Comportamiento más sofisticado

Uso de clases de biblioteca para implementar algunas funcionalidades más avanzadas

Conceptos principales a ser cubiertos

- Uso de bibliotecas de clases.
- Lectura de documentación.
- Escritura de documentación.

La biblioteca de clases de Java

- Miles de clases.
- Decenas de miles de métodos.
- Muchas clases útiles que hacen la vida más fácil.
- Un programador Java competente debe ser capaz de trabajar con bibliotecas.

Para trabajar con la biblioteca

Debería:

- Conocer por su nombre algunas clases importantes;
- Saber como encontrar otras clases.

Recordar:

- Sólo es necesario conocer la interfaz, no la implementación.

Un Sistema de Soporte Técnico

- Un sistema de diálogo textual.
- Idea baseda en *'Eliza'* de Joseph Weizenbaum (MIT, 1960s).
- *Explore…*

Ej.: 5.1

La estructura del ciclo principal

```
boolean terminado = false;

while(!terminado) {

    hacer algo ...

    if(condición de salida) {
        terminado = true;
    }
    else {
        hacer algo más ...
    }
}
```

Cuerpo del ciclo principal

```
String entrada = lector.getEntrada();
...
String respuesta =
   contestador.generarRespuesta();
System.out.println(respuesta);
```

La condición de salida

```
String entrada = lector.getEntrada();

if(entrada.startsWith("bye")) {
    terminado = true;
}
```

- ¿De dónde viene '`startsWith`'?
- ¿Qué es? ¿Qué hace?
- ¿Cómo podemos encontrarlo?

Lectura de la documentación de una clase

- Documentación de las bibliotecas de Java en formato HTML.
- Legibles en un navegador.
- Class API: *Application Programmers' Interface*
- Descripción de las Interfaces para todas las clases de la biblioteca.

Ej.: 5.2, 5.3, 5.4, 5.5, 5.6

Interfaz vs. Implementación

La documentación incluye

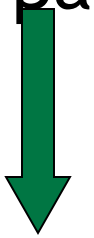
- el nombre de la clase;
- una descripción general de la clase;
- una lista de constructores y métodos;
- Los valores de retorno y los parámetros de los constructores y los métodos;
- una descripción del propósito de cada constructor y método.

la *interfaz* de la clase

Interfaz vs. Implementación

*La documentación **no** incluye*

- campos *private* (la mayoría de los campos son *private*);
- métodos *private*;
- Los cuerpos para cada método (código fuente).



la *implementación* de la clase

Usando clases de la biblioteca

- Las clases de la biblioteca deben importarse usando la instrucción **`import`** (excepto para clases de **`java.lang`**).
- Pueden usarse como clases del proyecto en desarrollo.

Ej.: 5.7, 5.8, 5.9

Paquetes y la instrucción *import*

- Las clases están organizadas en paquetes.
- Se puede importar una sola clase:

```
import java.util.ArrayList;
```

- Se puede importar todo el paquete:

```
import java.util.*;
```

Comentario: igualdad de *String*

verifica identidad

```
if(entrada == "chau") {
    ...
}
```

verifica igualdad

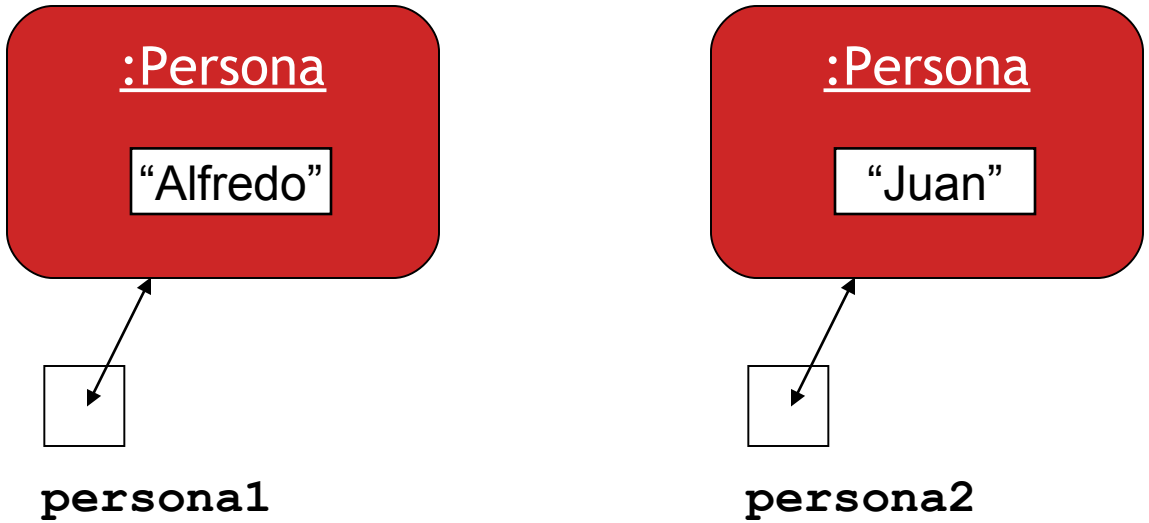
```
if(entrada.equals("chau")) {
    ...
}
```

