UNIDAD 0

ESTANDARES DE CODIFICACIÓN

DENIFICIÓN

- Conjunto de convenciones establecidas de ante mano (denominaciones, formatos, etc.) para la escritura de código.
- Son parte de las llamadas buenas practicas o mejores practicas, estas son un conjunto no formal de reglas, que han ido surgiendo en las distintas comunidades de desarrolladores con el paso del tiempo.
- Bien aplicadas pueden incrementar la calidad del código, notablemente.
- Para que funcionen las convenciones, cada persona que escribe software debe seguir la convención. Todos.

¿POR QUÉ?

- El 80% del coste del código de un programa va a su mantenimiento.
- Casi ningún software lo mantiene toda su vida el autor original.
- Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo.
- Si distribuimos nuestro código fuente como un producto, necesitamos asegurarnos de que esta bien hecho y presentado como cualquier otro producto.

INTERFACES

¿Qué son?

- Colección de métodos y propiedades abstractas, en las cuales se indica que se debe hacer, pero sin llegar a utilizarlos.
- En otras palabras, La interfaz define métodos , pero solo su nombre y parámetros , no su implementación. Llamados también Protocolos
- Contrato que una clase (la que implementa la interfaz) se compromete a cumplir
- Seguro para los clientes, que saben qué comportamiento se encontrarán en las clases que implementan la interfaz
- Puede incluir constantes.
- En java desde la versión 8, ya se permiten implementación de métodos por defecto

¿Para qué sirven?

- Organizar la programación.
- Obligar a que ciertas clases utilicen los mismos métodos (nombres y parámetros).
- Establecer relaciones entre clases que no estén relacionadas.
- Para que los objetos no relacionados se comuniquen entre sí

Ejemplo de Interfaz en Java

- La interfaz Comparable especifica un método compareTo() que las clases deben implementar.
- Esto significa que un método aparte de ordenamiento, por ejemplo, puede ordenar cualquier objeto que implemente la interfaz Comparable, sin tener que saber nada sobre la naturaleza interna de la clase (excepto que dos de estos objetos pueden compararse mediante compareTo())

ORGANIZACIÓN DE UN PROYECTO

¿POR QUÉ?

- La organización de un proyecto es importante por varias razones:
  - El 80% del coste del código de un programa va a su mantenimiento.
  - Casi ningún software lo mantiene toda su vida el autor original.
  - Un proyecto de software esta compuesto muchos paquetes lógicos que agrupan un conjunto de responsabilidades.
  - Si distribuimos nuestro código fuente como un producto, necesitamos asegurarnos de que esta bien hecho, organizado y presentado como cualquier otro producto.
- Por esto es una practica básica aplicar la separación de intereses a nuestro proyecto dividiendo responsabilidades en nuestro código.

Arquitectura de Software

- Es el conjunto de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y sus interfaces, con los que se compone el sistema.
- Existen buenas practicas con respecto a la arquitectura lógica de un sistema de software, las cuales han sido escritas en forma de patrones.
- Algunos patrones conocidos son:
  - El patrón de arquitectura de software en Capas (Layer)
  - El patrón arquitectónico Microkernel
  - El patrón de arquitectura Microservicios

