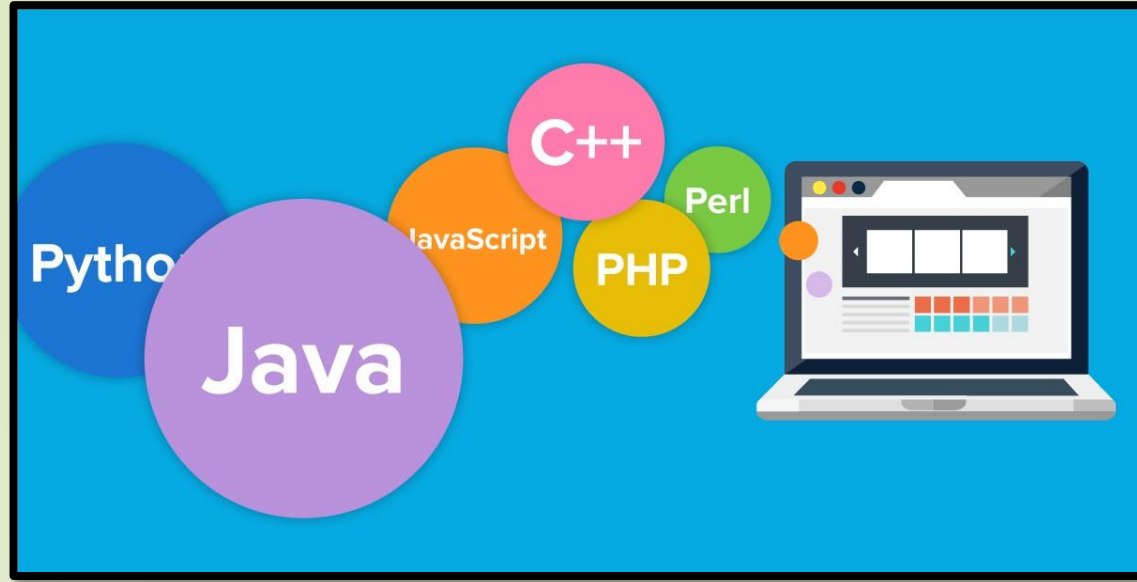




Desarrollo de Aplicaciones Web y Multiplataforma: Programación

DOCENTE: Daniel López Lozano





Tema 4.

Programación Orientada a Objetos. Paquetes Clases y Métodos

Índice de contenidos

- ❑ **Utilización de lenguajes orientados a objetos.**
- ❑ **Paquetes, clases y objetos.**
- ❑ **Paquete java.lang**
 - ✓ Clase Math.
 - ✓ Clase String.
- ❑ **Paquete java.util**
 - ✓ Clase Random.
 - ✓ Clase Date.
 - ✓ Clase Arrays.
- ❑ **Clases, métodos y modularización.**

¿Qué es la POO?



Características

Capacidades

- ❑ En este primer tema dedicado a la Programación Orientada a Objeto vamos a empezar a utilizar el concepto de clase para reutilizar código.
- ❑ En este caso vamos a modularizar nuestro código en bloques, cada uno de ellos con un nombre
- ❑ Dichos bloques vamos a poder invocarlos las veces que necesitemos desde cualquier parte de nuestro código fuente.
- ❑ Esto nos aporta la capacidad de poder organizar nuestro código.

- ❑ Los **métodos** son el mecanismo de Java para **modularizar mediante objetos** nuestro programa.
- ❑ Ahora vamos a centrarnos en comprender **el concepto de función** aplicable a cualquier lenguaje de programación y que son llamados **métodos** en los lenguajes orientados a objetos.
- ❑ Pensemos durante un momento que estamos haciendo un programa de **gestión de empleados**.
- ❑ Mientras hacemos ese programa, imaginemos también lo siguiente:

- ❑ Necesitamos calcular el **sueldo medio de dos departamentos** de una empresa en nuestro programa.
- ❑ Hasta ahora, el único pensamiento factible, era pensar que para poder hacer esto, **era necesario** que fuésemos al primer sitio del programa donde nos hiciese falta calcular el **sueldo medio del primer departamento** y realizar el código necesario.
- ❑ Tenemos que pensar que la forma de proceder es exactamente igual para los departamentos pero con **otras datos y variables que no se pisen**.

