

# Lenguajes de Programación 1

Unidades 1 y 2

# Lenguaje de Programación Java

- **Fuertemente tipado:** Cada variable y objeto corresponde a un tipo. Los arreglos comprenden colecciones del mismo tipo.
- **Orientado a Objetos:** Emplea este paradigma de programación para la resolución de problemas.
- **Multiplataforma:** Capaz de ejecutarse en diferentes sistemas operativos y arquitecturas de ordenadores.

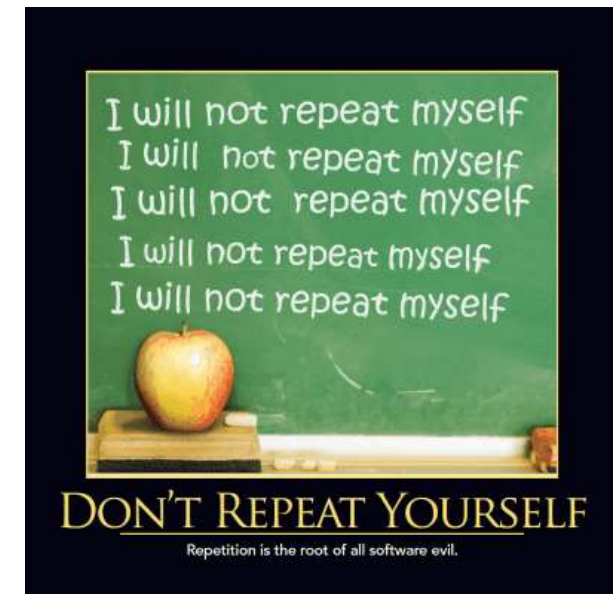


Indonesia



# 1. Funciones

- Java: Función principal de nombre **main**.
- Principio **DRY**: Don't Repeat Yourself!
- Permite implementar el código fuente una sola vez para emplearlo cuantas veces se requiera.
- Partes que lo constituyen:
  - Modificar de acceso: private, public, protected.
  - Nombre de la función: de preferencia expresando un verbo.
  - Parámetros: Conjunto de variables con sus correspondientes tipos cuyos valores serán expresados en tiempo de ejecución.



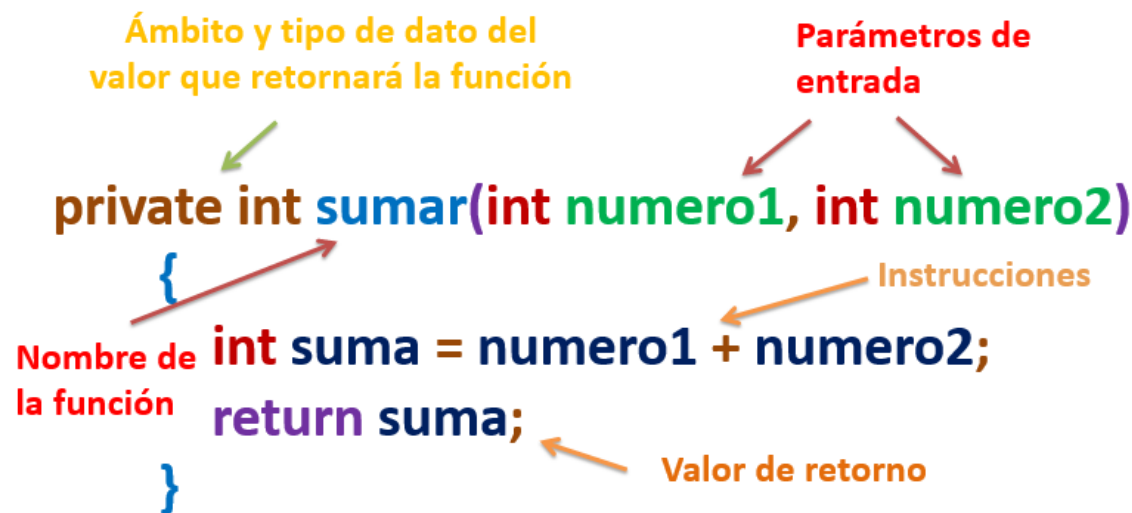
## 1.1 Funciones sin retorno

- Comprenden bloques de código fuente que se ejecutan al ser la función invocada pero, por lo general, no devuelven ningún tipo de valor en el nombre de la función.
- Ciertos lenguajes, por defecto, retornan el valor de 0 en el nombre de la función si no existen errores de ejecución, y el valor de 1 si la función se ejecuta con error (o viceversa).
- Pueden hacer uso de la función **return** para culminar la función si así se requiere.
- **void**: Tipo de dato que indica la ausencia de uno.



