UNIVERSIDAD DE EL SALVADOR

FACULTAD MULTIDISCIPLINARIA DE OCCIDENTE
DEPARTAMENTO DE INGENIERÍA Y ARQUITECTURA.

GUIA 1

"CONCEPTOS PARTE 1, DESARROLLO DE GUIA 1"

ASIGNATURA: PROGRAMACION II

Catedrático:

Ing. William Virgilio Zamora Girón

Instructor:

Br. Sigfrido Ernesto Gómez Guinea.

Alumnos:

Br. Marcello Ernesto Solórzano Flores.

Br. Jonathan Otoniel González Aguirre.

Br. Antonieta Carolina Aragón Cruz.

Grupo: #2

Ciclo I -2018

Guía número uno programación dos ingeniería.

Instrucciones: Resuelva los cuestionamientos presentados a continuación de manera concisa, lógica y ordenada.

Objetivo: Realizar un repaso general de la programación orientada a objetos (POO) y rememorar conocimientos de la misma, dando un enfoque a la resolución de problemas cotidianos de ingeniería, guiando al alumno a hacer uso de la misma para resolverlo.

Parte I: Responder de manera clara y precisa sobre los conceptos y cuestionamientos.

1-¿Qué es la programación orientada a objetos (POO)?

La programación orientada a Objetos básicamente define una serie de conceptos y técnicas de programación para representar acciones o cosas de la vida real basada en objetos, a diferencia de otras formas de programación como por ejemplo la estructurada, con la POO trabajamos de manera distinta vinculando diferentes conceptos tales como clases, objetos, métodos, propiedades, estados, herencia, encapsulación entre otros, generando cada vez interrelaciones en nuestro desarrollo en pro del funcionamiento del sistema principal, definiendo el programa como un conjunto de estos objetos relacionados entre si.

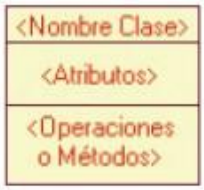
2-¿Cuáles son los conceptos y nociones básicas de la POO?

Veamos algunos de los conceptos principales.....

1. **Clases.**

Una clase se compone por tres partes fundamentales:

- ✓ **Nombre**
- ✓ **Atributos**
- ✓ **Métodos**



2. **Objeto.**

Al igual que las clases se componen de tres partes fundamentales:

- ✓ **Estado**
- ✓ **Comportamiento**
- ✓ **Identidad**

3. **Herencia.**
4. **Encapsulamiento**

Se representa por 3 niveles :

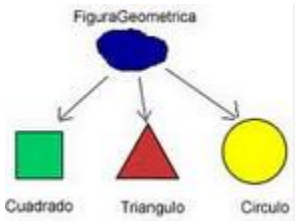
- ✓ **Público**
- ✓ **Protegido**
- ✓ **Privado**

5. **Clases Abstractas.**
6. **Interfaces.**
7. **Polimorfismo.**

3-Explique de manera clara que es:

-Polimorfismo.

Este tal vez sea uno de los conceptos de la programación orientada a objetos más usados pero muchas veces sin saber que se aplica ya que el concepto inicialmente puede ser un poco confuso, básicamente mediante el polimorfismo programamos de forma general en lugar de hacerlo de forma específica, se usa cuando se trabajen con la herencia y objetos de características comunes los cuales comparten la misma superClase y árbol jerárquico, al trabajar con este concepto optimizamos y simplificamos en gran medida nuestro trabajo.



Básicamente podemos definirlo como la capacidad que tienen los objetos de comportarse de múltiples formas sin olvidar que para esto se requiere de la herencia, en si consiste en hacer referencia a objetos de una clase que puedan tomar comportamientos de objetos descendientes de esta.

