TEMA 2

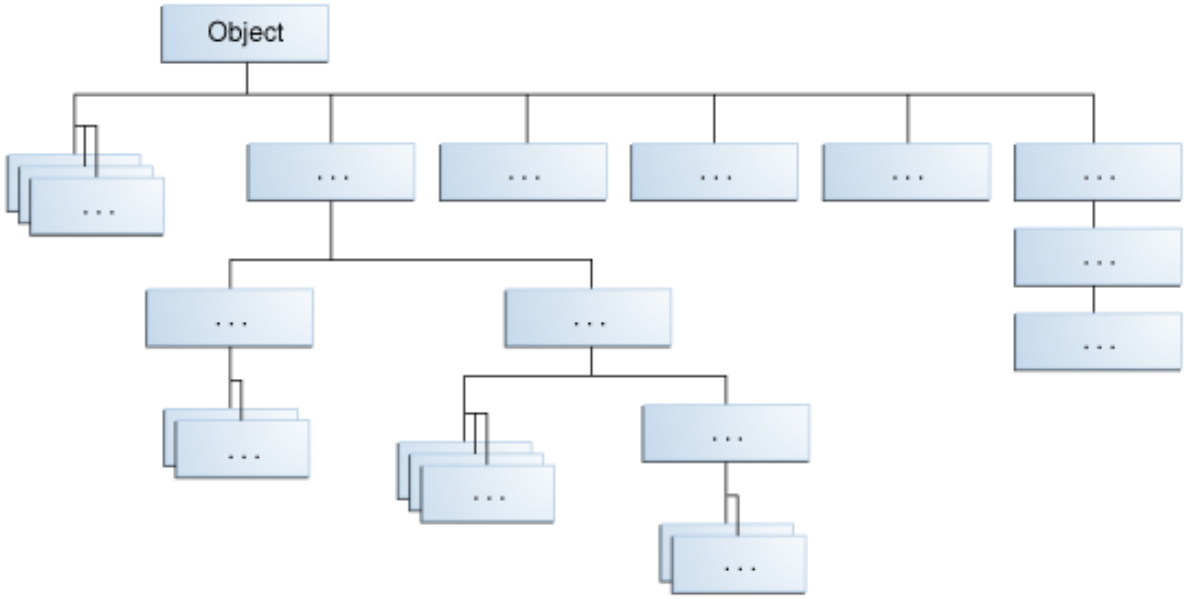
2.2 MÉTODOS DE OBJECT

2.2 MÉTODOS DE OBJECT

Clase Object

- Todas las clases son ella misma y *Object* en simultaneo (**polimorfismo**).
- Poseen métodos por defecto que no tenemos que escribir.

Ref: Oracle Java Tutorials (Subclasses)



2.2 MÉTODOS DE OBJECT

Clase Object

- Métodos por defecto en *Object* y que tienen todas las clases por defecto.
  - https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Ref: Oracle. (2014). *Class Object (Java Platform SE 8)*. Oracle.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
protected Object	clone()	Creates and returns a copy of this object.
boolean	equals(Object obj)	Indicates whether some other object is "equal to" this one.
protected void	finalize()	Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	getClass()	Returns the runtime class of this Object.
int	hashCode()	Returns a hash code value for the object.
void	notify()	Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll()	Wakes up all threads that are waiting on this object's monitor.
String	toString()	Returns a string representation of the object.
void	wait()	Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
void	wait(long timeout)	Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
void	wait(long timeout, int nanos)	Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

2.2 MÉTODOS DE OBJECT

El método Clone()

- **clone()** en *Object* realiza una copia superficial (*shallow copy*) del objeto.
- Respecto a sus atributos:
  - Los atributos primitivos (**int**, **double**, **boolean**, etc.) se copian por valor → se duplican correctamente.
  - Los atributos que son referencias a objetos o un vector, se copian la referencia → el clon y el original apuntan al mismo objeto en memoria.
- Requiere que incluya en la cabecera de la clase ***implements Cloneable***

```
Persona p1 = new Persona("Ana", 20);
Persona p2 = (Persona) p1.clone();
```

2.2 MÉTODOS DE OBJECT

El método Clone()

```
class Persona implements Cloneable {
    String nombre;
    int edad;
    int[] notas;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
        this.notas = new int[]{5, 7, 9};
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone(); // shallow copy
    }
}
```

