Ejercicios Teóricos

1. Dado el código mostrado en 1, mencione qué pasa y justifique su respuesta.

```
1 public class Reactivo1 {
2    public static void main(String args[]) {
3       int a[] = {1,2,3,4,5,6};
4       int i = a.length - 1;
5
6       while(i>=0){
7          System.out.print(a[i]);
8          i--;
9       }
10   }
11 }
```

Figure 1: Código 1

   (a) 123456

   (b) 65432

   (c) Se lanza una excepción en tiempo de ejecución

   (d) 654321

   Justificación: El valor de la variable i es inicializado con el número máximo de elementos del arreglo a menos 1, es decir, $i = 6 - 1 = 5$. Una vez entrando al ciclo while, se imprimirá el contenido del arreglo del elemento a[5] al a[0]. Todo será impreso en una sola línea ya que la función print no incluye salto de línea.

2. Dado el código mostrado en 2, mencione qué pasa y justifique su respuesta.

   (a) Error en tiempo de ejecución

   (b) 0

   (c) Error

   (d) El programa no compila

```java
public class Reactivo2 {
   public static void main(String args[]) {
      int x = 0, y = 10;

      try {
         y/=x;
      }
      System.out.println("División entre 0");
      catch(Excepcion e) {
         System.err.println("Error");
      }
   }
}
```

Figure 2: Código 2

(e) División entre 0

Justificación: El programa no compila ya que la estructura del bloque `try-catch` es incorrecta. Después de especificar el inicio del bloque del que se desean atrapar todas sus posibles excepciones, `try{...}`, debe seguir al menos un bloque `catch(Excepcion){...}`.

3. Dado el código mostrado en 3, mencione qué pasa y justifique su respuesta.

```java
public class Reactivo3 {
   public static void main(String args) {
      float resultado = (5 + 4) * 3 / 2 + 1;

      System.out.println("Resultado de la operación: " + resultado);
   }
}
```

Figure 3: Código 3

(a) Error en tiempo de ejecución

(b) Imprime Resultado de la operación: 14.0

(c) Imprime Resultado de la operación: 14

(d) Error de compilación

(e) Ninguna de las anteriores

Justificación: Dado que la línea 2 del código mostrado en 3 hace mención a la función `public static void main(String)`, ésta no coincide con la signatura del método principal `public static void main(String[])` requerida por Java para arrancar el programa. Dado que no es obligatorio que una clase declare esta función, no se presentan errores de compilación, ni en tiempo de ejecución.

4. Dado lo siguiente, seleccione las tres opciones correctas y justifique su respuesta. A y E son clases.
   B y D son interfaces.
   C es un clase abstracta.

   (a) La clase F implementa B,D{ }
   (b) La clase F implementa B{ }
   (c) La clase F extiende A,E{ }
   (d) La clase F extiende E{ }
   (e) La clase F implementa B,C{ }

   Justificación:

   El inciso a Es correcto porque en Java una clase puede implementar varias interfaces.

   El inciso b Es correcto, ya que, en especial, una clase puede implementar una sola interfaz.

   El inciso c Es incorrecta, debido a que Java no permite herencia múltiple.

   El inciso d Es correcto porque una clase puede extender a otra clase (solo una).

   El inciso e Es incorrecto porque un clase no puede implementar una clase abstracta. **Más bien una clase puede extender a una clase abstracta para redefinir sus métodos abstractos**, de tal forma que puedan implementarse.

5. Dado el código mostrado en 4, mencione qué pasa y justifique su respuesta.

```java
public class Clase {
   public static void main(String []args) {
      int a[] = {1,7,034,98,021,89};
      int b[][] = { {1,27,4},{19,6,3},{21,98} };

      System.out.println(a[3] == b[2][1]);
      System.out.println(b[1] == b[2]);
      System.out.println(b[0][3] == a[3]);
   }
}

```

Figure 4: Código 4

(a) true
true
false

(b) true
Error en tiempo de ejecución
Error en tiempo de ejecución

(c) true
false
Error en tiempo de ejecución

(d) true
false
true

Justificación: La primera comparación consta de los elementos a[3]=98 con el elemento b[2][1]=98, al ser ambos iguales se imprime `true`. En la segunda se compara b[1] y b[2], como b es un arreglo bidimensional (un arreglo de arreglos), entonces b[1] y b[2] son arreglos y por lo tanto, objetos; de esta forma al compararlos con `==` es comparar las referencias de memoria de dichos objetos, y al no hacer referencia a las mismas localidades de memoria, imprime `false`. La tercera se trata de acceder al cuarto elemento del primer arreglo de b (todos los índices en Java comienzan en 0), por lo que se lanza un error en tiempo de ejecución

al intentar a un índice que no existe. Ver 5

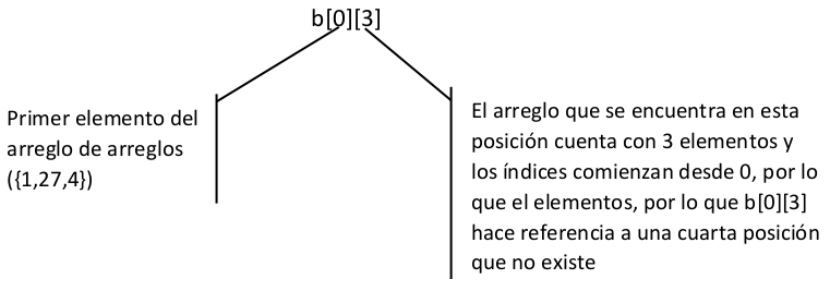

Figure 5: Ejemplo de uso incorrecto de índices en arreglos

6. Dados los códigos mostrados en la figura 6 y asumiendo que todos se encuentran en el mismo paquete; describa a detalle qué es lo que sucede. Indique cuál es la diferencia entre las líneas 2 y 6 de la clase Reactivo5; y si hay alguna utilidad práctica en ellas (¿Qué se logra?).