## 1.1 Funciones con retorno

- Retornan un determinado tipo de dato en el nombre de la función.
- Esta funcionalidad se realiza a través de la sentencia **return**.
- Es el tipo de función más comúnmente empleado.

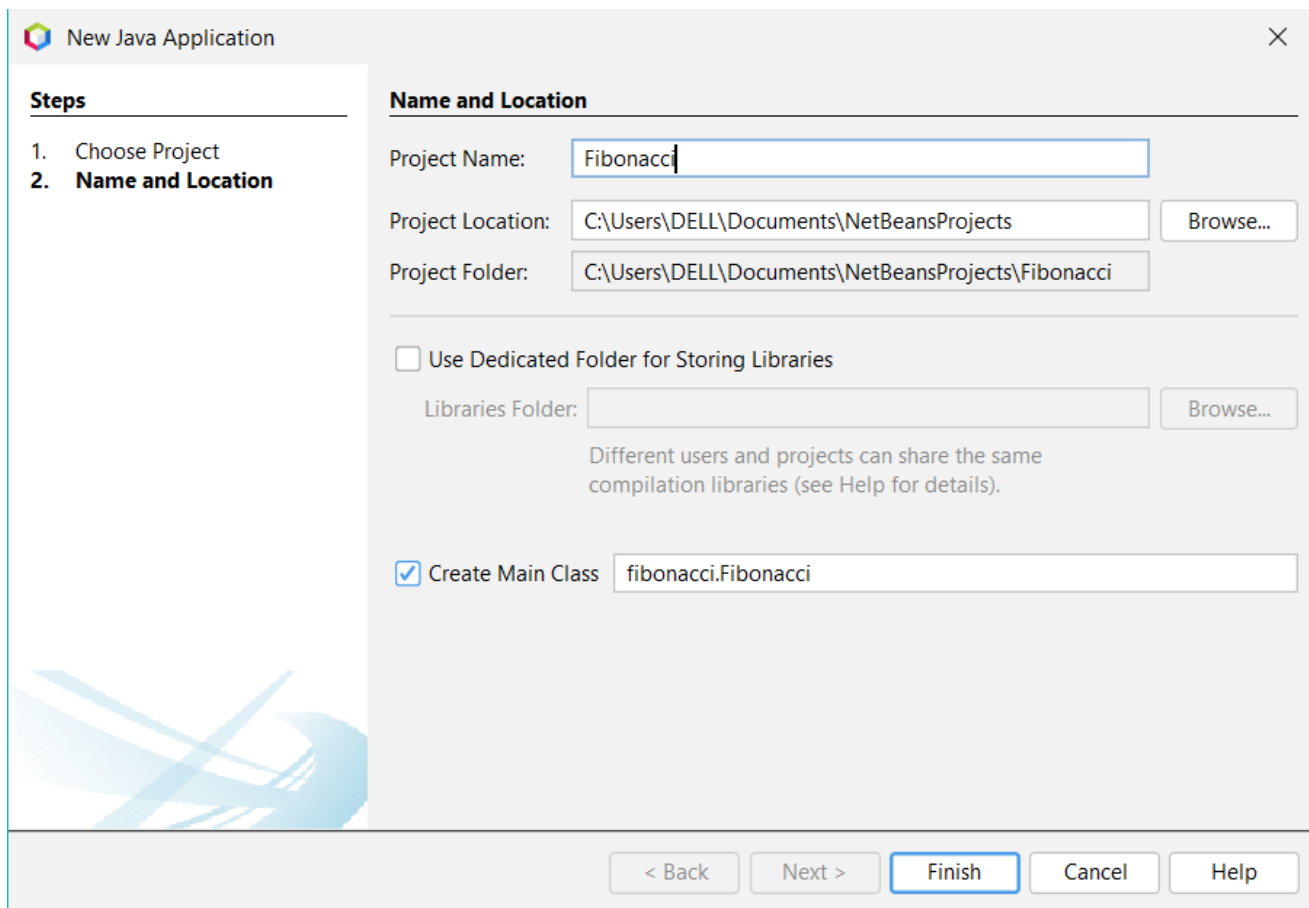


The diagram shows a C++ function definition with several annotations:

- Ámbito y tipo de dato del valor que retornará la función** (green arrow) points to `private int`.
- Parámetros de entrada** (red arrows) points to `int numero1, int numero2`.
- Instrucciones** (orange arrow) points to `suma = numero1 + numero2;`.
- Nombre de la función** (red arrow) points to `suma` in `return suma;`.
- Valor de retorno** (orange arrow) points to `return` in `return suma;`.

```
private int sumar(int numero1, int numero2)
{
    int suma = numero1 + numero2;
    return suma;
}
```

# Serie de Fibonacci



New Java Application

**Steps**

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name:

Project Location:  Browse...

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:  Browse...