2.2 MÉTODOS DE OBJECT

El método Clone()

```
public class Main {
    public static void main(String[] args) throws CloneNotSupportedException {
        Persona p1 = new Persona("Ana", 20);
        Persona p2 = (Persona) p1.clona();

        p1.edad=40
        System.out.println(p1.edad); // 40
        System.out.println(p2.edad); // 20

        p2.nombre = "Luis";
        System.out.println(p1.nombre); // Ana (p2.nombre ahora apunta a otro objeto)

        p2.notas[0] = 10;
        System.out.println(p1.notas[0]); // 10 (comparten el mismo array)
    }
```

2.2 MÉTODOS DE OBJECT

El método equals()

- **equals()** en *Object* ompara referencias en memoria, no el contenido de los objetos.
  - Es equivalente a usar ==.

```
Persona o1 = new Persona(Juan, 20);
Persona o2 = new Persona(Juan, 20);

System.out.println(o1.equals(o2)); // false (distintas referencias)
System.out.println(o1 == o2);      // false (distintas referencias)
```

2.2 MÉTODOS DE OBJECT

El método equals()

```
class Persona {
    String nombre;
    int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;  // misma referencia
        if (obj == null || getClass() != obj.getClass()) return false;
        Persona otra = (Persona) obj;
        return this.edad == otra.edad &&
               this.nombre.equals(otra.nombre);
    }
}
```

2.2 MÉTODOS DE OBJECT

El método equals()

```
public class Main {
    public static void main(String[] args) {
        Persona p1 = new Persona("Ana", 20);
        Persona p2 = new Persona("Ana", 20);

        System.out.println(p1 == p2);      // false (referencias distintas)
        System.out.println(p1.equals(p2)); // true (contenido igual)
    }
}
```

2.2 MÉTODOS DE OBJECT

El método equals()

```
public class Main {
    public static void main(String[] args) {
        Persona p1 = new Persona("Ana", 20);
        Persona p2 = new Persona("Ana", 20);

        System.out.println(p1 == p2);      // false (referencias distintas)
        System.out.println(p1.equals(p2)); // true (contenido igual)
    }
}
```

EJERCICIOS BÁSICOS

2.2 MÉTODOS DE LA CLASE "*OBJECT*"

USO BÁSICO DE EQUALS()

Ejercicio:

1. Crea una clase Persona con atributos nombre (String) y edad (int).
2. Crea dos objetos distintos con los mismos valores.
3. Compara con == y con equals().
4. Analiza la diferencia entre comparar referencias y contenidos.

SOBRESCRIBIR EQUALS()

Ejercicio:

1. Sobrescribe en Persona el método equals(Object o) para que compare por nombre y edad.
2. Vuelve a crear dos objetos con mismos valores y compara con equals().
3. Analiza qué ha cambiado respecto al comportamiento por defecto.

MÉTODO HASHCODE()

Ejercicio:

1. Añade el atributo dni a la clase Persona.
2. Sobrescribe hashCode() usando el campo dni.
3. Inserta dos personas con el mismo DNI en un HashSet y muestra cuántos elementos tiene.
4. Analiza por qué es importante mantener consistencia entre equals() y hashCode().

MÉTODO TOSTRING()

Ejercicio:

1. Sobrescribe en Persona el método toString() para que devuelva un texto como:

   "Persona[nombre=Ana, edad=20, dni=1234]".

2. Crea un objeto y muéstralo directamente con System.out.println().
3. Analiza qué ocurre si no sobrescribimos toString().

MÉTODO CLONE()

Ejercicio:

1. Haz que Persona implemente la interfaz Cloneable.
2. Sobrescribe clone() para devolver una copia superficial.
3. Declara un atributo array de enteros notas.
4. Clona un objeto y modifica el array de notas en el clon.
5. Analiza qué ocurre y por qué comparten el mismo array.

