Kevin Andrés Alvarez Herrera

202203038

Clases Abstractas

Una **clase abstracta** es una clase base para otras clases llamadas “clases concretas” o clases reales.

La finalidad de esta clase consiste en ocultar lo complicado de nuestro código y ofrecernos funciones de alto nivel, sencillas de utilizar para interactuar con nuestra aplicación sin conocer cómo funciona por dentro.

Con la abstracción captamos los comportamientos y atributos de un objeto, ocultando detalles y solo mostrando información esencial.

Una clase abstracta es una clase común que posee atributos y métodos, y tiene como mínimo un método abstracto, además puede contener métodos normales.

Esta clase no puede ser instanciada, solo puede heredarse, o sea no se puede usar para crear un objeto.

La creación del objeto se hará a través de sus clases hijas, las cuales están obligadas a implementar los métodos abstractos definidos en la clase principal.

Para poder implementar los métodos, la subclase debe hacerlo a través del uso de la palabra reservada ***extends***. Cuando se crea

la subclase la misma “extiende” esos comportamientos desde una clase padre, en este caso abstracta y así el objeto que se creará luego heredará todos los atributos y métodos definidos previamente.

Para definir una clase como abstracta, utilizamos la palabra reservada ***abstract***.

Una clase abstracta está restringida en su uso, no permite crear objetos, para la creación de los mismos se deben heredar las características y métodos a otra clase, conocida como clase hija o subclase.

Un método abstracto, sólo puede ser usado y estar definido dentro de una clase abstracta.

El método abstracto no tiene cuerpo y no están implementadas sus funciones, es obligatorio redefinir a través de una clase hija que lo hereda, esta es la clase “real” que proporcionará el cuerpo para dicho método.

La subclase es la encargada de implementar estas funciones, de manera obligatoria.

La clase abstracta sirve para implementar clases parcialmente.

Ejemplo de clase abstracta en Java

```
  public abstract class Felino {
    // clase abstracta

    public abstract void Alimentarse(); //metodo abstracto, vacio

}
```

Ejemplo de Clase que hereda o subclase en Java

```
  public  class Gato extends Felino
  {
public void Alimentarse()
    {
    System.out.println("El Gato come croquetas");
    }
  }
```

Ejemplo de Clase para crear el objeto en Java

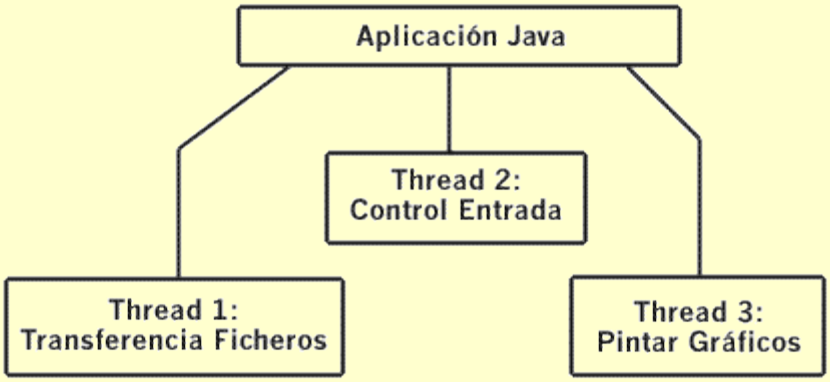
```
class Main{

        public static void main(String[] args) {
            Gato gatito = new Gato();
            gatito.Alimentarse();
            }
         }
```

Hilos de trabajo

Considerando el entorno *multithread* (multihilo), cada *thread* (hilo, flujo de control del programa) representa un proceso individual ejecutándose en un sistema. A veces se les llama *procesos ligeros* o *contextos de ejecución*. Típicamente, cada hilo controla un único aspecto dentro de un programa, como puede ser supervisar la entrada en un determinado periférico o controlar toda la entrada/salida del disco. Todos los hilos comparten los mismos recursos, al contrario que los *procesos*, en donde cada uno tiene su propia copia de código y datos (separados unos de

otros). Gráficamente, los hilos (*threads*) se parecen en su funcionamiento a lo que muestra la figura siguiente:



Hay que distinguir *multihilo* (multithread) de *multiproceso*. El *multiproceso* se refiere a dos programas que se ejecutan "*aparentemente*" a la vez, bajo el control del Sistema Operativo. Los programas no necesitan tener relación unos con otros, simplemente el hecho de que el usuario desee que se ejecuten a la vez.

Multihilo se refiere a que dos o más tareas se ejecutan "*aparentemente*" a la vez, dentro de un mismo programa.

Se usa "*aparentemente*" en ambos casos, porque normalmente las plataformas tienen una sola CPU, con lo cual, los procesos se ejecutan en realidad "*concurrentemente*", sino que comparten la CPU. En plataformas con varias CPU, sí es posible que los procesos se ejecuten realmente a la vez.

Tanto en el multiproceso como en el multihilo (multitarea), el Sistema Operativo se encarga de que se genere la ilusión de que todo se ejecuta a la vez. Sin embargo, la multitarea puede producir programas que realicen más trabajo en la misma cantidad de tiempo que el multiproceso, debido a que la CPU está compartida entre tareas de un mismo proceso. Además, como el multiproceso está implementado a nivel de sistema operativo, el programador no puede intervenir en el planteamiento de su ejecución; mientras que en el caso del multihilo, como el programa debe ser diseñado expresamente para que pueda soportar esta característica, es imprescindible que el autor tenga que planificar adecuadamente la ejecución de cada hilo, o tarea.

Actualmente hay diferencias en la especificación del intérprete de Java, porque el intérprete de *Windows '95* conmuta los hilos de igual prioridad mediante un algoritmo circular (round-robin), mientras que el de *Solaris 2.X* deja que un hilo ocupe la CPU indefinidamente, lo que implica la inanición de los demás.

Clase Thread

- **Las clases derivadas de `Thread` deben de incluir un método:**

```
public void run ()
```

  - ✓ Este método especifica realmente la tarea a realizar.

- **La ejecución del método `run` de un `Thread` puede realizarse concurrentemente con otros métodos `run` de otros `Thread` y con el método `main`.**