Segundo ensayo SFIA

[1].- ¿Cuál es es resultado del siguiente código? Elija todas las que apliquen.

```
1: public class _C {
2:   private static int $;
3:   public static void main(String[] main) {
4:     String a_b;
5:     System.out.print($);
6:     System.out.print(a_b);
7: } }
```

A. Error de compilación en línea 1
B. Error de compilación en línea 2
C. Error de compilación en línea 4
D. Error de compilación en línea 5
E. **Error de compilación en línea 6**
F. 0null
G. nullnull

---

[1] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

E.

Los caracteres `$` y `a_b` están permitidos para su uso en referencias así que este problema queda descartado.

`String a_b;`, es una variable local, estas variables necesitan ser inicializadas, lo que no ocurrió y por tanto la línea 6 no compila por que la referencia `a_b` no ha sido inicializada.

[2].- ¿Cuál es el resultado del siguiente fragmento de código?

```
String s1 = "Java";
String s2 = "Java";
StringBuilder sb1 = new StringBuilder();
sb1.append("Ja").append("va");
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));
System.out.println(sb1.toString() == s1);
System.out.println(sb1.toString().equals(s1));
```

A. true se imprime exactamente una vez
B. true se imprime exactamente dos veces
C. **true se imprime exactamente tres veces**
D. true se imprime exactamente cuatro veces
E. el código no compila

C. `s1 & s2` son los mismos objetos en el `string pool`, `sb1` es una referencia al objeto `StringBuilder`.

`s1 == s2`, es verdadero, ya que son el mismo objeto en el `string pool`.

`s1.equals(s2)`, es verdadero, ya que ambos tienen el valor `"Java"`.

`sb1.toString() == s1` es falso ya que `sb1.toString()` es un string, pero no el mismo.

`sb1.toString().equals(s1)` es verdadero ya que el valor de sus cadenas son los mismos.

---

[2] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[3].- ¿Cuál es la salida del siguiente código? Elija todas las que aplican.

```
1: interface HasTail { int getTailLength(); }
2: abstract class Puma implements HasTail {
3:   protected int getTailLength() {return 4;}
4: }
5: public class Cougar extends Puma {
6:   public static void main(String[] args) {
7:     Puma puma = new Puma();
8:     System.out.println(puma.getTailLength());
9:   }
10:
11:  public int getTailLength(int length) {return 2;}
12:}
```

A. 2
B. 4
C. **No compila por la línea 3.**
D. **No compila por la línea 5.**
E. **No compila por la línea 7.**
F. No compila por la línea 11.
G. No se puede especificar la salida con el código dado.

---

[3] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

C, D, E.

`int getTailLength();` es público en la interface, por lo tanto la clase `Puma` NO hace una correcta sobrescritura ya que lo deja `protected`, fallando la línea 3.

La clase `Cougar` extiende de `Puma` que posee un error de sobrescritura, mientras no se solucione sus hijos arrastrarán el error en su declaración, por este motivo la línea 5 no compila.

`Puma puma = new Puma();` es una instancia de una clase abstracta, lo que no está permitido, por eso falla.

[4].- ¿Cuál es la salida del siguiente programa?

```
1: public class FeedingSchedule {
2:   public static void main(String[] args) {
3:     boolean keepGoing = true;
4:     int count = 0;
5:     int x = 3;
6:     while(count++ < 3) {
7:       int y = (1 + 2 * count) % 3;
8:       switch(y) {
9:       default:
10:       case 0: x -= 1; break;
11:       case 1: x += 5;
12:       }
13:     }
14:     System.out.println(x);
15: } }
```

A. 4
B. 5
C. **6**
D. 7
E. 13
F. El código no compila por error en la línea 7

C

```
// recordemos que `count++ < 3` primero, compara y luego aumenta su valor, por
lo tanto entra a la función con el valor ya aumentado justo después de la
comparación.
```

---

[4] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

```
// PRIMER CICLO (count = 0) y luego (count = 1)
y = (1 + 2 * 1) % 3
  = (1 + 2) % 3 = 3 % 3
  = 0
// case 0: x -= 1; break;
x = 3 - 1 = 2

// SEGUNDO CICLO (count = 1) y luego (count = 2)

y = (1 + 2 * 2) % 3
  = (1 + 4) % 3 = 4 % 3
  = 2

// se ejecuta el caso default
// default: case 0: x -= 1; break;
x = 2 - 1 = 1

// TECER CICLO (count = 2) y luego (count = 3)
y = (1 + 2 * 3) % 3
  = (1 + 6) % 3 = 7 % 3
  = 1
// case 1: x += 5;
x = 1 + 5 = 6

// luego (count = 3), por lo que no se cumple la condición del
// while, finalizando el ciclo.

// el default está sobre los cases, es decir que al buscar
// algún case, no encuentra ninguno, entra al default, este no
// tiene un break, por lo que continúa descarreado ejecutando
// a destajo el código que se encuentra debajo de él, hasta
// que un break termine con el swith.
9:        default:
10:       case 0: x -= 1; break;
11:       case 1: x += 5;

// es decir, que en el caso que no sea ni 0 ni 1, entrará
// a los dos siempre y cuando no se tope con un break.
```

