03 Java Creating Using Arrays

Problema 1

Consider the following code:

```
public class ArrayTest {

    static int[][] table = new int[2][3];

    public static void init() {
        for (int x = 0; x < table.length; x++) {
            for (int y = 0; y < table[x].length; y++) {

                //insert code to initialize

            }
        }
    }

    public static void multiply() {
        for (int x = 0; x < table.length; x++) {
            for (int y = 0; y < table[x].length; y++) {

                //insert code to multiply

            }
        }
    }
}
```

Which of the following options can be used in the code above so that the init method initializes each table element to the sum of its row and column number and the multiply method just multiplies the element value by 2?

table[x, y] = x+y;
and
table[x, y] = table[x, y]*2;

table[x][y] = x+y;
and
table[x][y] = table[x][y]*2;

table[[x] [y]] = x+y;
and
table[[x] [y]] = table[[x] [y]]*2;

table(x, y) = x+y;
and
table(x, y) = table(x, y)*2;

La respuesta es [0][1].
This code snippet illustrates correct syntax for accessing array elements in a multi dimensional array. All other options are syntactically incorrect and will not compile.

Prolema 2

What would be the result of compiling and running the following program?

```
class SomeClass{
   public static void main(String args[]){
      int size = 10;
      int[] arr = new int[size];
      for (int i = 0 ; i < size ; ++i) System.out.println(arr[i]);
   }
}
```

You have to select one option:

- The code will fail to compile, because the int[] array declaration is incorrect.
- The program will compile, but will throw an `IndexArrayOutOfBoundsException` when run.
- The program will compile and run without error, and will print nothing.
- The program will compile and run without error and will print null ten times.
    - Here, all the array elements are initialized to have 0.
- The program will compile and run without error and will print 0 ten times.
    - It correctly will declare and initialize an array of length 10 containing int values initialized to have 0.

Explanation:
// Hace un array de int de dimension 10, sin valor inicial por lo que al ser primitivo es 0 en los 10 espacios del array, iterara sólo '0' 10 veces.

Elements of Arrays of primitive types are initialized to their default value (i.e. 0 for integral types, 0.0 for float/double and false for boolean)
Elements of Arrays of non-primitive types are initialized to null.

Problema 3

Which of the following option(s) correctly declare(s)
a variable that can directly reference an array of 10 ints?

You have to select 2 options:

- int[ ] iA
- int[10] iA
    - Size of the array is NEVER specified on the Left Hand Side.
- int iA[ ]
- Object[ ] iA
    - Here, iA is an array of Objects. It cannot hold an array of integers.
- Object[10] iA
    - Size of the array is NEVER specified on the LHS.

Explanation:
Note that an array of integers IS an Object :
Object obj = new int[]{ 1, 2, 3 }; // is valid.
But it is not an array of objects.
Object[] o = new int[10]; // is not valid.

Difference between the placement of square brackets:
int[] i, j; //here i and j are both array of integers.
int i[], j; //here only i is an array of integers. j is just an integer.

Problema 4

Given:

```
String[][] matrix = new String[2][2];
matrix[0][0] = "petrol";
matrix[1][0] = "diesel";
matrix[0][1] = "manual";
matrix[1][1] = "auto";
```

Which option will print petrol:diesel:manual:auto:?

`for(String[] row : matrix)`
`  for (String spec : row)`
`  System.out.print(spec+":");`
 	
This will print petrol:manual:diesel:auto:.
	`for(int i = 0; i<2; i++) {`
	`  for(int j = 0; j<2; j++){`
	
	`System.out.print(matrix[i][j]+":");`
	`  }`
	`}`
	This will print petrol:manual:diesel:auto:.

`for(int i = 0; i<2; i++) {`
`  for(int j = 0; j<2; j++){`
 	
`System.out.print(matrix[j][i]+":");`
`  }`
`}`
	`System.out.println("");`
	`for(int i = 1; i<2; i++) {`
	`  for(int j = 1; j<2; j++){`
	
	`System.out.print(matrix[j][i]+":");`
	`  }`
	`}`

`System.out.println("");`
`for(int i = 1; i<2; i++) {`
`  for(int j = 1; j<2; j++){`
 	
`System.out.print(matrix[i][j]+":");`
`  }`
`}`
This will print just one element auto.
	`for(int i = 2; i>0; i--) {`
	`  for(int j = 0; j<2; j++){`
	
	`System.out.print(matrix[j][i]+":");`
	`  }`
	`}`
	It will throw an ArrayIndexOutOfBoundsException because when i is 2, matrix[0][2] will be trying to access the third element of the array referred to by matrix[0], which is out of bounds.

Explanation:
Array indexing starts with 0. Thus, the given code really creates the following array structure:

```
matrix ==>  { matrix[0] , matrix[1] }
matrix[0] ==>  { matrix[0][0],  matrix[0][1]} ==> { "petrol", "manual"}
matrix[1] ==>  { matrix[1][0],  matrix[1][1]} ==> { "diesel", "auto"}
```

// Fijate que te ayuda la declaracion, la j se cambia con i y debe empezar en '0' para mostrar

Problema 5

Which of the following are valid code fragments:

You have to select 2 options:

- `new Object[]{ "aaa", new Object(), new ArrayList(), 10};`
  - 10 is a primitive and not an Object but due to auto-boxing it will be converted into an Integer object and that object will then be stored into the array of Objects.

- `new Object[]{ "aaa", new Object(), new ArrayList(), {} };`
  - {} is not a valid way to create an Object here. However, it is valid while creating an array. For example, the following are valid: String[] sa = { }; //assigns a valid String[] object of length 0 to sa Object arr[][] = new Object[][] {new String[5], {} }; //assigns a valid Object[] object of length 0 to arr[1].

- `new Object[]{ "aaa", new Object(), new ArrayList(), new String[]{""} };`
  - Every array is an Object so new String[]{""} is also an Object and can be placed in an array of objects.

- `new Object[1]{ new Object() };`
  - You can't specify array length if you are initializing it at the same place.

Explanation:
1. An array of objects can store Objects of any class.
2. Primitives (i.e. int, byte, char, short, boolean, long, double, and float) are NOT objects.
3. An array (of primitives as well as of objects) is an Object.