Different users and projects can share the same compilation libraries (see Help for details).

☒ Create Main Class

< Back Next > **Finish** Cancel Help

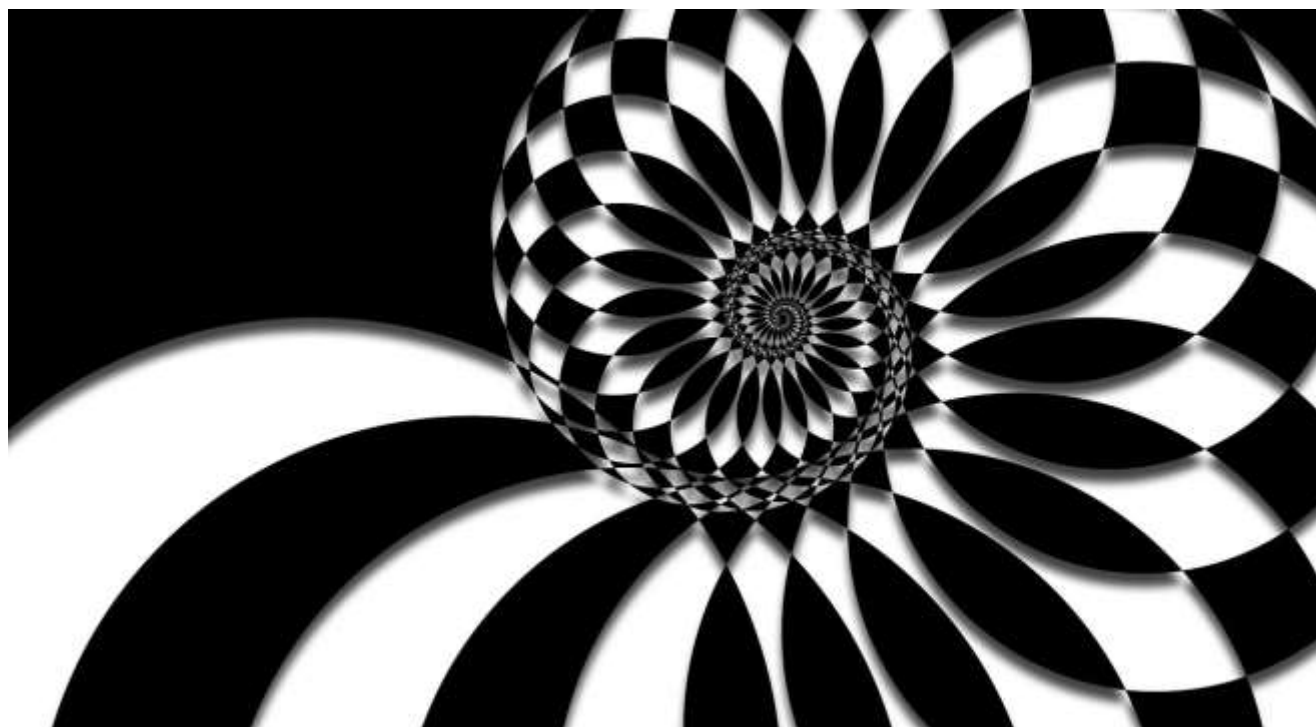
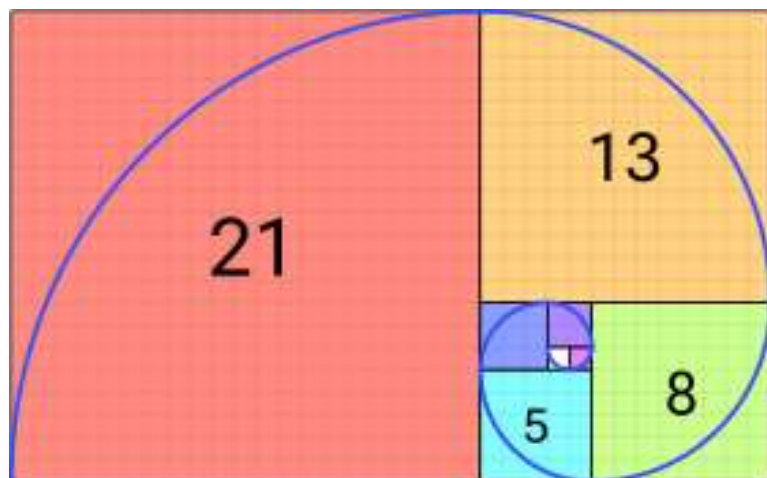