[5].- ¿Cuál es la salida del siguiente fragmento de código?

```
13: System.out.print("a");
14: try {
15:   System.out.print("b");
16:   throw new IllegalArgumentException();
17: } catch (RuntimeException e) {
18:   System.out.print("c");
19: } finally {
20:   System.out.print("d");
21: }
22: System.out.print("e");
```

A. abe
B. abce
C. abde
D. **abcde**
E. el código no compila
F. Cae en un excepción

D. El código comienza a ejecutarse e imprime `a` y `b` en las líneas 13 y 15. La línea 16 lanza una excepción, que se captura en la línea 17. Imprimiendo `c`

Después de que la línea 18 imprima c, el bloque `finally` se ejecuta y se imprime `d`. Luego, la instrucción de prueba finaliza y se imprime `e` en la línea 22.

---

[5] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[6].- ¿Cuál es la salida del siguiente programa?

```
1: public class MathFunctions {
2:     public static void addToInt(int x, int amountToAdd) {
3:     x = x + amountToAdd;
4:   }
5: public static void main(String[] args) {
6:   int a = 15;
7:   int b = 10;
8:   MathFunctions.addToInt(a, b);
9:   System.out.println(a); } }
```

A. 10
B. **15**
C. 25
D. Error de compilación en línea 3
E. Error de compilación en línea 8
F. Ninguna de las anteriores

B. El código se compila correctamente, por lo que las opciones D y E son incorrectas.

El valor de a no se puede cambiar con el método `addToInt`, independientemente de lo que haga el método, ya que solo **se pasa una copia** de la variable al parámetro x. Por lo tanto, a no cambia y la salida en la línea 9 es 15.

---

[6] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[7].- ¿Cuál es el resultado del siguiente código?

```
int[] array = {6,9,8};
List<Integer> list = new ArrayList<>();
list.add(array[2]);
list.set(1, array[1]);
list.remove(0);
System.out.println(list);
```

A. `[8]`
B. `[9]`
C. Algo como `[Ljava.lang.String;@160bc7c0`
D. Ocurre una excepción.
E. El código no compila.

B. Un arreglo puede usar un `anonymous initializer` porque está en la misma línea que la declaración. `ArrayList` utiliza el operador de diamante `<>` permitido desde Java 7. Esto especifica que el tipo coincide con el de la izquierda sin tener que volver a escribirlo.

Después de agregar los dos elementos, la lista contiene `[6, 8]`. Luego reemplazamos el elemento en el índice 1 con 9, resultando en `[6, 9]`. Finalmente, eliminamos el elemento en el índice 0, dejando `[9]`.

La opción C es incorrecta porque los arreglos generan algo así. No `ArrayList` Estos tienen el `toString` sobrescrito.

---

[7] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[8].- ¿Cuál es la salida del siguiente código?

```
1: public class Deer {
2:   public Deer() { System.out.print("Deer"); }
3:   public Deer(int age) { System.out.print("DeerAge"); }
4:   private boolean hasHorns() { return false; }
5:   public static void main(String[] args) {
6:     Deer deer = new Reindeer(5);
7:     System.out.println(","+deer.hasHorns());
8:   }
9: }
10: class Reindeer extends Deer {
11:   public Reindeer(int age) { System.out.print("Reindeer"); }
12:   public boolean hasHorns() { return true; }
13: }
```

A. **DeerReindeer, false**
B. DeerReindeer, true
C. ReindeerDeer, false
D. ReindeerDeer, true
E. DeerAgeReindeer, false
F. DeerAgeReindeer, true
G. El código no compila por la línea 7
H. El código no compila por la línea 12

---

[8] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

A. El código compila sin problemas, por lo que las opciones G y H se descartan.

Primero, el objeto `Reindeer` es instanciado usando el constructor que toma un valor `int`: `public Reindeer(int age) { System.out.print("Reindeer"); }`. Acá no hay una llamada explicita al padre usando `super`, se hace automáticamente:

```
// de esto:
public Reindeer(int age) { System.out.print("Reindeer"); }
// a esto:
public Reindeer(int age) {
  super();
  System.out.print("Reindeer");
}

// esto hace que se ejecute el constructor del paadre:
public Deer() { System.out.print("Deer"); }
// ya que super() sin parámetros es Deer().
```

Esto hace que se imprima `Deer` y luego `Reindeer`, partiendo el resultado con `DeerReindeer`, por lo tanto sólo tenemos como posibles correctas a las alternativas A y B.

La línea `System.out.println(","+deer.hasHorns());` es ejecutada a continuación. La clase `Reindeer` posee este método, que parece sobrescribir al del padre, pero no es así. El padre tiene este método privado, por lo tanto `deer.hasHorns()` en realidad llama a `private boolean hasHorns() { return false; }` ya que en realidad `Reindeer` no lo está sobrescribiendo, esto se puede demostrar si agrega la anotación `@Override` nos arrojará error. Por lo tanto se ejecuta `private boolean hasHorns() { return false; }`, y al estar dentro de la misma clase, no tenemos problemas de acceso.

Si quisiéramos que se ejecutara el método de la clase `Reindeer` hay que hacer un casteo.