Los *Strings* siempre deben compararse con **`.equals`**

Ej.: 5.10, 5.11

Identidad vs. Igualdad (1)

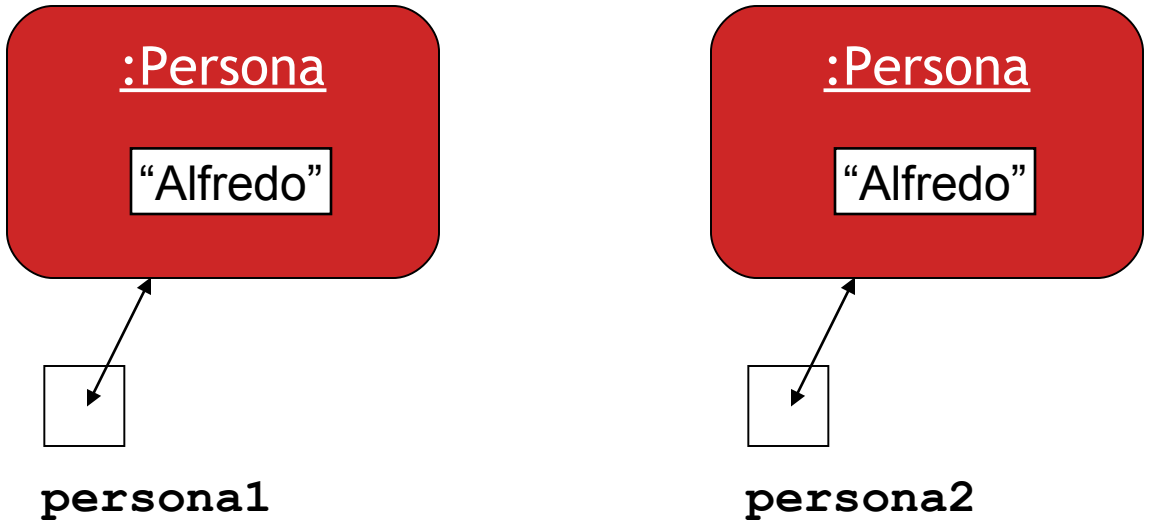
Otros objetos (no-String):



¿ persona1 == persona2 ?

Identidad vs. Igualdad (2)

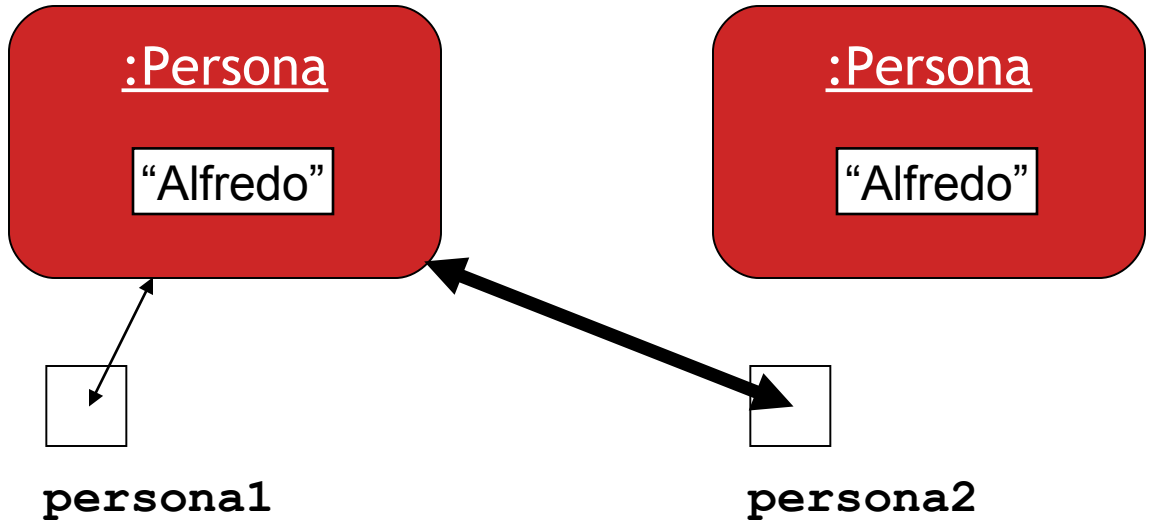
Otros objetos (no-String):




```
¿ persona1 == persona2 ?
```

