

# Programación 2

**Tecnicatura en Desarrollo de Aplicaciones  
Informáticas**

Jueves 19 de Agosto de 2021

# Objetivos

— — —

- Conocer los conceptos básicos de la Programación Orientada a Objetos
- Pensar y diseñar en términos de Objetos
  - Representar problemas en términos de objetos que interactúan en relación cliente/servidor
  - Clasificar los conceptos de un problema, de acuerdo a distintas relaciones

# Docentes

— — —

Prof. Dr. Luis Berdun

Prof Dr. Marcelo Armentano

Prof. Dr. Ariel Monteserin

Ing. Sebastian Vallejos

Andrés Mozo

Yago Lacoste

Soledad Merino

Franco Bayugar

# Organización de la Materia

— — —

- 4 Parcialitos (para promoción)
  - 1 Examen Parciales (con su Recuperatorio y prefinal)
  - 1 Trabajo Práctico Especial
- 
- Condiciones para la Promoción
    - $\text{Nota Examen} + \text{\#Parcialitos aprobados} = 10$
    - La nota del examen debe ser mayor igual que 7
    - TP Especial aprobado

# Fechas Importantes

---

1er Parcialito      --      Jueves 26 de Agosto

2do Parcialito      --      Jueves 9 Septiembre

3er Parcialito      --      Martes 28 de Septiembre

4to Parcialito      --      Jueves 7 de Octubre

PARCIAL              --      28 de Octubre

Recuperatorio      --      11 de Noviembre

Prefinal              --      25 de Noviembre

# Programación Orientada a Objetos

\_\_\_\_\_

## Una revolución Industrial (reutilización del software)



# ¿Que se busca?

---

- Incremento
  - Productividad
  - Calidad del software
  - Comprensión del software
  - Tiempo útil del software



