**Tabla**

|  |  |
| --- | --- |
| **Polimorfismo** | Un concepto en la programación orientada por objetos que les permite a las clases tener varias formas y comportarse como sus superclases. |
| **Sobrescribir métodos** | Implementar métodos en una subclase que tiene el mismo prototipo (los mismos parámetros, nombre de método, y tipo de retorno) que otro método en la superclase. |
| **final** | Una palabra clave en Java usada para limitar las subclases a partir de la extensión de una clase, sobrescribiendo métodos o cambiando datos. |
| **inmutable** | Una propiedad de una case estática que hace que la clase no pueda ser extendida o que sus datos sean cambiados. |
| **Sobrecarga de métodos** | Implementar un método con el mismo nombre que otro método en la misma clase que tiene diferentes parámetros o un tipo de retorno diferente. |
| **Enlace Dinámico** | El proceso por el cual Java es capaz de determinar qué método invocar cuando los métodos han sido sobrescritos |
| **Abstract** | Una palabra clave en Java que permite que las clases sean extendidas, pero las clases no pueden ser instanciadas (construidas)y cuando se aplican los métodos, se indica que los métodos deberían ser implementados en todas las subclases de la clase. |

Punto #1. En eclipse…Este código ejecuta lo siguiente:

* Inside Class B's Method!
* Inside Class C's Method!
* Inside Class A's Method!

Punto #2.

Existen algunas diferencias entre las clases abstractas e interfaces Por ejemplo, las clases abstractas pueden tener tanto métodos abstractos como concretos, mientras que las interfaces solo pueden tener métodos abstractos. Además, una clase puede heredar de una sola clase abstracta, pero implementar múltiples interfaces. Las clases abstractas pueden tener constructores, lo cual no es posible en las interfaces.

Las clases abstractas se utilizan para establecer una base común para varias clases relacionadas que comparten algunos métodos abstractos y otros métodos concretos.

En cambio las interfaces se utilizan para definir un conjunto de métodos que cualquier clase que implemente la interfaz debe tener En algunas situaciones, puede ser beneficioso utilizar ambas estructuras juntas para crear jerarquías de clases más complejas con especificaciones de métodos compartidos.

Punto #3.

|  |  |
| --- | --- |
| **Código** | **¿Siempre compila, a veces compila, no compila?** |
| **a = new B();** | Siempre compilará porque B es un tipo de A y se puede crear una instancia de B y asignarla a una variable de tipo A. |
| **d = new C();** | No compilará porque no se puede crear una instancia de C o D, ya que C es abstracta y D hereda de ella. |
| **b.methodA();** | Siempre compilará porque B implementa A, lo que garantiza una implementación para methodA(). |
| **e.methodA();** | Siempre compilará porque E hereda de B, que implementa A, lo que significa que E también implementa methodA(). |
| **c = new C();** | Siempre compilará porque se puede crear una nueva instancia de C sin problemas. |
| **(D)c.methodC();** | No compila porque C es abstracta y no se puede crear una instancia directamente. |

Punto #4. Eclipse clase main y clase factorial.