Problema 6

What will be the result of attempting to compile and run the following class?

```
public class TestClass{
    public static void main(String args[ ] ){
       int i = 1;
       int[] iArr = {1};
       incr(i) ;
       incr(iArr) ;
       System.out.println( "i = " + i + "  iArr[0] = " + iArr [ 0 ] ) ;
    }
    public static void incr(int   n ) { n++ ; }
    public static void incr(int[ ] n ) { n [ 0 ]++ ; }
}
```

You have to select 1 option:

- The code will print i = 1 iArr[0] = 1
- The code will print i = 1 iArr[0] = 2
- The code will print i = 2 iArr[0] = 1
- The code will print i = 2 iArr[0] = 2
- The code will not compile. //The code will not compile.

Explanation:
Arrays are proper objects (i.e. iArr instanceof Object returns true) and Object references are passed by value (so effectively, it seems as though objects are being passed by reference). So the value of reference of iArr is passed to the method incr(int[] i); This method changes the actual value of the int element at 0.

Video: https://youtu.be/b35eNdLEsZw

Problema 7

Consider the following code:

```
//INSERT CODE HERE
a[0][0] = 1;
a[0][1] = 2;


a[1][0] = 3;
a[1][1] = 4;
a[1][2] = 5;
a[1][3] = 6;
```

What can be inserted independently in the above code so that it will compile and run without any error or exception?

```
int[][] a = new int[2][];
```

This will instantiate only the first dimension of the array. The elements in the second dimension will be null. In other words, a will be instantiated to two elements but a[0] and a[1] will be null and so a[0][0] (and access to all other such ints) will throw a NullPointerException.

```
int[][] a = new int[2][4];
```

This is correct because it will instantiate both the dimensions of the array. i.e. a will be initialized with 2 references to int arrays a[0] and a[1]. Further, the arrays pointed to by a[0] and a[1] will also be initialized with size 4.

```
int[][] a = new int[4][2];
```

This will initialize a to an array of size 4 and each element of this array will be initialized to an int array of size 2. Therefore, a[0][2], a[0][3], a[1][2], and a[1][3], will cause an ArrayIndexOutOfBoundsException to be thrown.

```
int[][] a = new int[2][];
a[0] = new int[2];
a[1] = new int[4];
```

Observe that this creates a jagged array. i.e. the elements in the second dimension of a are not of same length. The first element in the second dimension is only of length 2 while the second element is of length 4. Since the given code doesn't need a[0][2] and a[0][3], it is ok.

```
int[][] a = new int[4][];
a[0] = new int[2];
a[1] = new int[2];
```

In this case, a[1][2] and a[1][3] will cause an ArrayIndexOutOfBoundsException to be thrown.

Problema 8

Consider the following program...

```
class ArrayTest{
   public static void main(String[] args){
      int ia[][] = { {1, 2}, null };
      for (int i = 0; i < 2; i++)
         for (int j = 0; j < 2; j++)
            System.out.println(ia[i][j]);
   }
}
```

Which of the following statements are true?
You have to select 1 option:

- It will not compile.
- It will throw an `ArrayIndexOutOfBoundsException` at Runtime.
- It will throw a `NullPointerException` at Runtime.
- It will compile and run without throwing any exceptions.
- None of the above.

Explanation:
It will throw a NullPointerException for ia[1][0] because ia[1] is null. Note that null is not same as having less number of elements in an array than expected. If you try to access ia[2][0], it would have thrown ArrayIndexOutOfBoundsException because the length of ia is only 2 and so ia[2] tries to access an element out of that range. ia[2] is not null, it simply does not exist.

Problema 9

The following class will print 'index = 2' when compiled and run.

```
class Test{
   public static int[ ] getArray() {  return null;  }
   public static void main(String[] args){
      int index = 1;
      try{
         getArray()[index=2]++;
      }
      catch (Exception e){  }  //empty catch
      System.out.println("index = " + index);
   }
}
```

Ya have to select 1 option:

- True
- False

Explanation:
If the array reference expression produces null instead of a reference to an array, then a NullPointerException is thrown at runtime, but only after all parts of the array reference expression have been evaluated and only if these evaluations completed normally.

This means, first index = 2 will be executed, which assigns 2 to index. After that null[2] is executed, which throws a NullPointerException. But this exception is caught by the catch block, which prints nothing. So it seems like NullPointerException is not thrown but it actually is.

In other words, the embedded assignment of 2 to index occurs before the check for array reference produced by getArray().

In an array access, the expression to the left of the brackets appears to be fully evaluated before any part of the expression within the brackets is evaluated. Note that if evaluation of the expression to the left of the brackets completes abruptly, no part of the expression within the brackets will appear to have been evaluated.

Problema 10

Which of these array declarations and instantiations are legal?
You had to select 4 options

- `int[ ] a[ ] = new int [5][4] ;`
  - This will create an array of length 5. Each element of this array will be an array of 4 ints.

- `int a[ ][ ] = new int [5][4] ;`
  - This will create an array of length 5. Each element of this array will be an array of 4 ints.

- `int a[ ][ ] = new int [ ][4] ;`
  - The statement int[ ][4] will not compile, because the dimensions must be created from left to right.

- `int[ ] a[ ] = new int[4][ ] ;`
  - This will create an array of length 4. Each element of this array will be null. But you can assign an array of ints of any length to any of the elements. For example:

```
a[0] = new int[10];//valid
a[1] = new int[4];//valid
a[2] = new int[]; //invalid because you must specify the length
a[3] = new Object[] //invalid because a[3] can only refer to an
array of ints.
```

  - This shows that while creating a one dimensional array, the length must be specified but while creating multidimensional arrays, the length of the last dimension can be left unspecified. Further, the length of multiple higher dimensions after the first one can also be left unspecified if none of the dimensions are specified after it. So for example,

```
a[][][][] = new int[4][3][3][5];
```

    is same as

```
a[][][][] = new int[4][][][];
```

    (Note that the first dimension must be specified.)

  - Thus, multidimensional arrays do not have to be symmetrical.

- `int[ ][ ] a = new int[5][4] ;`
  - This will create an array of length 5. Each element of this array will be an array of 4 ints.

Explanation
The [] notation can be placed both before and after the variable name in an array declaration.

```
int[] ia, ba;  // here ia and ba both are int arrays.
int ia[], ba; //here only 'ia' is int array and ba is an int.
```

Multidimensional arrays are created by creating arrays that can contain references to other arrays .

