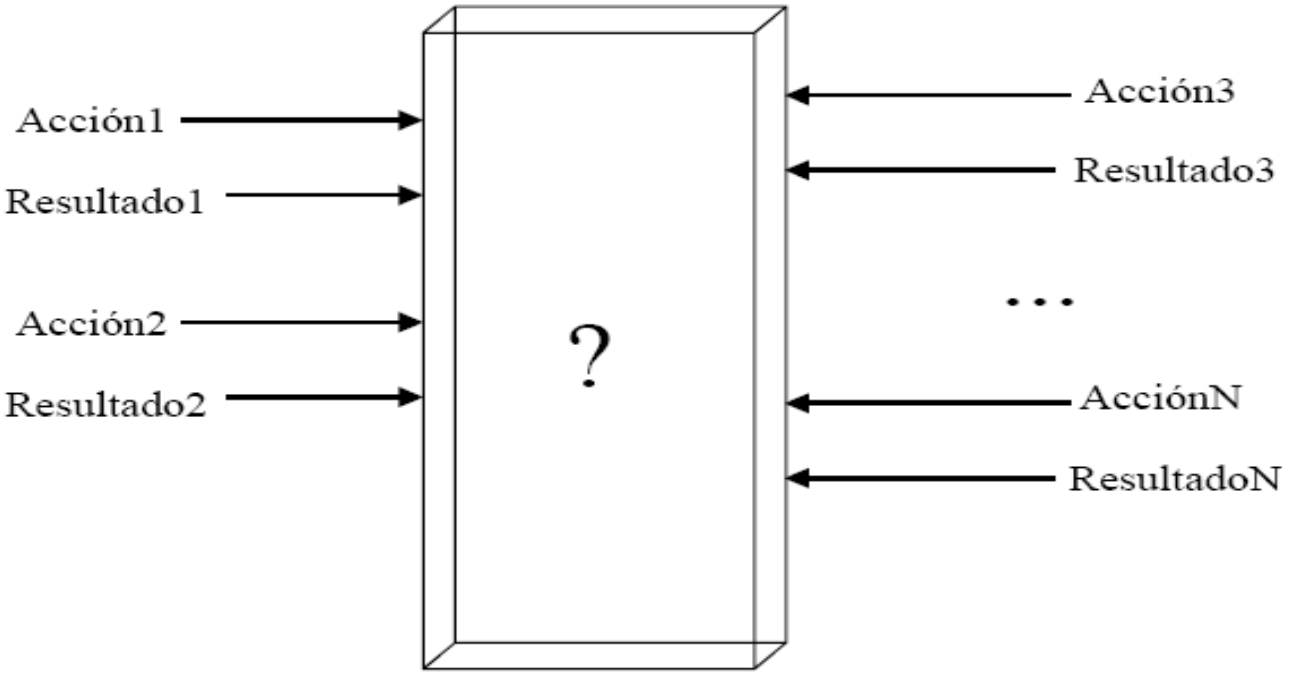
Abstracción

La abstracciones un ejercicio mental que consiste en la comprensión de los atributos tangibles de un objeto. La abstracción se centra en la visión externa de un objeto, y por tanto, sirve para delimitar el comportamiento esencial de un objeto de su implementación ("lo interno").

Materialmente hablando, entonces la abstracción es observar la utilidad de un objeto sin fijarse en cómo se consigue tal utilidad.

Programación Orientada a Objetos

Abstracción...



Abstracción(...)

La abstracción nos lleva a un conocimiento primario de los objetos. Este conocimiento es incrementado cuando se descubre el funcionamiento del objeto, es decir, el cómo y por qué interiormente.

Por lo tanto, la abstracción nos puede llevar a una comprensión total de la implementación del comportamiento de un objeto. Se dice entonces que de lo abstractose puede concluir en lo concreto.

Abstracción...

Ejemplo: la máquina de refrescos.