# Ciclo de Vida del Software

----

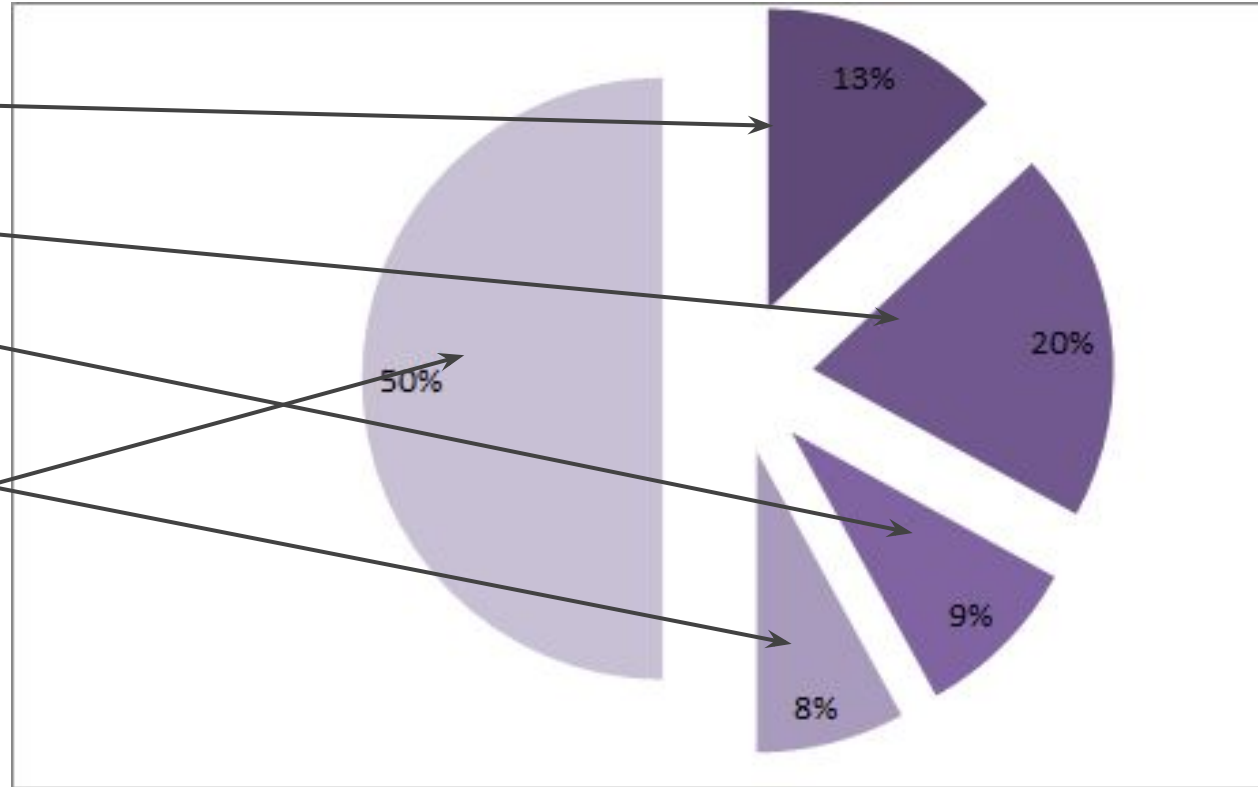
Análisis

Diseño

Implementación

Testing

Mantenimiento



La  
te  
in  
se

Alrededor de seis horas a la semana invierten los profesionales informáticos en corregir los errores y deficiencias de las conocidas como deudas técnicas, según el estudio [State of Technical Debt 2021](#) de la tecnológica Stepsize. Es decir, pierden casi un día de trabajo a la semana para solventar

o a la



Este tipo de problemas heredados no sólo dificulta el funcionamiento de las herramientas actuales, sino que, además, **puede complicar o llegar a impedir el desarrollo de nuevas funciones**, lo que fuerza a que se adopten soluciones muy limitadas.

interrupciones en el funcionamiento de la herramienta, lo que hace que los informáticos tengan que intervenir para solventarlos y dediquen parte de su jornada a reparar en lugar de a desarrollar.

# Aplicación Orientada a Objetos

---

Conjunto de **objetos** que interactúan mediante el **envío de mensajes** para cumplir un conjunto de **objetivos**



# Orientación a Objetos

---

- Énfasis en las **abstracciones** de los datos
- Las **funciones** y los **datos** son encapsulados en entidades **fuertemente relacionadas**
- Facilita el mantenimiento por **especialización**
- Correlación directa con las entidades del dominio

# Brecha Semántica

---

Reduce la brecha de representación entre el modelo de software y el dominio que se representa

MODELO

DOMINIO



# Definiciones

# ¿Qué es un Objeto?

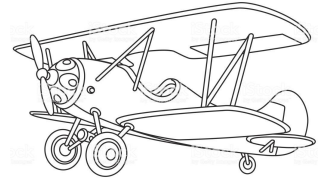
---

- Un Componente de **software**
- Una entidad almacenada en **memoria**
- Un objeto encapsula **datos** y **comportamiento** en una unidad

# Objetos : Conceptos Reales

---

**En un sistema de control de tráfico aéreo:**  
aviones, pistas, torres de control, etc.



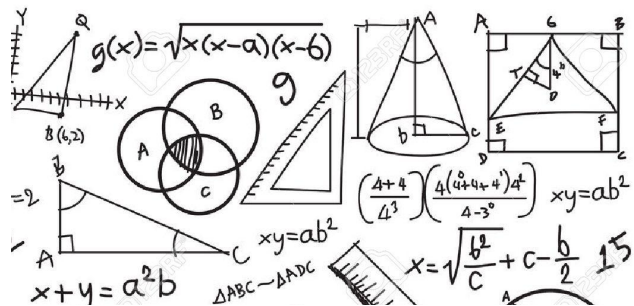
**En un sistema de alumnos de una Facultad:**  
Alumno, Curso, Docente, etc.





