CURSOS INTERSEMESTRALES

PROTECO

# Manipulación de Datos

Java Básico 2019-1

# Recordatorio

Animal
int patas
Animal perro
perro.caminar()
perro.nombre

Abstracción
Encapsulamiento
Herencia
Polimorfismo

# Nombre de Variable

- El primer carácter tiene que ser una letra (a, b, c...), el carácter guión bajo (_) o el dólar ($).

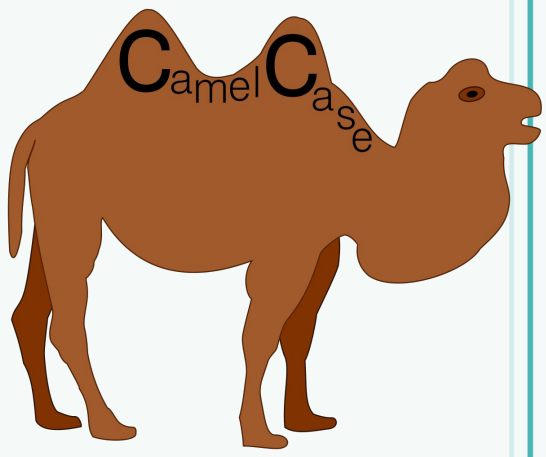
- Los siguientes caracteres pueden ser letras, el guión bajo, el dólar o dígitos (0, 1, 2...).

```
_123
123-123
123$123
$123
abc
```

PROTECO

# CamelCase

- UpperCamelCase, cuando la primera letra de cada una de las palabras es mayúscula. Ejemplo: *EjemploDeUpperCamelCase*.
- lowerCamelCase, igual que la anterior con la excepción de que la primera letra es minúscula. Ejemplo: *ejemploDeLowerCamelCase*.




PROTECO

# Paquete

Un paquete es un conjunto de clases, interfaces, enumeraciones, excepciones, errores, anotaciones y otros paquetes que cumplen un determinado objetivo.

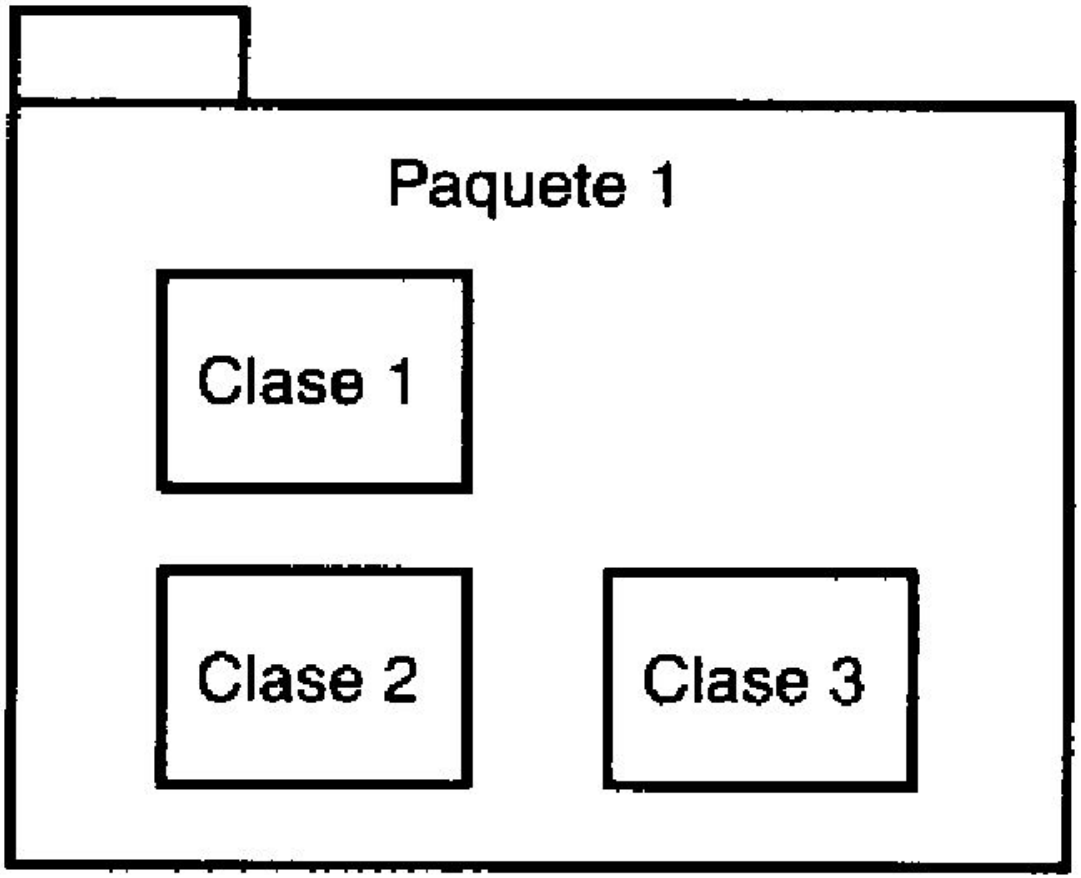
Así como las clases siguen la notación UpperCammelCase, los paquetes siguen la siguiente notación:

Todo el nombre va con minúsculas
Para separar palabras se usa puntos.

Por ejemplo: este.es.un.ejemplo

PROTECO

# Representación




PROTECO

# Paquetes Importantes

**java.lang:** Posee todas las clases necesarias para el funcionamiento de Java, no se requiere importar ninguna clase de este paquete puesto que todas las clases del paquete se importan por defecto. Contiene clases como String, Clases Envolventes, Clase Math, Object

**java.util:** Posee clases con funciones útiles (aunque no indispensables) para el funcionamiento de Java. Cómo Scanner, las ArrayList.

PROTECO

# Paquete Importantes

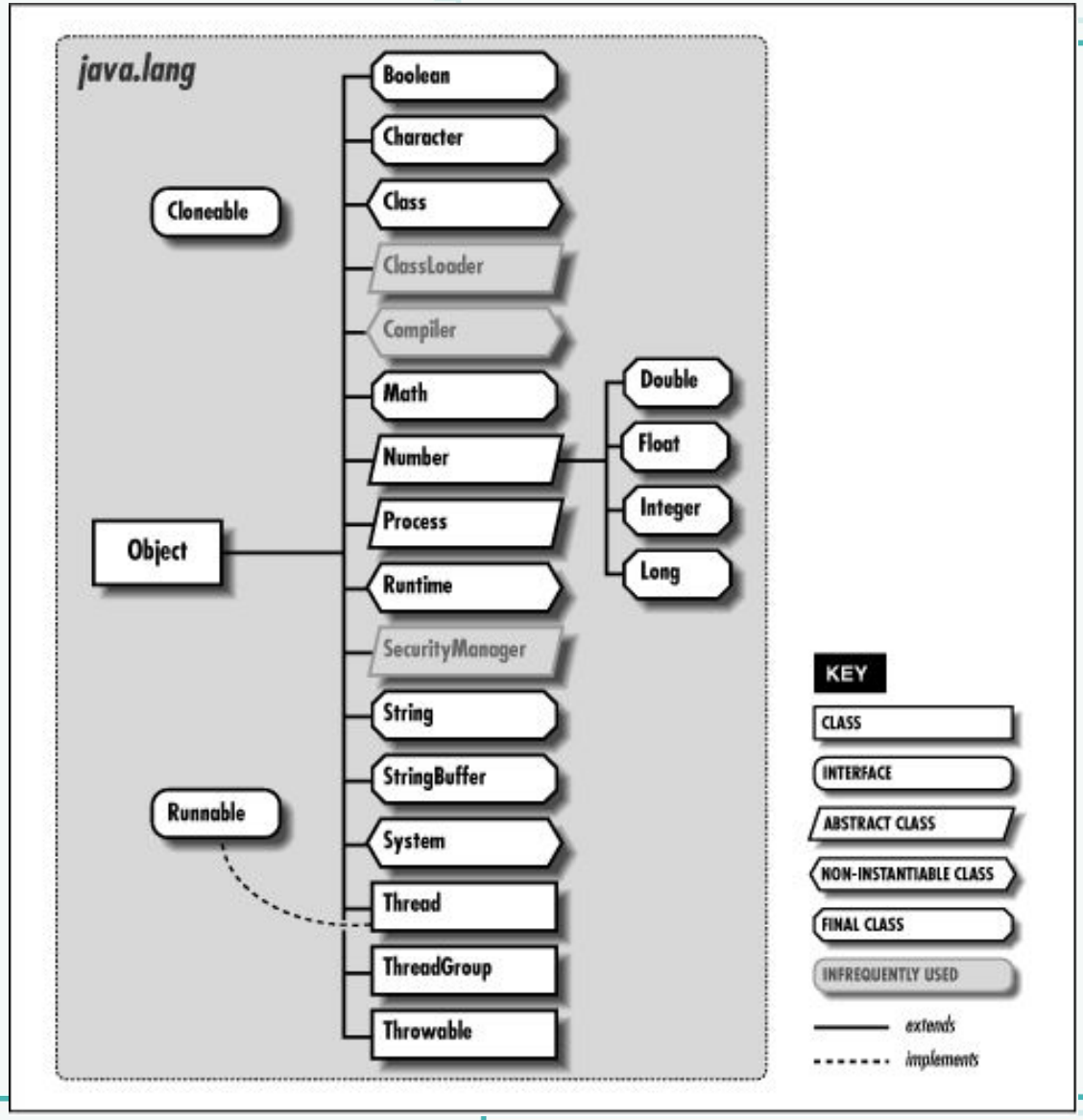
**javax.swing:** Posee clases para realizar interfaces gráficas.