Abstracción

- Recibe monedas o billetes
- Permite seleccionar el refresco
- Entrega el refresco

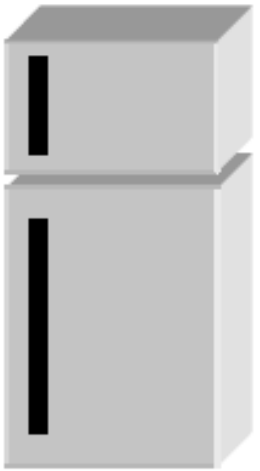
Implementación (concreto)

- Almacena el dinero y lo calcula
- Genera señales internas para escoger el refresco
- Libera el refresco

Paradigma orientado a objetos

Modelo de programación que basa su filosofía en interacciones entre objetos. Cada objeto tiene un comportamiento definido y que puede ser aprovechado por otros objetos.

- Colaborativos y cooperativos
- Reusables
- Distribuibles
- Localizables



Refrigerador

Paradigma OO(...)

Un objeto es un ente que tiene estado, comportamiento e identidad englobados en una sola unidad.

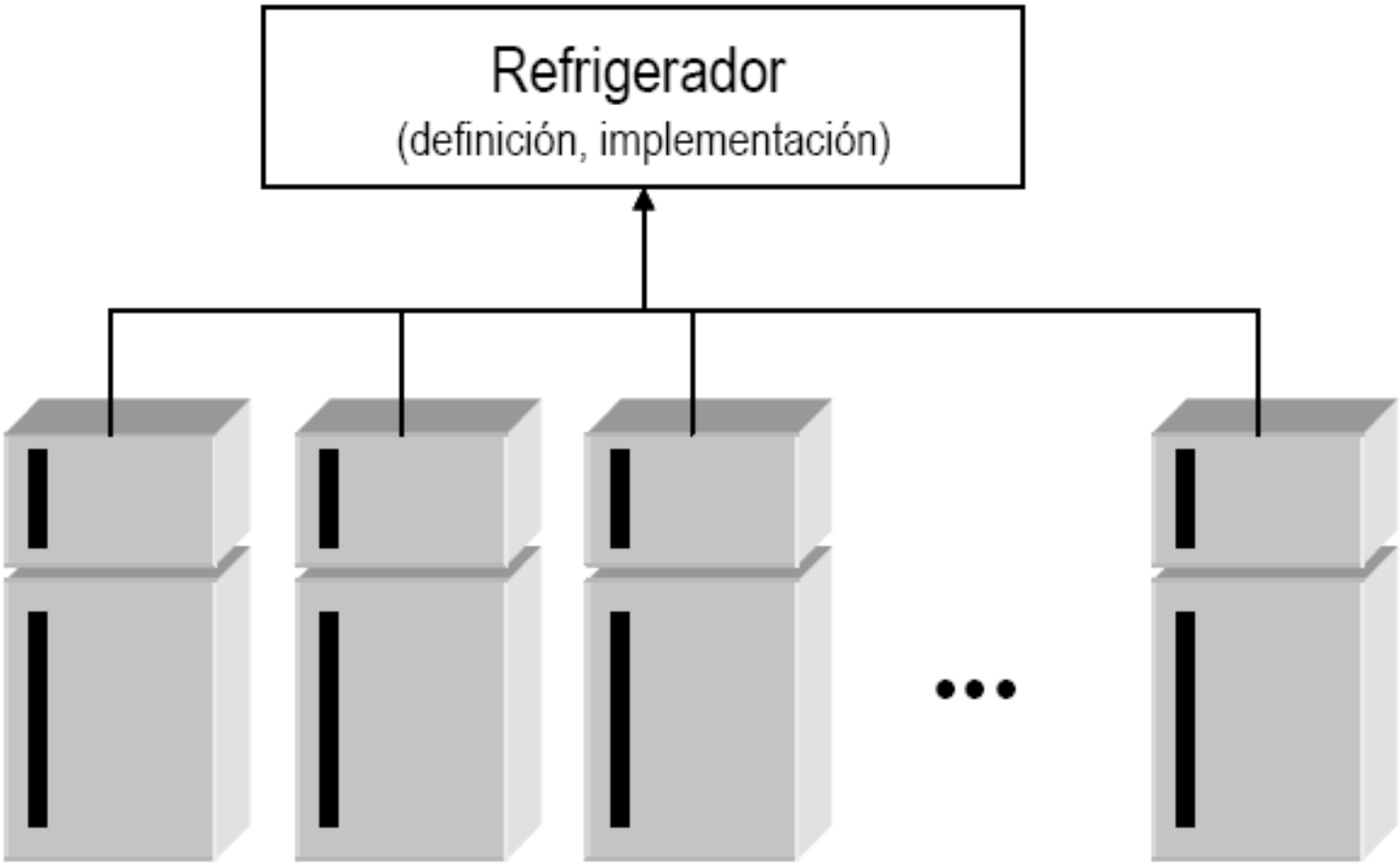
- La *identidad* de un objeto es la propiedad que lo distingue de todos los demás objetos y lo hace único.
- El *estado* de un objeto abarca todas las propiedades (estáticas) del mismo más los valores actuales (dinámicos) de cada propiedad.
- El *comportamiento* de un objeto es la suma de los actos y reacciones en términos de sus cambios de estado y los eventos generados.

