Sistemas Distribuidos - Tarea 1: Fundamentos de Java

1. ¿Qué es JDK, JRE y JVM?

- JDK: Las siglas significan Java Development Kit (Kit de Desarrollo de Java). Son las herramientas necesarias para el desarrollo de aplicaciones y componentes usando el lenguaje de programación Java. Estas herramientas permiten el desarrollo, monitoreo y depuración de cada programa.
- JRE: Las siglas significan Java Runtime Environment (Ambiente de Ejecución de Java). Es una capa de software que se ejecuta sobre el sistema operativo. Proporciona las clases de bibliotecas y otros recursos que un programa Java específico necesita para ejecutarse. No incluye las herramientas para desarrollar proyectos de Java.
- JVM: Las siglas significan Java Virtual Machine (Máquina Virtual de Java). Es el software que ejecuta las instrucciones de una clase de Java en cualquier plataforma.

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

Tipo de dato	Memoria	Rango
byte	8 bits	-128, 127
short	16 bits	-32768, 32767
int	32 bits	-2147483648, 2147483647
long	64 bits	-9223372036854775808, 9223372036854775807
float	32 bits	-3.402823e38, 3.402823e38
double	64 bits	-1.79769313486232e308, 1.79769313486232e308
chart	16 bits	'\u000', '\ufff'

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

En Java, es el proceso de convertir un tipo de dato a otro, se utiliza principalmente para realizar operaciones entre distintos tipos de datos.

4. ¿Qué es una clase y un objeto?

- Una clase es un plano o plantilla que define qué datos (atributos) y qué comportamiento (métodos) tendrá un tipo de entidad en el programa.
  Sirve para agrupar estructura y comportamiento: especifica qué información guarda y qué operaciones se pueden hacer sobre ella.
- Un objeto es una instancia concreta de una clase: ocupa memoria y tiene valores reales para los atributos.
  Cada objeto mantiene su propio estado (por ejemplo, nombre = "Ana", edad = 25) y puede ejecutar los métodos definidos por su clase.
  Se pueden crear muchos objetos desde la misma clase; son entidades independientes creadas a partir de la misma receta.
- Ejemplo:

```
public class Persona {
```

```
    private String nombre;
    private int edad;

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

    public void saludar() {
        System.out.println("Hola, soy " + nombre + " y tengo " + edad
+ " años.");
    }

    public static void main(String[] args) {
        // Crear (instanciar) objetos a partir de la clase Persona
        Persona p1 = new Persona("Ana", 25);
        Persona p2 = new Persona("Luis", 30);

        p1.saludar(); // Hola, soy Ana y tengo 25 años.
        p2.saludar(); // Hola, soy Luis y tengo 30 años.
    }
}
```

5. ¿Qué son las clases wrapper? ¿Para qué se usan y cómo se hacen las conversiones de datos primitivos a objetos mediante clases wrapper?

Las clases wrapper son aquellas clases que envuelven a los tipos de datos primitivos de Java. Cada primitivo (int, char, boolean, etc.) tiene su wrapper correspondiente (Integer, Character, Boolean, ...). Los wrappers se usan para tratar valores primitivos como objetos, esto nos permite tener null como valor, usar métodos o pasar a colecciones que requieren objetos. Se le conoce como “boxing” al uso de wrapper para transformar un dato primitivo a objeto, y hay dos formas de hacerlo; de forma automática, donde para un int i, tenemos un Integer I = i, o de forma manual, donde para un int i, podemos usar Integer.valueOf(i).

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

- Una variable local primitiva se almacena directamente en la pila (stack), guardando su valor inmediato, mientras que una variable tipo objeto en la pila solo guarda la referencia, mientras que el contenido real del objeto vive en el montón (heap).
- El garbage collector libera automáticamente memoria en el heap eliminando objetos que ya no tienen referencias activas.

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

En Java, un arreglo es una estructura de datos que permite almacenar un conjunto de elementos del mismo tipo en posiciones contiguas de memoria. Cada elemento se accede mediante un índice que empieza en 0 y termina en longitud - 1. Se utilizan para manejar colecciones fijas de datos (como números, cadenas u objetos) cuando se conoce de antemano su tamaño.

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

En Java, el arreglo String[] args en el método main recibe los argumentos que el usuario escribe al ejecutar el programa desde la consola. Cada argumento se guarda como una cadena en una posición del arreglo, en el orden en que se ingresó. Es posible acceder a estos argumentos como args[0], args[1], etc. y convertirlos al tipo de dato que sea necesario.

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

En Java, un package es un mecanismo para agrupar clases, interfaces y subpaquetes relacionados dentro de un mismo espacio de nombres. Se usan para organizar el código, evitar conflictos de nombres entre clases y facilitar la reutilización de librerías. Para usar una clase de otro paquete, se importa con la instrucción import y el nombre completo del paquete. Un ejemplo es el famoso `import java.util.*;`

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

En Java, la clase String representa secuencias de caracteres inmutables, es decir, no se pueden modificar después de crearse y cada cambio genera un nuevo objeto. Se utiliza para trabajar con texto y es una de las clases más usadas en el lenguaje, por eso tiene soporte directo con comillas dobles.