- ❑ Seguramente copiaríamos ese código que habíamos escrito, lo pegaríamos y adaptaríamos en el segundo sitio donde nos hiciese falta el sueldo medio del otro departamento.
- ❑ Esta problemática se podría acentuar si nos hiciese falta calcularlo para cincuenta departamentos.
- ❑ Precisamente esa es la esencia de la aparición de las funciones(métodos) ya que escribiremos como calcular la media una sola vez y utilizarlo todas la veces que queramos sin necesidad de copiar y pegar.

Declaración del Método:

[acceso] tipo_devuelto nombre (tipo arg1, tipo arg2...)
{...}

Donde:

- ❑ **Acceso:** Por ahora siempre public. Define quien puede utilizar este método.
- ❑ **Tipo devuelto:** Al finalizar el método, este puede devolver un valor, este puede ser un tipo primitivo o un objeto de una clase, o nada (void).
- ❑ **Nombre:** Es el identificador del método.
- ❑ **Argumentos:** Un método puede recibir ninguno, uno o más argumentos, estos se declaran indicando su tipo su identificador y si son más de uno se separan por comas.

¿Por qué usar métodos?

```
public class Principal{  
    public static void main(String[] args) {  
        int n1, n2, primero, segundo;  
        n1 = sc.nextInt();  
        n2 = sc.nextInt();  
        if(n1 < n2){  
            primero = n1;  
        }else{  
            primero = n2;  
        }  
        System.out.println("Su primer número es "+ primero);  
        n1 = sc.nextInt();  
        n2 = sc.nextInt();  
        if(n1 < n2){  
            segundo = n1;  
        }else{  
            segundo = n2;  
        }  
        System.out.println("Su segundo número es "+ segundo);  
    }  
}
```

Usando funciones mejoramos el código

```
public class Matematicas{  
    public static int menor(int a,int b){  
        int resultado;  
        if(a<b){  
            resultado=a;  
        }else{  
            resultado=b;  
        }  
        return resultado;  
    }  
  
    public static int mayor(int a,int b){  
        int resultado;  
        if(a<b){  
            resultado=b;  
        }else{  
            resultado=a;  
        }  
  
        return resultado;  
    }  
    ...  
}
```

Usando funciones mejoramos el código

```
public class Principal{  
    public static void main(String[] args){  
        Scanner teclado=new Scanner(System.in);  
        int n1,n2,primero,segundo;  
  
        System.out.println("Dime dos numeros: ");  
        n1=teclado.nextInt();  
        n2=teclado.nextInt();  
        primero=Matematicas.menor(n1,n2);//n1 sera a y n2 sera b  
        System.out.println("El menor es : "+primero);  
  
        System.out.println("Dime otros dos numeros: ");  
        n1=teclado.nextInt();  
        n2=teclado.nextInt();  
        segundo=Matematicas.menor(n1,n2);//n1 sera a y n2 sera b  
        System.out.println("El menor es : "+segundo);  
    }  
}
```

Otra forma de hacer lo mismo

```
public static void main(String[] args){  
    Scanner teclado=new Scanner(System.in);  
    int n1,n2;  
  
    System.out.println("Dime dos numeros: ");  
    n1=teclado.nextInt();  
    n2=teclado.nextInt();  
    System.out.println("Su menor es : "+Matematicas.menor(n1,n2));  
  
    System.out.println("Dime otros dos numeros: ");  
    n1=teclado.nextInt();  
    n2=teclado.nextInt();  
    System.out.println("Su menor es : "+Matematicas.menor(n1,n2));  
}
```

Partes clave

```
public static int menor(int a,int b){  
    //Las variables a y b son los parametros de entrada  
    //Son los nombres a utilizar dentro del metodo  
    //Se corresponden posicionalmente en su invocacion  
    int resultado;  
    //Esta variable solo es visible en este bloque  
    if(a<b){  
        resultado=a;  
    }else{  
        resultado=b;  
    }  
    //El return siempre al final porque retorna  
    //al programa donde estaba con el resultado  
    return resultado;  
}
```

PROHIBIDO. Mal diseño de métodos

```
//Nunca hacerlo así  
public static int menor(){  
    int resultado,a,b;  
    System.out.println("Dime dos numeros: ");  
    a=teclado.nextInt();  
    b=teclado.nextInt();  
    if(a<b){  
        resultado=a;  
    }else{  
        resultado=b;  
    }  
    System.out.println("El menor es : "+resultado);  
}  
  
//Un metodo solo debe hacer una accion  
//No se debe mezclar la entrada o salida  
//con la accion del metodo
```