   La función declarada en la línea 2 del código mostrado en 6, espera cualquier objeto que **implemente** la interfaz `IFunciones`, sin importar su clase padre, es decir, se asegura que el argumento cumpla con el contrato descrito en la interfaz. Su utilidad es permitir generar aplicaciones con una arquitectura de acoplamiento holgado (principio de inversión de control (IoC)). Coloquialmente podría responder a la premisa "quiero a alguien que cumpla con el trabajo, ¡quien sea!".

   En el caso de la función declarada en la línea 6 del mismo código, se espera un objeto que sea subclase de `AbstractFunciones`, lo cual permite asegurar un "tipo o categoría" de objeto en particular. Su utilidad es permitir el reuso de funciones aplicadas a distintos objetos de una clase padre en común, de la que no es posible generar una instancia. i.e., `AbstractFunciones objeto = new AbstractFunciones()` no es permitido por el compilador.

7. De la siguiente lista ¿cuáles son identificadores válidos en Java?

   (a) $aluda
   (b) Saluda
   (c) 5saluda
   (d) @saluda

```
public interface IFunciones {
   public void saluda(String nombre);
}
// fin IFunciones

public class FuncionesImpl implements IFunciones{
   @Override
   public void saluda(String nombre) {
      System.out.println("Hola " + nombre +
         ". Esta es la implementación de una interfaz");
   }
}
// fin FuncionesImpl

public abstract class AbstractFunciones {
   public abstract void saluda(String nombre);
}
// fin AbstractFunciones

public class Funciones extends AbstractFunciones{
   @Override
   public void saluda(String nombre) {
      System.out.println("Hola " + nombre +
         ". Esta es la implementación de una clase abstracta");
   }
}
// fin Funciones
```

```
1 public class Reactivo5 {
2    private void metodo(IFunciones funciones) {
3       funciones.saluda("Bruno Díaz");
4    }
5
6    private void metodo(AbstractFunciones funciones) {
7       funciones.saluda("Bruno Díaz");
8    }
9
10   public static void main(String[] args) {
11      System.out.println("Llamado a interfaz");
12      Reactivo5 r5 = new Reactivo5();
13
14      r5.metodo( new Funciones() );
15      r5.metodo( new FuncionesImpl() );
16   }
17 }
```

Figure 6: Clases Abstractas e Interfaces

(e) _aluda

8. Describa cuál es la diferencia entre *String* y *StringBuffer*. De algunos ejemplos que ayuden a clarificar su respuesta.

   Los objetos String son *inmutables* y los objetos StringBuffer son *mutables*. Eso implica que cada vez que modificamos un `String`, se crea un objeto nuevo; mientras que esto no ocurre con `StringBuffer`. Los objetos `String` se almacenan en el *Constant String Pool*, que es un repositorio o almacén de cadenas de valores de Strings. Esto se hace con el fin de que si creamos otro objeto `String` con el mismo valor, no se cree un nuevo objeto, sino que se use el ya existente y se asigne la referencia al objeto ya creado. Por elcontrario, los objetos `StringBuffer` se almacenan en el heap (área de datos en tiempo de ejecución destinada a todos los hilos del JRE) que es otro espacio de memoria usado en tiempo de ejecución para almacenar las instancias de clases, objetos y arrays.

   Un ejemplo de esto, es cuando necesitamos eliminar un carácter de una cadena, cambiar un carácter por otro, o convertir un carácter a mayúscula. Lo correcto es usar un objeto de tipo `StringBuffer` y no un `String`, ya que por cada operación que se realice sobre el String, se estaría generando objetos `String` nuevos.

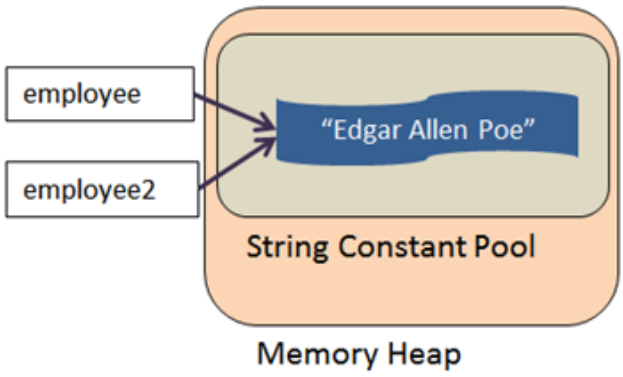

Figure 7: String Pool