Otro nombre que recibe el objeto es el de tipo de dato abstracto.

Paradigma OO(...)

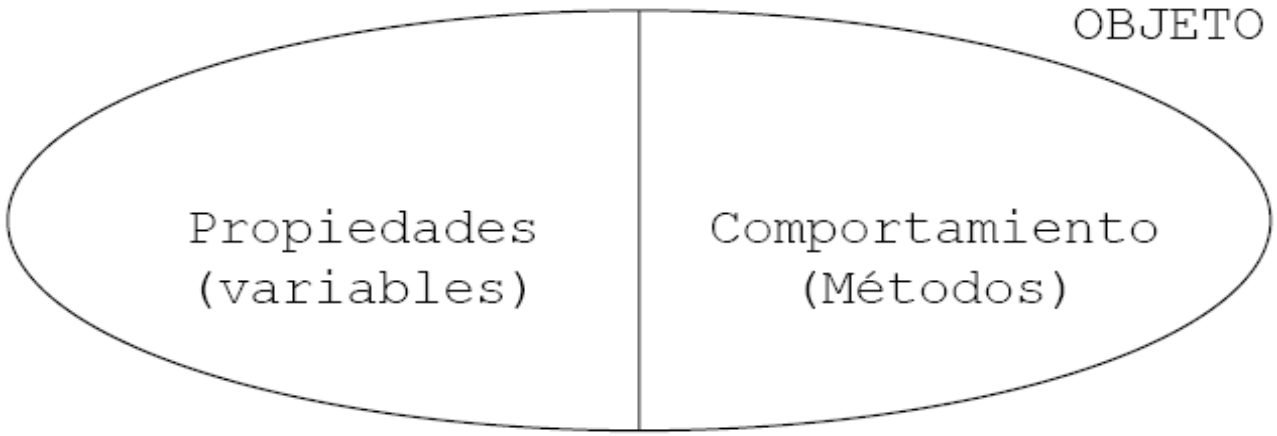
Una clase es la definición formal de un objeto en los términos de estructura y comportamiento común. Así podemos usar la definición de una clase para crear objetos de ese tipo de clase, esto es, crear objetos que contengan todos los componentes especificados en la clase.

Paradigma OO(...)



Paradigma OO(...)

Se llama encapsulamiento a la conjugación de propiedades y comportamiento de un objeto. Esto logra también que se oculte la implementación y variables de ese objeto.



Paradigma OO(...)

Una definición formal de una clase se compone de:

- Campos. Estos son variables que almacenan datos referentes al objeto. También son conocidos como miembros.
- Funciones. Estos son las operaciones que se pueden realizar sobre objetos de esa clase. También son conocidos como métodos.

Los campos pueden ser tipos de datos primitivos u objetos. Los métodos se asemejan a la estructura formal de las funciones.