[9].- *¿Cuáles sentencias sobre el siguiente código son verdades? Elija todas las que apliquen.*

```
1: import java.util.*;
2: public class Grasshopper {
3:   public Grasshopper(String n) {
4:    name = n;
5:   }
6:   public static void main(String[] args) {
7:     Grasshopper one = new Grasshopper("g1");
8:     Grasshopper two = new Grasshopper("g2");
9:     one = two;
10:    two = null;
11:    one = null;
12:  }
13:  private String name; }
```

A. Inmediatamente después de la línea 9, ningún objeto `Grasshopper` es elegible por el `garbage collection`.
B. Inmediatamente después de la línea 10, ningún objeto `Grasshopper` es elegible por el `garbage collection`.
C. **Inmediatamente después de la línea 9, solo un objeto `Grasshopper` es elegible por el `garbage collection`.****
D. **Inmediatamente después de la línea 10, solo un objeto `Grasshopper` es elegible por el `garbage collection`.**
E. Inmediatamente después de la línea 11, solo un objeto `Grasshopper` es elegible por el `garbage collection`.
F. **El código compila.**
G. El código NO compila.

---

[9] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

C, D, F.

Inmediatamente después de la línea 9, solo `Grasshopper g1` es elegible para la recolección de basura ya que tanto `one` como `two` apuntan a `Grasshopper g2`.

Inmediatamente después de la línea 10, solo tenemos `Grasshopper g1` elegible para la recolección de basura. La referencia `one` apunta a `g1` y la referencia `two` es nula.

Inmediatamente después de la línea 11, ambos objetos `Grasshopper` son elegibles para la recolección de basura ya que tanto uno como dos apuntan a nulo. El código sí compila. Aunque es tradicional declarar variables de instancia al principio de la clase, no es obligatorio hacerlo.

[10].- ¿Cuál es la salida del siguiente programa?

```
1: public class FeedingSchedule {
2:   public static void main(String[] args) {
3:     int x = 5, j = 0;
4:     OUTER: for(int i=0; i<3; )
5:     INNER: do {
6:       i++; x++;
7:       if(x > 10) break INNER;
8:       x += 4;
9:       j++;
10:    } while(j <= 2);
11:    System.out.println(x);
12:  } }
```

A. 10
B. **12**
C. 13
D. 17
E. El código no compila por error en la línea 4
F. El código no compila por error en la línea 6

---

[10] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

B. El código compila correctamente, así que las opciones E y F, son incorrectas.

En la primera iteración del bucle externo i es 0, por lo que el bucle continúa.

En la primera iteración del bucle interno, i se actualiza a 1 y x a 6. La sentencia `if-then` no se ejecuta, x se incrementa a 10 y j a 1

En la segunda iteración el `inner loop (dado que j = 1 and 1 <= 2)` i es actualizado a 2 y x a 11. En este punto, el `if-then` se evaluará como verdadera durante el resto de la ejecución del programa , lo que hace que el flujo salga del `inner loop` cada vez que se alcanza.

En la segunda iteración del `inner loop (dado que j = 1 y 1 <= 2)`, i se actualiza a 2 y x a 11. En este punto, la rama `if-then` se evaluará como verdadera durante el resto de la ejecución del programa , lo que hace que el flujo salga del bucle interno cada vez que se alcanza.

En la segunda iteración del `OUTER` (dado que i = 2), i se actualiza a 3 y x a 12. Como antes, el `INNER` se rompe ya que x todavía es mayor que 10.

En la tercera iteración de `OUTER`, se rompe, ya que i ya no es menor que 3. El valor más reciente de x, 12, se imprime, por lo que la respuesta es la opción B.

[11].- ¿Cuál es la salida del siguiente programa?

```
1:  public class Egret {
2:    private String color;
3:    public Egret() {
4:      this("white");
5:    }
6:    public Egret(String color) {
7:      color = color;
8:    }
9:    public static void main(String[] args) {
10:     Egret e = new Egret();
11:     System.out.println("Color:" + e.color);
12:   }
13: }
```

A. Color:
B. **Color: null**
C. Color:White
D. Error de compilación en línea 4
E. Error de compilación en línea 10
F. Error de compilación en línea 11

B. La línea 10, llama al constructor de las lineas `[3-5]`, este constructor llama a otro constructor de las lineas `[6-8]`. El problema con este constructor, es que al hacer `color = color;` y no `this.color = color;`, la variable de instancia `color`, no es afectada con el cambio y conserva su valor por defecto `null`.

---

[11] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[12].- ¿Cuál es la salida del siguiente programa?

```
1: public class BearOrShark {
2:   public static void main(String[] args) {
3:     int luck = 10;
4:     if((luck>10 ? luck++: --luck)<10) {
5:       System.out.print("Bear");
6:     } if(luck<10) System.out.print("Shark");
7:   } }
```

A. Bear.
B. Shark.
C. **BearShark**.
D. El código no compila por error en la línea 4.
E. El código no compila por error en la línea 6.
F. El código se compila sin ningún problema pero no produce ninguna salida.

C. La expresión `(luck>10 ? luck++: --luck)` evalúa `--luck` es decir que primero `luck` se decremento en una unidad, quedando en `9`.