- ❑ Una cuestión importante en Java es el paso de parámetros a métodos.
- ❑ Los tipos simples se pasan por valor, esto quiere decir que si modificamos su valor dentro del método, dicha modificación no afecta a la variable.

```
//método que suma 1 al entero recibido  
public static void sumaEntero (int num){  
    num=num+1;  
}  
...  
int x=45;  
sumaEntero(x);  
System.out.println(x);  
//Sigue valiendo 45
```


- ❑ Los arrays y los objetos en general se pasan por referencia.
- ❑ En otros lenguajes se les llamas punteros en lugar de referencias.
- ❑ Si se modifica su contenido dentro de un método, dicho cambio permanece fuera del método, a diferencia de los tipos simples en el que este cambio se perdía.

```
public static void Sumar(int [] numeros)
{
    for(int i=0;i<numeros.length;i++)
    {
        numeros[i]++;
    }
}
```

```
int [] lista=new int[5];
...
Sumar(numeros);
//Todos los numeros incrementados en 1
```

Códigos para manejar arrays

A veces no podemos evitar meter entrada de datos, siendo este caso super necesario para no repetir código

```
public static int [] crea(int tam, int limite){  
    Random r=new Random();  
    int [] a=new int[tam];  
  
    for(int i=0; i<a.length; i++){  
        a[i]=r.nextInt(limite);  
    }  
    return a;  
}
```

Mejor devolver un String

```
public static void mostrar(int[] a){  
    for(int i=0;i<a.length;i++){  
        System.out.println(a[i]);  
    }  
}
```

```
public static int [] suma(int [] s1, int [] s2){  
    int [] res=new int[s1.length];  
  
    for(int i=0; i<res.length; i++){  
        res[i]=s1[i]+s2[i];  
    }  
    return res;  
}
```

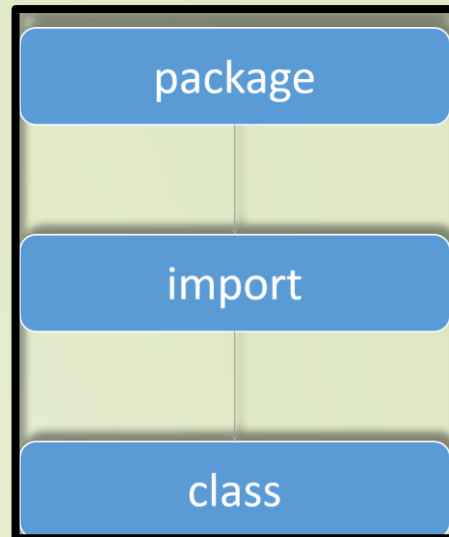
Códigos para manejar matrices

```
public static int [][] creaMatriz(int filas,int col,int limite){  
    Random r=new Random();  
    int [][] mat=new int[filas][col];  
  
    for(int i=0; i<mat.length; i++){  
        for(int j=0; j<mat[i].length; j++){  
            mat[i][j]=r.nextInt(limite);  
        }  
    }  
    return mat;  
}
```

```
public static void muestraMatriz(int [][] mat){  
    for(int i=0; i<mat.length; i++){  
        for(int j=0; j<mat[i].length; j++){  
            System.out.println(mat[i][j]);  
        }  
    }  
}
```

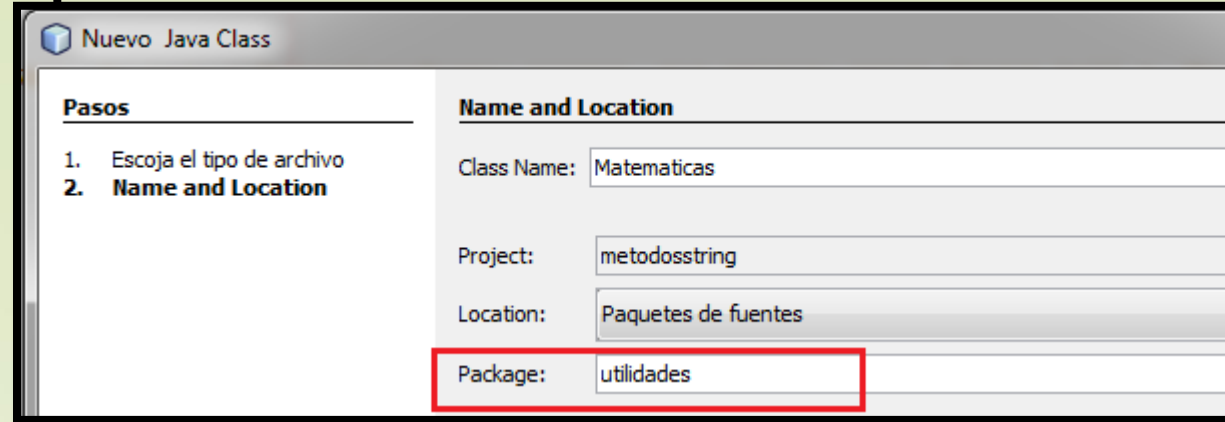
```
public static int [][] sumaMatriz(int [][] s1, int [][] s2){  
    int [][] res=new int[s1.length][s1[0].length];  
  
    for(int i=0; i<res.length; i++){  
        for(int j=0; j<res[0].length; j++){  
            res[i][j]=s1[i][j]+s2[i][j];  
        }  
    }  
    return res;  
}
```