# Objetos: Entidades Abstractas

- Una fórmula matemática



- Un evento (click del mouse)



# Vista Dinámica de los Objetos

---

- Los objetos se crean y se destruyen en forma **dinámica**
- Los objetos tienen su propia **identidad** y **encapsulan estado y comportamiento**
- Las **variables de instancia** mantienen **referencias a otros objetos**
- El **comportamiento** de los objetos es definido por los **métodos**

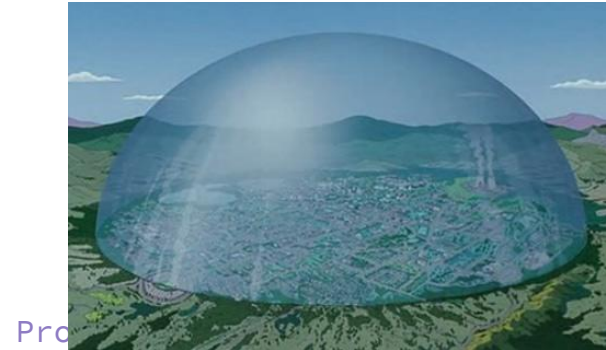
# Los Objetos

— — —

“Revelan qué pueden hacer y qué información pueden proporcionar, pero **no** revelan cómo lo hace o cómo lo conoce.”



## ENCAPSULAMIENTO



# Encapsulamiento

— — —

- El encapsulamiento se refiere al **ocultamiento de información o detalles**



# Encapsulamiento

---

- Los objetos **encapsulan** sus **datos**
  - Los datos en los objetos son **privados**
  - Desde “el mundo exterior” no se puede **acceder** o **modificar** sus datos



# Encapsulamiento

---

- Los **métodos** son (típicamente) **públicos**
  - Desde “el mundo exterior” se puede **enviar mensajes** que invoquen a los **métodos**



# Programa Orientado a Objetos

— — —

*Un programa orientado a **objetos** consiste de objetos que **interactúan** con otros **objetos** mediante el envío de **mensajes** de uno a otro.*

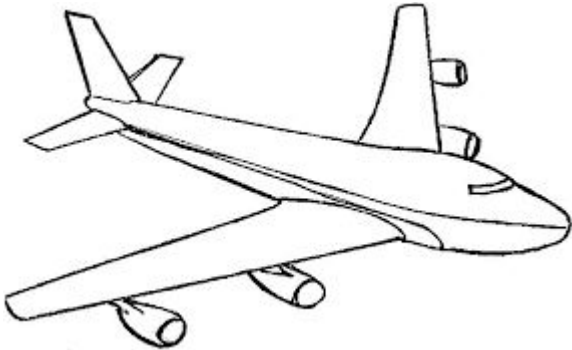


# Responsabilidades y Colaboraciones

---

## Los Objetos tienen responsabilidades

Un Avión es responsable por conocer su hora de aterrizaje

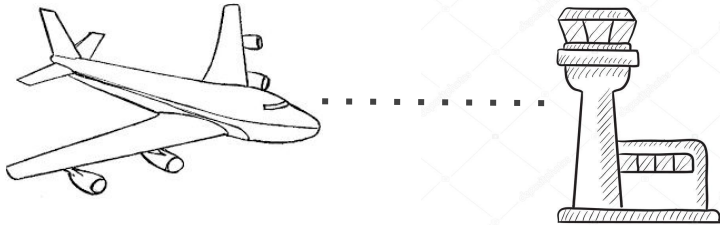




# Responsabilidades y Colaboraciones

---

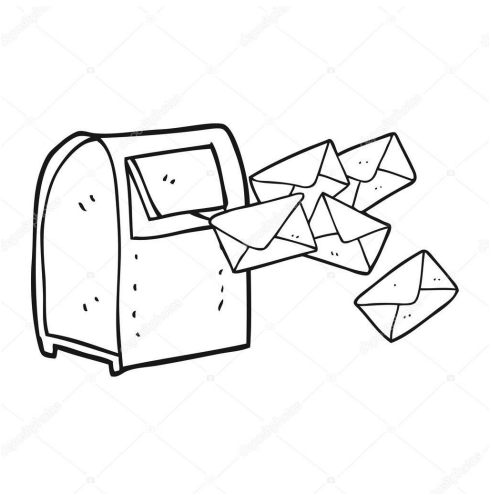
- Los objetos colaboran entre sí para cumplir sus responsabilidades
  - Un avión puede colaborar con la Torre de Control para calcular el tiempo de aterrizaje
  - Análogo a la forma en que colabora la gente de acuerdo a su especialidad y conocimiento.



