

## Tarea 1

### 1. ¿Qué es JDK, JRE y JVM?

- **JVM (Java Virtual Machine):** Ejecuta el bytecode .class, independiente del sistema operativo.
- **JRE (Java Runtime Environment):** Incluye la JVM y las bibliotecas necesarias para ejecutar programas.
- **JDK (Java Development Kit):** Incluye el JRE más herramientas de desarrollo como javac (compilador), javadoc, etc.

### 2. ¿Cuáles son los tipos de dato primitivos, cuanta memoria ocupa cada uno y cuáles son sus rangos?

Java tiene 8 tipos primitivos: byte, short, int, long, float, double, char y boolean. Cada uno tiene un tamaño en bits fijo y un rango específico de valores. Se usan para almacenar datos simples y son más eficientes que los objetos.

Tipo	Bits	Rango aproximado	Ejemplo
byte	8	-128 a 127	byte b=1;
short	16	-32,768 a 32,767	short s=2;
int	32	$-2^{31}$ a $2^{31}-1$	int i=3;
long	64	$-2^{63}$ a $2^{63}-1$	long l=4L;
float	32	1.4E-45 a 3.4E38	float f=1.5f;
double	64	4.9E-324 a 1.79E308	double d=1.5;
char	16	'\u0000' a '\uFFFF'	char c='A';
boolean	≈1B	true o false	boolean ok=true;

### 3. ¿Qué es el casteo(casting) y para que se utiliza?

El casting es convertir un valor de un tipo de dato en otro. Puede ser implícito (widening) cuando se convierte de un tipo más pequeño a uno más grande. Es explícito (narrowing) cuando el programador fuerza la conversión hacia un tipo más pequeño.

```
int i = 10;
long l = i;      // widening implícito
short s = (short) i; // narrowing explícito
double d = 3.14;
int j = (int) d;  // j = 3
```

#### 4. ¿Qué es una clase y un objeto?

Una clase es un molde que define atributos y métodos. Un objeto es una instancia real de esa clase que ocupa memoria y tiene estado. Las clases permiten organizar el código, y los objetos representan entidades concretas.

```
class Persona {
    String nombre;
    Persona(String n){ this.nombre = n; }
    void saludar(){ System.out.println("Hola, soy " + nombre); }
}

public class Demo {
    public static void main(String[] args){
        Persona p = new Persona("Ana"); // objeto
        p.saludar();
    }
}
```

#### 5. ¿Qué son las clases wrapper, para qué se usan y como se hacen las conversiones de datos primitivos a objetos mediante clases wrapper?

Son clases que envuelven los tipos primitivos (ejemplo: `int` → `Integer`). Se usan en colecciones genéricas, porque estas solo aceptan objetos. Permiten conversiones, métodos utilitarios y también pueden contener null.

```
int x = 5;
Integer obj = x;    // autoboxing
int y = obj;        // unboxing
int z = Integer.parseInt("42");
```

6. ¿Cuál es la diferencia al almacenar en la memoria una variable local y una variable tipo objeto? ¿Para qué sirve el garbage collector?

Las variables locales se almacenan en el stack, y desaparecen al terminar el método. Los objetos se almacenan en el heap, donde viven hasta que ya no tienen referencias. El Garbage Collector libera automáticamente esos objetos, evitando fugas de memoria.

7. ¿Qué son los arreglos y como se utilizan?

Son estructuras que guardan múltiples elementos del mismo tipo en posiciones contiguas. Tienen un tamaño fijo que se define al momento de crearlos. Permiten acceder a los elementos mediante un índice.

```
int[] nums = {1,2,3};  
for (int i : nums) System.out.println(i);
```

8. ¿Cómo se le pueden pasar argumentos al programa mediante el arreglo String[] args?

Es el parámetro que recibe el método main. Contiene los valores que se pasan al ejecutar el programa desde la línea de comandos. Sirve para trabajar con datos externos sin necesidad de escribirlos en el código.

```
public class Echo {  
    public static void main(String[] args){  
        for (String a : args)  
            System.out.println(a);  
    }  
}  
// Ejemplo: java Echo hola mundo
```

9. ¿Qué es un package, para que se usan y como se importan?

Un package es un espacio de nombres que organiza clases relacionadas. Ayuda a evitar conflictos entre nombres de clases. Para usar clases de otro paquete se importa con `import paquete.Clase;`

```
package com.ejemplo;  
import java.util.List;
```

## 10. ¿Qué es la clase String y cuáles son sus métodos más importantes?

Representa cadenas de caracteres y es inmutable, lo que significa que su valor no cambia. Cuando se manipula un String se crea una nueva cadena en memoria. Ofrece muchos métodos como `length()`, `substring()`, `equals()` y `split()`.

```
String s = " Hola Mundo ";  
System.out.println(s.trim().substring(0,4)); // "Hola"
```

## 11. ¿Para qué se usa la palabra reservada this?

`this` es una referencia al objeto actual. Se usa para distinguir entre atributos y parámetros con el mismo nombre. También sirve para llamar a otros constructores o devolver la instancia actual.

```
class Punto {  
    int x;  
    Punto(int x){ this.x = x; } // distingue campo y parámetro  
}
```

## 12. ¿Qué es la herencia?

Permite que una clase (subclase) herede atributos y métodos de otra (superclase). Facilita la reutilización del código y la creación de jerarquías lógicas. Con herencia, las subclases pueden extender o modificar el comportamiento de la clase base.

```
class Animal { void hablar(){ System.out.println("..."); } }  
  
class Perro extends Animal { @Override void hablar(){  
    System.out.println("Guau"); } }
```

## 13. ¿Qué es el polimorfismo?

Es la capacidad de un objeto de adoptar múltiples formas. Una referencia del tipo padre puede apuntar a diferentes subclases. El método ejecutado depende del objeto real y no del tipo de referencia.

```
Animal a = new Perro();  
a.hablar(); // "Guau"
```

14. ¿Qué es el @Override y para que sirven los métodos toString() y equals()? \

@Override indica que un método redefine otro existente en la superclase. toString() devuelve una representación en texto del objeto. equals() compara si dos objetos son equivalentes en contenido, no en memoria.

```
class Persona {  
    String nombre;  
    @Override public String toString(){ return nombre; }  
    @Override public boolean equals(Object o){  
        return nombre.equals(((Persona)o).nombre);  
    }  
}
```

15. ¿Qué es la sobrecarga de métodos?

Sucede cuando existen varios métodos con el mismo nombre pero con distinta lista de parámetros. Puede variar el número de argumentos, el orden o los tipos. Se resuelve en tiempo de compilación.

```
int suma(int a, int b){ return a+b; }  
double suma(double a, double b){ return a+b; }
```

16. ¿Qué es el manejo de excepciones?

Son eventos que ocurren cuando algo falla en la ejecución. Java maneja errores con try-catch-finally, evitando que el programa se detenga bruscamente. También se pueden crear y lanzar excepciones personalizadas con throw o throws.

```
try {  
    int r = 10 / 0;  
} catch (ArithmeticException ex) {  
    System.out.println("Error: " + ex.getMessage());  
} finally {  
    System.out.println("Siempre se ejecuta");  
}
```

## 17. ¿Para que se utiliza static?

Define miembros que pertenecen a la clase y no a instancias específicas. Se usa para constantes, métodos utilitarios, bloques de inicialización estáticos y el main. Los miembros estáticos se comparten entre todos los objetos de la clase.

```
class Util {  
    static final double PI = 3.14159;  
    static int sumar(int a,int b){ return a+b; }  
}
```