Paradigma OO(...)

Una clase en Java se define mediante la palabra reservada **class** y enseguida, el identificador de la clase. Las propiedades y el comportamiento se definen dentro del cuerpo de la clase.

```
class NombreClase{
   //Propiedades
   ...
   //Comportamiento
}
```

Paradigma OO(...)

Un objeto es también conocido como una instancia de la clase a la que pertenece. Entonces al crearse la instancia, el objeto contendrá los campos definidos en la clase.

Los campos pueden clasificarse como:

- Variables de instancia, y
- Variables de clase

Paradigma OO(...)

Variables de instancia

- Cada objeto tendrá su propia copia local de cada variable definida en clase
- Estas variables existen cuando se genera la instancia

Variables de clase

- Son variables que existen en la clase y solo existe una sola copia para todas las instancias.
- El valor es compartido y el mismo para todas las instancias.
- Estas variables existen AÚN que no exista ni una instancia de esa clase.

Paradigma OO(...)


```
public class Circulo {
   //variable de clase
   static double PI = 3.14;
   //variables de instancia
   double x;
   double y;
   double radio;
}
```

Paradigma OO(...)

Un objeto puede contener también métodos y se definen con la misma nomenclatura que las funciones.

Los métodos pueden clasificarse como:

- Métodos de instancia, y
- Métodos de clase

Al igual que los campos, los métodos tienen las mismas características: un método de instancia, existe cuando se crea la instancia; un método de clase existe aún cuando no se genere la instancia.

Paradigma OO(...)


```
public class Circulo {
    //variable de clase
    static double PI = 3.14;
    //variables de instancia
    double x;
    double y;
    double radio;
    double area() {
        return PI*radio*radio;
    }
    static double pi() {
        return PI;
    }
}
```

Paradigma OO(...)

Suponga la clase A con una variable de instancia y un método de clase:

```
public class A {
    int x;
    static int cuadradoX() {
        return x*x;
    }
}
```

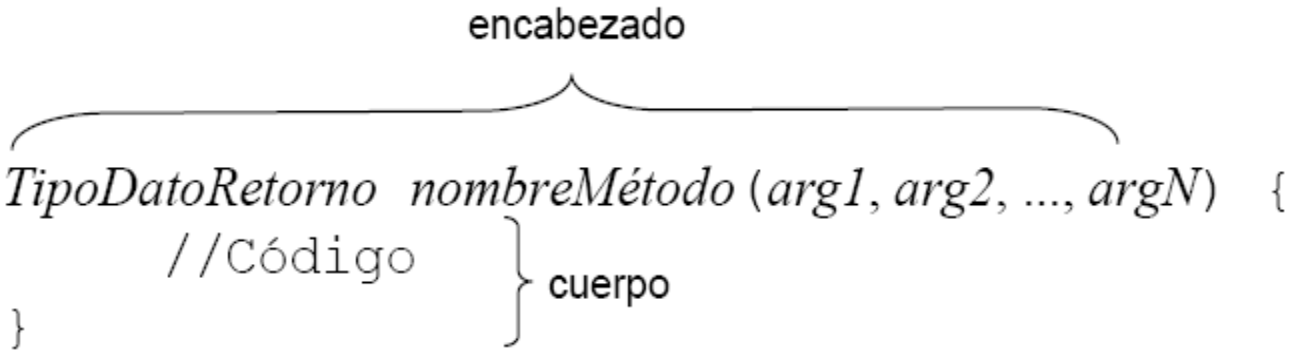
Esto marca el error:

```
"A.java": non-static variable x cannot be referenced
froma static context at line?, column?,
```

¿Por qué?

Paradigma OO(...)

La forma general de definición de métodos es:



En caso que el tipo de dato retorno no sea void entonces, el código deberá al menos contener una línea con la sentencia return.