# Colaboración

— — —

*La Colaboración se da a través del envío de **mensajes***



# Mensajes y Métodos

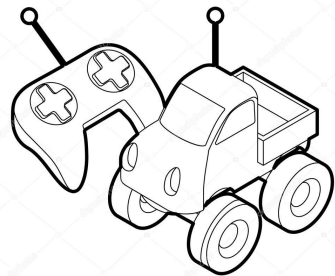
---

- Todo el computo es realizado por los **objetos**
- La única forma de interactuar con un objeto es mediante el envío de un **mensaje** a este

# Mensajes y Métodos

---

- **Mensaje:** Señal que se le envía a un *objeto* (receptor) para invocar un *método*



- **Método:** Comportamiento de un *objeto* ejecuta cuando el objeto recibe un mensaje

# Ejemplos de Objetos

— — —

En un sistema de control aéreo como seria un Vuelo? qué datos tiene que responsabilidades posee?

# Clase Vs Instancia

# Programa Orientado a Objetos

---

- Generalmente se necesitan muchos objetos de un mismo tipo en un programa
  - Alumnos, Aviones, Cuentas de Banco, Empleados

# Clase

---

Molde para crear objetos con un determinado comportamiento y estado

Instancias



Clase





# Clase Vs Instancia

— — —

## CLASE

Un molde que define a las instancias

Un creador- una fábrica para crear objetos de un determinado tipo

## INSTANCIA

Es instanciada (**creada**) por una clase

Ocupa espacio en memoria

Mantiene un estado

Posee comportamiento

Conoce a qué clase pertenece

# Ejemplos de Clases Vs Instancias

---

**clase** Persona

**objetos** Juan, Pedro, Carlos

Los Objetos son las variables y son los que tienen los valores de una clase

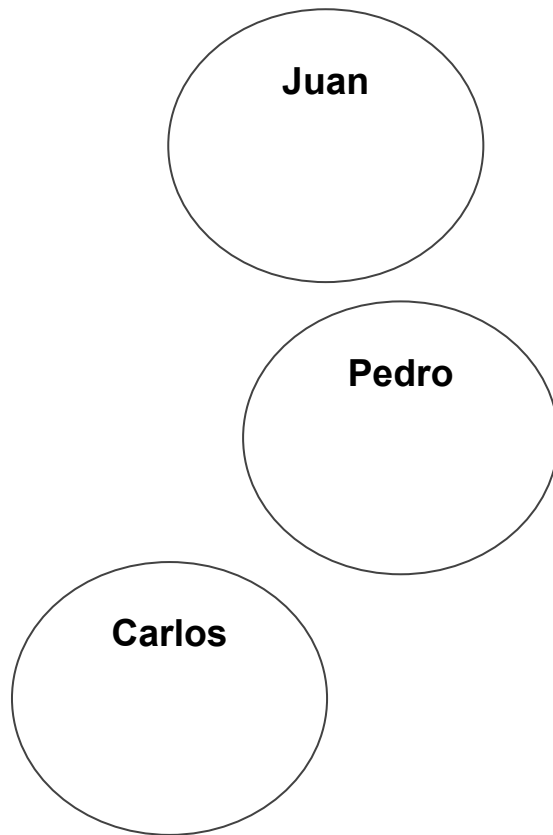
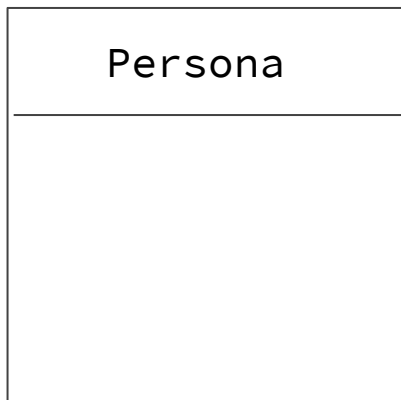
Normalmente las clases no son plurales! NO existen clases del estilo Personas, Empleados, Alumnos, ya que modelan a 1 Objeto no a una lista