- ❑ Un **paquete** en Java es un contenedor de clases que permite agrupar las **distintas partes** de un programa cuya funcionalidad tienen **elementos comunes**.
- ❑ El uso de paquetes es un **mecanismo** que permite **facilitar** la reutilización de código.



- ❑ Cuando creamos un conjunto de métodos en una o varias clases lo normal es que estén en paquetes distintos al de la clase principal.
- ❑ Esto es para separar conceptos y lo observamos en Random, Date, System otras clases utilizadas.
- ❑ No tendría sentido que clases compartan paquete si de alguna no están relacionadas directamente.
- ❑ Si nosotros queremos realizar lo mismo con nuestras clases que solucionan problemas a parte y después las usamos en el main, cuando se cree la clase deberemos asegurarnos que están en un paquete distinto.

❑ Por ejemplo:



Nuevo Java Class

Pasos

1. Escoja el tipo de archivo
2. **Name and Location**

Name and Location

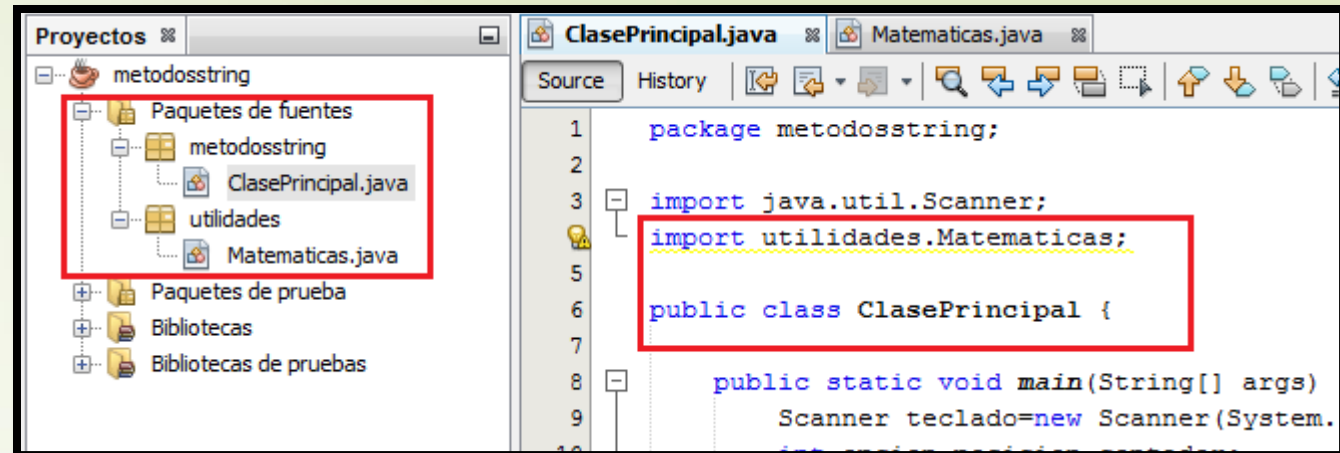
Class Name: Matematicas

Project: metodosstring

Location: Paquetes de fuentes

Package: utilidades

- ❑ Esto nos permitiría en un futuro reutilizar el código sin necesidad de copiar y pegar códigos directamente.
- ❑ También implica hacer un import en el main y/o donde se use.