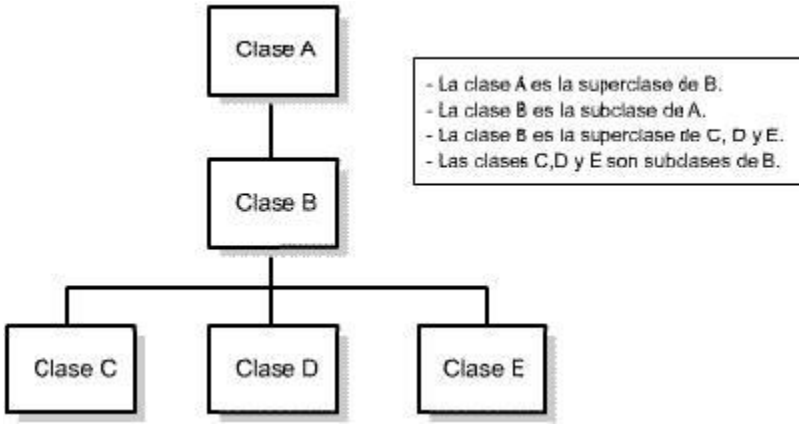
Con el polimorfismo usamos la generalización olvidando los detalles concretos de los objetos para centrarnos en un punto en común mediante una clase padre.

Tomando como ejemplo la imagen anterior, podemos decir que un objeto de la clase **FiguraGeometrica** puede usarse para referirse a cualquier objeto de cualquier **subClase** de **FiguraGeometrica**..... en otras palabras una figura geométrica puede ser un cuadro, un triángulo, un cuadrado o cualquier figura que en términos generales sea geométrica..... (**Ejemplo1**) (**Ejemplo2**)

Veamos este proceso como se representa en Java.

```java
class FiguraGeometrica{

}

class Cuadrado extends FiguraGeometrica {

}

class Triangulo extends FiguraGeometrica {

}

class Circulo extends FiguraGeometrica{

}

public class Principal{

   public void metodo(){
 /**Puedo crear objetos concretos*/
 FiguraGeometrica miFiguraGeometrica = new FiguraGeometrica();
 Cuadrado miCuadro=new Cuadrado();

 /**Puedo crear objetos polimorficos*/
 miFiguraGeometrica=miCuadro;

 /**Objeto Cuadrado de tipo FiguraGeometrica*/
 FiguraGeometrica miCuadrado= new Cuadrado();
 /**Objeto Circulo de tipo FiguraGeometrica*/
 FiguraGeometrica miCirculo=new Circulo();
 /**Objeto Triangulo de tipo FiguraGeometrica*/
 FiguraGeometrica miTriangulo=new Triangulo();
  }
}
```

-Abstracción.



Abstraer significa simplificar.

A partir de ahora en adelante vamos a trabajar con clases y objetos, que son el elemento principal del lenguaje Java (y cualquier lenguaje orientado a objetos).

Al principio se puede hacer un poco difícil de entender, pero con constancia y empeño se entenderá y veréis como la programación orientada a objetos ofrece muchísimas ventajas.

-Herencia.

La herencia en java representa lo que conocemos de herencia en el mundo real, básicamente mediante esta obtenemos las características o rasgos comunes de nuestros padres o abuelos, en java es el mismo enfoque permitiendo la creación de nuevas clases basadas en clases ya existentes, con las cuales podemos obtener las características de las clases padres, heredando campos, atributos, métodos o funcionalidades.

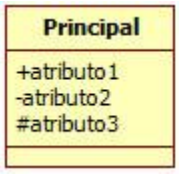
En Java solo se puede heredar de una sola clase padre y se representa mediante la palabra **extends (Ver Ejemplo...)**

```
1  public class Animal{
2   public String tamaño;
3
4   public void comer(){
5    /**Comportamiento.....*/
6   }
7  }
8
9   class Perro extends Animal{
10
11  public int dientes;
12
13  public void correr(){
14   /**Comportamiento.....*/
15  }
16 }
17
18  class Paloma extends Animal{
19
20  public int plumas;
21
22  public void volar(){
23   /**Comportamiento.....*/
24  }
25 }
```

