uso inválido
  }
}
```

Nunca usaremos variables de instancia públicas porque van en contra del concepto de encapsulación. Usaremos siempre variables de instancia privadas y ofreceremos métodos para manipularlas.

Respecto a los métodos, éstos pueden ser:

PUBLICO (public): Visible fuera del ámbito de la clase. Cualquier objeto puede invocarlo

PRIVADO (private): Sólo visible dentro del ámbito de la clase. Solo puede invocarse desde la misma clase.

Los métodos privados generalmente se usan para cálculos o tareas internas o auxiliares

```
public class Mascota{
   private String nombre;
   ….
   public void setNombre(String nn)
   public String getNombre()
}

public class TestMascota{

  public static main (String arg){
     Mascota miMasc = new Mascota("Cata");
     miMasc.setNombre("Catalina");
     System.out.print(miMasc.getNombre());
```

```
    }
}
```

7. PALABRA RESERVADA THIS

La palabra this tiene un significado especial para los objetos (es una palabra reservada), se usa dentro de una clase para representar el nombre del objeto que invoca los métodos. En Java podemos omitir this.

Si se omite se entiende que el nombre de una variable de instancia se refiere al objeto que invoca el método.

```
public class Mascota
{
  // Variables de instancia
    private String nombre;
    private int edad;
    private double peso;

  // Interfaz
    public void setEdad(int edad) {
                this.edad = edad;
    }

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

    //...getEdad,getPeso,setPeso,getNombre
 public String toString( ) {
      return "Nombre = " + this.getNombre() +
           " Edad = " + this.getEdad() +
           " Peso = " + this.getPeso();
    }
    // toString y toString1 hacen lo mismo
  public String toString1( ) {
      return "Nombre = " + this.nombre +
           " Edad = " + this.edad+
           " Peso = " + this.peso;
    }
```

9. GENERACION DE OBJETOS

Para crear un objeto se utiliza la sentencia new. Cuando se declara un objeto sólo se reserva lugar en memoria para guardar la referencia al objeto.

```
Mascota miPerro;
```

La sentencia de instanciación (new) es la que reserva lugar en la memoria para guardar la información (atributos) del objeto

```
miPerro = new Mascota();
```

Al usar la sentencia new Especie() se invoca a un método constructor de la clase. Por defecto, Java ofrece un constructor vacío, el cual reserva lugar en memoria sin asignar ningún valor a los atributos

```
Mascota miPerro = new Mascota();
```

Se requiere la implementación de un constructor para instanciar valores pasados por parámetros como se ve a continuación:
Ej: Se crea con un valor para cada atributo

```
miPerro = new Mascota("Titan", 8, 3.4);
```

Otra posibilidad es crear un clon del objeto existente.

```
otroPerro = new Mascota(miPerro);
```

```
public class Mascota{
  // Variables de instancia
  private String nombre;
  private int edad;
  private double peso;

  // Ejemplos de constructores
  public Mascota(String nomb, int edad) {
            nombre = nomb;
            this.edad = edad;
            peso = 1.0; //se inicializa en un valor mínimo
  }
 // Constructores con todos los parámetros
  public Mascota(String nomb,int edad,double peso) {
            this.nombre = nomb;
            this.edad = edad;
            this.peso = peso;
    }
  // Constructor de un clon
  public Mascota(Mascota m) {
            this.nombre = m.nombre;
            this.edad = m.edad;
            this.peso = m.peso;
    }
 }
```

10. VARIABLES Y METODOS ESTATICOS

Hay dos tipos de métodos y variables, de instancia y de clase.

- Los de instancia, no utilizan la palabra clave static

```
public void setNombre(String nn)
String nombre
```

- Los estáticos (o de la clase), utiliza la palabra clave static

```
public static String readLine()
static int version = 10;
```

Las variables estáticas se usan generalmente para definir una constante,

```
public static final double PI
```

Las variables estática son variables compartidas por todos los objetos en la clase, tiene sentido cuando todos los objetos de la misma clase comparten el mismo valor. Un ejemplo puede ser para la clase Persona, la variable estática mayoríaEdad, dado que lo comparten todas las personas, debe usarse con cuidado cualquier objeto puede modificarla inapropiadamente.

```
public class Persona
```

```
private static int mayoriaEdad
```

Variables Estáticas: La misma variable estática es accedida por cualquier objeto de la clase. Puede estar inicializada o no. Se declaran anteponiendo la palabra static. Todas las instancias de la clase (todos los objetos instanciados de la clase) guardan las mismas variables de clase o estáticas. La variable (y su valor) son únicos para todos los objetos instanciados a partir de esa clase.
Existe una única copia de la variable de instancia, que comparten entre todos los objetos

Las **variables de instancia**; se declaran dentro de la definición de una clase sin anteponerle la palabra clave static. Cualquier objeto instanciado de esa clase contiene su propia copia de toda variable de instancia.
Un objeto es una instancia de una clase, y cada objeto tiene su propia copia de un dato miembro, entonces ese dato miembro es una variable de instancia.

Metodos de instancia, son métodos no-estáticos, están asociados con un objeto – el comportamiento del método depende del objeto y por lo tanto "no-estático".

Deben ser invocados por un objeto, antes de usarlo hay que declarar y crear el objeto

```
Ej. el método setEdad, Clase Mascota
 public void setEdad(String nn)


                    Mascota e = new Mascota("Juan");
                    e.setEdad(12);
```

Metodos estáticos o de clase
No se necesita crear un objeto de la clase para invocar un método estático. Si se invoca desde la clase en la que se encuentra definido, basta con escribir su nombre. Si se le invoca desde una clase distinta, debe anteponerse a su nombre, el de la clase en la que se encuentra. Usa el nombre de la clase en lugar del nombre de un objeto para invocarlo.

Se usan para proveer librerías de métodos útiles y relacionados.
Algunos métodos estáticos conocidos

- `método main – //el comienzo de un programa`
- `TecladoIn: define métodos para la entrada por teclado`

No se necesita crear un objeto TecladoIn

- `String s = TecladoIn.readLine();`

La clase Math: define métodos para cálculos matemáticos, es provista por Java, no se necesita crear un objeto Math para invocarlos.

- `Ejemplo: Math.sqrt(45); Math.max(4,30); etc.`