CLASES
Una clase es un tipo definido por el usuario que describe los atributos y los métodos de los objetos que se crearan a partir de la misma.

```
class nombre_clase
{
 cuerpo de la clase
}
```

Derechos de Acceso:

- Acceso privado (private): Los elementos privados solo se pueden usar dentro de la clase que los define, nunca desde ninguna otra clase.
- Acceso de paquete: No se pone nada. El acceso a estos componentes es libre dentro del paquete en el que se define la clase.
- Acceso protegido (protected): los elementos protegidos solo se pueden usar dentro de la clase que los define, aquellas clases que la extiendan y cualquier clase en el mismo paquete.
- Acceso publico (public): Dicho elemento se puede usar libremente.

COMPOSICION

La composicion es el agrupamiento de uno o varios objetos y valores como atributos que conforman el valor de los distintos objetos de una clase.

PAQUETES
Los paquetes son agrupaciones de clases, interfaces y otros paquetes (subpaquetes), normalmente relacionados entre si. Los paquetes proporcionan un mecanismo de encapsulacion de mayor nivel que las clases. Los paquetes permiten unificar un conjunto de clases e interfaces relacionados funcionalmente. Por ejemplo, el paquete java engloba una serie de paquetes con utilidades de soporte al desarrollo y ejecucion de la aplicación. Contiene a su vez, los subpaquetes util o lang.
Para indicar que la clase que se esta escribiendo pertenece a un paquete, la primera secuencia debe tener la sintaxis:

```
package nombrePaquete;
```

cuando en una clase se quieren utilizar componentes que están en otro paquete diferente, se añade una declaración de importación, que puede tener las siguientes formas:

```
import Matricula.Alumno; // importacion de la clase “Alumno” del paquete “Matricula”
import Matricula.*; // importacion del paquete “Matricula”, incluye la clase Alumno.
```

ATRIBUTOS
Los atributos permiten guardar información de un objeto. Por ejemplo para un alumno se necesita saber la siguiente información: el nombre, los apellidos, la materia en que esta inscrito, si su horario es de mañana o de tarde, etc.
La declaración de atributos se hace de la siguiente manera:

```
enum Horario {MAÑANA, TARDE} //posibles horarios
class Alumno{
    String nombre;
    String apellidos;
```

```
    int añoDeNacimiento;
    int numeroCarnet; //para identificar al alumno
    String grupo;
    Horario horario=Horario.MAÑANA;
}
```

METODOS

Los métodos sirven para definir el comportamiento del objeto en sus interacciones con otros objetos. Siguiendo con el ejemplo del alumno, se puede solicitar su nombre, asignarle grupo, etc.

```
enum Horario {MAÑANA, TARDE} //posibles horarios
class Alumno{
    String nombre;
    String apellidos;
    int añoDeNacimiento;
    int numeroCarnet; //para identificar al alumno
    String grupo;
    Horario horario=Horario.MAÑANA;
       public String ingresaGrupo() { ... }
       public void imprimeGrupo(String nuevoGrupo) { ... }
}
```

CONSTRUCTORES.

Un constructor es un procedimiento especial de una clase que es llamado automáticamente siempre que se crea un objeto de esa clase. Su función es iniciar el objeto.

```
Alumno alumno1=new Alumno();
```

Esta operación invoca al constructor por defecto, que se proporciona automáticamente y tiene el mismo nombre que la clase.

Una clase puede definir varios constructores con distintos tipos de argumentos, a esto se le conoce como SOBRECARGA.

DESTRUCTORES

Es un procedimiento especial de una clase que es llamado automáticamente siempre que se destruye un objeto de esa clase. Su función es realizar cualquier final en el momento de destruir el objeto.

REFERENCIA this

La palabra clave this hace referencia a los miembros de la propia clase en el objeto actual.

```
public class MiClase {
    int i;
    public MiClase() {
       i = 10;
       }
    // Este constructor establece el valor de i
    public MiClase( int valor ) {
       this.i = valor; // i = valor
       }
    // Este constructor también establece el valor de i
    public MiClase( int i ) {
       this.i = i;
       }
    public void Suma_a_i( int j ) {
```

```
        i = i + j;
        }
}
```

HERENCIA

La herencia establece una relación esun entre clases. La herencia introduce la capacidad de extender clases, donde la clase original se denomina padre (o madre), clase base o superclase, y la nueva clase denominada clase hija, derivada o subclase. Así, una clase derivada es una clase base. Por ejemplo, Alumno es una Persona. Esta relación se puede representar haciendo una clase Alumno extienda (herede) de la clase Persona.