Paradigma OO(...)


```
public class A {
    public int inc(int j) {
        ++j;
        return j;
    }
}
...
A a;
int j = 10, i;
...
i = a.inc(j);   //¿Cuánto vale i y j?
```

La variable *this*

Esta variable implícita siempre se refiere a la instancia actual. Por medio de esta referencia, se pueden acceder a los campos y métodos del objeto en turno.

```
class Clock{
    int hour= 12;
    void setHour(int hour) {
        hour= hour;
    }
}
...
Clock c;
...
c.setHour(24); //¿Cuánto vale c.hour?
```

La variable *this*

Como se ve en el anterior ejemplo, tenemos un caso de ambigüedad que puede ser resuelto con la referencia this.

```
class Clock{
    int hour= 12;
    void setHour(int hour) {
        this.hour= hour;
    }
}
```

Parámetro

Variable de instancia

Acceso a campos y métodos

- Campos y métodos de instancia

```
A a;
...
a.i = 5;
a.hazAlgo();
```

- Campos y métodos de clase

```
A.dato = 6; //dato y ejecuta()
A.ejecuta(); //fueron definidos con la palabra static
```

Acceso a campos y métodos

Ejercicio 1:

```
public class OpsMat {
        public static void main(String[] args) {
            double d = 762;
            System.out.println(Math.sqrt(d));
        }
}
```

Ejercicio 2:

Haga un programa que obtenga dos números aleatorios y determine cuál de los dos es el más grande utilizando EXCLUSIVAMENTE la clase Math.

Constructores

Un constructores un método especial que no devuelve ningún tipo de dato, que posee el mismo nombre de la clase y que tiene la finalidad de:

- Crear espacio en memoria para el objeto
- Inicializar las variables de instancia

```
class Clock{
    int hour;
    Clock() {
        hour = 12;
    }
    void setHour(int hour) {
        this.hour= hour;
    }
}
```

Constructores(...)

Cuando un objeto es declarado para su uso posterior, es imperativo construir el objeto mediante una llamada al constructor.

```
class A {
    A() {
        ...
    }
}
...
A a = new A();
```

Constructores(...)

Ejercicio 3:

- Defina una clase Prueba con un constructor sin parámetros.
- Defina un método imprime() tal que deberá imprimir el mensaje "Esta es una prueba"
- En la misma clase defina el método main
- En este método, declare un objeto obj de clase Prueba
- Sin construir el objeto, mande llamar el método imprime() de obj.

¿Qué sucede?

Constructores(...)

Inicialmente, todos los objetos cuando son declarados apuntan a null. Al invocar un campo o método de la instancia generará un error (excepción).

A a; ⟶ a = null

...

a = new A(); ⟶ a = 0x12345

En realidad, cada instancia es un apuntador a memoria.

Constructores(...)


```
class A{
    int b;
    A() {
        b = 12;
    }
}
...
A a = new A();
A c = a;
a.b = 15;
System.out.println(c.b); //¿?
```

Sobrecarga de métodos

Hay ocasiones que resulta útil tener un mismo identificador de método para diferentes método con diferente funcionalidad. Ejemplo:

```
class Calculadora {
    double suma(double d1, double d2) {
        return d1 + d2;
    }
    int suma(int i1, int i2) {
        return i1 + i2;
    }
}
```

Sobrecarga de métodos(...)

A esto se le conoce como sobrecarga. Existen reglas para ello:

- El identificador debe ser el mismo.
- El encabezado de cada método sobrecargado debe ser diferente, esto es, debe variar ya sea en el tipo de dato de retorno, o la cantidad de parámetros y/o los tipos para esos parámetros

Sobrecarga de constructores

De la misma forma, es posible definir sobrecarga para los constructores creando un repertorio amplio de maneras de construir un objeto.

```
class Clock{
    int hour;
    Clock() {
        hour= 12;
    }
    Clock(int hour) {
        this.hour = hour;
    }
}
```

```
Clock c1 = new Clock();
Clock c2 = new
Clock(13);
c1 = c2;
¿c1.hour?
```