# Ejemplos de Clases Vs Instancias

---

**clase** Persona

**objetos** Juan, Pedro, Carlos

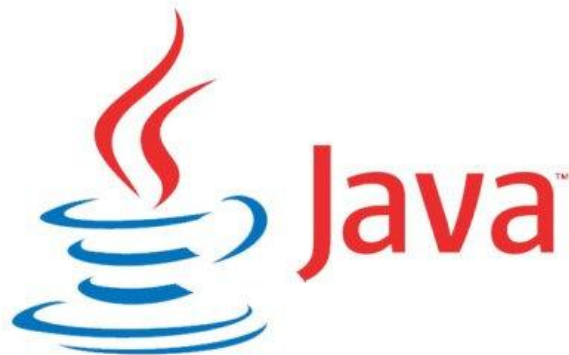


# Java y Programación Orientada a Objetos

# JAVA

---

El lenguaje **Java** es un lenguaje de **programación orientado a objetos**, que permite desarrollar aplicaciones para diferentes sistemas (es decir, aplicaciones **multiplataforma**)



# JAVA

---

¿Utilizar el lenguaje Java **garantiza** que tengamos una solución correctamente orientada a objetos (con todas sus bondades)?

**No**, el lenguaje Java brinda al desarrollador cierta flexibilidad que puede ser usada de forma libre y, a veces, poco orientado a objetos

# P00

---

¿Una solución orientada a objetos garantiza el éxito de la aplicación?

**No**, si ésta no cumple con los requisitos del cliente (por ejemplo, no es lo que el cliente pedía, es demasiado lenta, etc.)

# Lenguaje y Plataforma Java

---

- Java es el nombre de un **lenguaje** de programación orientado a objetos
- También es el nombre de una **plataforma de desarrollo** de aplicaciones (conjunto de herramientas que permiten construir y correr aplicaciones)
- Tanto lenguaje como plataforma están pensados para desarrollar aplicaciones **multiplataforma**, es decir, que se puedan ejecutar en Linux, Windows, Android, u otro sistema



# Plataforma Java

---

Es un conjunto de herramientas de software (programas y bibliotecas) que permiten desarrollar aplicaciones multiplataforma

A menudo, las aplicaciones para la plataforma Java son programadas usando el lenguaje de programación **Java** (aunque no es la única forma)

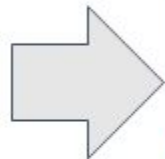
# Bytecode

---

El lenguaje intermedio de la plataforma Java

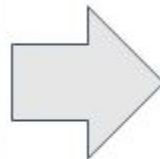
Código Fuente  
en lenguaje  
Java (es un  
archivo de  
texto legible)

ejemplo.java



Compilador Java

Por ej:  
javac.exe en  
Windows



Código en  
formato  
bytecode (no  
es texto  
fácilmente  
legible)