Problema 11

Consider the following array definitions:

```
int[] array1, array2[];
int[][] array3;
int[] array4[], array5[];
```

Which of the following are valid statements?

You had to select 3 options

- `array2 = array3;`
- `array2 = array4;`
- `array1 = array2;`
- `array4 = array1;`
- `array5 = array3;`

Explanation:
// para mi lo fácil es decir que solo esas opciones señalan un array de 2 dimensiones.y esto es porqueya sea que desde el inicio se dijo array de 2 dimensiones como en 3, o array[] de array[] como en el caso de 2, 4 y 5. El único array de una dimensión de int es 1.

There is a subtle difference between: `int[] i;` and `int i[];` although in both the cases, i is an array of integers. The difference is if you declare multiple variables in the same statement such as: `int[] i, j;` and `int i[], j;`, j is not of the same type in the two cases.
In the first case, j is an array of integers while in the second case, j is just an integer.

Therefore, in this question:
array1 is an array of int array2, array3, array4, and array5 are arrays of int arrays
Therefore, option 1, 2 and 5 are valid.

Problema 12

What will the following program print?

```
class Test{
   public static void main(String[] args){
      int i = 4;
      int ia[][][] = new int[i][i = 3][i];
      System.out.println( ia.length + ", " + ia[0].length+", "+
ia[0][0].length);
   }
}
```

You had to select 1 option

- It will not compile.
- 3, 4, 3
- 3, 3, 3
- 4, 3, 4
- 4, 3, 3

Explanation:
// En mi idea se asigna y una vez asignado el siguiente ya tiene esa nueva asignación de 3

In an array creation expression, there may be one or more dimension expressions, each within brackets. Each dimension expression is fully evaluated before any part of any dimension expression to its right. The first dimension is calculated as 4 before the second dimension expression sets 'i' to 3.
Note that if evaluation of a dimension expression completes abruptly, no part of any dimension expression to its right will appear to have been evaluated.

Problema 13

Which of the options will create and initialize a matrix of ints as shown below?

	1		
	1		
	1		
	1		
	1		

You had to select 1 option

```
int[][] matrix = new int[5][4];
for(int i=1;i<=5; i++) matrix[i][1]  = 1;
```

```
int[][] matrix = new int[5][4];
for(int i=0;i<5; i++) matrix[i][1]  = 1;
```

```
int[][] matrix = new int[4][5];
for(int i=1;i<=5; i++) matrix[i][1]  = 1;
```

```
int[][] matrix = new int[5][4];
for(int i=1;i<=5; i++) matrix[i][2]  = 1;
```

Explanation:
// Mi explicacion fácil es ver bien que aunque las opciones 1 y 3 esten bien en las dimensiones, comienzan con i=1 por lo que nunca pondrá el valor en `matrix[i][1] = 1` en la primera fila.
// Practicamente son 2 pasos y se explican adelante: ver dimension y comienzo de for

You can visualize the given table in two ways - 1. As a table with 5 rows and 4 columns, where the second column of each row has a 1. In this case, you can define your matrix as new int[5][4]. Thus, the elements that you need to populate are: `[0][1]`, `[1][1]`, `[2][1]`, `[3][1]`, and `[4][1]`. This can be easily done using the for loop:

```
for(int i=0; i<5;i++) matrix[i][1] = 1;
```

Remember that array indexing starts with 0. Therefore the addresses of the rows are from 0 to 4 and the address of the second column is 1. 2. As a transposed table with 4 rows and 5 columns, where all the columns of the second row has a 1. In this case, you can define your matrix as `new int[4][5]`. Thus, the elements that you need to populate are: `[1][0]`, `[1][1]`, `[1][2]`, `[1][3]`, and `[1][4]`. This can be easily done using the for loop:

```
for(int i=0; i<5;i++) matrix[1][i] = 1;
```

Remember that array indexing starts with 0. Therefore the address of second row is 1 and the addresses of the columns are from 0 to 4.

Problema 14

Consider the following class…

```
class Test{
   public static void main(String[ ] args){
      int[] a = { 1, 2, 3, 4 };
      int[] b = { 2, 3, 1, 0 };
      System.out.println( a [ (a = b)[3] ] );
   }
}
```

What will it print when compiled and run ?
You had to select 1 option

- It will not compile.
- It will throw ArrayIndexOutOfBoundsException when run.
- It will print 1.
- It will print 3.
- It will print 4

Explanation:
// A mis palabras, se asigna en a[3] el valor de b[3] el cual es cero, por lo que lo que se imprime es
// a[0] que en este caso es 1

In an array access, the expression to the left of the brackets appears to be fully evaluated before any part of the expression within the brackets is evaluated.
In the expression `a[(a=b)[3]]`, the expression a is fully evaluated before the expression `(a=b)[3];` this means that the original value of a is fetched and remembered while the expression `(a=b)[3]` is evaluated. This array referenced by the original value of a is then subscripted by a value that is element 3 of another array (possibly the same array) that was referenced by b and is now also referenced by a.
So, it is actually `a[0] = 1`.
Note that if evaluation of the expression to the left of the brackets completes abruptly, no part of the expression within the brackets will appear to have been evaluated.

Problema 15

What will the following code snippet print?

```
int index = 1;
String[] strArr = new String[5];
String   myStr  = strArr[index];
System.out.println(myStr);
```

You had to select 1 option

- nothing
- null
- It will throw ArrayIndexOutOfBounds at runtime.
- It will print some junk value.
- None of the above.

Explanation:
// Se Inicia un array de String con 5, y mandas una String en el index 1, pero inicializado es ‘null’

When you create an array of Objects (here, Strings) all the elements are initialized to null. So in the line 3, null is assigned to myStr.
Note that. empty string is "" (`String str = "";`) and is not same as null.

Problema 16

What will the following program print?

```
public class TestClass{
  public static void main(String[] args){
    String str = "111";
    boolean[] bA = new boolean[1];
    if( bA[0] ) str = "222";
    System.out.println(str);
  }
}
```

You had to select 1 option

- 111
- 222
- It will not compile as bA[0] is uninitialized.
- It will throw an exception at runtime.
- None of the above.