- ❑ Un método es lo mismo que una función, pero que se encuentra en medio de la definición de una clase.
- ❑ Por lo tanto, en Java, si somos puristas, nunca deberíamos hablar de funciones, sino de métodos, aunque es normal caer en el uso de función.
- ❑ Un método es una parte de código que realiza alguna **tarea** en concreto y que además puede ser invocado desde algún otro sitio.
- ❑ Podemos pensar en un método como en un **subprograma**.

- ❑ Las palabras **public** y **static** son modificadores relacionados con la POO y para no liarnos por ahora los pondremos sin pensarlo.
- ❑ Es posible que esa tarea tenga un resultado y dicho resultado es de un tipo en concreto que se debe especificar.
- ❑ Si la función no devuelve nada, el tipo de dato que se usa aquí es **void**.
- ❑ Por ultimo, lo que hay que especificar es su nombre como **identificador valido de Java**.

¿Cuándo usar funciones?

Poder reutilizar códigos

- ❑ Si hay código que vayamos a utilizar en sitios diferentes, pues hacemos una función con ese código común, y así lo escribimos una sola vez.
- ❑ Una vez escrita la función, ya la podemos usar allá donde queramos.

Programar algo de cierta complejidad

- ❑ Imaginemos que estamos haciendo un programa de contabilidad.
- ❑ En ese programa en un solo sitio, queremos calcular la cuota de una hipoteca.
- ❑ Como no es un cálculo directo y se necesitan muchos pasos, tomamos la decisión **de encapsular todos esos cálculos en una función.**

- ❑ De esta manera obtenemos muchas ventajas sobre el código que programamos.
- ❑ Si posteriormente queremos utilizarlo, pues ya la tenemos.
- ❑ Esto **simplifica** la programación y aumenta la **reutilización**.
- ❑ **Si tenemos un error** en nuestro programa en el cálculo de la hipoteca, es más fácil llegar a él y corregirlo.

Dividir un trozo de código en partes más pequeñas

- ❑ Supongamos que tenemos una función que ocupa **1000 líneas** (código complejo).
- ❑ Como es una función muy grande, **decidimos trocearla** en varias funciones, de manera que la función original se limita a llamar a esas nuevas funciones que han de ser más pequeñas.
- ❑ Esta razón va asociada a la anterior.

Hacer que el programa quede más legible

- ❑ Cuando nos acostumbremos a usar funciones, vemos que los nombres de las mismas deben representar aquello que hacen, precisamente para saber lo que estamos haciendo en cada momento.

Motivos de diseño.

- ❑ Esto se verá un poco mejor cuando lleguemos al tema de la herencia e interfaces en Java.

- ❑ En <https://docs.oracle.com/javase/7/docs/api/> disponemos de la **documentación completa** de todos los paquetes de Java.
- ❑ Toda clase incluso definida por nosotros **se encuentra en un paquete**.
- ❑ En el IDE un paquete se simboliza mediante una **carpeta** que contiene todas las **clases** que forman parte del mismo paquete.
- ❑ Si una clase no pertenece a un paquete se incluye en el **paquete por defecto**, algo poco recomendable.