La sintaxis en Java para la extensión de clases (herencia) es la siguiente:

class ClaseDerivada extends ClaseBase { ... }

en el caso de Alumno:

class Alumno extends Persona { ... }

POLIMORFISMO

De acuerdo con la compatibilidad ascendente, se puede asignar a una variable de una clase una referencia de cualquier objeto de la misma clase o de cualquier clase derivada de ella. Polimorfismo es la capacidad de que diferentes objetos de una misma jerarquía tengan comportamientos distintos aunque reciban el mismo mensaje.

La idea detrás del polimorfismo es poder diseñar nuestro código teniendo en cuenta que todos las clases que derivan de una clase base heredan los métodos de la clase base.

```
abstract public class Animal {
    private String nombre;
    public Animal(String nombre) { this.nombre = nombre; }
    public String getNombre()    { return nombre;      }
    abstract public void habla();
}
public class Perro extends Animal {
    public Perro(String nombre) { super(nombre); }
    public void habla() {
            System.out.println(getNombre()+": Warf, Guau, etc.");
    }
}
public class Gato extends Animal {
    public Gato(String nombre) { super(nombre); }
    public void habla() {
        System.out.println(getNombre()+": Miau, Pffff, etc.");
    }
}
public class TestAnimal {
    public static void main(String[] args) {
        // un array de Animal
        Animal[] animales = {
            new Perro("Kujo"),
            new Perro("Fido"),
             new Gato("Garfield"),
             new Gato("Patán")
         };
         for( Animal a : animales)
```

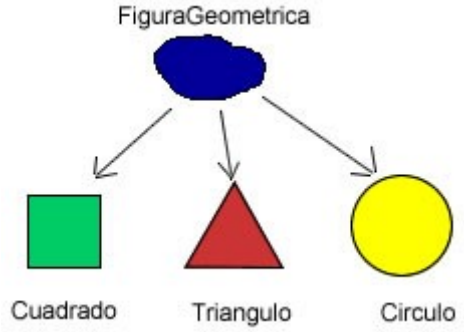
```
            a.habla();
        } // end main
}   // end class
```

La salida:

```
Kujo: Warf, Guau, etc.
Fido: Warf, Guau, etc.
Garfield: Miau, Pffff, etc.
Patán: Miau, Pffff, etc.
```

CLASES ABSTRACTAS

Las denominadas clases abstractas son las que representan el mayor grado de abstracción.
De hecho, las clases abstractas presentan un nivel de "abstracción" tan elevado que no sirven para instanciar objetos de ellas.



¿Cuándo es una clase abstracta?
En cuanto uno de sus métodos no tiene implementación (en Java, el método abstracto se etiqueta con la palabra reservada abstract).
¿Cuándo se utilizan clases abstractas?
Cuando deseamos definir una abstracción que englobe objetos de distintos tipos y queremos hacer uso del polimorfismo.

```
public abstract class Figura
{
  protected double x;
  protected double y;
  public Figura (double x, double y)
  {
    this.x = x;
    this.y = y;
  }
  public abstract double area ();
}
public class Circulo extends Figura
{
  private double radio;
  public Circulo (double x, double y, double radio)
  {
    super(x,y);
    this.radio = radio;
  }
```

```
  public double area ()
  {
    return Math.PI*radio*radio;
  }
}
public class Cuadrado extends Figura
{
  private double lado;
  public Cuadrado (double x, double y, double lado)
  {
    super(x,y);
    this.lado = lado;
  }
  public double area ()
  {
    return lado*lado;
  }
}
```

INTERFACES

Las interfaces se declaran con la palabra reservada interface de manera similar a como se declaran las clases abstractas.
En la declaración de una interfaz, lo único que puede aparecer son declaraciones de métodos (su nombre y signatura, sin su implementación) y definiciones de constantes simbólicas.
Una interfaz no encapsula datos, sólo define cuáles son los métodos que han de implementar los objetos de aquellas clases que implementen la interfaz. Para indicar que una clase implementa una interfaz se utiliza la palabra reservada implements.
La clase debe entonces implementar todos los métodos definidos por la interfaz o declararse, a su vez, como una clase abstracta (lo que no suele ser especialmente útil):

```
  abstract class SinArea implements Figura
  {
  }
```