Explanation:
// Lo que ayuda mucho es que `boolean[] bA =new boolean[1]` es como si inicializaras un boolean primitivo y este comienza con `'false'`, no para por el if y sol imprimirá el valor original del 'String str'

All the arrays are initialized to contain the default values of their type. This means,
`int[] iA = new int[10];` will contain 10 integers with a value of 0.
`Object[] oA = new Object[10];` will contain 10 object references pointing to null.
`boolean[] bA = new boolean[10];` will contain 10 booleans of value false.

So, as `bA[0]` is `false`, the if condition fails and str remains `111`.

Problema 17

Given the following code :

```
public class TestClass {

    int[][] matrix = new int[2][3];

    int a[] = {1, 2, 3};
    int b[] = {4, 5, 6};

    public int compute(int x, int y){
        //1 : Insert Line of Code here
    }

    public void loadMatrix(){
        for(int x=0; x<matrix.length; x++){
            for(int y=0; y<matrix[x].length; y++){
                //2: Insert Line of Code here
            }
        }
    }
}
```

What can be inserted at //1 and //2?
You had to select 1 option

- `return a(x)*b(y);` and `matrix(x, y) = compute(x, y);`
  - (and) are used to call a method on an object. To access array elements, you need to use [and].
- `return a[x]*b[y];` and `matrix[x, y] = compute(x, y);`
- `return a[x]*b[y];` and `matrix[x][y] = compute(x, y);` `//Es el que vi correcto declarado`
- `return a(x)*b(y);` and `matrix(x)(y) = compute(x, y);`
  - a(x), b(y), and matrix(x)(y) are invalid because a, b, and matrix are not methods.
- `return a[x]*b[y];` and `matrix[[x][y]] = compute(x, y);`
  - [[x][y]] is invalid syntax.

The correct syntax to access any element within an array is to use the square brackets - [ ]. Thus, to access the first element in an array, you would use array[0].

Problema 18

Given:

```
public class FunWithArgs {
    public static void main(String[][] args) {
        System.out.println(args[0][1]);
    }
    public static void main(String[] args) {
        FunWithArgs fwa = new FunWithArgs();
        String[][] newargs = {args};
        fwa.main(newargs);
    }
}
```

The above program is compiled with the command line:
`javac FunWithArgs.java`
and then run with:
`java FunWithArgs a b c`

What will be the output?
You had to select 1 option

- It will not compile.
- It will throw ArrayIndexOutOfBoundsException at run time.
- It will print b
- It will print null

Explanation.
// Pensé que no leeria un array de entrada por ser de 1 dimensión pero tanto para imprimir en el método main y en el principal puede ser accedido. Eso quiere decir que args[0][1] es el primer array (y unico) y en el segundo valor, que en este caso es 'b'

There is no problem with the code. The main method is just overloaded. When it is run, the main method with String[] will be called. This method then calls the main with String[][] with an argument { {"a", "b", "c"} } Thus, args[0][1] refers to "b", which is what is printed.

Problema 19

What will be the result of trying to compile and execute the following program?

```
public class TestClass{
   public static void main(String args[] ){
      int i = 0 ;
      int[] iA = {10, 20} ;
      iA[i] = i = 30 ;
      System.out.println(""+ iA[ 0 ] + " " + iA[ 1 ] + "  "+i) ;
    }
}
```

You had to select 1 option

- It will throw ArrayIndexOutOfBoundsException at Runtime.
- Compile time Error.
- It will print 10 20 30
- It will print 30 20 30
- It will print 0 20 30

Explanation:
// De alguna manera ya lo pude ver más rápido, iA[i] en el momento inicial es como si tuviera el valor iA[0], y lo que hace es asignar a iA[0] y a `i` el valor de 30, por lo que el print ya es fácil de ver.

The statement `iA[i] = i = 30 ;` will be processed as follows:
`iA[i] = i = 30; => iA[0] = i = 30 ;  =>  i = 30; iA[0] = i ; =>  iA[0] = 30 ;`

Here is what JLS says on this:
1 Evaluate Left-Hand Operand First
2 Evaluate Operands before Operation
3 Evaluation Respects Parentheses and Precedence
4 Argument Lists are Evaluated Left-to-Right

For Arrays: First, the dimension expressions are evaluated, left-to-right. If any of the expression evaluations completes abruptly, the expressions to the right of it are not evaluated.

Problema 20

Consider the following code:

```
public class ArrayTest {

    static int[][] table = new int[2][3];

    public static void init() {
        for (int x = 0; x < table.length; x++) {
            for (int y = 0; y < table[x].length; y++) {

                //insert code to initialize

            }
        }
    }

    public static void multiply() {
        for (int x = 0; x < table.length; x++) {
            for (int y = 0; y < table[x].length; y++) {

                //insert code to multiply

            }
        }
    }
}
```

Which of the following options can be used in the code above so that the init method initializes each table element to the sum of its row and column number and the multiply method just multiplies the element value by 2?

You had to select 1 option

- `table[x, y] = x+y;` and `table[x, y] = table[x, y]*2;`
- `table[x][y] = x+y;` and `table[x][y] = table[x][y]*2;`
  - This code snippet illustrates correct syntax for accessing array elements in a multi dimensional array. All other options are syntactically incorrect and will not compile.
- `table[[x] [y]] = x+y;` and `table[[x] [y]] = table[[x] [y]]*2;`
- `table(x, y) = x+y;` and `table(x, y) = table(x, y)*2;`

// Es el que está bien escroto

04 Java

Loop Constructs

Problema 1

What will the following code print when compiled and run?

```
class Test{
    public static void main(String args[]){
        int c = 0;
        A: for(int i = 0; i < 2; i++){
            B: for(int j = 0; j < 2; j++){
                C: for(int k = 0; k < 3; k++){
                    c++;
                    if(k>j) break;
                }
            }
        }
        System.out.println(c);
    }
}
```

You have to select 1 option:

- 7
- 8
- 9
- 10
- 11

Explanation:
The point to note here is that a break without any label breaks the innermost outer loop. So in this case, whenever k>j, the C loop breaks. You should run the program and follow it step by step to understand how it progresses.

Problema 2

Which of the following code snippets will compile without any errors?
(Assume that the statement int x = 0; exists prior to the statements below.)