-Encapsulamiento:

Este concepto es uno de los más importantes en términos de seguridad dentro de nuestra aplicación, la encapsulación es la forma de proteger nuestros datos dentro del sistema, estableciendo básicamente los permisos o niveles de visibilidad o acceso de nuestros datos

Se representa por 3 niveles:



Público: Se puede acceder a todos los atributos o métodos de la clase. **Protegido**: Se puede acceder a los atributos o métodos solo en la misma jerarquía de herencia. **Privado**: Solo se puede acceder a los atributos o métodos de la clase en la que se encuentran.

Con la Encapsulación mantenemos nuestros datos seguros, ya que podemos evitar que por ejemplo se hagan modificaciones al estado o comportamiento de un objeto desde una clase externa, una buena práctica es trabajar con métodos **setter** y **getter** que permiten manipular nuestros datos de forma segura.

En Java lo representamos así:

```
1  public class Principal{
2
3    public String atributo1;
4    private String atributo2;
5    protected String atributo3;
6
7    /**Métodos Set y Get para trabajar con las variables*/
8
9    public String getAtributo2() {
10  return atributo2;
11   }
12
13   public void setAtributo2(String atributo2) {
14  this.atributo2 = atributo2;
15   }
16
17 }
```

4-¿Por qué es preferible usar POO a programación estructurada?

La programación orientada a objetos trata de amoldarse al modo de pensar del hombre y no al de la máquina.

El elemento básico de este paradigma no es la función (elemento básico de la programación estructurada), sino un ente denominado objeto.

Un objeto es la representación de un concepto para un programa, y contiene toda la información necesaria para abstraer dicho concepto: los datos que describen su estado y las operaciones que pueden modificar dicho estado, y determinan las capacidades del objeto.

Java incorpora el uso de la orientación a objetos como uno de los pilares básicos de su lenguaje.

5-¿Qué ventajas posee al usar JAVA para el desarrollo de aplicaciones?

Java ha sido probado, ajustado, ampliado y probado por toda una comunidad de desarrolladores, arquitectos de aplicaciones y entusiastas de Java. Java está diseñado para permitir el desarrollo de aplicaciones portátiles de elevado rendimiento para el más amplio rango de plataformas informáticas posible. Al poner a disposición de todo el mundo aplicaciones en entornos heterogéneos, las empresas pueden proporcionar más servicios y mejorar la productividad, las comunicaciones y colaboración del usuario final y reducir drásticamente el costo de propiedad tanto para aplicaciones de usuario como de empresa. Java se ha convertido en un valor impagable para los desarrolladores, ya que les permite:

- Escribir software en una plataforma y ejecutarla virtualmente en otra
- Crear programas que se puedan ejecutar en un explorador y acceder a servicios Web disponibles
- Desarrollar aplicaciones de servidor para foros en línea, almacenes, encuestas, procesamiento de formularios HTML y mucho más
- Combinar aplicaciones o servicios que utilizan el lenguaje Java para crear aplicaciones o servicios con un gran nivel de personalización
- Escribir aplicaciones potentes y eficaces para teléfonos móviles, procesadores remotos, microcontroladores, módulos inalámbricos, sensores, gateways, productos de consumo y prácticamente cualquier otro dispositivo electrónico

JavaFX está basado en Java. La plataforma JavaFX permite a los desarrolladores de la aplicación crear e implementar fácilmente aplicaciones de Internet enriquecidas (RIA) que se comportan de la

misma forma en distintas plataformas. JavaFX amplía la potencia de Java permitiendo a los desarrolladores utilizar cualquier biblioteca de Java en aplicaciones JavaFX. Los desarrolladores pueden ampliar sus capacidades en Java y utilizar la tecnología de presentación que JavaFX proporciona para crear experiencias visuales que resulten atractivas.