Identidad vs. Igualdad (3)

Otros objetos (no-String):

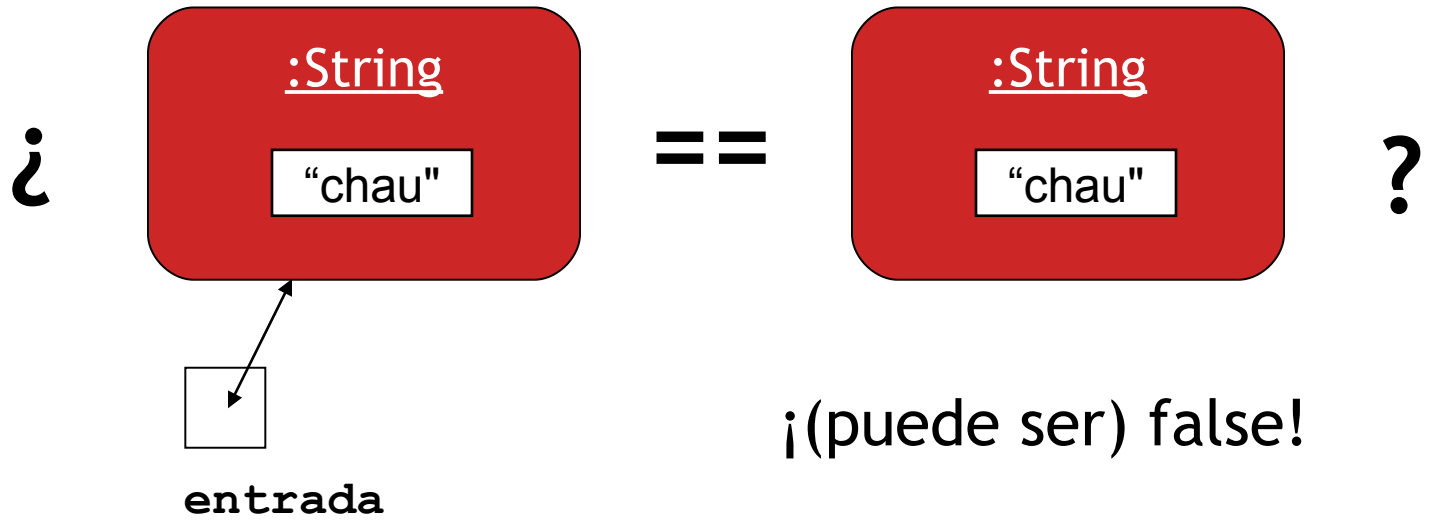


¿ persona1 == persona2 ?

Identidad vs. Igualdad (*Strings* - 1)

```
String entrada = lector.getEntada();
if(entrada == "chau") {
    ...
}
```