0 1 1 2 3 5 8 13 21





## Serie de Fibonacci



# Serie de Fibonacci

```
1 package fibonacci;
2 import java.util.Scanner;
3
4 // Comentario aqui
5 public class Fibonacci {
6
7     // Comentario aqui
8     public static void main(String[] args) {
9         Scanner lectura = new Scanner (System.in);
10        System.out.println("Ingrese cantidad de digitos: ");
11        int num = Integer.parseInt(lectura.next());
12
13        if (num <= 2) {
14            System.out.print("Número no válido");
15            return;
16        }
17
18        int n1 = 0;
19        int n2 = 1;
20        int aux;
21        System.out.print(n1 + " " + n2 + " ");
22
23        for (int i=0; i<num-2; i++) {
24            aux = n1;
25            n1 = n2;
26            n2 = generarNumero(aux, n2);
27            System.out.print(n2 + " ");
28        }
29
30        System.out.println();
31    }
```

```
// Comentario aquí
public static int generarNumero(int n1, int n2) {
    return n1+n2;
}
```

Ahora implementa la serie usando un método sin retorno!



## 1.1 Recursividad

- Estrategia a través de la cual una función se invoca a sí misma.
- Concepto teórico de fácil comprensión pero comúnmente de difícil implementación práctica.
- Por lo general, todo problema iterativo puede ser implementado a la vez a través de recursividad.





```
1 package fibonacci;  
2 import java.util.Scanner;  
3  
4 // Comentario aquí  
5 public class Fibonacci {  
6  
7     // Comentario aquí  
8     public static void main(String[] args) {  
9         Scanner lectura = new Scanner (System.in);  
10        System.out.println("Ingrese cantidad de dígitos: ");  
11        int num = Integer.parseInt(lectura.next());  
12  
13        if (num <= 2) {  
14            System.out.println("Número no válido");  
15            return;  
16        }  
17  
18        for (int i=0; i<num; i++) {  
19            System.out.print(generarNumero(i) + " ");  
20        }  
21  
22        System.out.println();  
23    }  
24
```



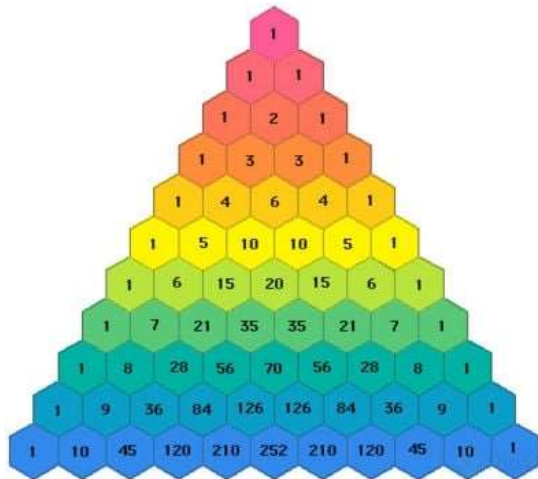


```
25 // Comentario aquí
26 public static int generarNumero(int numero) {
27     if (numero == 0)
28         return 0;
29     else if (numero ==1)
30         return 1;
31     else
32         return generarNumero(numero-1)+generarNumero(numero-2);
33 }
34 }
```