Java™ Platform, Standard Edition 7

API Specification

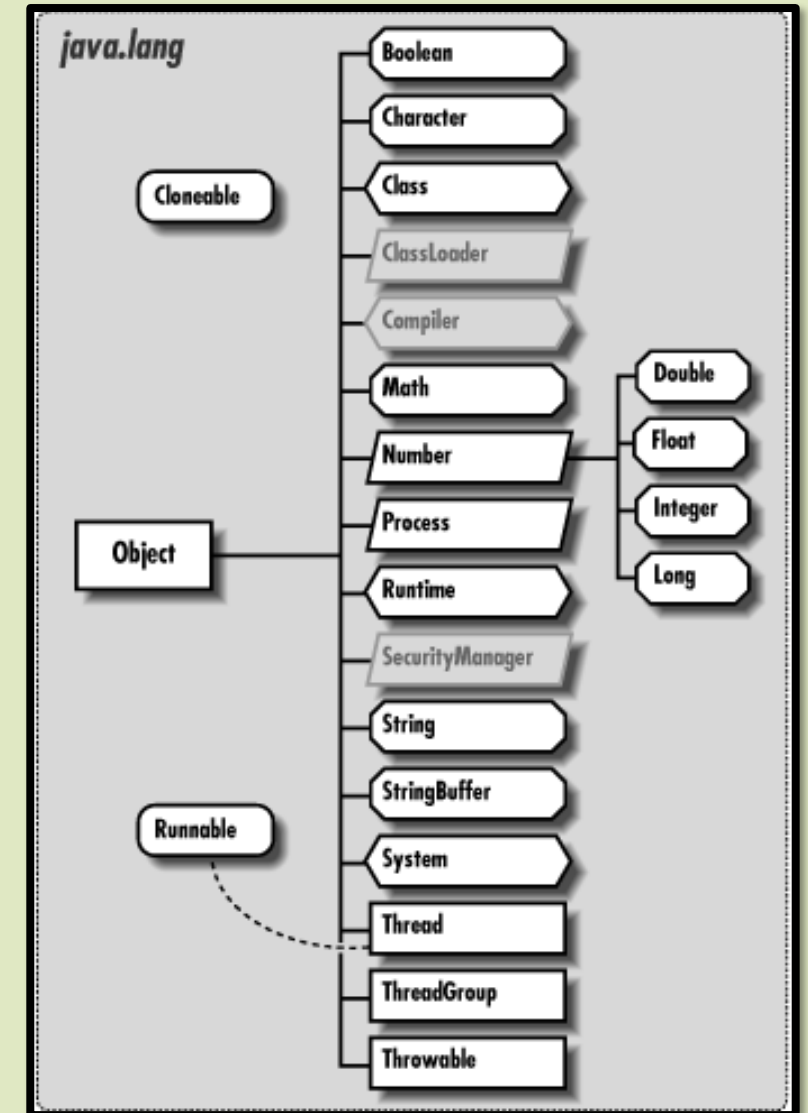
This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing <i>beans</i> – components based on the JavaBeans™ architecture.

- ❑ El paquete de lenguaje Java, también conocido como **java.lang**, contiene las clases que son el **corazón del lenguaje**.
- ❑ Contiene clases **imprescindibles** para el **funcionamiento** de los programas.
- ❑ Es un paquete que **se importa automáticamente**.

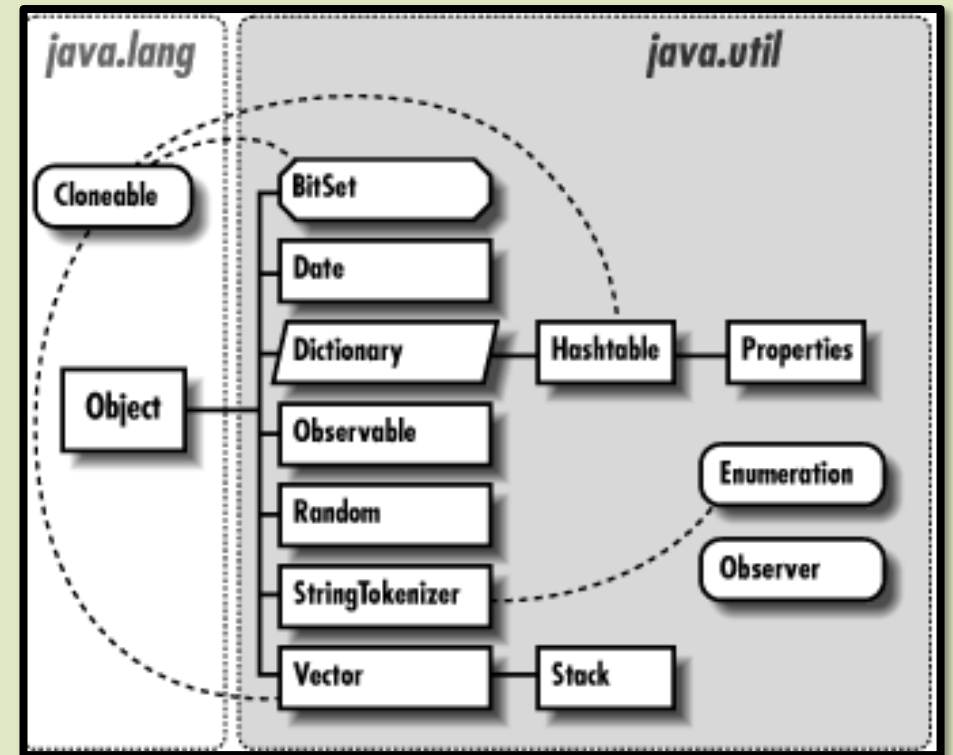


- ❑ La clase **Math** define constantes y operaciones matemáticas más complejas de las habituales.
- ❑ Es un caso particular ya que es **estática**, eso quiere decir que **no se pueden crear instancias** de la clase **Math**.

```
//Caracteristicas
Math.PI;
Math.E;

//Capacidades
Math.pow(3.5,2.3); //Devuelve la potencia de dos numeros
Math.round(5.136); //Redondea por defecto un numero
Math.sqrt(4); //Devuelve la raiz cuadrada
Math.sin(90); //Devuelve la funcion seno
```