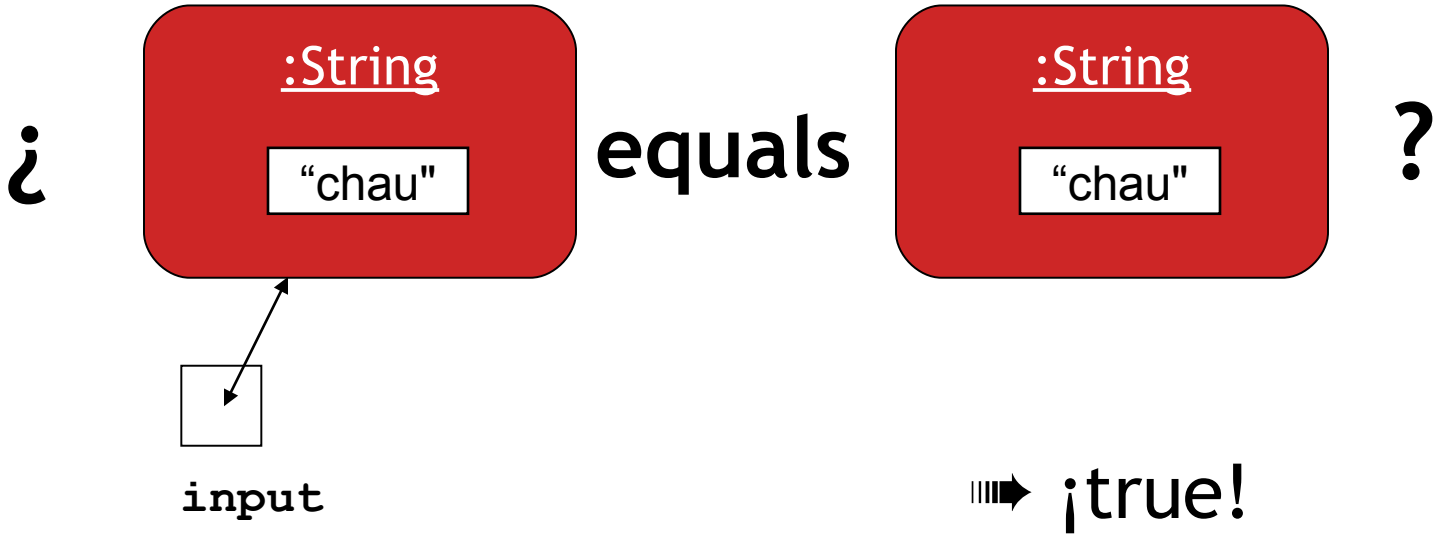
== verifica identidad



Identidad vs. Igualdad (*Strings* - 2)

```
String entrada = lector.getEntrada();
if(entrada.equals("chau")) {
    ...
}
```

equals verifica igualdad



Comportamiento Aleatorio

- La biblioteca de clases **`Random`** puede usarse para generar números pseudoaleatorios

```
import java.util.Random;
...
Random generadorDeAzar = new Random();
...
int indice1 = generadorDeAzar.nextInt();
int indice2 = generadorDeAzar.nextInt(100);
```

Ej.: 5.12, 5.13, 5.14, 5.15, 5.16, 5.17, 5.18, 5.19, 5.20, 5.21, 5.22

Generación de respuestas aleatorias

```
public Contestador()
{
    generadorDeAzar = new Random();
    respuestas = new ArrayList<String>();
    rellenarRespuestas();
}

public String generarRespuesta()
{
    int indice =
              generadorDeAzar.nextInt(respuestas.size());
    return respuestas.get(indice);
}

public void rellenarRespuestas()
{
    ...
}
```

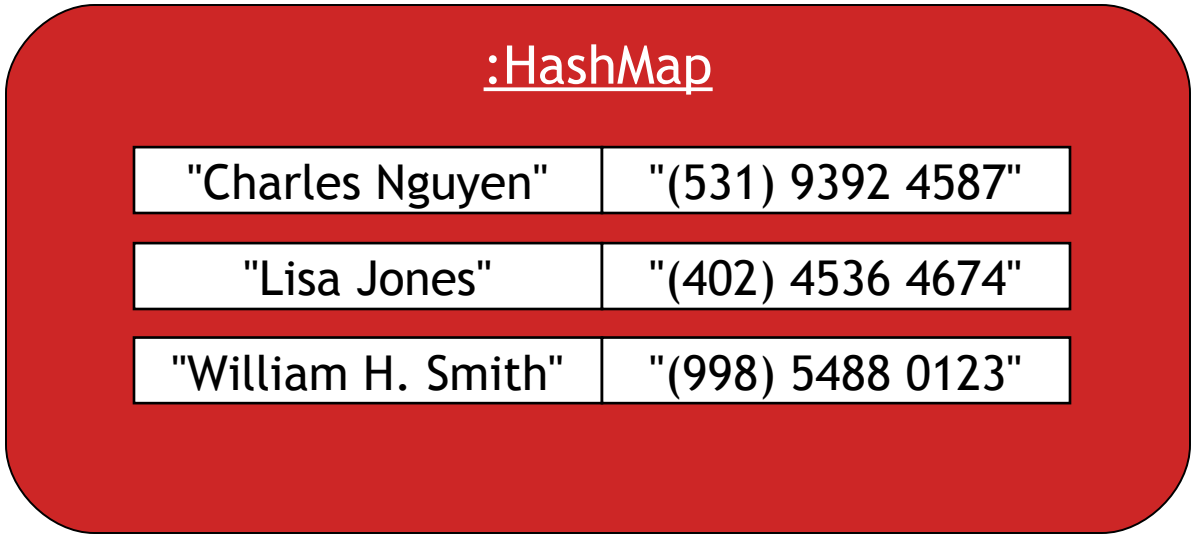
Mapas

- Los Mapas son colecciones que contienen pares de valores.
- Un Par consiste de una llave y un valor.
- La búsqueda trabaja suministrando una llave y recuperando un valor.
- Un ejemplo: una agenda.

