Cuestionario

Rojo Mata Daniel

---

1. **Explica con tus propias palabras que son las referencias en java.**

   Las referencias son variables que almacenan direcciones de memoria en lugar de valores directos. Por ejemplo, cuando se crea un objeto en Java, la variable de referencia guarda la ubicación en memoria donde reside el objeto. Esto permite manipular y acceder a los objetos indirectamente a través de la referencia, lo que facilita la gestión de la memoria y el trabajo con estructuras de datos complejas.

2. **¿Cuáles son las diferencias de las referencias en Java con los punteros en otros lenguajes de programación como C?**

   En Java, las referencias están fuertemente tipadas y gestionadas por un recolector de basura, lo que garantiza la seguridad del tipo y automatiza la gestión de memoria. Por otra parte, los punteros en C son genéricos y requieren una gestión manual de la memoria por parte del programador. Además, las referencias en Java pueden ser nulas, indicando la falta de objeto, mientras que en C, los punteros no se inicializan automáticamente en *NULL*, lo que puede llevar a problemas de acceso no deseado a la memoria.

3. **Explica en tus propias palabras que es el recolector de basura de Java?**

   Es un componente que se encarga de gestionar automáticamente la memoria del programa. Por ejemplo, algunos objetos pueden volverse inaccesibles porque ya no hay referencias que apunten a ellos. El recolector de basura identifica y libera automáticamente la memoria ocupada por estos objetos inaccesibles para que pueda ser utilizada por otros objetos en el futuro.

4. **¿Qué pasa si en el método main de la case Referencias ejecutamos las siguientes instrucciones considerando los objetos ya declarados? Justifica tu respuesta considerando las referencias de los objetos.**

```
1       System.out.println( p1 );
2       System.out.println( c1 );
3       System.out.println( p2 );
```

Si las variables $p1$, $c1$ y $p2$ son referencias a objetos que han sido inicializados previamente, las instrucciones previas imprimirán las direcciones de memoria a las que apuntan estas referencias. Por ejemplo, si p1 es una referencia a un objeto de tipo *Persona*, la línea de código 1 imprimirá la dirección de memoria de este objeto.

5. **Considerando que: Persona p2 = new Persona("pepe"); y Persona p3 = new Persona("pepe"); ¿Qué regresa p2.equals(p3)? Justifica tu respuesta.**

   Por defecto, el método *equals()* en la clase *Object* compara las referencias de los objetos, es decir, verifica si dos referencias apuntan al mismo objeto en memoria. En el caso de las variables p2 y p3, aunque contienen objetos con el mismo contenido ("pepe"), son dos objetos diferentes en memoria porque se crearon con new Persona("pepe") por separado. Es por ello que *p2.equals(p3)* es *false*.

6. **Considerando que: Coche c1 = new Coche(çhevy"); y Coche c2 = new Coche('chevy'); ¿Qué regresa c1.equals(c2)? Justifica tu respuesta.**

   Si la clase Coche no ha sobrescrito el método equals(), entonces c1.equals(c2) se comportará de la misma manera que en la clase Object. En este caso, equals() comparará las referencias de los objetos y devolverá false porque c1 y c2 son dos objetos diferentes en memoria, a pesar de que contienen el mismo nombre (çhevy").

7. **Considerando la clase Persona ¿Por qué no se modifica el valor de x en el siguiente código? Justifica tu respuesta.**

   Cuando se pasa x como argumento al constructor de la clase Persona, se pasa una copia del valor de x, no la variable en sí. Esto significa que el valor original de x (que es 10 en este caso) se copia en un nuevo espacio de memoria para ser utilizado por el objeto p4. Cuando se imprime p4.edad, se imprime el valor copiado de x, que es 10. Luego, cuando se modifica p4.edad a 15, esto no tiene ningún efecto en la variable x fuera del objeto p4. Las modificaciones en p4.edad no afectan a x fuera del objeto p4.