## Tarea 2: Triángulo de Pascal con Funciones

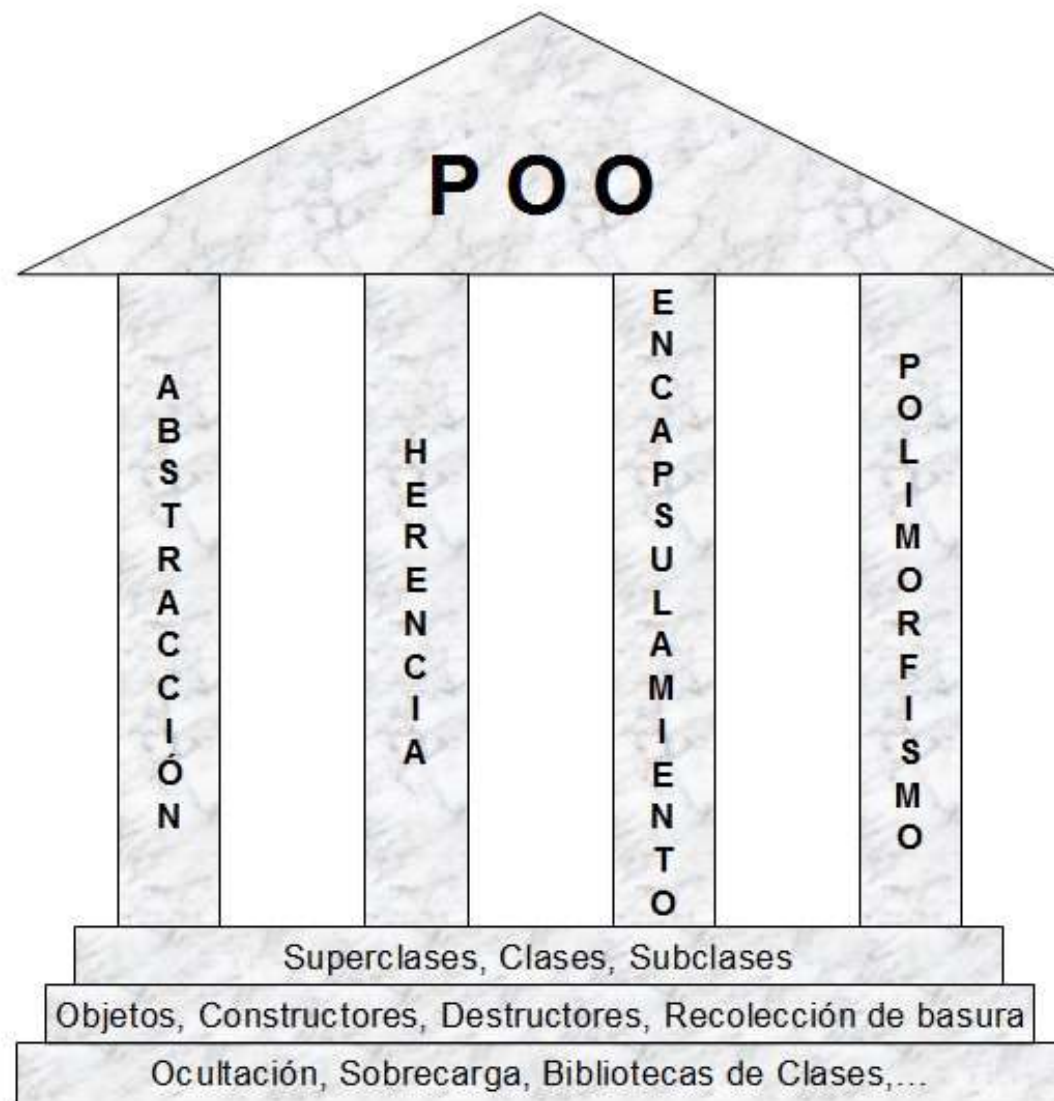
- Implementar una solución iterativa y recursiva del triángulo de Pascal.
- Puedes usar ChatGPT, pero por favor procura entender el código fuente y comentarlo adecuadamente...
- El parámetro de entrada será el número de filas del triángulo.
- Crea tantas funciones como consideres necesario.



## 2. Programación Orientada a Objetos

- **POO**: Programación orientada a objetos (**OOP**: *Object Oriented Programming*).
- Es un paradigma de programación empleado para la implementación de soluciones de software haciendo uso de objetos y las relaciones entre éstos.
- Permite crear aplicaciones escalables y flexibles así como también la reusabilidad del código fuente.
- Se basa en conceptos clave como:
  - Abstracción (clase)
  - Encapsulamiento
  - Herencia
  - Polimorfismo







## Tarea 3

- **Ensayo:** Importancia de la Programación Orientada a Objetos en la implementación de soluciones de software.
- Mínimo 1 página, máximo 3 páginas.
- Criterio personal (no se permiten herramientas de IA).
- Cuidado con el plagio! (Citar de ser necesario).



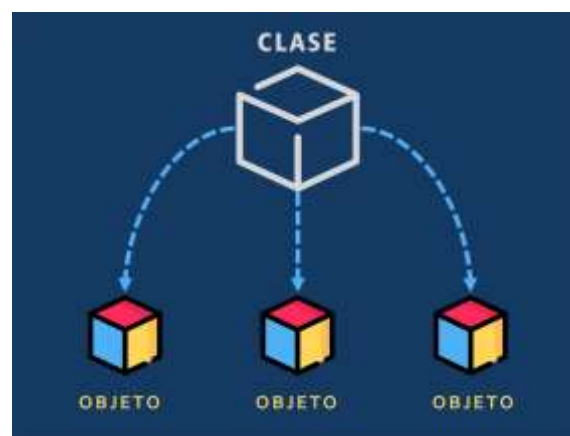
## 3. Clases

- Comprende un modelo para la abstracción de un concepto del mundo real.
- Está conformada principalmente por:
  - Atributos
  - Constructores
  - Métodos
  - Propiedades de acceso
- Debe ser nombrada en singular comenzando por una letra mayúscula:
  - Estudiante
  - Persona
  - Empleado
  - Transaccion
  - Reporte