Patrón de arquitectura software en Capas

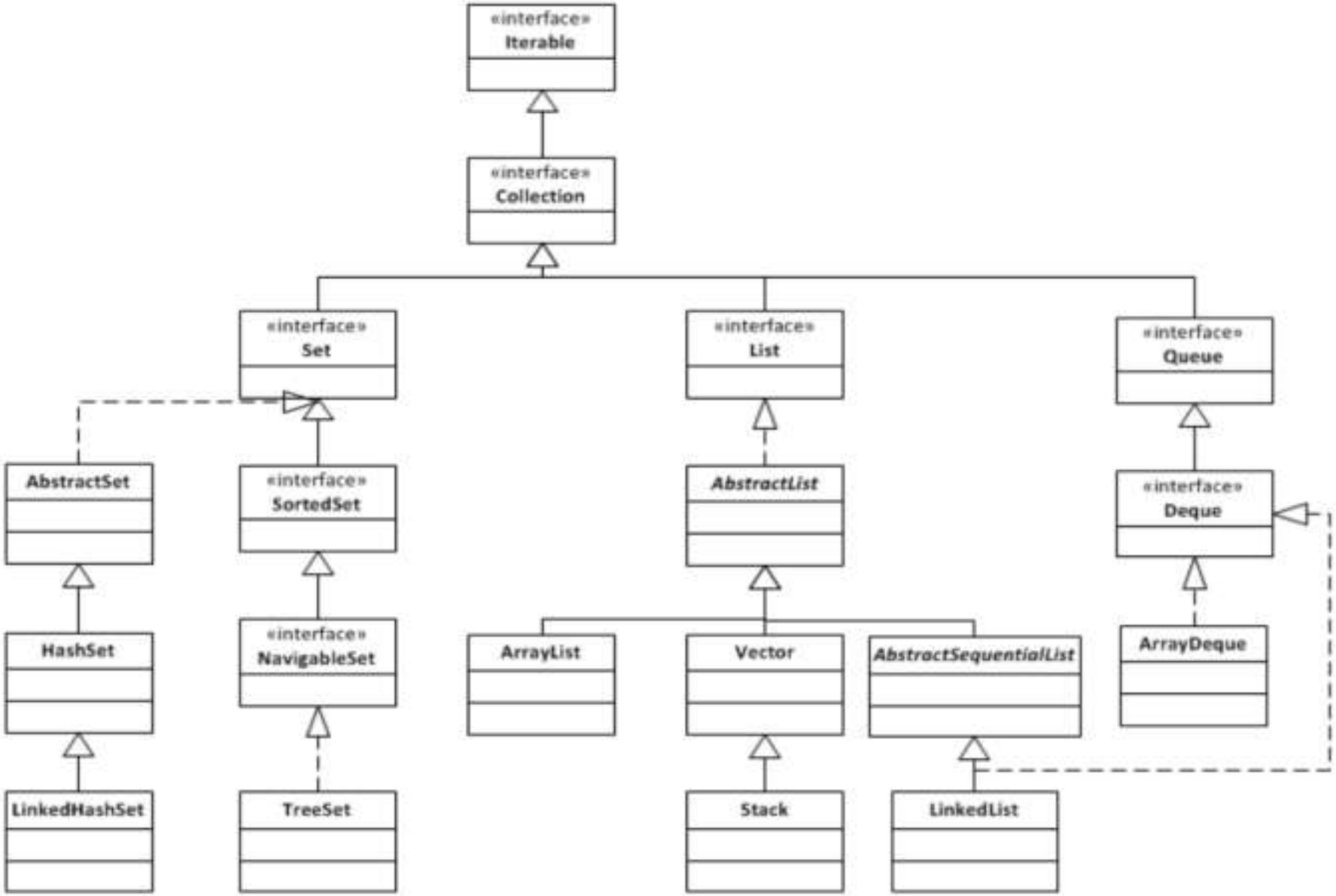
- Trata de separar la estructura lógica de un sistema en capas separadas, donde cada capa tiene responsabilidades distintas y a la vez relacionadas.
- La colaboración y acoplamiento es desde las capas más altas a las más bajas, evitando el acoplamiento de las capas bajas a las más altas.
- Define un modelo general de N-niveles.
- El objetivo y el número de capas varía de una aplicación a otra y entre dominios de aplicación.

Colecciones en Java

Colección en Java

- Son un núcleo de abstracciones de utilidad (interfaces) e implementaciones ampliamente útiles.
- Se encuentran en el paquete java.util
- Permite almacenar y organizar objetos de manera útil para un acceso eficiente.
- Las interfaces proporcionan métodos para todas las operaciones comunes y las implementaciones concretas especifican la decisión de las operaciones no permitidas.

Colecciones



Genéricos en Java

Introducción

- Java permite definir clases con variables o parámetros que representan tipos => Clases genéricas
- Esto es posible, cuando el tipo de un dato no afecta al tratamiento que se le va a dar a ese dato.
- Un mecanismo similar está disponible en lenguajes como C#, C++ o Ada.
- Antes se simulaba el genérico utilizando Object

Motivación (1/2)

- Supongamos que en un programa necesitamos una clase ParEnteros:

```
public class ParEnteros {
        private int a, b;
        public ParEnteros(int a, int b){
                this.a = a;
                this.b = b;
        }
        public ParEnteros swap (){
                return new ParEnteros(b, a);
        }
}
```

Motivación (2/2)

- Pero también necesitamos un clase ParCaracteres con los mismos métodos que la anterior.
- La implementación de los métodos es la misma, porque no depende del tipo del dato manipulado => abstraemos el tipo convirtiéndolo en un parámetro genérico T.

Clase Par Genérica

```
public class Par<T> {
    private T a, b;
    public Par(T a, T b){
        this.a = a;
        this.b = b;
    }
    public Par<T> swap (){
        return new Par<T>(b, a);
    }
}
```