Ej.: 5.23, 5.24

Uso de mapas

- Un mapa con *Strings* como llaves y valores asociados



Ej.: 5.25, 5.26, 5.27, 5.28, 5.29, 5.30

Uso de mapas

```
HashMap <String, String> agenda =
                new HashMap<String, String>();

agenda.put("Charles Nguyen", "(531) 9392 4587");
agenda.put("Lisa Jones", "(402) 4536 4674");
agenda.put("William H. Smith", "(998) 5488 0123");


String numeroTelefonico = agenda.get("Lisa Jones");
System.out.println(numeroTeleonico);
```

Ej.: 5.31

Uso de conjuntos

```
import java.util.HashSet;
import java.util.Iterator;
...
HashSet<String> miConjunto = new
   HashSet<String>();

miConjunto.add("uno");
miConjunto.add("dos");
miConjunto.add("tres");

Iterator<String> it = miConjunto.iterator();
while(it.hasNext()) {
     llamar it.next() para obtener el siguiente
     objeto
     hacer algo con ese objeto
}
```

Compare esto con el código de ArrayList!

Ej.: 5.32

Descomponiendo Cadenas

```
public HashSet<String> getEntrada()
{
    System.out.print("> ");
    String linea =
        lector.lineaSiguiente().trim().toLowerCase();

    String[] arregloDePalabras = linea.split();
    HashSet<String> palabras =
            new HashSet<String>();

    for(String palabra : arregloDePalabras) {
        palabras.add(palabra);
    }
    return palabras;
}
```

Ej.: 5.33, 5.34, 5.35, 5.36, 5.37, 5.38, 5.39, 5.40, 5.41, 5.42

Documentación de clases

- Sus propias clases deben documentarse en la misma forma que lo están las bibliotecas de clases.
- Otras personas deben ser capaces de usar sus clases sin leer la implementación.
- ¡Haga de su clase una “clase de biblioteca”!

Elementos de documentación

La documentación para una clase debería incluir:

- El nombre de la clase.
- Un comentario que describa el propósito general y las características de la clase.
- Un número de versión.
- Los nombres de los autores.
- La documentación para cada constructor y cada método.

Elementos de documentación

La documentación para cada constructor y cada método debería incluir:

- El nombre del método.
- El tipo del valor retornado.
- Los nombres y tipos de los parámetros.
- Un descripción del propósito y función del método.
- Una descripción de cada parámetro.
- Una descripción del valor retornado.

javadoc

Comentario sobre la clase:

```
/**
 * La clase Contestador representa un objeto
 * generador de respuestas. Es usado para gener una
 * respuesta automática.
 *
 * @author     Michael Kölling and David J. Barnes
 * @version    1.0  (30.Mar.2006)
 */
```

javadoc

Comentario para un método:

```
/**
 * Lee una línea de texto de la entrada standard
 *(la terminal de texto), y la retorna como un
 * conjunto de palabras.
 *
 * @param  prompt  Un prompt a imprimir en la pantalla.
 * @return Un conjunto de cadenas, en el que cada String
 *         es uno de las palabras que escribió el usuario
 */
public HashSet<String> getInput(String prompt)
{
    ...
}
```