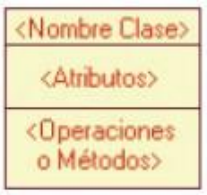
Para obtener más información sobre la tecnología de JavaFX, visite el sitio web de JavaFX.

6-Desglose cada uno de los siguientes conceptos en el ámbito de la POO:

-Clase.

Las clases son uno de los principales componentes de un lenguaje de programación, pues en ellas ocurren todos los procesos lógicos requeridos para un sistema, en si podemos definirlas como estructuras que representan objetos del mundo real, tomando como objetos a personas, lugares o cosas, en general las clases poseen propiedades, comportamientos y relaciones con otras clases del sistema. (**Ver Ejemplo...**)

una clase se compone por tres partes fundamentales:



Nombre: Contiene el Nombre de la Clase.
Atributos: Representan las propiedades que caracterizan la clase.
Métodos: Representan el comportamiento u operaciones, la forma como interactúa la clase con su entorno.

En java se representa así:

```
1  /**Principal define el nombre de la Clase*/
2  public class Principal {
3
4   public String atributo="Esto es un atributo";
5
6   /**Esto es un método, donde se definen las operaciones*/
7   public void metodo(){
8    /**aqui van las sentencias que definen
9     * el comportamiento del método*/
10  }
11 }
```

-Método.

Los métodos son como una Función.

Lo que pasa que los métodos están dentro de una clase, es decir, que son funciones que solo utilizan los objetos de esta Clase, en este caso, los coches que creemos.

Se pueden hacer tantos métodos como se quieran para la Clase.

En esta Clase Coche, vamos a crear solo 3, que son los siguientes:

- Método para añadir más Km al coche.
- Método para saber cuántos Km lleva recorrido el coche.
- Método para saber de qué color es el coche.

Como ya digo, se pueden hacer más, por ejemplo, un método para saber qué marca es el coche, otro para modificar el color del coche, etc...Pero con estos 3 nos valdrá para aprender a crear los métodos, que es lo que nos interesa, y esto es una Clase de ejemplo.

Debajo del constructor de la Clase, vamos a programar los métodos, de esta forma:

```
18      //Métodos
19
20      //Añadir más Km al coche
21      public void AñadirKm (double km_de_mas)
22      {
23          km = km + km_de_mas; //Se añaden los km de más a los que ya tenía el coche
24      }
25
26      //Saber los Km que lleva recorrido el coche
27      public double ObtenerKm ()
28      {
29          return km;
30      }
31
32      //Saber color del coche
33      public String ObtenerColor()
34      {
35          return color;
36      }
```

AulaFacil.com

El primer Método, le suma la cantidad que le pases por parámetro, a la cantidad de Km que ya tenía el coche.

El segundo, simplemente devuelve los Km que tiene el Coche.

(Esta es la forma de acceder a las variables "private", si fuese "public", no tendríamos que hacer este método, puesto que se podría acceder a la variable directamente).

Y el tercero, realiza la misma operación que el segundo. Este tipo de métodos son muy comunes en las Clases.

-Constructor.

El constructor siempre tiene el mismo nombre que la Clase y se utiliza para construir objetos de esa Clase.

Los constructores no tienen ningún valor de retorno, ni siquiera "void".

Los constructores se forman así:

```
1   package aulafacil;
2
3   public class Coche {
4
5       //Atributos
6       private String marca; //Marca del coche
7       private String color; //Color del coche
8       private double km; //Kms recorridos
9
10      //Constructor
11      public Coche (String m, String c, double kilometros)
12      {
13          marca = m; //Asigna la marca
14          color = c; //Asigna el color
15          km = kilometros; //Asigna los Km
16      }
17  }
```

AulaFacil.com

El constructor si lleva delante la palabra public.

Si os dáis cuenta es muy parecido a una función. Lo que pasa que el constructor no tiene valor de retorno, no es necesario.

Porque lo que hace esta Clase al llamar a su constructor es crear un objeto de la Clase.