Quedando `9 < 10` por tanto se ejecuta `System.out.print("Bear");`

Saliendo del bloque `if-then` continúa hasta llegar `if(luck<10) System.out.print("Shark");`, el valor de `luck`no ha cambiado, aún es `9`, por lo que se ejecuta.

---

[12] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[13].- ¿Cuál de las siguiente sentencias se puede insertar en la línea en blanco para que le código compile? Elija todas las que apliquen.

```
public interface CanSwim {}
public class Amphibian implements CanSwim {}
class Tadpole extends Amphibian {}
public class FindAllTadPole {
  public static void main(String[] args) {
    List<Tadpole> tadpoles = new ArrayList<Tadpole>();
    for(Amphibian amphibian : tadpoles) {
    ___________ tadpole = amphibian;
} } }
```

A. **CanSwim**
B. `Long`
C. **Amphibian**
D. Tadpole
E. **Object**

se recomienda usar netbeans para poder probar bien.

[13] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

A, C, E.

El `for-each loop`, automáticamente hace `cast` de `Tadpole` a `Amphibian` para la referencia `amphibian`,

El bucle `for-each` convierte automáticamente cada objeto `Tadpole` en una referencia de `amphibian`, que no requiere `cast` explícito porque `Tadpole` es una subclase de `amphibian`.

A partir de ahí, cualquier clase o interfaz padre de la cual `Amphibian` herede, se permite un `casteo implícito` . Esto incluye `CanSwim`, la interfaz que implementa `Amphibian` y `Object`, desde donde se extienden todas las clases, por lo que las opciones A y E son correctas.

La opción C también es correcta, ya que la referencia se está `casteando` al mismo tipo, por lo que no se requiere un `casteo` explícito. La opción B es incorrecta, ya que `Long` no es padre de `Amphibian`.

La opción D también es incorrecta, aunque se requeriría un `cast explícito` a `Tadpole` en el lado derecho de la expresión para permitir que el código se compile.

[14].- ¿Qué cambios individuales si los hubiera, permitiría compilar el siguiente código? Elija todas las que apliquen.

```
1: public interface Animal { public default String getName() { return
null; } }
2: interface Mammal { public default String getName() { return null; } }
3: abstract class Otter implements Mammal, Animal {}
```

A. El código compila sin problemas
B. Eliminar el modificador `default` y la implementación de método de la línea 1
C. Eliminar el modificador `default` y la implementación de método de la línea 2
D. **Eliminar el modificador `default` y la implementación de método de la línea 1 y 2**
E. Cambiar el valor de retorno de la línea 1 de `null` a `Animal`.
F. **Sobrescribir el método `getName()` con el método abstracto en la clase `Otter`**
G. **Sobrescribir el método `getName()` con un método concreto en la clase `Otter`**

D, F, G.

El código no compila, por que una clase, no puede implementar dos interfaces, que tengan métodos `default` con la misma firma. A menos que se sobre escriba en la clase que implementa las interfaces con un método abstracto o concreto.

Por lo tanto, la opción A es correcta junto con la F y la G.

Una alternativa, sería hacer los métodos `default` pasarlos a `abstractos` ya que en este caso SI estantería permitido.

---

[14] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[15].- ¿Cuál de las siguientes líneas puede ser insertada en la línea 11 para imprimir *true*?

```
10:  public static void main(String[] args) {
11:    // INSERT CODE HERE
12:  }
13:  private static boolean test(Predicate<Integer> p) {
14:    return p.test(5);
15:  }
```

A. **`System.out.println(test(i -> i == 5));`**
B. `System.out.println(test(i -> {i == 5;}));`
C. **`System.out.println(test((i) -> i == 5));`**
D. `System.out.println(test((int i) -> i == 5);`
E. `System.out.println(test((int i) -> {return i == 5;}));`
F. **`System.out.println(test((i) -> {return i == 5;}));`**

A, C, F. `test()` es una `functional programming interface` Es la firma de método para su correcta implementación y uso.

Toma un parámetro y retorna un `boolean`

`i -> i == 5` , `(i) -> i == 5`, `(i) -> {return i == 5;}` son las formas correctas de una función lambda.

---

[15] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[16].- ¿Cuál de las siguientes impresiones imprime una fecha que representa el primero de abril del 2015?

A. `System.out.println(LocalDate.of(2015, Calendar.APRIL, 1));`

B. **`System.out.println(LocalDate.of(2015, Month.APRIL, 1));`**

C. `System.out.println(LocalDate.of(2015, 3, 1));`

D. **`System.out.println(LocalDate.of(2015, 4, 1));`**

E. `System.out.println(new LocalDate(2015, 3, 1));`

F. `System.out.println(new LocalDate(2015, 4, 1));`

Las nuevas API de fecha agregadas en Java 8, usan métodos estáticos en lugar de un constructor para crear una nueva fecha, haciendo que las opciones E y F sean incorrectas.

Los meses se indexan comenzando con 1 en esta API, lo que hace que las opciones A y C sean incorrectas. La opción A usa las constantes de `Calendar`, antiguas que se indexan desde 0 `public final static int APRIL = 3;`. Por lo tanto, las opciones B y D son correctas.

