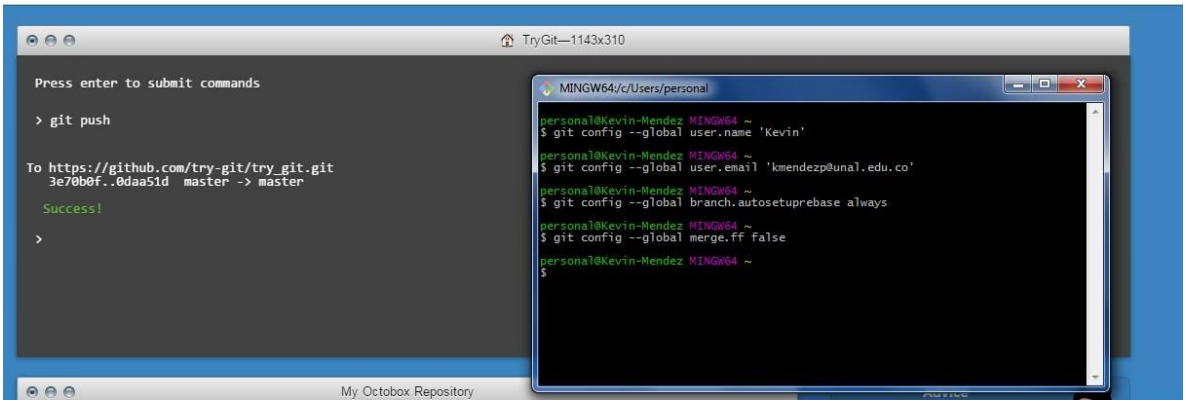

1) Abstracción: La abstracción se puede definir como una operación intelectual que separa las cualidades de un objeto para considerarlas aisladamente o para analizar al objeto en su pura esencia o noción. Básicamente es la capacidad de separar los elementos (al menos mentalmente) para poder verlos de forma singular. Como cuando describimos el cuerpo humano y decimos cabeza, brazo(s), pierna(s), etc. En los lenguajes de programación orientada a objetos, el concepto de Clase es la representación y el mecanismo por el cual se gestionan las abstracciones. Gracias a la abstracción podemos representar las características esenciales de un objeto sin preocuparnos de las restantes características (no esenciales). La abstracción se centra en la vista externa de un objeto, de modo que sirva para separar el comportamiento esencial de un objeto de su implementación. Ejemplo: En

POO, podemos considerar una *Persona* como un objeto que tiene propiedades (como nombre, altura, peso, color de pelo, color de ojos, etcétera) y métodos (como hablar, mirar, andar, correr, parar, etcétera). Gracias a la abstracción, otro objeto *Tren* puede manipular objetos *Persona* sin tener en cuenta sus propiedades ni métodos ya que sólo le interesa, por ejemplo, calcular la cantidad de personas que están viajando en él en ese momento, sin tener en cuenta ninguna otra información relacionada con dichas personas, tales como la altura, el nombre, el color de ojos, etcétera.

2) Clase: Es una plantilla para la creación de objetos de datos según un modelo predefinido. Las clases son utilizadas para representar conceptos o identidades. Las clases permiten abstraer los datos y sus operaciones asociadas al modo de una caja negra. Las clases se componen de elementos, llamados miembros, de varios tipos, campos de datos: almacenan el estado de la clase por medio de variables, estructuras de datos e incluso otras clases y métodos: subrutinas de manipulación de dichos datos. Ejemplo: La clase vehículo puede tener como propiedades: la placa, el tipo de motor, la ubicación, y puede tener como operaciones disponibles: ubicar al vehículo, asignar tipo de motor y asignar placa.

3) Objeto: es una representación abstracta de un ente de la vida real, este es interpretado por medio de atributos y métodos. Ejemplo: Un automóvil puede ser representado con atributos: color, marca, tipo, placa y modelo; además de métodos como: acelerar, prender luces y cambiar marcha.

4) Atributo: Los atributos describen el estado del objeto. Un atributo consta de dos partes, un nombre de atributo y un valor de atributo. Los objetos simples pueden constar de tipos primitivos, tales como enteros, caracteres, boolean, etc. Los objetos complejos pueden constar de pilas, conjuntos, listas, arrays, etc, o incluso de estructuras recursivas de alguno o todos de sus elementos. Ejemplo:

```
class Coche {

        public marca;

        public guardaMarca(parametro) {

                this.marca = parametro;

        }

}
```

En este ejemplo la variable marca es un atributo o característica del objeto coche.