8. **Explica con tus propias palabras qué es el paso por referencia y da un ejemplo donde se use.**

   Paso por referencia es un concepto que implica que, al pasar un argumento a una función o método, en realidad se está pasando la dirección de memoria (la referencia) del objeto, en lugar del valor real del objeto en sí. Esto significa que cualquier modificación realizada en el objeto dentro de la función se reflejará fuera de esa función, ya que ambos están apuntando a la misma ubicación de memoria.

```
1       public static void main(String[] args) {
2           Persona persona = new Persona();
3           persona.nombre = "AntiguoNombre";
4
5           cambiarNombre(persona);
6
7           System.out.println("Nombre después del cambio: " + persona.nombre);
8       }
```

   El método *cambiarNombre*() modifica el atributo nombre del objeto persona. Debido al paso por referencia, los cambios realizados dentro de *cambiarNombre*() se reflejan fuera de la función y se imprime 'NuevoNombre' en la salida.

9. **Para que sirve null en el contexto de las referencia de Java. Da un ejemplo donde algo haga referencia a null**

   Representa la ausencia de un objeto o la falta de valor. Las variables de referencia que se establecen en null no apuntan a ningún objeto en la memoria.

```
1           class Persona {
2               String nombre;
3
4               public Persona(String nombre) {
5                   this.nombre = nombre;
6               }
7           }
```

```
8
9           public static void main(String[] args) {
10              Persona persona = new Persona("Ejemplo");
11
12              // persona contiene una referencia a un objeto Persona
13              System.out.println("Nombre: " + persona.nombre);
14
15              // Ahora establecemos la referencia a null,
16              // liberando el objeto anterior
17              persona = null;
18
19              // Esto causará un error
20              System.out.println("Nombre: " + persona.nombre);
21          }
```

En el ejemplo anterior se inicializa un objeto Persona y luego se establece la referencia persona a *null*. Si se intenta acceder a *persona.nombre* después de establecerla como null, se producirá un *NullPointerException* porque null no apunta a ningún objeto y no se puede acceder a las propiedades de un objeto nulo.

10. **¿Qué es lo que se imprimirá en la terminal? Justifica tu respuesta**

```
1           Object [] arreglo = new Object [5];
2           arreglo [0] = new Persona ( " jose " );
3           arreglo [3] = new Coche ( " Mustang " );
4               for ( int i = 0; i < arreglo . length ; i ++){
5                   System.out.println( arreglo [ i ]);}
```

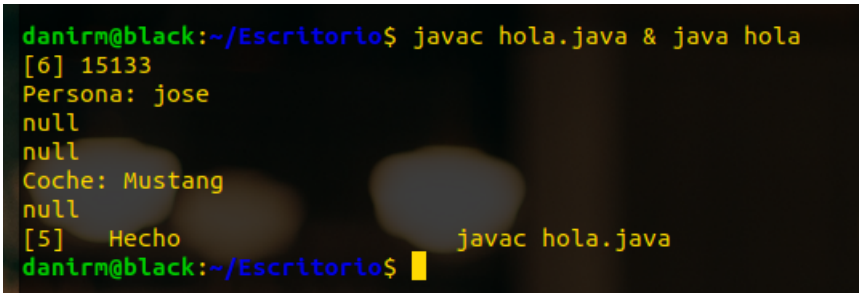
Para realizar lo anterior, se hizo un código que trate de simular la impresión del fragmento dado de la siguiente manera:

```
1       class Persona {
2           String nombre;
```

```
3           public Persona(String nombre) {
4               this.nombre = nombre;
5           }
6
7           public String toString() {
8               return "Persona: " + nombre;
9           }
10      }
11
12      class Coche {
13          String modelo;
14          public Coche(String modelo) {
15              this.modelo = modelo;
16          }
17
18          public String toString() {
19              return "Coche: " + modelo;
20          }
21      }
22
23      public class hola {
24          public static void main(String[] args) {
25              Object[] arreglo = new Object[5];
26              arreglo[0] = new Persona("jose");
27              arreglo[3] = new Coche("Mustang");
28
29              for (int i = 0; i < arreglo.length; i++) {
30                  System.out.println(arreglo[i]);
31              }
32          }
```

```
33          }
```

La impresión en terminal es la siguiente:



Se imprime null por los 'elementos' no existentes en el arreglo.