---

16 FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[17].- ¿El `Bytecode` esta en un archivo con extensión?

A. `.bytecode`
B. `.bytes`
C. `.class`
D. `.exe`
E. `.javac`
F. `.java`

C. El archivo compilado en `bytecode` tiene la extensión .class.

---

17 FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[18].- ¿Cuál de las siguientes son `checked exception`?

A. **Exception**
B. IllegalArgumentException
C. **IOException**
D. NullPointerException
E. NumberFormatException
F. StackOverFlowError

A, C. Un `checked exception` es un tipo de excepción que debe ser capturada o declarada en el método en el que se lanza.

`Exception` es la clase base y es un `checked exception` al igual que `IOException`, las demás extienden a `RuntimeException`, serían `unchecked exceptions`

La opción F no es una excepción, es un `throwable`.

---

[18] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[19].- ¿Cuáles de los siguientes son identificadores de java válidos? Elija todos los que corresponda.

A. **A$B**
B. **_helloWorld**
C. true
D. java.lang
E. **Public**
F. 1980_s.

Solución: A es correcta, ya que está permitido el uso del signo `$` en los identificadores. **B es correcta** ya que también está permitido el uso de el signo `_` en los identificadores. **C no es correcta**, ya que la palabra `true` es una palabra reservada de Java. **D no es correcta**, ya que el uso del punto (.), no está permitido en los identificadores. **E es correcta**, ya que Java es `case sensitive` entonces `Public` no es la palabra reservada `public`, por lo tanto está permitido su uso. **F no es correcta** ya que el primer carácter del identificador no es una letra o `$ o _`.

---

[19] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[20].- ¿Cuál es la salida del siguiente programa?

```
1:  public class WaterBottle {
2:  private String brand;
3:  private boolean empty;
4:  public static void main(String[] args) {
5:      WaterBottle wb = new WaterBottle();
6:      System.out.print("Empty = " + wb.empty);
7:      System.out.print(", Brand = " + wb.brand);
8:  }}
```

A. La línea 6 genera un error de compilación.
B. La línea 7 genera un error de compilación.
C. No hay ninguna salida.
D. **Empty = false, Brand = null.**
E. Empty = false, Brand =.
F. Empty = null, Brand = null.

Solución: D es correcta ya que los campos `boolean` se inicializan en `false` y las referencias se inicializan en `null`.

---

[20] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[21].- *¿Cuál de las siguientes alternativas es correcta según el siguiente código? Elija todas las que apliquen.*

```
4: short numPets = 5;
5: int numGrains = 5.6;
6: String name = "Scruffy";
7: numPets.length();
8: numGrains.length();
9: name.length();
```

A. La línea 4, genera un error de compilación.
B. **La línea 5, genera un error de compilación.**
C. La línea 6, genera un error de compilación.
D. **La línea 7, genera un error de compilación.**
E. **La línea 8, genera un error de compilación.**
F. La línea 9, genera un error de compilación.
G. El código compila sin problemas.

Solución: La opción A (línea 4) compila, porque short es un tipo integral de 16-bits, soporta el 5. Opción B (línea 5) genera un error de compilación, porque int es un tipo integral, pero 5.6 es un tipo de punto flotante. La opción C (línea 6) se compila porque la cadena de caracteres es soportada por `String`. Las opciones D y E (líneas 7 y 8) no se compilan porque short y int son primitivas y no poseen métodos. La opción F (línea 9), compila porque `length()` está definido en `String`.

---

[21] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[22].- *Dadas las siguientes clases, ¿Cuál de los siguientes códigos propuestos, pueden ser insertados en* `INSERTE EL CÓDIGO ACÁ`*, para que el programa compile?(Escoja todos lo que apliquen)*

```
package aquarium;

public class Water { boolean salty = false; }

package aquarium.jellies;

public class Water { boolean salty = true; }

package employee;
// INSERTE EL CÓDIGO ACÁ

public class WaterFiller { Water water; }
```

A. **`import aquarium.*;.`**
B. **`import aquarium.Water; import aquarium.jellies.*;.`**
C. **`import aquarium.*; import aquarium.jellies.Water;.`**
D. `import aquarium.*; import aquarium.jellies.*;.`
E. `import aquarium.Water; import aquarium.jellies.Water;.`
F. . Ninguno de estos cambios haría que la aplicación compile.

---

[22] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

Solución: La opción A es correcta porque importa todas las clases en el paquete `aquarium`, incluido `aquarium.Water`. Las opciones B y C son correctas porque importa `Water` por nombre de clase. Dado que **la importación por nombre de clase tiene prioridad sobre los comodines**, estos se compilan. La opción D es incorrecta porque Java no sabe cuál de las dos clases `Water` en el comodín usar. La opción E es incorrecta porque no se puede especificar el mismo nombre de clase en dos importaciones. Finalmente F es incorrecta ya que el código compila con A, B, C.