Los métodos más importantes son:

- **length():** Devuelve la cantidad de caracteres.
- **charAt(int index):** Obtiene el carácter en una posición.
- **substring(int inicio, int fin):** Extrae una parte del texto.
- **equals(String s):** Compara si dos cadenas son iguales.
- **equalsIgnoreCase(String s):** Compara ignorando mayúsculas/minúsculas.
- **toUpperCase() / toLowerCase():** Convierte a mayúsculas o minúsculas.
- **trim():** Elimina espacios al inicio y final.
- **replace(char a, char b):** Reemplaza un carácter por otro.
- **contains(CharSequence s):** Revisa si incluye una subcadena.
- **split(String regex):** Divide la cadena en un arreglo según un separador.

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

En Java, la palabra reservada “this” es una referencia al objeto actual (la instancia desde la cual se está ejecutando un método). Se usa principalmente para; diferenciar atributos de parámetros locales con el mismo nombre, llamar a otros constructores de la misma clase y pasar la referencia del objeto actual a otro método o clase.

12. ¿Qué es la herencia? (ejemplificar con un código breve y funcional)

La herencia que permite que una clase (llamada subclase o hija) reutilice y extienda los atributos y métodos de otra clase (llamada superclase o padre). Sirve para reutilizar código, especializar comportamientos y representar relaciones “es un”.

Ejemplo:

```
// Clase padre (superclase)
class Animal {
    public void comer() {
        System.out.println("El animal está comiendo.");
    }
}
```

```
// Clase hija (subclase) que hereda de Animal
class Perro extends Animal {
    public void ladrar() {
        System.out.println("El perro ladra: ¡Guau!");
    }
}

public class Main {
    public static void main(String[] args) {
        Perro p = new Perro();
        p.comer();   // Método heredado de Animal
        p.ladrar();  // Método propio de Perro
    }
}

Salida:
El animal está comiendo.
El perro ladra: ¡Guau!
```

13. ¿Qué es el polimorfismo? (ejemplificar con un código breve y funcional)

El polimorfismo es la capacidad de un objeto de adoptar múltiples formas, es decir, que una referencia de tipo padre pueda apuntar a objetos de diferentes subclases y ejecutar el comportamiento correspondiente. Se usa para escribir código más flexible, donde los métodos pueden trabajar con la superclase sin importar la subclase específica.
Ejemplo:

```
// Superclase
class Animal {
    public void hacerSonido() {
        System.out.println("El animal hace un sonido.");
    }
}

// Subclases que sobrescriben el método
class Perro extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("El perro ladra: ¡Guau!");
    }
}

class Gato extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("El gato maúlla: ¡Miau!");
    }
}
```

```
    public class Main {
        public static void main(String[] args) {
            Animal a1 = new Perro(); // Referencia tipo Animal → objeto
Perro
            Animal a2 = new Gato();  // Referencia tipo Animal → objeto
Gato

            a1.hacerSonido(); // Llama versión de Perro
            a2.hacerSonido(); // Llama versión de Gato
        }
    }

    Salida:
    El perro ladra: ¡Guau!
    El gato maúlla: ¡Miau!
```

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

- En Java @Override es una anotación que indica que un método en una subclase está sobrescribiendo un método de la superclase. Sirve para que el compilador verifique que realmente existe el método en la clase padre y evitar errores de escritura.
- toString() es un método heredado de la clase Object que devuelve una representación en texto del objeto. Por defecto muestra algo como Clase@hashcode, pero se sobrescribe para mostrar información útil.
- equals() también viene de Object y se usa para comparar objetos.
- Por defecto compara referencias (si apuntan al mismo objeto en memoria), pero se sobrescribe para comparar por contenido.

15. ¿Qué es la sobrecarga de métodos? (ejemplificar con un código breve y funcional)

En Java, la sobrecarga de métodos es cuando una clase tiene varios métodos con el mismo nombre, pero diferente lista de parámetros. Sirve para que un mismo comportamiento pueda funcionar con distintos tipos o cantidades de datos sin cambiar el nombre del método.
Ejemplo:

```
class Calculadora {
    // Suma dos enteros
    public int sumar(int a, int b) {
        return a + b;
    }

    // Suma tres enteros
    public int sumar(int a, int b, int c) {
        return a + b + c;
    }

    // Suma dos números decimales
    public double sumar(double a, double b) {
        return a + b;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Calculadora calc = new Calculadora();
        System.out.println(calc.sumar(2, 3));        // 5
        System.out.println(calc.sumar(1, 2, 3));     // 6
        System.out.println(calc.sumar(2.5, 3.5));    // 6.0
    }
}
```

16. ¿Qué es el manejo de excepciones? (ejemplificar con un código breve y funcional)

En Java, el manejo de excepciones es el mecanismo que permite detectar y controlar errores en tiempo de ejecución sin que el programa se detenga abruptamente.
Se hace usando las palabras clave; try, catch, finally y, opcionalmente, throw o throws.
Ejemplo:

```
public class Main {
    public static void main(String[] args) {
        try {
            int resultado = 10 / 0; // Esto genera ArithmeticException
            System.out.println("Resultado: " + resultado);
        } catch (ArithmeticException e) {
            System.out.println("¡Error! No se puede dividir entre
cero.");
        } finally {
            System.out.println("Este bloque se ejecuta siempre.");
        }
    }

    Salida:
    ¡Error! No se puede dividir entre cero.
    Este bloque se ejecuta siempre.
```

17. ¿Para que se utiliza static? (véase: https://www.youtube.com/watch?v=mvBX4-5-A4o)

En Java, la palabra reservada “static” se utiliza para definir variables de clase, las cuales serán el mismo valor para cada instancia. Esto nos ayuda a ahorrar memoria ya que sólo se guarda en un solo lugar, así, todas las instancias usan la misma variable.