Memoria

Opción Múltiple

1. La línea de código `int numero = 5;`
    a. Crea un objeto `Integer` en el heap y guarda la referencia en `numero`.
    b. Crea un objeto `int` en el heap y guarda la referencia en `numero`.
    c. Crea otra variable de tipo `int` en el stack y guarda la referencia en `numero`.
    d. Crea la variable `numero` en el stack y guarda el valor 5.

2. La línea de código `Integer numero = 5;`
    a. Crea un objeto `Integer` en el heap y guarda la referencia en `numero`.
    b. Crea un objeto `int` en el heap y guarda la referencia en `numero`.
    c. Crea un objeto `Integer` en el stack y guarda la referencia en `numero`.
    d. Crea la variable `numero` en el stack y guarda el valor 5.

3. Dado el siguiente código, ¿qué afirmación es correcta?

```
public static void aprobarMateria(Alumno alumno){
    alumno.aprobar("AYED", 7);
}

public static void main(String[] args) {
    Alumno alumno = new Alumno();
    aprobarMateria(alumno);
}
```

    a. El objeto Alumno se pasa por copia, por lo que cualquier cambio dentro del método no se verá reflejado.
    b. El objeto Alumno se pasa por referencia, por lo que cualquier cambio dentro del método se verá reflejado.
    c. La variable alumno se pasa por copia, creando además una copia del objeto al que hace referencia.
    d. La variable alumno se pasa por copia, por lo que ambas variables hacen referencia al mismo objeto Alumno.

Verdadero / Falso

1. Para usar el heap es obligatorio usar la palabra reservada `new`, creando en el proceso un objeto.
    a. Verdadero
    b. Falso

2. El *Garbage Collector* (GC) debe ser llamado explícitamente para liberar la memoria dinámica no utilizada.
    a. Verdadero
    b. Falso

3. El siguiente código:

```
public static void sumar(Integer[] arreglo) {
    for (int i = 0; i < arreglo.length; i++) {
        arreglo[i] += 1;
    }
}

public static void main(String[] args) {
    Integer[] arreglo1 = {1, 2, 3, 4, 5};
    Integer[] arreglo2 = arreglo1;
    sumar(arreglo2);
    System.out.println(Arrays.toString(arreglo1));
}
```

   Imprime por pantalla `[1, 2, 3, 4, 5]`.
    a. Verdarero
    b. Falso

Desarrollo

1. Dibujar el mapa de memoria para cada uno de los instantes, indicando en qué momento actúa el Garbage Collector.

```
Fecha fechaNacimiento = new Fecha(20,6,2005);
Fecha fechaPatria = new Fecha(25,5,1810);
Persona juan = new Persona("Juan", fechaNacimiento);
```

```
Feriado revolucion = new Feriado(“Revolucion de Mayo”,
fechaNacimiento);
// Instante 1


revolucion.cambiarFecha(fechaPatria);
fechaNacimiento = new Fecha(18,8,2006);
Persona maría = new Persona(“Maria”, fechaNacimiento);
fechaNacimiento = null;
// Instante 2

Persona hermano = new Persona(“Jorge”, fechaPatria);
maria = null;
// Instante 3
```

2. Realice el seguimiento de memoria para el siguiente código, diagramando el estado final del stack y del heap. Considere que la lista es doblemente enlazada, y que el *Garbage Collector* (GC) no liberó ningún objeto no referenciado.

```
public static void main(String[] args) {
    Lista<Integer> lista = new Lista<>();
    Integer i;
    for (i = 1; i <= 5; i++) {
        lista.agregar(i);
    }
    lista.eliminar(2);
    lista.agregar(6, 0); // dato, indice.
    lista.eliminar();
}
```

Notas: La variable `i` está para que la diagramen en el stack, apuntando finalmente al número 5 de la lista. Es doblemente enlazada para que diagramen el anterior y siguiente, y el primero y último en la lista. Como es un template se tiene que usar `Integer`, y hay que diagramarlo en el heap porque son objetos.