TEMA 2

2.3 CLASES ENUMERADAS

2.3 CLASES ENUMERADAS

Tipos enumerados

- Los tipos enumerados son clases que limitan la creación de objetos a los especificados explícitamente en la implementación de la clase

  Ej.: la clase Mes con únicamente 12 objetos que cada uno gestiona su número máximo de días (28, 30 o 31) y los literales en varios idiomas ("enero", "january")
- La única limitación respecto a una clase normal es que, si incorporan constructores, deben ser privados para evitar la creación de nuevos objetos distintos a los ofertados por la implementación;
- Es importante comprender que cada constante de enumeración es un objeto de su tipo de enumeración;
- La sentencia switch acepta expresiones de tipo enumerado y, en tal caso, las posibles constantes serán los objetos especificados en el enumerado.

```
enum <enumerado> {
<enumerado1>[(<argumentos>)],
…
<enumeradoN>[(<argumentos>)];

<definiciónAtributos>

<definiciónMétodos>
}
```

donde cada <enumeradoX> es **public static final** por definición

2.3 CLASES ENUMERADAS

Tipos enumerados

- No se pueden crear objetos enum explícitamente y, por lo tanto, no podemos invocar el constructor enum directamente. Sólo se hace en la declaración del tipo enumerado.
- Una enumeración puede definir constructores, agregar métodos y tener variables de instancia.
- ***Los tipos enumerados no soportan mecanismos de derivacion (herencia)***

```
//Uso de un constructor, una variable de instancia y un método.
enum Transporte{
    COCHE(60), CAMION(50), AVION(600), TREN(70), BARCO(20);
    private int velocidad; //velocidad típica de cada transporte
    //Añadir un onstructor
    Transporte(int s){velocidad=s;}
    //Añadir un método
    int getVelocidad(){return velocidad;}
}
```

EJERCICIOS BÁSICOS

2.3 CLASES ENUMERADAS

CREACIÓN DE UN ENUM

Ejercicio:

1. Crea un enum llamado DiaSemana con los valores: LUNES, MARTES, …, DOMINGO.
2. Declara una variable de este tipo y asígnale un valor.
3. Imprime el valor en pantalla.

ENUM CON ATRIBUTOS

Ejercicio:

1. Crea un enum llamado Mes con los 12 meses.
2. Cada mes debe guardar como atributo el número de días (28, 30 o 31).
3. Declara un método getDias() que devuelva ese valor.
4. Desde el main, imprime cuántos días tiene febrero.

ENUM EN UN SWITCH

Ejercicio:

1. Crea una variable del tipo DiaSemana.
2. Usa un switch para imprimir un mensaje distinto si es fin de semana o día laborable.
3. Reflexiona: ¿qué ventaja tiene usar enum en lugar de String o int para este caso?

MÉTODOS EN ENUM

Ejercicio:

1. Amplía el enum Mes para que cada mes tenga también un atributo con su nombre en inglés.
2. Declara un método getNombreIngles().
3. Desde el main, imprime los 12 meses con sus nombres en inglés.