Sobrecarga de (...)


```
public class Circulo {
   double x;
   double y;
   double r;
   public Circulo(double x, double y, double r) {
      this.x = x; this.y = y; this.r = r;
   }
   public Circulo(double r) { this(0.0, 0.0, r); }
   public Circulo(Circulo c) { this(c.x, c.y, c.r); }
   public Circulo() { this(0.0, 0.0, 1.0); }
}
```

Sobrecarga de (...)


```
class ClaseEjemplo{
    int var1, var2;
    ClaseEjemplo() {
        var1 = -1;
        var2 = -1;
    }
    ClaseEjemplo(int var1, int var2) {
        this.var1 = var1;
        this.var2 = var2;
    }
}
```

Sobrecarga de (...)


```
public static void main(String[] args) {
    ClaseEjemplo m = new ClaseEjemplo(5,10);
    ClaseEjemplo n = new ClaseEjemplo();
    //Cuanto vale m.var1?
    //Cuanto vale n.var2?
}
```

La clase String

Esta clase permite utilizar valores en forma de cadena y las operaciones sobre los mismos. NO es un arreglo de caracteres aunque ofrece métodos para operar sobre algunos de sus caracteres.

```
String cad = "Esta es una cadena";
System.out.println(cad.charAt(3));
System.out.println("Longitud: " + cad.length());
```

La clase String(...)

Comparación de cadenas

Existen dos formas de comparar cadenas:

- Comparación exacta

```
String cad = "Cadenita";
boolean igual = cad.equals("cadenita");
```

- Comparación sin sensitividad

```
igual = cad.equalsIgnoreCase("cadenita");
```

Las cadenas NO se pueden comparar directamente con los operadores de igualdad (==)

La clase String(...)

Concatenación de cadenas

Existen dos formas de concatenar cadenas:

- Usando la función concat(String)

```
String cad = "Cadenita";
cad = cad.concat(" y cadenota");
```

- Usando el operador +

```
cad = cad + " y más cadenitas";
```

Cuando se intenta concatenar un objeto o una variable a una cadena, Java intenta convertir ese dato a su representación cadenizada.

Más sobre arreglos

A diferencia de C++, los arreglos también son objetos, similares a los arreglos de creación dinámica en C++.

```
int[] arreglo = new int[5];
System.out.println("Tamaño: " + arreglo.length);
```

Parámetros a la JVM

Se dice que una clase es ejecutable (o es un programa en Java) si tiene el método estático main.

Este método define un parámetro que es un arreglo de cadenas (String[] args).

Cada cadena corresponde a un parámetro enviado desde la línea de comando, de tal forma que args[0]es el primer parámetro, args[1]el segundo, y así sucesivamente en orden de aparición.

Parámetros a la JVM(...)

Ejercicio 6:

```
public class Parametros {
   public static void main(String[] args) {
      for (int i=0; i<args.length; i++) {
         System.out.println(args[i]);
      }
   }
}

java Parametros p1 p2 p3 p4
```

Tareas

- Haga un programa que reciba como parámetro una lista de números. El programa deberá ordenarlos de manera descendente e imprimir la lista ordenada.

- Haga un programa que reciba una cadena como parámetro e imprima todas las subcadenas posibles de esa cadena.

Tareas(...)

- Desarrolle una clase Hex2IPconvertidora de direcciones IP a formato hexadecimal y viceversa. La clase recibe dos parámetros:
  - El primer parámetro es “-hex” (convertir dirección a IP) o “-ip” (convertir de IP a hexadecimal)
  - El segundo parámetro es una cadena hexadecimal de 8 dígitos (cuando el primer parámetro es “-hex”) o una dirección IP (cuando el primer parámetro es “-ip”)
- La clase definirá, al menos, dos métodos NO ESTÁTICOS correspondientes a cada uno de los parámetros. La misma clase define el método main.

Tareas(...)


```
C:\> java Hex2IP –hex “0A05FF01”
10.5.255.1

C:\>java Hex2IP –ip “10.5.255.1”
0A05FF01

C:\>java Hex2IP –hex “GH34VTPP”
Error
```