Fijáos como le pasamos por parámetros las variables necesarias para la "creación" del coche.

Es decir, cada que vez queramos crear un nuevo Coche, por parámetro hay que pasarle la marca, color, y km, para que se asignen esos valores a los atributos de la Clase.

-Librería.

Una librería en Java se puede entender como un conjunto de clases, que poseen una serie de métodos y atributos. Lo realmente interesante de

estas librerías para Java es que facilitan muchas operaciones. De una forma más completa, las librerías en Java nos permiten reutilizar código, es decir que podemos hacer uso de los métodos, clases y atributos que componen la librería evitando así tener que implementar nosotros mismos esas funcionalidades.

Imaginemos cómo hacer un código para conseguir que nuestra aplicación imprima algo por pantalla o conseguir que nuestra aplicación pueda hacer uso de arreglos dinámicos con cualquier tipo de dato. Por medio de librerías, para imprimir texto en pantalla en Java basta con usar System.out.print() y se puede hacer uso de ArrayList, para tener arreglos dinámicos y demás, en Java, esto es así de fácil y todo es gracias a las librerías, la clase System que pertenece a la librería java.lang (ésta es la librería estándar de Java y no es necesario importarla). Ocurre de la misma forma con el ArralyList, no podríamos hacer uso de éste si no existiera la librería java.util. Cabe mencionar que estas librerías no solo poseen estas clases sino que poseen una gran cantidad de clases Java adicionales que nos permiten llevar a cabo operaciones muy complejas con gran facilidad.

Lo más interesante de las librerías en Java, es que nosotros también podemos crearlas y hacer uso de ellas al interior de nuestros proyectos. Básicamente un paquete en Java puede ser una librería, sin embargo una librería Java completa puede estar conformada por muchos paquetes más. Al importar un paquete podemos hacer uso de las clases, métodos y atributos que lo conforman, así que eso de las librerías no es solo trabajo de otros sino que podemos crear nuestras propias librerías y usarlas, incluso podemos hacer uso de un proyecto completo nuestro al interior de otro.

-Registro.

Ya se ha visto como variables simples pueden almacenar una sola pieza de información y como arreglos pueden almacenar un conjunto de ellas del mismo tipo y al mismo tiempo, estos dos mecanismos pueden manejar una gran variedad de situaciones, pero a menudo se necesita trabajar sobre datos de diversos tipos, en este caso ni variables escalares ni arreglos son adecuados.

Para resolver estos problemas los lenguajes de programación proveen de un tipo de dato especial llamado registros.

Un registro es una variable especial que tiene la capacidad de almacenar datos de diferentes tipos.

Sin embargo JAVA, usa en su lugar una CLASE.

Este método tiene la ventaja de que además de incluir los campos tradicionales de un registro (en forma de atributos), también puede incorporar una serie de métodos que permiten procesar de manera más fácil los campos o atributos de la clase.

Ejemplo;

class alumno{

void alumno(){};

static String nombre= new String();

static int edad;

void inicializar(){

alumno.nombre="pepe el toro";

alumno.edad=18; };

void desplegar(){

System.out.println(alumno.nombre);

System.out.println(alumno.edad);

};

} termina la clase

Programa ejemplo;

```
import java.io.*;
class prog33{
public static void main(String[] args) {
alumno.inicializar();
alumno.desplegar();
} // cierra main
} // cierra clase
class alumno{
void alumno(){};
static String nombre= new String();
static int edad;
static void inicializar(){
```

```
alumno.nombre="pepe el toro";
alumno.edad=18; };
static void desplegar(){
System.out.println(alumno.nombre);
System.out.println(alumno.edad);
};
} // temina la clase
```

-Dato.

Los primeros lenguajes de programación no usaban objetos, solo variables. Una variable podríamos decir que es **un espacio de la memoria del ordenador a la que asignamos un contenido** que puede ser un valor numérico (sólo números, con su valor de cálculo) o de tipo carácter o cadena de caracteres (valor alfanumérico que constará sólo de texto o de texto mezclado con números).