You have to select 3 options:

- `while (false) { x=3; }`
- `if (false) { x=3; }`
- `do{ x = 3; } while(false);`
    - In a do- while, the block is ALWAYS executed at least once because the condition check is done after the block is executed. Unlike a while loop, where the condition is checked before the execution of the block.
- `for( int i = 0; i< 0; i++) x = 3;`

Explanation:
while (false) { x=3; } is a compile-time error because the statement x=3; is not reachable;
Similarly, for(int i = 0; false; i++) x = 3; is also a compile time error because x= 3 is unreachable. In if(false){ x=3; }, although the body of the condition is unreachable, this is not an error because the JLS explicitly defines this as an exception to the rule. It allows this construct to support optimizations through the conditional compilation. For example, if(DEBUG){ System.out.println("beginning task 1"); } Here, the DEBUG variable can be set to false in the code while generating the production version of the class file, which will allow the compiler to optimize the code by removing the whole if statement entirely from the class file.

Problema 3

What will the following code print when compiled and run?

```
public class TestClass{
   public static void main(String[] args){
        int[] arr = { 1, 2, 3, 4, 5, 6 };
        int counter = 0;
        for (int value : arr) {
           if (counter >= 5) {
               break;
           } else {
            continue;
           }
           if (value > 4) {
             arr[counter] = value + 1;
           }
           counter++;
        }
        System.out.println(arr[counter]);
   }
}
```

You have to select 1 option

- It will not compile.
    - Observe that the line if (value > 4) { and the rest of the code in the for loop will not execute in any case. It is therefore unreachable code and the compiler will complain about it.
- It will throw an exception at run time.
- 5
- 6
- 7
- 8

Problema 4

Consider the following code:

```
public static void main(String[] args) {
   int[] values = { 10, 30, 50 };
        for( int val : values ){
            int x = 0;
            while(x<values.length){
             System.out.println(x+" "+val);
              x++;
            }
        }
}
```

How many times is 2 printed out in the output?
You have to select 1 option

- 0
- 1
- 2
- 3

This is a simple while loop nested inside a for loop.
The for loop loops three times - once for each value in values array.
Since, values.length is 3, x is incremented two times for each for loop iteration before the condition x<values.length returns false.
Therefore, it prints:
0 10
1 10
2 10
0 30
1 30
2 30
0 50
1 50
2 50

Problema 5

What will the following program print?

```
class LoopTest{
    public static void main(String args[]) {
        int counter = 0;
        outer:
        for (int i = 0; i < 3; i++) {
            middle:
            for (int j = 0; j < 3; j++) {
                inner:
                for (int k = 0; k < 3; k++) {
                    if (k - j > 0) {
                        break middle;
                    }
                    counter++;
                }
            }
        }
        System.out.println(counter);
    }
}
```

You have to select 1 option:

- 2
- 3
- 6
- 7
- 9

Explanation:
To understand how this loop works let us put some extra print statements in the innermost loop:
System.out.println("i="+i+" j="+j+" k="+k);

```
if(k-j>0){
     System.out.println("breaking middle "+j);
     break middle;
}
counter++;
```

This is what it prints:	breaking middle 0	breaking middle 0	breaking middle 0
i=0 j=0 k=0	i=1 j=0 k=0	i=2 j=0 k=0	3
i=0 j=0 k=1	i=1 j=0 k=1	i=2 j=0 k=1	

The key is that the middle loop is broken as soon as k-j becomes > 0. This happens on every second iteration of inner loop when k is 1 and j is 0. Now, when middle is broken inner cannot continue. So the next iteration of outer starts.

Problema 6

Which of the following code fragments compile without any error?
Assume that Math.random() returns a double between 0.0 and 1.0 (not including 1.0).
You have to select 3 options

```
for(;Math.random()<0.5;){
    System.out.println("true");
}
```

The second expression in a for loop must return a boolean, which is happening here. So this is valid.

```
for(;;Math.random()<0.5){
    System.out.println("true");
}
```

Here, the first part (i.e. the init part) and the second part (i.e. the expression/condition part) part of the for loop are empty. Both are valid. (When the expression/condition part is empty, it is interpreted as true.)
The third part (i.e. the update part) of the for loop does not allow every kind of statement. It allows only the following statements here: Assignment, PreIncrementExpression, PreDecrementExpression, PostIncrementExpression, PostDecrementExpression, MethodInvocation, and ClassInstanceCreationExpression. Thus, Math.random()<0.5 is not valid here, and so this will not compile.

```
for(;;Math.random()){
    System.out.println("true");
}
```

This is a valid never ending loop that will keep printing true.

```
for(;;){
    Math.random()<.05? break : continue;
}
```

This is an invalid use of ? : operator. Both sides of : should return some value. Here, break and continue do not return anything. However, the following would have been valid: for(;Math.random()<.05? true : false;){ }

```
for(;;){
    if(Math.random()<.05) break;
}
```

El bucle for(;;) es una forma de crear un bucle infinito en Java. En este caso, no se especifican condiciones de inicio, condición de continuación o expresiones de incremento, por lo que el bucle se ejecutará de forma indefinida hasta que se alcance una condición de salida, que es cuando Math.random() < 0.05 se evalúa como true.

The three parts of a for loop are independent of each other. However, there are certain rules for each part.

Problema 7

What will the following program print?

```
class Test{
    public static void main(String args[]){
        int c = 0;
        boolean flag = true;
        for(int i = 0; i < 3; i++){
            while(flag){
                c++;
                if(i>c || c>5) flag = false;
            }
        }
        System.out.println(c);
    }
}
```

You have to select 1 option

- 3
- 4
- 5
- 6
- 7

Explanation:
In the first iteration of for loop, the while loop keeps running till c becomes 6. Now, for all next for loop iteration, the while loop never runs as the flag is false. So final value of c is 6.

Problema 8

What can be inserted in the following code so that it will print exactly 2345 when compiled and run?

```
public class FlowTest {

    static int[] data = {1, 2, 3, 4, 5};

    public static void main(String[] args) {
        for (int i : data) {
            if (i < 2) {
                //insert code1 here
            }
            System.out.print(i);
            if (i == 3) {
                //insert code2 here
            }
        }
    }
}
```

You have to select 2 options

```
break;
and
//nothing is required
```

```
continue;
and
//nothing is required
```

```
continue;
and
continue;
```

```
break;
and
continue;
```

```
break;
and
break;
```

Explanation:
This is a very simple loop to follow if you know what break and continue do. break breaks the nearest outer loop. Once a break is encountered, no further iterations of that loop will execute. continue simply starts the next iteration of the loop. Once a continue is encountered, rest of the statements within that loop are ignored (not executed) and the next iteration is started.