ejemplo.class

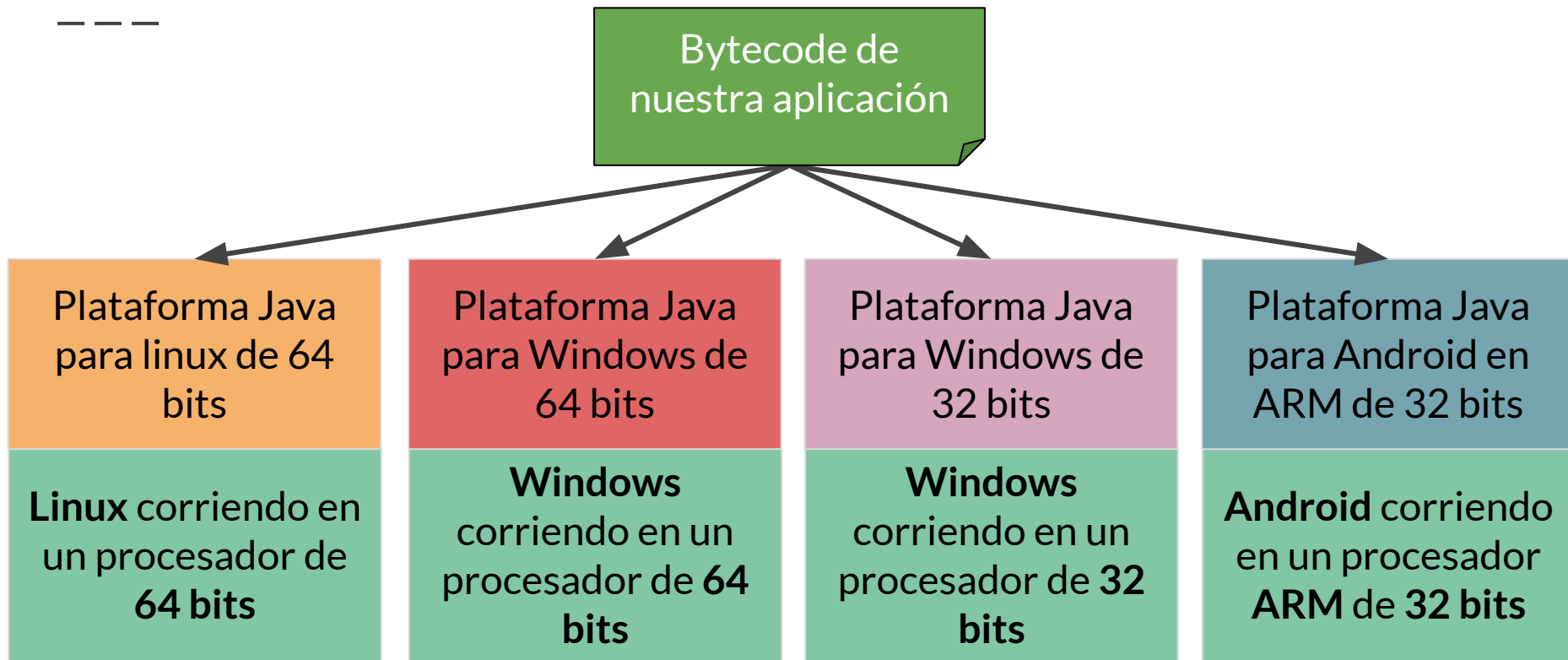
# ByteCode

---

En la plataforma Java existe un programa, llamado **Máquina Virtual** o Virtual Machine (VM), que traduce Bytecode al código de la máquina actual

**Bytecode** es un lenguaje similar al código de máquina, pero no depende del tipo de CPU

# Multiplataforma: Ejecución



# ByteCode

---

La desventaja de este esquema es que la plataforma Java (en especial la VM) tiene que funcionar en el sistema actual.

Por ej: Windows 32/64 bits, Linux, MacOS, etc.

Afortunadamente, existen versiones para los sistemas más populares.

# Plataforma Java Edición Estándar

---

Incluye todo lo necesario para ejecutar programas escritos en Java en una PC o un servidor

**JRE (Java Runtime Environment):** solo para ejecutar aplicaciones Java (lo debería tener instalado el usuario final de nuestra aplicación)

**JDK (Java Development Kit):** contiene el JRE y, además, software que usan los desarrolladores para monitorear y encontrar errores en los programas Java (lo deberíamos tener instalado nosotros)

# Máquina Virtual

— — —

- Es la parte central del JRE: Ejecuta la aplicaciones
- Es un programa, se ejecuta indicándole una aplicación Java, previamente compilada a bytecode
- Por ejemplo (en Windows):
  - Primero compilamos (traducimos) nuestra app a bytecode
    - **C:\> javac.exe MiProgramaEnJava.java**
  - Se crea un archivo MiProgramaJava.class, se indica a la maquina virtual usando el primer argumento
    - **C:\> java.exe MiProgramaEnJava**

# Máquina Virtual - Manejo de RAM

— — —

- En Java, a diferencia de otros lenguajes, el manejo de memoria RAM se supone automático
- La máquina virtual se encarga de buscar estructuras que no se usan más (por ejemplo, arreglos)
- La parte de la máquina virtual que hace esta “limpieza” se denomina Recolector de Basura, o Garbage Collector (GC)



# Ejemplo: Avión

---

El avión posee información, por ejemplo capacidad, modelo, estado (aterrizando, despegando, en Vuelo, en tierra)

También tiene responsabilidades

Cuales? Qué otros atributos podemos definir?