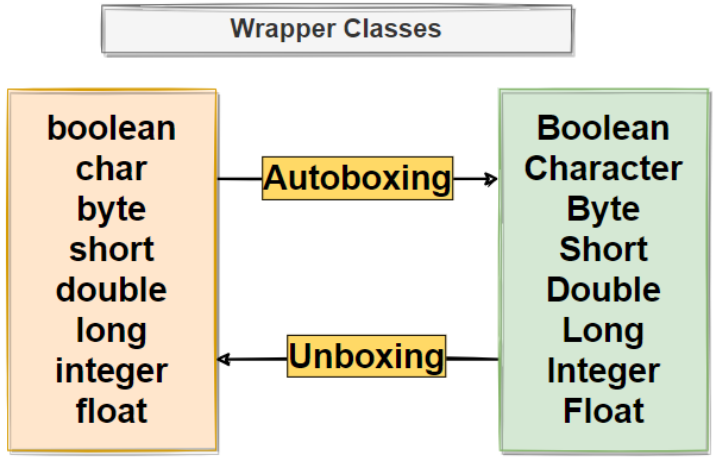
TEMA 2

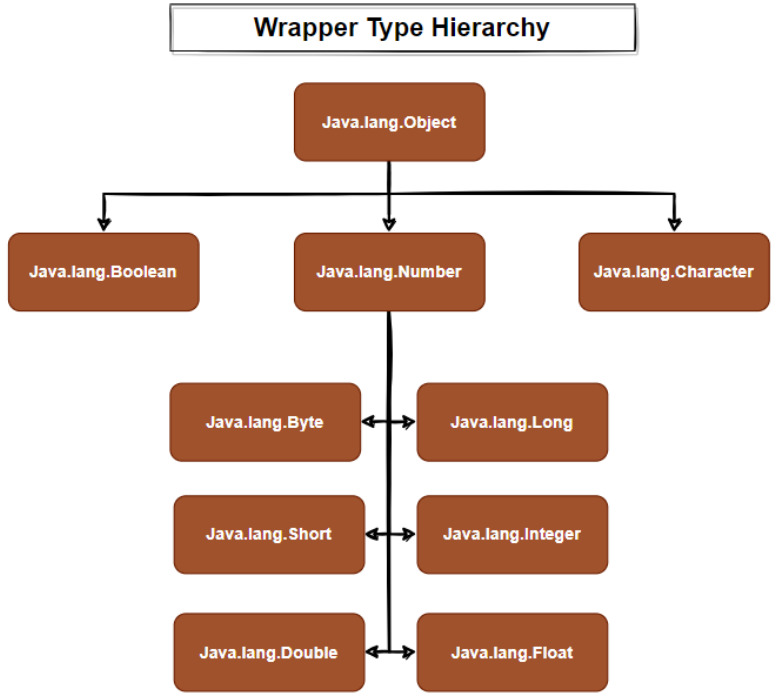
2.4 CLASES DE RECUBRIMIENTO

2.4 CLASES DE RECUBRIMIENTO

Clases de recubrimiento (wrapper classes)

- Sirven para para aglutinar funciones de conversión entre tipos primitivos y cadenas de caracteres junto con otras funciones auxiliares particulares de cada clase.
- Existen clases de recubrimiento para las siguientes clases:
  - Clase: Byte
  - Clase: Short
  - Clase: Integer
  - Clase: Long
  - Clase: Float
  - Clase: Double
  - Clase: Boolean
  - Clase: Character





2.4 CLASES DE RECUBRIMIENTO

Clases de recubrimiento (wrapper classes)

Clase Character

- public static String toString(char)
- public static char[] toChars(int)
- public static boolean isLowerCase(char)
- public static boolean isUpperCase(char)
- public static boolean isTitleCase(char)
- public static boolean isDigit(char)
- public static boolean isDefined(char)
- public static boolean isLetter(char)
- public static boolean isLetterOrDigit(char)
- public static char toLowerCase(char)
- public static char toUpperCase(char)
- public static char toTitleCase(char)

Clase Integer

- public static final int MIN_VALUE;
- public static final int MAX_VALUE;
- public static final int SIZE;
- public static String toString(int, int)
- public static String toHexString(int)
- public static String toOctalString(int)
- public static String toBinaryString(int)
- public static String toString(int)
- public static void getChars(int, int, char[])
- public static int stringSize(int)
- public static int parseInt(String, int)
- public static int parseInt(String)

2.4 CLASES DE RECUBRIMIENTO

Clases de recubrimiento (wrapper classes)

- Autoboxing (wraping automático): es el proceso de conversión automática que realiza el compilador de Java para que un tipo primitivo pase a ser un objeto utilizando para ello su clase de envoltura ("Wrapper").

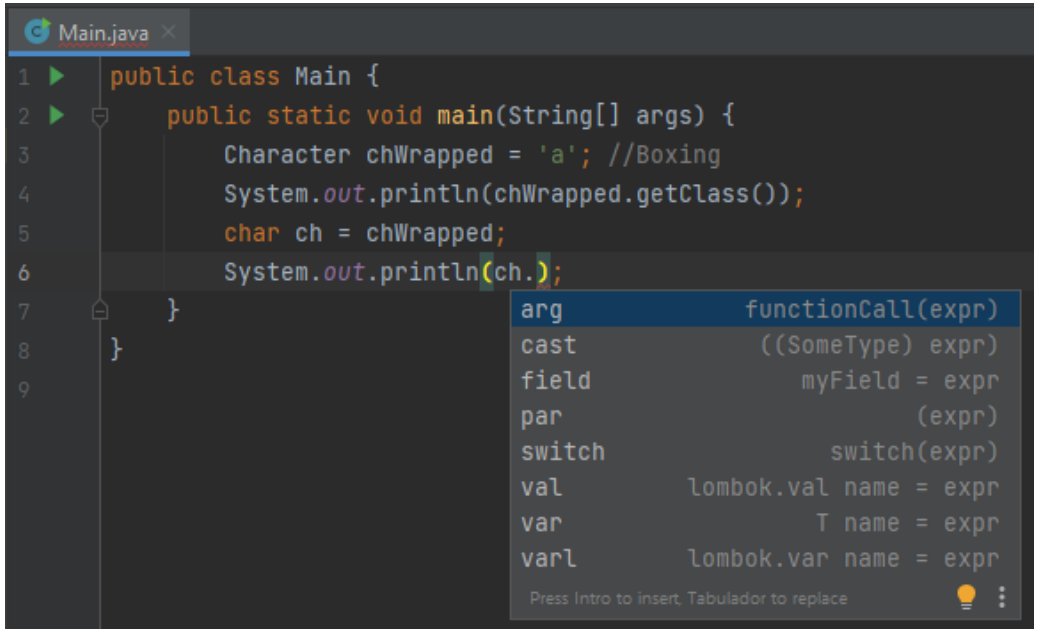
```
public class Main {
   public static void main(String[] args) {
       Character chWrapped = 'a'; //Autoboxing
       System.out.println(chWrapped.getClass());
   }
}
```

```
public class Main {
   public static void main(String[] args) {
       char ch = 'a';
       System.out.println(((Character) ch).getClass());
   }
}
```