[23].- *Dada las siguientes clases, ¿Cuál es el máximo número de `imports` que podemos remover, para que la aplicación pueda ser compilada?*

```
package aquarium; public class Water { }

package aquarium;
import java.lang.*;
import java.lang.System;
import aquarium.Water;
import aquarium.*;

public class Tank {
    public void print(Water water) {
        System.out.println(water);
    }
}
```

A. 0.
B. 1.
C. 2.
D. 3.
E. **4.**
F. El código no compila.

Solución: Las dos primeras importaciones se pueden eliminar porque `java.lang` se importa automáticamente. Las segundas dos importaciones se pueden eliminar porque `Tank` y `Water` están en el mismo paquete, lo que hace que la opción **E, sea la correcta**. Si `Tank` y `Water` estuvieran en paquetes diferentes, se podría eliminar una de ellas. En ese caso, la respuesta sería la opción D, pero este no es el caso ya que están todas en el mismo paquete `aquarium`.

---

[23] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[24].- *Dada la siguiente clase, ¿Cuál de las siguientes llamadas desde la consola imprimirá `Blue Jay`? (Escoja todas las que aplican)*

```
public class BirdDisplay {
    public static void main(String[] name) {
        System.out.println(name[1]);
}}
```

A. `java BirdDisplay Sparrow Blue Jay.`
B. **`java BirdDisplay Sparrow "Blue Jay".`**
C. `java BirdDisplay Blue Jay Sparrow.`
D. `java BirdDisplay "Blue Jay" Sparrow.`
E. `java BirdDisplay.class Sparrow "Blue Jay".`
F. `java BirdDisplay.class "Blue Jay" Sparrow.`
G. Este código, no compila.

Solución: La opción B es correcta porque los arreglos comienzan a contar desde cero y las cadenas con espacios deben estar entre comillas. La opción A es incorrecta porque da salida es `Blue`. C es incorrecto porque la salida es `Jay`. La opción D es incorrecta porque la salida es `Sparrow`. Las opciones E y F son incorrectas porque generan un error: `Could not find or load main class Bird- Display.class.`

---

[24] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[25].- *¿Cuál de las siguientes propuestas, podrían reemplazar ____ para que una aplicación pueda ser llamada desde la línea de comandos? (Escoja todas las que apliquen)*

```
public static void main( ______ )
```

A. **`String[] _names.`**
B. `String[] 123.`
C. **`String abc[].`**
D. **`String _Names[].`**
E. **`String... $n.`**
F. `String names.`
G. Ninguna de las anteriores.

Solución: La opción **A es correcta** porque es la firma tradicional del método main () y las variables pueden comenzar con guiones bajos. Las opciones **C y D son correctas** porque el operador de arreglo puede estar después del nombre de la variable. La opción **E es correcta** porque se permiten `varargs...` en lugar de un arreglo. La opción B es incorrecta porque las variables no pueden comenzar con un dígito. La opción F es incorrecta porque el argumento debe ser un arreglo o `varargs...`. La opción F es una firma de método correcta. Sin embargo, no se puede ejecutar desde la línea de comandos porque tiene el tipo de parámetro incorrecto.

---

[25] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[26].- ¿Cuál de las siguientes propuestas, sería un punto de entrada válido, para que una aplicación pueda ser llamada desde la línea de comandos?(Escoja todas aquellas que apliquen)

A. `private static void main(String[] args).`
B. `public static final main(String[] args).`
C. `public void main(String[] args)`
D. `public static void test(String[] args).`
E. **`public static void main(String[] args).`**
F. `public static main(String[] args).`
G. Ninguna de las anteriores.

Solución: La opción **E es correcta** , ya que es la firma canónica del método `main()`, es necesario memorizarla. La opción A es incorrecta porque el método `main()` debe ser público. Opciones B y F son incorrectos porque el método `main()` debe tener un tipo de retorno `void`. La opción C es incorrecta porque el método `main()` debe ser estático. La opción D es incorrecta porque el método `main()` debe llamarse `main`.

---

[26] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[27].- ¿Cuál de las siguientes líneas de códigos compila? (Escoja todas las que aplican)

A. **`int i1 = 1_234;.`**

B. `double d1 = 1_234_.0;.`

C. `double d2 = 1_234._0;.`

D. `double d3 = 1_234.0_;.`

E. **`double d4 = 1_234.0;.`**

F. ninguna es correcta.

Solución: Se permiten guiones bajos siempre que estén directamente entre otros dos dígitos. Esto significa que las opciones **A y E son correctas**. Las opciones B y C son incorrectas porque el subrayado es adyacente al punto decimal. La opción D es incorrecta porque el guión bajo es el último carácter.

---

[27] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[28].- ¿Cuál de las siguientes propuestas es verdadera?(Escoja todas las que apliquen)

```
public class Bunny {
    public static void main(String[] args) {
        Bunny bun = new Bunny();
}}
```

A. **`Bunny` es una clase.**
B. `bun` es una clase.
C. `main` es una clase.
D. `Bunny` es una referencia a un objeto.
E. **`bun` es una referencia a un objeto.**
F. `main` es una referencia a un objeto.
G. Ninguna de las anteriores.

Solución: `Bunny` es una clase, puede verse en la declaración: `public class Bunny` por lo que **A es correcto**.`bun` es una referencia a un objeto por lo que **E es correcto**. `main()` es un método.