# Clase en Java

---

Primer paso, crear la clase

```
public class Avion {
```

```
....
```

```
}
```

Esto se guarda en un archivo **Avion.java**

*Convenciones: Las clases siempre comienzan con mayusculas*

# Clase en Java

---

Tenemos la clase, es decir tenemos el molde que crea la instancia.

```
Avion air314 = new Avion()
```



# Atributos

— — —

- Primero tenemos que establecer qué cosas va a tener nuestro avión: **Capacidad**. Un **nombre**.
  - Podría tener más cosas: color, modelo, motor, distancia recorrida, etc. **Son cosas que tienen que ver con el Avión**
- No tiene sentido poner un atributos que pertenecen otras entidades “precio” o “número de ticket de recital”  
Esto se suele denominar alta **cohesión** (los atributos del avión tienen relación con la entidad avión)

# Clase Avion - Atributos

— — —

```
public class Avion {  
    int capacidad;  
  
    String estado;  
  
    String color;  
  
    Fabrica marca;  
  
}
```

# Métodos

---

```
tipo_retorno nombreMetodo(tipo_arg1 arg1, ...){  
    return retorno;  
}
```

```
void nombreMetodo(tipo_arg1 arg1, ...){  
}
```

Nota: Normalmente atributos y métodos sigue notación **CamelCase**, comúnmente los métodos son verbos

# Clase Avión - Métodos

---

Supongamos que queremos cambiar el color del avión o preguntarle su color

```
public class Avion {  
    .... // los atributos anteriores  
    public void setColor(String unColor){  
        color = unColor;  
    }  
    public String getColor () {  
        return color;  
    }  
}
```

# Clase Vuelo

---

```
public class Vuelo {  
    Avion avionQueVuela;  
    Aeropuerto origen;  
    Aeropuerto destino;
```

...



# Clase Vuelo- Métodos

---

Supongamos que queremos saber a pista en la que aterriza el vuelo

```
public int getPistaAterrizaje() {  
    return (destino.getTorreControl()).getPista(avionQueVuela);  
    //          ( TorreControl )      .getPista(AVION)  
    //          (PISTA)  
}
```

# Definición de clases en Java- Métodos y atributos

---

**¿hay algún orden entre ellos?** No, los métodos, atributos, pueden ser declarados en cualquier orden

**Entonces...¿Es recomendable ponerlos en cualquier orden?**

**¡No!** Por convención de código, se recomienda colocar atributos **primero** y métodos **después**.

Nota: Todos los desarrolladores Java suelen adherir a las convenciones, para facilitar compartir el código en equipos de desarrollo.

# Constructor de un Objeto

---

El constructor puede verse como el método invocado para la construcción de un objeto. Cuando hacemos:

```
Avion a234 = new Avion()
```

Invocamos al constructor sin argumentos de la clase avión.

El constructor es quien se encarga de crear el objetos y setear los valores iniciales del mismo

# Constructor de un Objeto JAVA

---

```
public class Avion {  
    int capacidad;  
    String estado;  
    String color;  
  
    public Avion() {    // CONSTRUCTOR SIN ARGUMENTOS  
        capacidad = 100;    //VALORES POR DEFECTO  
        estado = "en Tierra";  
        color = "blanco";  
    }  
    //CONTINUA
```

# Constructor de un Objeto JAVA

---

El constructor puede poseer parámetros. UN objeto puede tener múltiples constructores

```
//CONTINUA CLASE AVION
```

```
public Avion (int cap, String col){//DOS PARAMETROS  
    capacidad = cap;  
    color = col;  
    estado = "en Tierra"; //DEFECTO  
}
```

# Constructor de un Objeto JAVA

---

UN objeto puede tener múltiples constructores todos ellos deben tener una **signatura** diferente. Cuando se llama a un constructor es cuando se decide cual se invoca.

```
Avion a234 = new Avion(); //100, blanco
```

```
Avion a235 = new Avion(25, "Rojo");
```