5) Método: Es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase, como es el caso de los métodos de clase o estáticos, como a un objeto, como es el caso de los métodos de instancia. Análogamente a los procedimientos en los lenguajes imperativos, un método consiste generalmente de una serie de sentencias para llevar a cabo una acción, un juego de parámetros de entrada que regularán dicha acción o, posiblemente, un valor de salida (o valor de retorno) de algún tipo. Ejemplo: Métodos para hallar área o perímetro de una clase figura a partir de los datos que proporciona la misma

6) Encapsulación: Es un método estructurado para controlar el acceso a los atributos y los métodos de los objetos instanciados de una clase, esto garantiza la integridad del objeto. Ejemplo: un atributo definido como private solo puede ser accesado en la misma clase, por lo que para trabajar con un atributo private se generan métodos de set y get, garantizando que el que quiera accesar al atributo tendrá que usar alguno de los dos métodos dependiendo del uso que le quiera dar al atributo.

7) Herencia: La herencia es específica de la programación orientada a objetos, donde una clase nueva se crea a partir de una clase existente. La herencia (a la que habitualmente se denomina subclases) proviene del hecho de que la subclase (la nueva clase creada) contiene los atributos y métodos de la clase primaria. La principal ventaja de la herencia es la capacidad para definir atributos y métodos nuevos para la subclase, que luego se aplican a los atributos y métodos heredados. Esta particularidad permite crear una estructura jerárquica de clases cada vez más especializada. La gran ventaja es que uno ya no debe comenzar desde cero cuando desea especializar una clase existente. Como resultado, se pueden adquirir bibliotecas de clases que ofrecen una base que puede especializarse a voluntad.

Ejemplo:

```
public class Mamifero
 {
   private int patas;
   private String nombre;
   public Mamifero(String nombre, int patas)
   {
     this.nombre = nombre;
     this.patas = patas;
   }


 public class Perro extends Mamifero
   {
   public Perro(String nombre){
     super(nombre, 4);
   }
 }

 public class Gato extends Mamifero{
```

```
  public Gato(String nombre){
    super(nombre, 4);
  }
}
```

En el ejemplo la clase mamífero es la clase padre de las clases perro y gato, donde se ve que las clases hijas utilizan los atributos de la clase padre.

8) Polimorfismo: Se refiere a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía. Ejemplo: puede haber polimorfismo en una clase figura, con un dato de entrada puede retornar área de un círculo, con dos datos área de un cuadrado y con tres datos el área de un triángulo.

9) Diferencia entre tipo primitivo y de referencia: un dato de tipo primitivo almacena directamente un valor que este comprendido entre su rango y tipo de dato, mientras que al instanciar un dato de referencia este almacena la dirección de memoria que contiene el objeto que se creó. Ejemplo: al declarar una variable b de tipo Int lo hacemos de la siguiente forma: Int b=543234; Esta guarda directamente el valor 543234 que está comprendido entre el rango de un tipo de dato int, Por otro lado al instanciar un arreglo de int, lo haríamos de esta forma: int arreglo []=new int [7]; con el new le estamos indicando que cree una dirección de memoria para el objeto de tipo arreglo en una dimensión y la asocie con la variable de dato de referencia.

10) Diferencia entre usar private, protected, public, default: Existen diferentes modificadores de acceso a las clases y sus métodos como public (que permite el acceso a todo desde cualquier clase) y private (que no permite el acceso a ninguna clase externa) y existe además un modificador de acceso intermedio que no es ni public ni private, sino algo intermedio que se denomina como “acceso protegido”, expresado con la palabra clave protected, que significa que las subclases sí pueden tener acceso al campo o método. El modificador de acceso protected puede aplicarse a todos los miembros de una clase, es decir, tanto a campos como a métodos o constructores. En el caso de métodos o constructores protegidos, estos serán visibles/utilizables por las subclases y otras clases del mismo package. El acceso protegido suele aplicarse a métodos o constructores, pero preferiblemente no a campos, para evitar debilitar el encapsulamiento. En ocasiones puntuales sí resulta de interés declarar campos con acceso protegido. La sintaxis para emplear esta palabra clave es análoga a la que usamos con las palabras public y private, con la salvedad de que protected suele usarse cuando se trabaja con herencia.

MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
No especificado	Sí	Sí	No	No
private	Sí	No	No	No

11) Clase abstracta: Es una clase que no se puede instanciar, se usa únicamente para definir subclases, una clase es abstracta cuando uno o más de sus métodos no tiene implementación, las clases abstractas se utilizan cuando deseamos definir una abstracción que englobe objetos de distintos tipos y queremos hacer uso del polimorfismo. Ejemplo: Figura es una clase abstracta, porque no tiene sentido calcular su área, pero sí la de un cuadrado o un círculo, si una subclase de figura no redefine área, deberá declararse también como clase abstracta.

12) Interfaz: es un conjunto de métodos y constantes, estas no poseen un constructor por lo que no se pueden instanciar como tal, una clase que las implemente deberá redefinir los métodos de la interfaz pero un objeto de esta clase si se podrá instanciar. Ejemplo: una interfaz con un método por defecto de área y perímetro, una clase llamada cuadrado implementa la interfaz y redefine los métodos anteriormente mencionados para poder trabajar con objetos de tipo cuadrado.