2.4 CLASES DE RECUBRIMIENTO

Clases de recubrimiento (wrapper classes)

- Auto Unboxing (deswraping automático): es el proceso de conversión automática que realiza el compilador de Java para que un objeto de clave Wrapper pase a ser un tipo primitivo perdiendo con ello sus métodos.



2.4 CLASES DE RECUBRIMIENTO

Clases de recubrimiento (wrapper classes)

- Boxing (wraping manual): el proceso de conversión no automático que realizamos con el fin de pasar un tipo primitivo a un objeto mediante su clase de envoltura ("Wrapper").

```
1.  public class Main {
2.      public static void main(String[] args) {
3.          int numPrimitive = 6;
4.          Integer numWrapper = Integer.valueOf(numPrimitive);
5.          System.out.println(numWrapper.getClass().getName() + " ¿Es un objeto? " +
    (numWrapper instanceof Object));
6.      }
7.  }
```

2.4 CLASES DE RECUBRIMIENTO

Clases de recubrimiento (wrapper classes)

- Unboxing (deswraping manual): el proceso de conversión no automático que realizamos con el fin de pasar un tipo no primitivo (Wrapper) a un tipo primitivo.

```
1.  public class Main {
2.      public static void main(String[] args) {
3.          int numPrimitive = 6;
4.          Integer numWrapper = Integer.valueOf(numPrimitive);
5.          System.out.println(numWrapper.getClass().getName() + " ¿Es un objeto? " +
    (numWrapper instanceof Object));
6.          int numUnWrapped = numWrapper.intValue();
7.          System.out.println(numUnWrapped);
8.      }
9.  }
```

EJERCICIOS BÁSICOS

2.4 CLASES DE RECUBRIMIENTO

CONVERSIÓN BÁSICA CON INTEGER

Ejercicio:

1. Declara una variable int numero = 42;.
2. Convierte el número a un objeto Integer usando el método valueOf().
3. Convierte ese objeto a una cadena con toString().
4. Imprime el resultado.

AUTOBOXING Y UNBOXING

Ejercicio:

1. Declara una lista de enteros: List<Integer> lista = new ArrayList<>();.
2. Añade a la lista valores primitivos (int) del 1 al 5.
3. Recorre la lista con un bucle for-each y suma todos los valores.
4. Analiza cómo actúa el autoboxing y el unboxing en este ejemplo.

MÉTODOS DE CHARACTER

Ejercicio:

1. Declara una variable char c = 'A';.
2. Usa métodos de la clase Character para comprobar:
    1. Si es letra (isLetter)
    2. Si es mayúscula (isUpperCase)
    3. Convertirlo a minúscula (toLowerCase)
3. Imprime los resultados en pantalla.

CONVERSIÓN DE CADENAS A NÚMEROS

Ejercicio:

1. Declara una cadena String s = "1234";.
2. Convierte la cadena a un int usando Integer.parseInt().
3. Suma 10 al resultado e imprime el valor final.
4. Analiza qué ocurre si la cadena no contiene un número válido.

CONVERSIÓN DE CARACTERES

Ejercicio:

1. Pide por teclado un carácter.
2. Comprueba si es un dígito con Character.isDigit().
3. Si lo es, conviértelo a entero con Character.getNumericValue().
4. Muestra su valor en binario usando Integer.toBinaryString().