Como ejemplo podemos definir una variable a que contenga 32 y esto lo escribimos como a = 32. Posteriormente podemos cambiar el valor de a y hacer a = 78. O hacer "a" equivalente al valor de otra variable "b" así: a = b.

Dado que antes hemos dicho que un objeto también ocupa un espacio de memoria: **¿en qué se parecen y en qué se diferencia un objeto de una variable?** Consideraremos que las variables son entidades elementales: un número, un carácter, un valor verdadero o falso… mientras que los objetos son entidades complejas que pueden estar formadas por la agrupación de muchas variables y métodos. Pero ambas cosas ocupan lo mismo: un espacio de memoria (que puede ser más o menos grande).

En los programas en Java puede ser necesario tanto el uso de datos elementales como de datos complejos. Por eso en Java se usa el término "Tipos de datos" para englobar a cualquier cosa que ocupa un espacio de memoria y que puede ir tomando distintos valores o características durante la ejecución del programa. Es decir, en vez de hablar de tipos de variables o de tipos de objetos, hablaremos simplemente de tipos de datos. Sin embargo, a veces "coloquialmente" no se utiliza la terminología de forma estricta: puedes encontrarte textos o páginas web donde se habla de una variable en alusión a un objeto.

En cualquier lenguaje de programación, se trabaja con datos. Por ello, en esta lección voy a mencionar los tipos de dato más utilizados (y básicos).

Hay más, pero son menos útiles.

Los tipos de datos que más vamos a usar son los siguientes:

- **int** = Tipo de dato entero. Su valor mínimo es -2,147,483,648 y el máximo 2,147,483,647 (incluído).

  Ejemplo: 12, 456, -315...

- **double** = Dato en coma flotante, para valores decimales.
  Ejemplo: 2.40 - 3.14 - 10.90.
- **boolean** = El tipo de dato boolean solamente tiene dos valores posibles: true (verdadero) y false (falso).
- **char** = El tipo de dato char es un solo carácter Unicode. Por ejemplo: 'c', 'a', '7', etc.

Veremos otro tipo de dato sumamente útil, que son las **cadenas de caracteres**, pero a estas les dedicaremos una lección aparte.

Los tipos de datos no se pueden mezclar entre sí, es decir, no podemos introducir un dato tipo "double" en un dato tipo "entero", puesto que no tienen nada que ver.

Estos son conceptos puramente teóricos, ahora con el uso de **variables**, vamos a entender mejor para qué sirven los tipos de datos.

-Valor.

Las variables son el elemento esencial de la programación.

En ellas será donde almacenemos todos los datos con los que vamos a trabajar.

Vamos a aplicar los conceptos vistos anteriormente a la práctica, es decir, veremos cómo se crean y modifican datos en Java.

La creación de variables se lleva a cabo de la siguiente forma:

<tipo de dato> <nombre de variable>

El tipo de dato puede ser cualquiera de los vistos en la lección anterior, y el nombre de la variable, el que queramos darle, siempre es recomendable darle un nombre orientativo.

Como ejemplo, vamos a crear un dato de cada tipo visto anteriormente.

En la siguiente imagen veréis como queda:

```
1  package aulafacil;
2
3  public class AulaFacil {
4
5      public static void main(String[] args) {
6
7          int num = 3;
8          double num_decimal = 10.85;
9          boolean bool = true;
10         char caracter = 'c';
11
12     }
13 }
14
```

AulaFacil.com

En este caso, a la vez que los he creado, les he asignado un valor.

Esto también se podría hacer por separado, sería de la siguiente forma:

int num;
num = 3;

De esta forma, primero se crea la variable tipo entero, y luego se le asigna el valor "3".

