Conceptos Java primera semana

Final

- Una clase **final** no puede ser heredada
- Un **final** en un método significa que no se le puede hacer @override
- Un **final** para objetos significa que no se puede cambiar la referencia a donde fue apuntada esa variable de referencia

Buffer de Inetgers

Solo en el caso de los Integers

- Hay un buffer donde existen los valores del -128 al 127, en este caso si dos variables de referencia tienen el mismo valor en este rango, apuntaran al mismo objeto
- Si esta fuera de este rango el valor, son distintos objetos

```
Integer uno = 100;
Integer dos = 100;

Integer tres = 200;
Integer cuatro = 200;

System.out.println(uno==dos);    //Da true
System.out.println(tres==cuatro);    //Da false
```

Sobre Constructores y Clases

- Se puede tener un método con el mismo nombre que el constructor, pero Es un método, esto lo sabemos porque si tiene un valor de retorno, No es un constructor
- En una clase, solo debe tener o un **super** o un **this**, No puede tener ambos y deben estar en la primera linea (Del constructor)
- Todas las clases heredan un constructor de la clase Object (al menos tiene uno).
- Se pueden tener múltiples constructores

- Si se tienen dos nombres de atributos iguales (Clase padre y clase hija) y se manda a llamar con el nombre de la clase padre aparecerá el dato de la clase padre

```
public class Canino {
    String nombre="Lobo";
    int edad=15;
}
//////////////////////////////////////////
public class Perro extends Canino{
    String nombre="Firulais";
    int edad=10;

    // Constructor
    public Perro(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}
//////////////////////////////////////////

public class Hola {
    public static void main(String[] args) {

        Canino p = new Perro("Kilian",12);
        System.out.println(p.nombre);    //Muestra Lobo
        System.out.println(((Perro)p).nombre); //Muestra kilian
    }
}
```

Modificadores de Acceso

- public, protected, private, default (JDK 1 - JDK 8)
- **public**: cualquiera tiene acceso
- **protected**: Jalan los del mismo paquete y acceso desde subclases independientemente de si se encuentran en el mismo paquete o en un paquete diferente.
- **private**: solo los métodos de la misma clase
- **Default**: Solo los del mismo paquete tiene acceso

	Default	Private	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	Yes	No	Yes	Yes
Same Package Non-Subclass	Yes	No	Yes	Yes
Different Package Subclass	No	No	Yes	Yes
Different Package N0on-Subclass	No	No	No	Yes

- Que un constructor sea private: significa que es Singleton

Static

- static → pertenece a la clase
- Se pueden usar antes de tener objetos creados
- Un método static no puede regresar algo que no sea static
- En Java, los bloques de inicialización estática se ejecutan antes de cualquier otra cosa en una clase.
- Se pueden importar solo elementos estaticos (metodos y propiedades)

```
import static com.curso.v1.Pato.*;
//En este caso importa todo lo estatico de la clase Pato

//igual se puede de esta forma
//Solo traeria la propiedad contador
import static com.curso.v1.Pato.contador;
```

Singleton

- El patrón Singleton es un patrón de diseño creacional que garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a esa instancia
- El patrón Singleton asegura que, independientemente de cuántas veces se solicite la creación de un objeto, siempre se devuelve la misma instancia.
- Se utiliza para controlar la creación de objetos y garantizar que haya una única instancia de una clase en todo el programa.
-

Herencia

- bajarle el tipo (Downcasting)

```
public static void main(String[] args) {

    Object cadena = new String("Java");
    System.out.println(((String)cadena).length());

    Ave ave1 = new Aguila("Aguila1");

    ((Aguila)ave1).volarAguila();
    ave1.volarAve(); //hereda
    ave1.volar(); //hereda
```

- Insteadof (Ejemplo)

```java
public static void main(String[] args) {

    Aguila a = getAguila();

    a.volarAguila();

    if (a instanceof AguilaReal)
        ((AguilaReal)a).volarAguilaReal();

    if (a instanceof AguilaCalva)
        ((AguilaCalva)a).volarAguilaCalva();

}

private static Aguila getAguila() {

    Aguila[] aguilas = {new Aguila("A")
            ,new AguilaReal("AR")
            ,new AguilaCalva("AC")};

    int i = new Random().nextInt(aguilas.length);

    return aguilas[i];
}
```

Herencia

Herencia de tipo → hacen referencia a las interfaces

Herencia de estado → No existe

Clases Abstractas

- Se puede ejecutar el método main desde una clase abstracta
- No se puede crear una instancia de una clase abstracta
- Puede tener métodos abstractos, o puede no tener ninguno
- Sirven mucho en la herencia, ya que evita que se instancia la clase mas general
- Un método abstracto no tiene cuerpo, osea que no tiene un comportamiento.

- Se esta obligado a definir el método abstracto de la clase padre, si no da error
- Si existe un metodo abstract la clase DEBE ser abstract
- Una clase abstracta puede implementar una clase abstracta. una clase abstracta puede implementar una interface. una interfaz no puede HEREDAR UNA CLASE ABSTRACTA
- Una interfaz pude heredar otra interfaz

Interfaz

- los atributos tienen que estar inicializados (son public, static y final).
- Los metodos son public y asbtract
- No pueden tener constructores
- No se pueden tener métodos con comportamiento
- Las interfaces son abstract, esto esta implícito en la firma de la interfaz
- cuando una interfaz hereda otra interfaz se pone con extends
- Una interfaz no puede heredar una clase abstracta ni una clase
- Una interfaz no puede heredar algo que no sea una interfaz
- A partir de java 8 puede tener metodos definidos que son static (tienen que ser public)
- A partir de java 8 puede tener metodos definidos que son default

Nombres de los archivos

Para el nombre de un archivo, o nadie es public, o solo un nombre de la clase se llama igual al archivo que es public (solo uno es public)

Bloques anonimos

- Primero se ejecutan los bloques anonimos por orden como se los va encontrando y despues se ejecuta el constructor de la clase

Cosas sueltas

Todas las variables locales necesitan estar inicializadas antes de usarlas. si no da error

El main original siempre tiene que ser void, acepta ser final, y tiene un arreglo de parametro

package-private → Default