- ❑ El paquete `java.util` contiene las clases de utilidades para el lenguaje Java que completan tareas rutinarias, pero de **cierta complejidad**.
- ❑ Las clases tiene que ver con manejo de **fechas**, **colecciones**, **números aleatorios**, **ficheros zip** etc.
- ❑ Este paquete nos ha servido para manejar el uso de teclado usando la **clase Scanner**.



- ❑ La **clase Date** representa fechas en formato universal y permite hacer distintas **operaciones** entre ellas.
- ❑ No es la forma más actual para usar fechas, pero tiene un **gran interés didáctico**.

```
//Creacion y capacidades de la clase Date  
Date hoy=new Date(); //Devuelve la fecha y hora de hoy  
hoy.getDay(); //Devuelve la hora  
hoy.getMinutes(); //Devuelve los minutos  
hoy.getTime(); //Devuelve la fecha y hora transformada en milisegundos  
Date f1=new Date(120,4,12); //Devuelve 12 de Mayo de 2020
```

- ❑ A diferencia de String que se puede usar como un tipo básico, la clase Date usa un **operador nuevo** llamado **new** que es la forma habitual de crear objetos en Java.

- ❑ Una cuestión bastante útil es saber cuánto tiempo ha transcurrido desde una fecha concreta o si es posterior o anterior a otra fecha dada.

```
Long fechaInicialMs = hoy.getTime();
Long fechaFinalMs = f1.getTime();
Long diferencia = fechaFinalMs - fechaInicialMs;
//Para saber cuando una fecha es anterior a otra
if(diferencia>0)
{
    System.out.println("Es posterior");
}else{
    System.out.println("Es anterior");
}

//Para saber cuantos dias han transcurrido entre una fecha y otra
double dias = Math.round(diferencia / (1000 * 60 * 60 * 24));
```

- ❑ La clase **Random** proporciona un generador de números aleatorios para programas que usen el azar o necesiten de un conjunto de datos de prueba.
- ❑ Para ello existe su constructor y una serie de métodos parecidos a la clase **Scanner** para obtener números aleatorios.

```
Random generador=new Random();//Semilla aleatoria por defecto  
Random otro_gen=new Random(3816);//Semilla aleatoria 3816  
  
generador.nextInt(); //Un numero entre 0 y el MAX int  
generador.nextInt(90); //Un numero entre 0 y 89
```

- ❑ Las **aplicaciones** de los números aleatorios en general pueden ser **el número resultante de tirar un dado**, **número premiado en un sorteo de lotería**, **ruleta de un casino**, **máquina tragaperras** y cualquier cuestión relacionada con el azar. También es interesante a la hora de hacer pruebas.
- ❑ Cada vez que creamos un objeto de la clase Random con la misma semilla obtendremos **la misma secuencia de números aleatorios**. Esto no es útil en el caso de **loterías**, pero puede ser útil en el caso de **juegos y/o simulaciones** que se repitan de la misma forma una y otra vez, etc.

Bibliografía

- ❑ **García de Jalón, j.:** “Aprende Java como si estuvieras en primero”. Editorial TECNUN. 2000
- ❑ **Holzner, S.:** “La biblia de JAVA 2”. Editorial Anaya Multimedia 2000.
- ❑ **Moreno Pérez, J.C.:** “C.F.G.S Entornos de desarrollo” Editorial RA-MA. 2012
- ❑ **Wikipedia, la enciclopedia libre.**
<http://es.wikipedia.org/>
Última visita: Octubre 2018.
- ❑ **López, J.C.:** “Curso de JAVA
<http://www.cursodejava.com.mx>
Última visita: Octubre 2018.
- ❑ **Documentación oficial Java JSE 8**
<http://docs.oracle.com/javase/8/>
Última visita: Octubre 2015.
- ❑ **Programación en castellano: Java.**
<http://www.programacion.net/java>
Última visita: Octubre 2015.