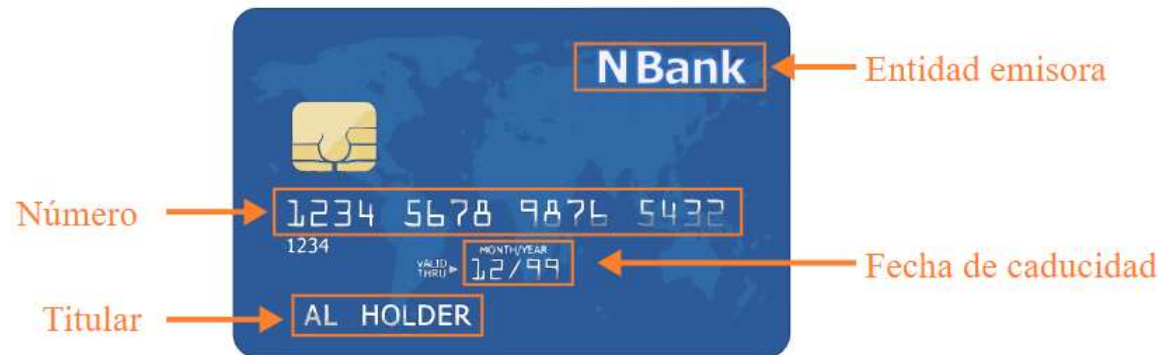
## 3.1 Objeto

- Un objeto es la instancia específica de una clase.
- En Java se crean a través de la sentencia **new** y la llamada al **constructor** respectivo.
- En Java **todo** es un objeto.
- Los objetos son almacenados en memoria hasta que el recolector de basura los destruye.
- Por lo tanto es buena práctica de programación limitar el tiempo de vida de los objetos.



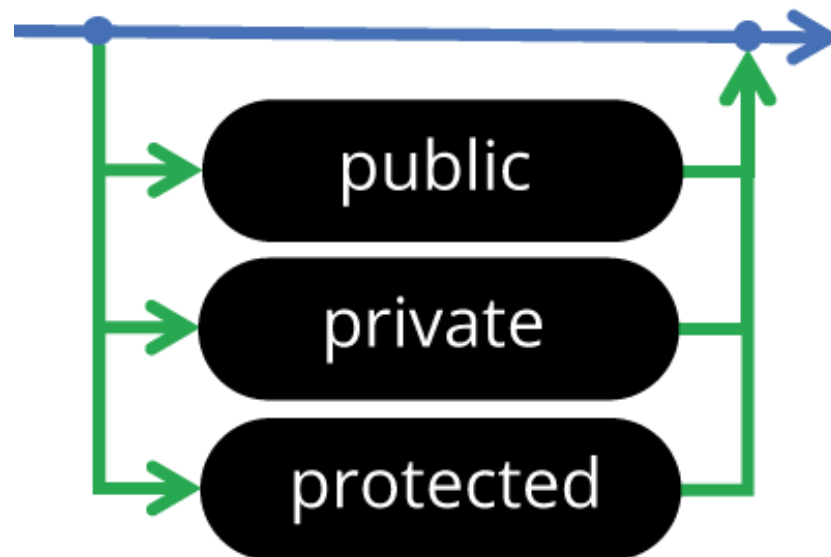
## 3.2 Atributos

- Corresponden a características que describen a la entidad que la clase abtrae.
- Pueden ser vistos como variables de una clase.
- Cada atributo tendrá un tipo de dato: Lenguaje fuertemente tipado.
- Los atributos pueden ser referencias hacia otras clases: Relaciones entre clases.
- Los atributos comúnmente tendrán el modificador de acceso **private**.



## 3.3 Modificadores de acceso

- **Private:** Accesible únicamente desde la misma clase.
- **Public:** Accesible desde cualquier parte.
- **Protected:** Accesible únicamente desde la misma clase y la clase que hereda.

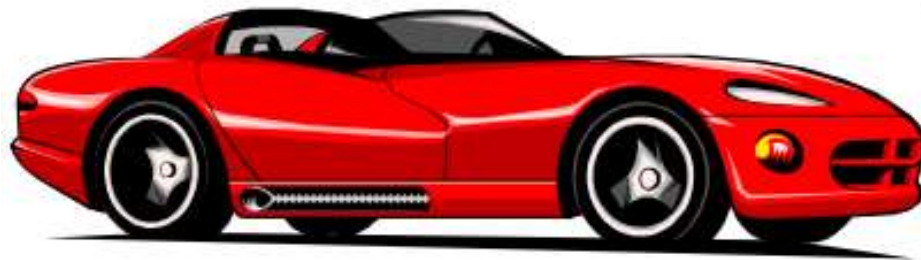


## 3.4 Métodos

- Corresponden a acciones o funcionalidades capaces de ejecutar la entidad que la clase abstrae.
- Por lo tanto son funciones declaradas e implementadas dentro de una clase.
- Como buena práctica se debe implementar por cada clase un método toString().
- Este método retornará como una cadena con la información completa del objeto como una cadena de texto.
- **Sobrecarga:** Métodos con el mismo nombre pero con diferente número y/o tipos de parámetros.
- Comúnmente, todos los métodos emplearán el modificador de acceso **public** con excepción de aquellos que tienen importancia sólo dentro de la misma clase.
- Atributos y métodos son accesibles a través del operador punto (.)