Ej.: 5.43, 5.44, 5.45, 5.46

Public vs. *Private*

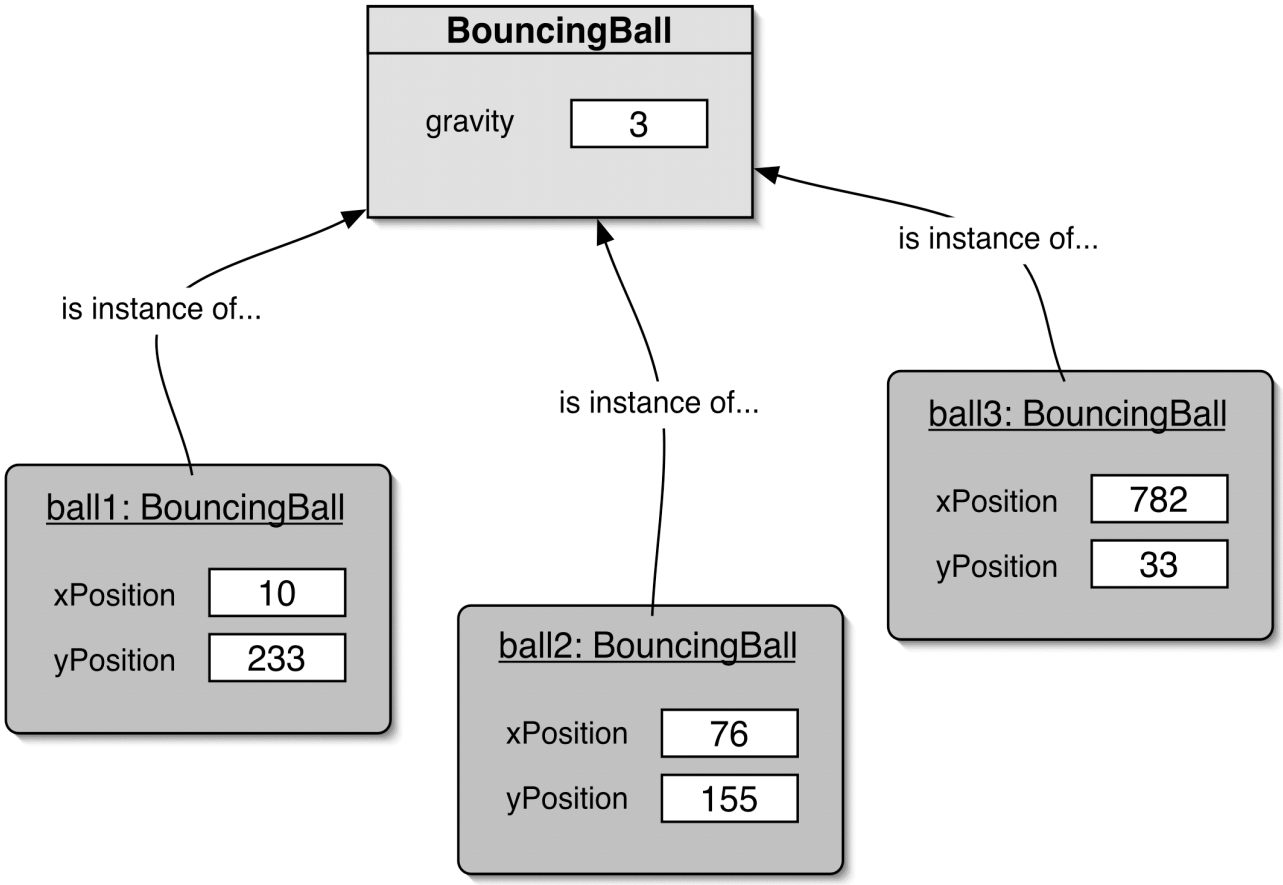
- Los atributos *public* (campos, constructores y métodos) son accesibles desde otras clases.
- Los campos no deberían ser *public*.
- Los atributos *private* sólo son accesibles dentro de la misma clase.
- Sólo deberían ser *public* los métodos que son para el uso de otras clases.

Ocultamiento de información

- Los datos que pertenecen a un objeto están ocultos para otros objetos.
- Se debe saber qué puede hacer un objeto y no cómo lo hace.
- El ocultamiento de la información incrementa el nivel de *independencia*.
- La independencia de los módulos es importante en los grandes sistemas y en su mantenimiento.

Ej.: 5.47, 5.48, 5.49, 5.50, 5.51, 5.52, 5.53, 5.54, 5.55, 5.56

Variables de clase



Constantes

```
private static final int gravity = 3;
```

- **`private`**: modificador de acceso, como es usual.
- **`static`**: variable de clase
- **`final`**: constante

Ej.: 5.57, 5.58, 5.59, 5.60

Repaso

- Java tiene una extensa biblioteca de clases.
- Un buen programador debe estar familiarizado con la biblioteca de clases.
- La documentación nos dice lo que necesitamos saber acerca de una clase (*interface*).
- La implementación está oculta (ocultamiento de la información).
- Las clases se documentan de tal manera que la interfaz pueda ser leída por sí misma (clase, comentario de método).

Ej.: 5.61, 5.62, 5.63