Problema 9

What will the following code print when run?

```
public class TestClass {
    public static void main(String[] args) throws Exception {
        String[] sa = {"a", "b", "c"};
        for(String s :  sa){
            if("b".equals(s)) continue;
            System.out.println(s);
            if("b".equals(s)) break;
            System.out.println(s+" again");
        }
    }
}
```

You have to select 1 option

```
a
a again
c
c again
```

```
a
a again
b
```

```
a
a again
b
b again
```

```
c
c again
```

Explanation:
To determine the output you have to run through the loop one iteration at a time in your mind:

Iteration 1: s is "a". It is not equal to "b" so, it will print "a", and then "a again".

Iteration 2: s is "b". It is equal to "b", so the first if will execute "continue", which mean the rest of the code in the loop will not be executed (thus b and b again will not be printed), and the next iteration will start. Note that the second if is not executed at all because of the continue in the first if.

Iteration 3: s is "c", both the if conditions are not satisfied. So "c" and "c again" will be printed.

Problema 10

What will be the result of attempting to compile and run the following program?

```
public class TestClass{
   public static void main(String args[]){
      int x  = 0;
      labelA:   for (int i=10; i<0; i--){
         int j = 0;
         labelB:
         while (j < 10){
            if (j > i) break labelB;
            if (i == j){
               x++;
               continue labelA;
            }
            j++;
         }
         x--;
      }
      System.out.println(x);
   }
}
```

You have to select 1 option:

- It will not compile.
- It will go in infinite loop when run.
- The program will write 10 to the standard output.
- The program will write 0 to the standard output.
- None of the above.

Explanation:
This is just a simple code that is meant to confuse you. Notice the for statement: for(int i=10; i<0; i--). i is being initialized to 10 and the test is i<0, which is false. Therefore, the control will never get inside the for loop, none of the weird code will be executed, and x will remain 0, which is what is printed.

Problema 11

You have been given an array of objects and you need to process this array as follows -
1. Call a method on each object from first to last one by one.
2. Call a method on each object from last to first one by one.
3. Call a method on only those objects at even index (0, 2, 4, 6, etc.)

Which of the following are correct?
You had to select 1 option

- Enhanced for loops can be used for all the three tasks.
- Enhanced for loop can be used for only the first task. For the rest, standard for loops can be used.
- Standard for loops can be used for tasks 1 and 2 but not 3.
- All the tasks can be performed either by using only standard for loops or by using only enhanced for loops.
- Neither standard for loops nor enhanced for loops can be used for all three tasks.

Explanation:
The enhanced for loop is tailor made for processing each element of a collection (or an array) in order. Most importantly, it does not give you an iterating variable that you can manipulate and that makes it impossible to change the order or to skip an element. Therefore, tasks 2 and 3 cannot be done by an enhanced for loop.

The standard for loop is very flexible. It can do pretty much anything. Here is how you can do task 2 and 3 using a standard for loop -
//processing in reverse

```
for(int i=arr.length-1; i>=0; i--){
   arr[i].m1();
}
```

//processing alternate

```
for(int i=0; i<arr.length; i=i+2){
   arr[i].m1();
}
```

Problema 12

How many times will the line marked //1 be called in the following code?

```
int x = 10;
do{
 x--;
 System.out.println(x);  // 1
}while(x<10);
```

You had to select 1 option

- 0
- 1
- 9
- 10
- None of these.

Explanation:
// entra a un bucle infinito ya que por el 'do while' empezará si o si una vez lo que está dentro y entonces x siempre será menor que 10, no saldrá del bucle hasta que se llegue al límite inferior del tipo int y en ese momento pasará al límite superior del int siendo un número positivo y saldrá del bucle. Ese límite es de millones por lo que no es ninguna de las opciones (La opción de ninguna de estas)

A do-while loop is always executed at least once. So in the first iteration, x is decremented and becomes 9. Now the while condition is tested, which returns true because 9 is less than 10. So the loop is executed again with x = 9. In the loop, x is decremented to 8 and the condition is tested again, which again returns true because 8 is less than 10.

As you can see, x keeps on decreasing by one in each iteration and every time the condition x<10 returns true. However, after x reaches -2147483648, which is its MIN_VALUE, it cannot decrease any further and at this time when x-- is executed, the value rolls over to 2147483647, which is Integer.MAX_VALUE. At this time, the condition x<10 fails and the loop terminates.

Problema 13

Consider the following code snippet:

```
for(int i=INT1; i<INT2; i++){
    System.out.println(i);
}
```

INT1 and INT2 can be any two integers.

Which of the following will produce the same result?
//El Print algo, es porque se piensa que INT1=1 e INT2=3

- `for(int i=INT1; i<INT2; System.out.println(++i));`
  - Prints: 2 and 3
- `for(int i=INT1; i++<INT2; System.out.println(i));`
  - Prints: 2 and 3
- `int i=INT1; while(i++<INT2) { System.out.println(i); }`
  - Prints: 2 and 3
- `int i=INT1; do { System.out.println(i); }while(i++<INT2);`
  - Prints: 1 2 and 3
- None of these.

Explanation:
In such a question it is best to take a sample data such as INT1=1 and INT2=3 and execute the loops mentally.

Eliminate the wrong options. In this case, the original loop will print:

```
=====ORIGINAL ====
1
2
```

Outputs of all the options are given above (Ignoring the line breaks).

Thus, none of them is same as the original.

Problema 14

What will the following code print when compiled and run?

```
public class DaysTest{

    static String[] days = {"monday", "tuesday", "wednesday", "thursday",
    "friday", "saturday", "sunday" };

    public static void main(String[] args) {

        int index = 0;
        for(String day : days){

            if(index == 3){
                break;
            }else {
                 continue;
            }
            index++;
            if(days[index].length()>3){
                days[index] = day.substring(0,3);
            }
        }
        System.out.println(days[index]);
    }
}
```

You had to select 1 option

- mon
- thu
- fri
- It will not compile.
- It will throw an exception at run time.