**java.math:** Posee clases que permiten trabajar con cualquier entero y con cualquier decimal.

**java.io:** Posee clases que continen funciones de entrada y salida.

PROTECO

# Paquete Lang




PROTECO

# Paquetes

Para importar una clase de un paquete se realiza de la siguiente forma (por ejem: importar la clase ArrayList de java.util):

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

Para importar todas las clases de un paquete se realiza de la siguiente forma (por ejem: importar todas las clases de java.util):

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

PROTECO

No es recomendable importar todas las clases de un paquete, porque hace al compilador un poco más lento y genera errores si tenemos dos o más clases con el mismo nombre.


PROTECO

# Creación de Paquetes

Podemos crear paquetes usando la palabra reservada `package nombre.del.paquete;`

De esta forma indicamos que estamos en un paquete llamado "nombre.del.paquete"

Todo lo que queramos usar (clases, interfaces, excepciones, etc.) que no se encuentre en nuestro paquete hay que importarlo.

Si no definimos un paquete se crea un paquete llamado `default`.

PROTECO

# Encapsulamiento

Encapsulamiento consiste en la ocultación de los atributos (campos) para que las demás clases (o determinadas) no puedan acceder directamente a ellos. En Java el encapsulamiento se representa con modificadores de acceso.

PROTECO

# Modificadores de Acceso

| Visibilidad | Public | Protected | Default | Private |
|---|---|---|---|---|
| Desde la misma Clase | SI | SI | SI | SI |
| Desde cualquier Clase del mismo Paquete | SI | SI | SI | NO |
| Desde una SubClase del mismo Paquete | SI | SI | SI | NO |
| Desde una SubClase fuera del mismo Paquete | SI | SI, a través de la herencia | NO | NO |
| Desde cualquier Clase fuera del Paquete | SI | NO | NO | NO |

PROTECO

# Modificadores de Acceso

| Visibilidad | Public | Protected | Default | Private |
|---|---|---|---|---|
| Desde la misma Clase | SI | SI | SI | SI |
| Desde cualquier Clase del mismo Paquete | SI | SI | SI | NO |
| Desde cualquier Clase fuera del Paquete | SI | NO | NO | NO |

PROTECO

# Notas

- Subclase hace referencia a conceptos de herencia (se ve posteriormente).
- Los modificadores de acceso se pueden aplicar a métodos, atributos, clases, interfaces, enumeraciones y excepciones.

PROTECO

# Modificadores

Un modificador permite realizar un cambio en una variable, método o clase.

**final** está relacionado con la incapacidad para modificar una variable, método o clase.

**static** está relacionado con la no instancia de las cosas, mejor dicho, en lugar de la pertenencia al objeto pertenece a la clase.

PROTECO

# Modificador: final

**Final** aplicado a un atributo, permite establecer dicho atributo como constante. Una constante debe poseer su nombre en mayúscula, y de preferencia poseer el modificador de acceso public.

**Final** aplicado a un método, establece que dicho método no puede ser sobreescrito en una clase hija

**Final** aplicado a una clase, establece que ninguna clase puede heredar de ella.

PROTECO

# Modificador: static

**Static** aplicado a un atributo, permite que esta sea igual para cualquier instancia de la clase y puede ser accedida (si los modificadores de acceso lo permiten) por otra clase sin la necesidad de crear una instancia.

**Static** aplicado a un método hace que ese método se pueda usar sin instanciar la clase.

**static** se puede aplicar a una clase, pero tiene que ser una clase anidada.

PROTECO

# Mutabilidad

Mutabilidad hace referencia a el alteramiento de un objeto. Existen dos tipos de objetos en cuanto a su mutabilidad:

* **Objetos Mutables:** Objetos que si se pasan como parámetro de una función ó si se utiliza alguno de sus métodos afectan su contenido.