[28] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[29].- ¿Cuál representa el orden, que deberían llevar las siguientes líneas de código, para lograr que el programa compile satisfactoriamente?

```
A: class Rabbit {}
B: import java.util.*;
C: package animals;
```

A. A, B, C.
B. B, C, A.
C. **C, B, A.**
D. **B, A.**
E. **C, A.**
F. A, C.
G. A, B.

Solución: Los paquetes y los `imports` son opcionales. Si ambos están presentes, el paquete debe estar primero, luego las importaciones, luego la declaración de la clase. La opción A es incorrecta porque la clase está antes del paquete y la importación. La opción B es incorrecta porque la importación es anterior al paquete. La opción F es incorrecta porque la clase está antes del paquete. La opción G es incorrecta porque la clase está antes de la importación.

---

[29] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[30].- ¿Cuál es la salida de la siguiente aplicación?

```
1:  public class Salmon {
2:    int count;
3:    public void Salmon() {
4:      count = 4;
5:    }
6:    public static void main(String[] args) {
7:      Salmon s = new Salmon();
8:      System.out.println(s.count);
9:  }}
```

A. **0.**
B. 4.
C. La compilación falla en la línea 3.
D. La compilación falla en la línea 4.
E. La compilación falla en la línea 7.
F. La compilación falla en la línea 8.

Mientras que el código en la línea 3 se compila, **no es un constructor porque tiene un tipo de retorno**. Es un método que tiene el mismo nombre que la clase. Cuando se ejecuta el código, se llama al constructor predeterminado y el conteo tiene el valor predeterminado (0) para un int por lo que **A es correcto**.

---

[30] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[31].- *¿Cuál de los siguientes operadores en Java, pueden ser usados con variables tipo boolean? (Escoja todas la que apliquen)*

A. ==
B. +
C. --
D. !
E. %
F. <=

A, D. La opción A, es el operador de igualdad y se puede usar en primitivas numéricas, valores booleanos y referencias de objetos. Las opciones B y C son operadores aritméticos y no pueden aplicarse a un valor booleano. La opción D es el operador de complemento lógico y se utiliza exclusivamente con valores booleanos. La opción E es el operador de módulo, que solo se puede utilizar con primitivas numéricas. Finalmente, la opción F es un operador relacional que compara los valores de dos números.

---

31 FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[32].- ¿Qué tipo de datos (o tipos), están permitidos para que el siguiente trozo de código compile? (Escoja todas la que apliquen)

```
byte x = 5;
byte y = 10;
_____ z = x + y;
```

A. `int`
B. `long`
C. `boolean`
D. `double`
E. `short`
F. `byte`

A, B, D. El valor `x + y` se promueve automáticamente a `int`, por lo que int y los tipos de datos que puedan promoverse automáticamente desde int funcionarán. Las opciones A, B, D son tales tipos de datos. La opción C no funcionará porque `boolean` no es un tipo de dato numérico. Las opciones E y F no funcionarán sin una conversión explícita (`cast`) a un tipo de datos más pequeño.

---

[32] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[33].- ¿Cuál es la salida de la siguiente aplicación?

```
1: public class CompareValues {
2:  public static void main(String[] args) {
3:   int x = 0;
4:   while(x++ < 10) {}
5:   String message = x > 10 ? "Greater than" : false;
6:   System.out.println(message+","+x);
7: }
8: }
```

A. `Greater than,10`
B. `false,10`
C. `Greater than,11`
D. `false,11`
E. El código no compila, por un error en la linea 4.
F. El código no compila, por un error en la linea 5.

F. En este ejemplo, el operador ternario tiene dos expresiones, una de ellas es un `String` y el otro es un `boolean`. Al operador ternario se le permite tener expresiones con distintos tipos, pero la clave aquí es la asignación a la referencia de `String` . El compilador sabe cómo asignar el primer valor de expresión como un `String`, pero la segunda expresión booleana no se puede establecer como una cadena; por lo tanto, esta línea no compilará. Es decir que no compila por que `message` solo puede ser un `String` y no un `boolean`.

---

[33] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[34].- ¿Qué cambios permitiría que el siguiente trozo de código compile? (Escoja todas la que apliquen)

```
3: long x = 10;
4: int y = 2 * x;
```

A. Ningún cambio es necesario, compila así como está.
B. `Castear` x en la línea 4 a `int`
C. Cambiar el tipo de dato de x en la línea 3 a un tipo `short`
D. `Castear 2 * x`, en la línea 4 a `int`
E. Cambiar el tipo de dato de `y` en la línea 4 a un tipo `short`
F. Cambiar el tipo de dato de `y` en la línea 4 a un tipo `long`

B, C, D, F.

El código no se compilará tal como está, por lo que la opción A no es correcta.

El valor `2 * x` se promueve automáticamente a `long` y no se puede almacenar automáticamente en y, que es un `int`. Las opciones B, C y D resuelven este problema reduciendo el valor de `long` a `int`.

La opción E no resuelve el problema y en realidad lo empeora al intentar colocar el valor en un tipo de datos más pequeño. La opción F resuelve el problema al cambiar `y` a tipo `log`.