Explanation:
//nunca entra al else y el compilador te diría que es innecesario, NO COMPILA.
Notice the statement :

```
if(index == 3){
                break;
             }else {
                  continue;
             }
```

In no situation can the control go beyond this statement in the for loop. Therefore, rest of the statements in the for loop are unreachable and so the code will not compile.

Problema 15

Which of these statements are valid when occurring by themselves in a method?
You had to select 3 options

- while ( ) break ;
    - The condition expression in a while header is required.

- do { break ; } while (true) ;

- if (true) { break ; } (When not inside a switch block or a loop)
    - You cannot have break or continue in an 'if' or 'else' block without being inside a loop. Note that the problem statement mentions, "...occuring by themselves". This implies that the given statement is not wrapped within any other block. Note: break with a label is possible in an if/else statement without a loop: label: if(true){ System.out.println("break label"); break label; //this is valid }

- switch (1) { default : break; }
    - You can use a constant in switch(...);

- for (; true ;) break ;

Explanation:
// Tal cual, se requiere la expresión del while, y el true puede tenerlo pero si está dentro de un ciclo.

It is not possible to break out of an if statement. But if the if statement is placed within a switch statement or a loop construct, the usage of break in option 3 would be valid.

Problema 16

What will the following program print?

```
class Test{
   public static void main(String args[]){
      int var = 20, i=0;
      do{
         while(true){
         if( i++ > var) break;
         }
      }while(i<var--);
      System.out.println(var);
   }
}
```

You had to select 1 option

- 19
- 20
- 21
- 22
- It will enter an infinite loop.

Explanation:
// El pedo es ese if,peor la explicación es:
// Cuando la condición del if se evalúa, primero se compara el valor actual de i con var. Luego, i se incrementa en 1 después de la comparación.
// Cuando i=20 => `( i++ > 20 )` => `false` entonces itera otra vez el `while` y suma i = 21
// Cuando i=21 => `( i++ > 20 )` => `true` entonces entra al `if`, sale del `while` infinito y suma
// `i=22`

When the first iteration of outer do-while loop starts, var is 20. Now, the inner loop executes till i becomes 21. Now, the condition for outer do-while is checked, while(22 < 20), [i is 22 because of the last i++>var check], thereby making var 19. And as the condition is false, the outer loop also ends. So, 19 is printed.

Problema 17

Consider the following class :

```
class Test{
   public static void main(String[] args){
      for (int i = 0; i < 10; i++) System.out.print(i + " ");  //1
      for (int i = 10; i > 0; i--) System.out.print(i + " ");  //2
      int i = 20;                  //3
      System.out.print(i + " ");   //4
   }
}
```

Which of the following statements are true?
You had to select 4 options

- As such, the class will compile and print "20 " (without quotes) at the end of its output.
- It will not compile if line 3 is removed.
    - If //3 is removed, 'i' will be undefined for //4
- It will not compile if line 3 is removed and placed before line 1.
- It will not compile if line 4 is removed and placed before line 3.
- Only Option 2, 3, and 4 are correct.

Explanation:
The scope of a local variable declared in 'for' statement is the rest of the 'for' statement, including its own initializer.
So, when line 3 is placed before line 1, there is a redeclaration of i in the first for() which is not legal.

As such, the scope of i's declared in for() is just within the 'for' blocks. So placing line 4 before line 3 will not work since 'i' is not in scope there.

Problema 18

Which of these for statements are valid?

```
1. for (int i=5; i=0; i--) { }

2.  int j=5;
      for(int i=0, j+=5;  i<j ; i++) {  j--;  }

3. int i, j;
    for (j=10; i<j; j--) { i += 2; }

4. int i=10;
    for ( ; i>0 ; i--) { }

5. for (int i=0, j=10; i<j; i++, --j) {;}
```

You had to select 1 option

- 1, 2
- 3, 4
- 1, 5
- 1 is not valid.
- 4, 5
- 5

No 1. uses '=' instead of '==' for condition which is invalid. The loop condition must be of type boolean.

No 2. uses 'j +=5'. Now, this statement is preceded by 'int i=0,' and that means we are trying to declare variable j. But it is already declared before the for loop. If we remove the int in the initialization part and declare i before the loop then it will work. But if we remove the statement int j = 5; it will not work because compiler will try to do j = j+5 and you can't use the variable before it is initialized. Although the compiler gives a message 'Invalid declaration' for j += 5, it really means the above mentioned thing.

No 3. i is uninitialized.

No 4. is valid. It contains empty initialization part.

No 5. This is perfectly valid. You can have any number of comma separated statements in initialization and incrementation part. The condition part must contain a single expression that returns a boolean. All a for loop needs is two semi colons :- for(; ;) {} This is a valid for loop that never ends. A more concise form for the same is : for(; ;);

Problema 19

What will be the output if you run the following program?

```
public class TestClass{
   public static void main(String args[]){
      int i;
      int j;
      for (i = 0, j = 0 ; j < 1 ; ++j , i++){
         System.out.println( i + " " + j );
      }
      System.out.println( i + " " + j );
   }
}
```

You had to select 1 option

- 0 0 will be printed twice.
- 1 1 will be printed once.
- 0 1 will be printed followed by 1 2.
- 0 0 will be printed followed by 1 1.
- It will print 0 0 and then 0 1.

Explanation:
j will be less than 1 for only the first iteration. So, first it will print 0, 0. Next, i and j are incremented. Because j is not less than 1 at the start of the loop, the condition fails and it comes out of the loop. Finally, it will print 1,1.

Problema 20

What will be the result of attempting to compile and run the following program?

```
class TestClass{
   public static void main(String args[]){
      boolean b = false;
      int i = 1;
      do{
         i++ ;
      } while (b = !b);
      System.out.println( i );
   }
}
```

You had to select 1 option

- The code will fail to compile, 'while' has an invalid condition expression.
    - It is perfectly valid because b = !b; returns a boolean, which is what is needed for while condition.
- It will compile but will throw an exception at runtime.
- It will print 3.
    - The loop body is executed twice and the program will print 3.
- It will go in an infinite loop.
- It will print 1.

Explanation:
Unlike the 'while(){}' loop, the 'do {} while()' loop executes at least once because the condition is checked after the iteration.