* **Objetos no Mutables:** Objetos que si se pasan como parámetro de una función ó si se utiliza alguno de sus métodos no pueden afectar su contenido.

PROTECO

# Compilación en java

En java para compilar usamos el mando javac, este posee opciones adicionales:

javac [opciones adicionales] [nombre de archivo .java]

las opciones adicionales más importantes son:

-d [ubicación]  esta opción (bandera) nos permite establecer donde queremos guardar los .class

-cp [ubicacion] esta bandera nos permite establecer dónde se encuentra algunos .class que necesitamos

PROTECO

# Compilación en Java

Por ejemplo, si deseamos compilar la class Ejemplo.java y queremos guardar la clase en una carpeta superior:

javac -d .. Ejemplo.java

Si queremos compilar todos los .java de una carpeta podemos usar el asterisco:

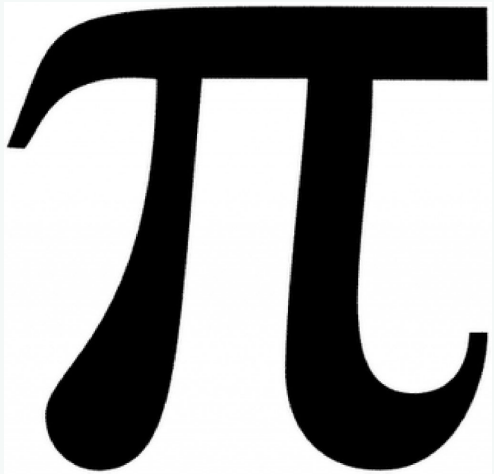
javac *.java

El comando java que sirve para ejecutar un programa posee tambien las banderas -d y -cp.

PROTECO

# Clase Math

La clase Math es una clase de utilidad (que posee métodos y atributos relacionados con matemáticas. Esta clase se encuentra en java.lang, así que no hay que importarla.



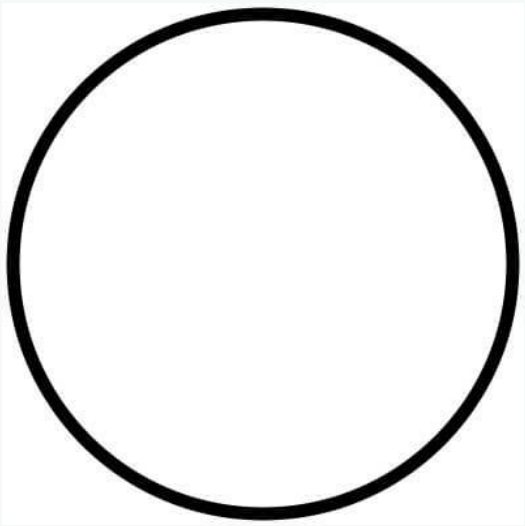
PROTECO

# Constantes

Posee dos constantes:

* **e:** La base del logaritmo natural

* **pi:** La relación entre la longitud de una circunferencia y su diámetro.



PROTECO

# Métodos

La clase Math incluye métodos relacionados con matemáticas, los cuales abarcan:

- Funciones trigonométricas
- Funciones exponencial y logarítmica
- Potencia y Raíz Cuadrada
- Redondeos de decimales
- Números Aleatorios
- Maximos y Minimos
- Valor Absoluto

PROTECO

# Clase String

La clase String pertenece a java.lang por lo que no hay que importarla, la clase String permite representar texto en un programa en Java, aunque no es la única clase que permite esto es la más intuitiva y más usada.

Cabe destacar que los objetos pertenecientes a String son inmutables, es decir, una vez establecidos su valor estos no cambian.

PROTECO

# Operadores Relacionales

| Operadores Relacionales | |
|---|---|
| > | Mayor que |
| < | Menor que |
| >= | Mayor o igual que |
| <= | Menor o igual que |
| != | Distinto |
| == | Igual |

PROTECO

# Clase String

Cualquier función que se le aplique a un objeto de tipo String crea un nuevo objeto como resultado de la acción.

La clase String es un poco compleja en el uso de constructores, dado el siguiente código:

```
String cadena = "Texto";
String cadena2 = "Texto";
String cadena3 = new String("Texto");
```

PROTECO

# Clase String

Si recordamos lo que el operador == hace, nos regresa true si a == b, a y b se encuentran en la misma dirección de memoria. El resultado de ese operador en las variables es:

* cadena == cadena2  -> True
* cadena2 == cadena3  -> False
* cadena == cadena3  -> False

PROTECO

# Clase String

Normalmente la máquina virtual de java hace que los objetos que tengan la misma cadena en su interior apunten a la misma dirección de memoria, pero al momento de usar el constructor se fuerza a apuntar a una nueva dirección de memoria a pesar que internamente posean la misma información, es una mala práctica.