■ Atributos:

- color
- velocidad
- ruedas
- motor

■ Métodos:

- arranca()
- frena()
- dobla()





#### Características

- Marca
- Color
- Tipo
- Precio
- No. de Puertas
- Tipo Combustible
- Cilindros
- Transmisión

#### Acciones

- Encender
- Avanzar
- Retroceder
- Detener
- Apagar

#### Características

- Nombre
- Especie
- Color
- Edad

#### Acciones

- Comer
- Dormir
- Correr



## 3.5 Constructores

- Un constructor es un tipo especial de método empleado para la creación de objetos, es decir, instancias específicas de una clase.
- Sobrecarga de constructores: Constructores con el mismo nombre pero con diferente número y/o tipos de parámetros.
- Los constructores deben necesariamente emplear el modificador de acceso **public**, a menos que se requiera ocultar el constructor del exterior.
- Los constructores necesariamente deberán poseer el mismo nombre de la clase.



## 3.6 Propiedades de acceso

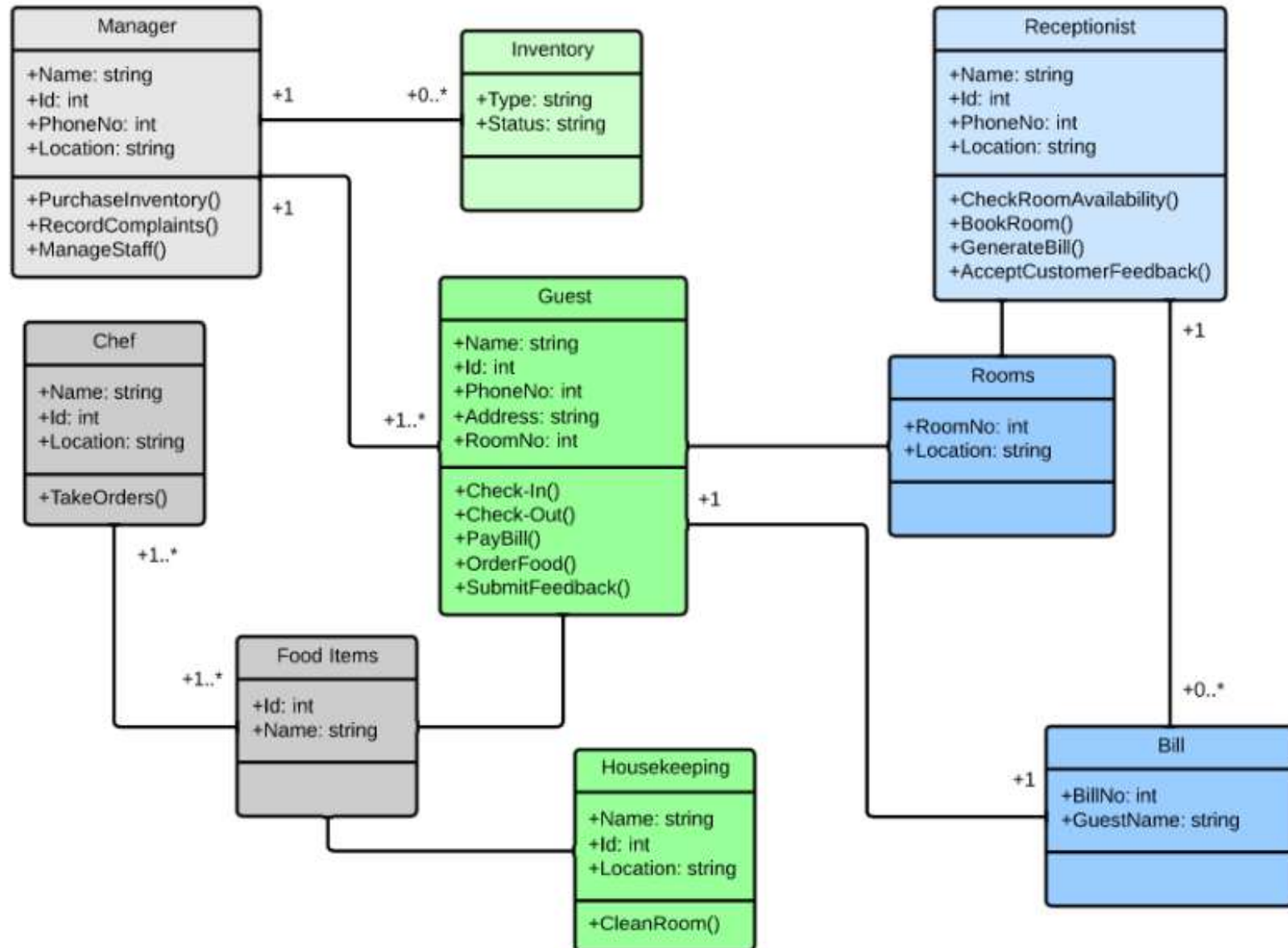
- Dado que los atributos comúnmente tendrán un modificador de acceso `private`, es buena práctica implementar getters y setters.
- **Getters:** Permiten acceder al valor almacenado en el atributo.
- **Setters:** Permiten establecer o actualizar el valor almacenado en el atributo.
- En su forma más básica pueden ser vistos como métodos públicos para lectura y escritura de cada atributo de la clase.