Ya solo nos queda comprobar que dichas variables tienen el contenido bien asignado. Para ello vamos a mostrar las variables por pantalla...¿Imagináis como? Como hemos hecho anteriormente, utilizaremos "***System.out.println***".

```
1  package aulafacil;
2
3  public class AulaFacil {
4
5      public static void main(String[] args) {
6
7          int num = 3;
8          double num_decimal = 10.85;
9          boolean bool = true;
10         char caracter = 'c';
11
12         System.out.println("Esto contiene la variable num: " + num);
13         System.out.println("Esto contiene la variable num_decimal: " + num_decimal);
14         System.out.println("Esto contiene la variable bool: " + bool);
15         System.out.println("Esto contiene la variable caracter: " + caracter);
16
17     }
18 }
19
```

AulaFacil.com

Esto nos mostrará el contenido de cada variable.

7-¿Cuáles son las características principales de los siguientes tipos de dato? (Limitantes, posibilidades, usos, etc).

- **-int.**

Tipo de dato entero. Su valor mínimo es -2,147,483,648 y el máximo 2,147,483,647 (incluído).

Ejemplo: 12, 456, -315...

-double.

Dato en coma flotante, para valores decimales.
Ejemplo: 2.40 - 3.14 - 10.90.

-String.

Dentro de un objeto de la clases *String* o *StringBuffer*, Java crea un array de caracteres de una forma similar a como lo hace el lenguaje C++. A este array se accede a través de las funciones miembro de la clase.

Los strings u objetos de la clase *String* se pueden crear explícitamente o implícitamente. Para crear un string implícitamente basta poner una cadena de caracteres entre comillas dobles. Por ejemplo, cuando se escribe

```
System.out.println("El primer programa");
```

Java crea un objeto de la clase *String* automáticamente.

Para crear un string explícitamente escribimos

```
String str=new String("El primer programa");
```

También se puede escribir, alternativamente

```
String str="El primer programa";
```

Para crear un string nulo se puede hacer de estas dos formas

```
String str="";
String str=new String();
```

Un string nulo es aquél que no contiene caracteres, pero es un objeto de la clase *String*. Sin embargo,

```
String str;
```

Está declarando un objeto *str* de la clase *String*, pero aún no se ha creado ningún objeto de esta clase.

-Object.

La clase **Object**, como ya se ha indicado anteriormente, es la clase raíz de todo el árbol de la jerarquía de clases Java, y proporciona un cierto número de métodos de utilidad general que pueden utilizar todos los objetos. La lista completa se puede ver en la documentación del API de Java, aquí solamente se tratarán algunos de ellos; por ejemplo, **Object** proporciona:

- Un método por el que un objeto se puede comparar con otro objeto
- Un método para convertir un objeto a una cadena
- Un método para esperar a que ocurra una determinada condición
- Un método para notificar a otros objetos que una condición ha cambiado
- Un método para devolver la clase de un objeto

-char.

El tipo de dato char es un solo carácter Unicode. Por ejemplo: 'c', 'a', '7', etc.

8-¿Que es la clase math en JAVA?

La clase *Math* tiene miembros dato y funciones miembro estáticas, vamos a conocer algunas de estas funciones, cómo se llaman y qué tarea realizan.

La clase *Math* define dos constantes muy útiles, el número π y el número e.

```
public final class Math {
    public static final double E = 2.7182818284590452354;
    public static final double PI = 3.14159265358979323846;
    //...
}
```

El modificador **final** indica que los valores que guardan no se pueden cambiar, son valores constantes

Se accede a estas constantes desde la clase *Math*, de la siguiente forma

```
System.out.println("Pi es " + Math.PI);
System.out.println("e es " + Math.E);
```

9-¿Qué es una base de datos?

Java Database Connectivity, más conocida por sus siglas **JDBC**, es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan los métodos de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la biblioteca de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión; para ello provee el localizador a la base de datos y los parámetros de conexión específicos. A partir de allí puede realizar cualquier tipo de tarea con la base de datos a la que tenga permiso: consulta, actualización, creación, modificación y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.