En generar, para comparar objetos hay que utilizar el método .equals() en vez del operador == porque este genera fallas en algunas ocasiones.

PROTECO

# Caracteres Especiales

| Secuencia de escape | Valor |
|---|---|
| `\b` | Retroceso o *backspace* (equivalente a `\u0008`) |
| `\t` | Tabulador (equivalente a `\u0009`) |
| `\n` | Nueva línea (equivalente a `\u000A`) |
| `\f` | Salto de página (equivalente a `\u000C`) |
| `\r` | Retorno de carro (equivalente a `\u000D`) |
| `\"` | Doble comilla (equivalente a `\u0022`) |
| `\'` | Comilla simple (equivalente a `\u0027`) |

PROTECO

# Clase StringBuilder

StringBuilder es una clase con la que se puede trabajar con cadenas, StringBuilder no es tan intuitivo como la clase String pero StringBuilder permite generar objetos de tipo mutable.

StringBuilder es más rápido que la clase String pero es menos intuitivo.

PROTECO

# Clases Envolventes

Las clases envolventes son clases que permiten funcionalidades extras a los tipos primitivos, estas funcionalidades son:

- Posibilidad de comportarse como un objeto en una determinada circunstancia (Autoboxing).
- Caster de un String a los tipos de datos primitivos.
- Contener métodos y atributos que dan una funcionalidad extra a los mismos.

PROTECO

| TIPO | ESPACIO | RANGO |
|---|---|---|
| long | 64 bits | -9,233372,036854,775808 a 9,233372,036854,775807 |
| int | 32 bits | -2147,483648 a 2147,483647 |
| short | 16 bits | -32768 a 32767 |
| byte | 8 bits | -128 a 127 |
| boolean | 1 bit | true (verdadero), false (falso) |
| char | 16 bits | unicode 0 a 65535 |
| double | 64 bits | ~~-1.~~7976931348623157e~~+308~~ (15 digitos significativos) |
| float | 32 bits | ~~-3.~~40282347e~~+38~~ (7 digitos significativos) |
| void | | |

PROTECO

# Tabla de Clases Envolventes

| Primitive Data Types | Wrapper Classes |
|---|---|
| int | Integer |
| short | Short |
| long | Long |
| byte | Byte |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

PROTECO

# Autoboxing

Convertir de un tipo primitivo a un objeto de su clase envolvente. Se realiza cuando:

* Pasado como parámetro a un método que espera como parámetro un objeto proveniente de una clase envolvente.

* Asignado a una variable que corresponda a una clase envolvente

# Unboxing

Convertir de un objeto de una clase envolvente a su respectivo tipo primitivo. Java lo realiza cuando:

- Cuando el objeto es pasado como parametro a un método que espera su correspondiente tipo primitivo.
- Cuando el objeto se asigna a una variable correspondiente a su tipo primitivo.

# Métodos de Clases Envolventes

Las Clases Envolventes poseen métodos relacionados con su tipo de datos.

Por ejemplo: las clases envolventes provenientes de números poseen los valores de máximos y mínimos

PROTECO

# Enumeración

Las enumeraciones son un tipo especial en java que se asemejan a una clase que establecen un conjunto predefinido de variables constantes, cada variable proveniente de una determinada enumeración debe valer un valor predefinido por esta.

Se debe usar enum **siempre** que se posea un conjunto fijo de constantes por su velocidad. Todos los enum heredan de java.lang.Enum por y poseen el método values() que regresa un arreglo de todas las constantes del enum.

PROTECO

# Enum

A un valor del enum se le pueden aplicar los siguientes métodos:

| Tipo | Metodo | Descripcion |
|---|---|---|
| String | name() | Regresa el nombre del valor del enum |
| int | ordinal() | Posicion de la constante dentro del enum |
| int | compareTo() | Compara dos constantes del enum segun su posicion |

PROTECO

# Creación Enum

Un enum se crea de forma similar a una clase:

```
public enum Ejemplo{
  VALOR1, VALOR2, VALOR3
}
```

- Los nombres de los valores van con mayúscula porque son constantes
- A un enum solo se le puede aplicar los modificadores de acceso private y default
- Un enum puede tener métodos, atributos y constructor.
- Se incluyen ejemplos en la carpeta "Enumeraciones"

PROTECO