---

[34] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[35].- ¿Cuál es la salida del siguiente trozo de código?

```
3: java.util.List<Integer> list = new java.util.ArrayList<Integer>();
4: list.add(10);
5: list.add(14);
6: for(int x : list) {
7:   System.out.print(x + ", ");
8:   break;
9: }
```

A. 10, 14,
B. 10, 14
C. 10,
D. El código no compila, por un error en la línea 7
E. El código no compila, por un error en la línea 8
F. El código contiene un `loop` infinito

C. Este código no contiene ningún error de compilación o un bucle infinito, por lo que las opciones D, E y F son incorrectas. La declaración `breack` en la línea 8 hace que el bucle se ejecute una vez y termine, por lo que la opción C es la respuesta correcta.

---

[35] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[36].- ¿Cuál es la salida del siguiente trozo de código?

```
3: int x = 4;
4: long y = x * 4 - x++;
5: if(y<10) System.out.println("Too Low");
6: else System.out.println("Just right");
7: else System.out.println("Too High");
```

A. `Too Low`
B. `Just Right`
C. `Too High`
D. Compila pero arroja un `NullPointerException`
E. No compila por un error en la línea 6
F. No compila por un error en la línea 7

F. El código no se compila, porque otras dos declaraciones no se pueden encadenar sin declaraciones adicionales `if-else`, por lo que la respuesta correcta es la opción F.

La opción E es incorrecta, ya que la Línea 6 por sí sola no causa problemas, el problema está en que se le agrega otro `else` en la linea 7. Una forma de arreglar este código para qué compile, sería agregar una instrucción `if-then (esto significa if{ then...})` en la línea 6. La otra solución sería eliminar la línea 7.

---

[36] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[37].- ¿Cuál es la salida del siguiente trozo de código?

```
1: public class TernaryTester {
2:   public static void main(String[] args) {
3:     int x = 5;
4:     System.out.println(x > 2 ? x < 4 ? 10 : 8 : 7);
5: }}
```

A. 5
B. 4
C. 10
D. 8
E. 7
F. No compila por un error en la línea 4

D. Como se aprendió en "Operadores ternarios", aunque no se requieren paréntesis, el uso de ellos aumenta considerablemente la legibilidad del código, como la siguiente declaración equivalente:

```
System.out.println((x > 2) ? ((x < 4) ? 10 : 8) : 7)
```

Primero aplicamos el operador ternario externo, ya que es posible que la expresión ternaria interna nunca se evalúe. Como `(x> 2)` es verdadero, esto reduce el problema a:

```
System.out.println((x < 4) ? 10 : 8)
```

---

[37] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

Como x es mayor que 2, la respuesta es 8, u opción D en este caso.

[38].- ¿Cuál es la salida del siguiente trozo de código?

```
3: boolean x = true, z = true;
4: int y = 20;
5: x = (y != 10) ^ (z=false);
6: System.out.println(x+", "+y+", "+z);
```

A. `true, 10, true`
B. `true, 20, false`
C. `false, 20, true`
D. `false, 20, false`
E. `false, 10, true`
F. No compila por un error en la línea 5

B. Este ejemplo es complicado debido al segundo operador de asignación integrado en la línea 5. La expresión `(z = false)` asigna el valor `false` a `z` y devuelve falso para toda la expresión. Como `(y != 10)` es verdadero, el lado izquierdo devuelve verdadero; por lo tanto, el `or exclusivo XOR, que es verdadero solo si las proposiciones son diferentes` (^) de la expresión completa asignada a `x` es verdadera.

La salida refleja estas asignaciones, sin cambios en y, por lo que la opción B es la única respuesta correcta. El código se compila y ejecuta sin problemas, por lo que la opción F no es correcta.

---

[38] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[39].- ¿Cuántas veces el siguiente código imprimirá "Hello World"?

```
3: for(int i=0; i<10 ; ) {
4:   i = i++;
5:   System.out.println("Hello World");
6: }
```

A. 9
B. 10
C. 11
D. No compila por un error en la línea 3
E. No compila por un error en la línea 5
F. Contiene un loop infinito

F. En este ejemplo, falta la instrucción de actualización del bucle `for`, lo cual está bien ya que **la instrucción es opcional**, por lo que la opción D es incorrecta.

La expresión dentro del bucle **aumenta `i`, pero luego asigna a `i` el valor anterior.** Por lo tanto, `i` termina el bucle con el mismo valor que comienza con: 0. Entonces el bucle se repetirá infinitamente, emitiendo la misma declaración una y otra vez porque `i` permanece en `0` después de cada iteración del bucle.

---

[39] FUTURO K-PAZ | MTI ® LUIS HERRERA G.

[40].- ¿Cuál es la salida del siguiente trozo de código?

```
3: byte a = 40, b = 50;
4: byte sum = (byte) a + b;
5: System.out.println(sum);
```

A. 40
B. 50
C. 90
D. No compila por un error en la línea 4
E. Un valor indefinido

D. La línea 4 genera un error de compilación `possible loss of precision`. El operador de de `casteo` tiene una mayor prioridad, por lo que se evalúa primero, `(byte) a`. Luego, se evalúa la suma, lo que hace que a y b se promuevan a valores del tipo `int`.

El valor 90 es un `int` y no se puede asignar a la suma de bytes sin una conversión explícita `(casteo)`, por lo que el código no se compila.

El código podría corregirse agregado paréntesis `(a + b)`, en cuyo caso la opción C sería la respuesta correcta.

---

[40] FUTURO K-PAZ | MTI ® LUIS HERRERA G.