## 3.7 Relaciones entre clases: Diagramas UML

- UML: Unified Modeling Language.
- UML representa la apuesta de tres de los más prestigiosos expertos en metodologías orientadas a objetos: Grady Booch, James Rumbaugh e Ivar Jacobson.
- Cada clase puede tener relación con otras clases.
- Un diagrama UML representa una o varias clases y las relaciones entre éstas.







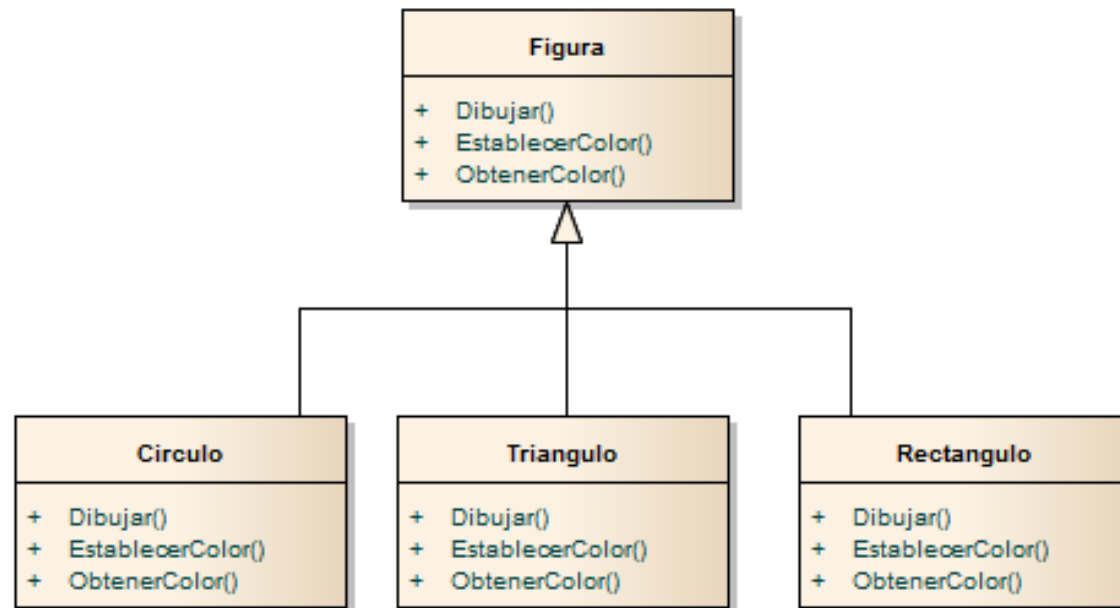
## 4. Encapsulamiento

- Cada objeto puede ejecutar acciones debido a los métodos implementados en la clase a la que pertenece.
- Sin embargo, esta funcionalidad implementada es indiferente para el resto de objetos con los que interactúa.
- **Caja negra:** El objeto toma como entrada una serie de parámetros, ejecuta su método y retorna un resultado de ser el caso.
- El encapsulamiento permite entonces que el cómo fue implementada una clase sea indiferente para otras clases.



## 5. Herencia

- Mecanismo a través del cual una clase hereda de la clase padre sus atributos y métodos.
- Permite definir una nueva clase que añada nuevas características sobre una clase ya existente.



## 5. Herencia

- Cuidado! En Java los constructores no se heredan.
- Por lo tanto cada clase deberá implementar sus propios constructores.
- Sin embargo, se pueden hacer uso de los constructores de la clase padre a través de la sentencia **super** para establecer los valores de los atributos heredados.



## 6. Polimorfismo

- Permite definir múltiples clases con funcionalidad diferente, pero con atributos y métodos comunes.
- Un método abstracto no proporciona una implementación, por lo que obliga a la clase hija a implementar la funcionalidad del método.
- Un método virtual puede ser sobrescrito, o utilizarse tal como está implementado en la clase madre (Anotación **@Override**).



