**Conceptos de Programación Orientada a Objetos:**

* **Abstracción**: exponer sólo los detalles esenciales.
* **Encapsulamiento**: agrupar datos + las operaciones sobre esos datos => DAT o tipo de dato abstracto (Data Abstract Type).
* **Herencia**: derivar un DAT de otro, por ejemplo, el xenomorpho de la película *Alien* (buen spoiler).
* **Ocultar información**: ocultar detalles de implementación que pueden cambiar.
* **Polimorfismo**: una entidad soporta diferentes significados en diferentes contextos.
  + **Coercitivo**: cast de tipos.
  + **De inclusión**: herencia y sobreescritura (de métodos)
  + **De sobrecarga** (de métodos).
  + **Paramétrico**: generics de Java. :mrmeeseeks: Mr. Meeseeks!

L de SOLID = [Principio de Substitución de Liksov](https://es.wikipedia.org/wiki/Principio_de_sustituci%C3%B3n_de_Liskov)

**WORKING CLASS:**

6.1.1. Fundamentos de clase: tipos de datos abstractos (ADT) Un tipo de datos abstracto es una colección de datos y operaciones que funcionan con esos datos.

Las operaciones describen los datos al resto del programa y permiten que el resto del programa cambie los datos. La palabra "datos" en "tipo de datos abstractos" se utiliza de forma flexible. Abstract data types are exciting because you can use them to manipulate real-world entities rather than low-level, implementation entities.

ADT y clases

Los tipos de datos abstractos forman la base para el concepto de clases. En lenguajes que soportan clases, puedes implementar cada tipo de datos abstracto en su propia clase. Las clases normalmente involucran los conceptos adicionales de herencia y polimorfismo. Una forma de pensar en una clase es como un tipo de datos abstracto más herencia y polimorfismo.

6.2.1. Buena Abstracción. La interfaz de la clase debe ofrecer un grupo de rutinas que claramente van juntas. Una clase que presenta una abstracción pobre sería una que contuviera una colección de funciones diversas.

### Encapsulación

La encapsulación consiste en unir en tener las variables de clase declaradas como private para evitar que cuando instanciemos un objeto podamos acceder a ellas (por seguridad). ¿Y entonces como accedemos a los valores de esas variables de clase? Mediante los métodos getter y setter que declaramos anteriormente.

### ****Herencia****

La herencia es uno de los conceptos más cruciales en la **POO.** La herencia consiste en que una **clase** puede heredar las variables y los métodos de otra clase (la clase que hereda es llamada **superclase** o **clase padre**). Esto significa que una subclase, aparte de los atributos y métodos propios, tiene incorporados los atributos y métodos **heredados**de la superclase. De esta manera se crea una **jerarquía de herencia**.

Si partimos de la clase Automovil que creamos anteriormente podríamos crear clases hijas de Automovil, la cuales heredarán todo lo que contenga la clase Automovil.

### Polimorfismo

El polimorfismo es la habilidad de una función, método, variable u objeto de poseer varias formas distintas. Podríamos decir que **un mismo identificador comparte varios significados diferentes**.

El propósito del polimorfismo es implementar un estilo de programación llamado **envío de mensajes** en el que los objetos interactúan entre ellos mediante estos mensajes, que no son más que llamadas a distintas funciones.

Nótese que las versiones

fueron originalmente referidos como JDK (Java Development Kits) y más tarde se cambió de nombre a J2SE

(Plataforma Java 2, Edición Estándar).

**Entorno de ejecución de Java (JRE)**

**Java Runtime Environment (JRE) es el entorno de software en el que se ejecutan los programas Java.**

**La interfaz de programación de aplicaciones Java (API) es el conjunto de componentes Java preempaquetados y listos para usar agrupados en bibliotecas. Java.awt; Java.swing**

**Las clases normalmente se ensamblan en bibliotecas que se almacenan físicamente en archivos JAR (Java Archive).**

**Para ubicar las clases, el cargador de clases primero encontrará las bibliotecas correspondientes y luego cargará las clases según las necesite el programa (lo que se denomina carga bajo demanda).**

**Finalmente, el cargador de clases del sistema carga el código desde las ubicaciones especificadas en la variable de entorno CLASSPATH, que está definida por el sistema.**