Problema 21

Given:

```
package loops;
public class JustLooping {
    private int j;
    void showJ(){
        while(j<=5){
            for(int j=1; j <= 5;){
                System.out.print(j+" ");
                j++;
            }
            j++;
        }
    }
    public static void main(String[] args) {
        new JustLooping().showJ();
    }
}
```

What is the result?
You had to select 1 option

- It will not compile.
    - There is no problem with the code. The variable j declared in the for loop shadows the instance member j inside the for loop.
- It will print 1 2 3 4 5 five times.
- It will print 1 3 5 five times.
- It will print 1 2 3 4 5 once.
- It will print 1 2 3 4 5 six times.

Explanation:
The point to note here is that the j in for loop is different from the instance member j. Therefore, j++ occuring in the for loop doesn't affect the while loop. The for loop prints 1 2 3 4 5.

The while loop runs for the values 0 to 5 i.e. 6 iterations. Thus, 1 2 3 4 5 is printed 6 times. Note that after the end of the while loop the value of j is 6.

Problema 22

What will be the result of attempting to compile and run the following program?

```
class TestClass{
   public static void main(String args[]){
      int i = 0;
      for (i=1 ;  i<5  ; i++) continue;  //(1)
      for (i=0 ;       ; i++) break;       //(2)
      for (    ; i<5?false:true ;    );     //(3)
   }
}
```

You had to select 1 option

- The code will compile without error and will terminate without problem when run.
- The code will fail to compile, since the continue can't be used this way.
- The code will fail to compile, since the break can't be used this way.
- The code will fail to compile, since the for statement in line 2 is invalid.
- The code will compile without error but will never terminate.
    - the condition part is 'false' so the control will never go inside the loop.

Explanation:
A continue statement can occur in and only in loops i.e. for/enhanced-for, while, and do-while loop. A continue statement means: Forget about the rest of the statements in the loop and start the next iteration. So,

`for (i=1 ;  i<5  ; i++) continue;` just increments the value of i up to 5 because of i++.

`for (i=0 ;       ; i++) break;` iterates only once because of the break so the value of i becomes 0.

`for (    ; i<5?false:true ;    );` never iterates because i is less than 5 (it is 0 because of //2) and the condition expression is false!

At the end of the code, the value of i is 0.

Problema 23

What will the following code print?

```
public class BreakTest{
  public static void main(String[] args){
    int i = 0, j = 5;
    lab1 : for( ; ; i++){
      for( ; ; --j)  if( i >j ) break lab1;
    }
    System.out.println(" i = "+i+", j = "+j);
  }
}
```

You had to select 1 option

- `i = 1, j = -1`
- `i = 1, j = 4`
- `i = 0, j = 4`
- `i = 0, j = -1`
- It will not compile.

Explanation:
The values of i and j in the inner most for loop change as follows:

```
i = 0 j = 5
i = 0 j = 4
i = 0 j = 3
i = 0 j = 2
i = 0 j = 1
i = 0 j = 0
i = 0 j = -1
```

Therefore, the final println prints `i = 0, j = -1`

05 Java Constructors, Methods, & Encapsulation

Problema 1

Given:

```
class Triangle{
    public int base;
    public int height;
    private final double ANGLE;

    public  void setAngle(double a){  ANGLE = a;  }

    public static void main(String[] args) {
        Triangle t = new Triangle();
        t.setAngle(90);
    }
}
```

Identify the correct statement(s).
You have to select 1 option

- the value of ANGLE will not be set to 90 by the setAngle method.
- An exception will be thrown at run time.
- The code will work as expected setting the value of ANGLE to 90.
- The code will not compile.

Explanation:
The given code has two problems: 1. If you declare a field to be final, it must be explicitly initialized by the time the creation of an object of the class is complete. So you can either initialize it immediately: private final double ANGLE = 0; or you can initialize it in the constructor or an instance block. 2. Since ANGLE is final, you can't change its value once it is set. Therefore the setAngle method will also not compile.

Problema 2

Given the following source code, which of the lines that are commented out may be reinserted without introducing errors?

```
abstract class Bang{
 //abstract void f();  //(0)
   final    void g(){}
 //final    void h(){} //(1)
   protected static int i;
   private int j;
}

final class BigBang extends Bang{
 //BigBang(int n) { m = n; } //(2)
   public static void main(String args[]){
      Bang mc = new BigBang();
   }
   void h(){}
 //void k(){ i++; } //(3)
 //void l(){ j++; } //(4)
   int m;
}
```

You have to select 1 option:

`final void h( ) { } //(1)`
It will fail because BigBang will try to override a final method.

`BigBang(int n) { m = n; }  //(2)`
It will fail since BigBang will no longer have a default constructor that is used in the main( ) method.

`void k( ) { i++; }  //(3)`

`void l( ) { j++; }  //(4)`
It will fail since the method will try to access a private member 'j' of the superclass.

`abstract void f( ) ;  //(0)`
If this line is inserted, then either the class BigBang will have to be declared abstract or it has to implement method f().

Explanation:
Default constructor (having no arguments) is automatically created only if the class does not define any constructors. So as soon as //2 is inserted the default constructor will not be created.

Problema 3

What will the following code print when run?

```
public class Noobs {
    public void m(int a){
        System.out.println("In int ");
    }
    public void m(char c){
        System.out.println("In char ");
    }
    public static void main(String[] args) {
        Noobs n = new Noobs();
        int a = 'a';
        char c = 6;
        n.m(a);
        n.m(c);
    }
}
```

You have to select 1 option:

- In int
  In char

- In char
  In int

- In int
  In int

- In char
  In char

- It will not compile.

Explanation:
It looks confusing but it is a simple question. Remember that whenever two methods are applicable for a method call, the one that is most specific to the argument is chosen.
In case of `m(a)`, a is an `int`, which cannot be passed as a char (because an int cannot fit into a char). Therefore, only `m(int)` is applicable.
In case of `m(c)`, c is a `char`, which can be passed as an int as well as a char. Therefore, both the methods are applicable. However, `m(char)` is most specific therefore that is chosen over m(`int`).

Problema 4

What should be placed in the two blanks so that the following code will compile without errors:

```
class XXX{
    public void m() {
        throw new RuntimeException();
    }
}
class YYY extends XXX{
    public void m() throws Exception{
      throw new Exception();
    }
}
public class TestClass {
    public static void main(String[] args) {
        ______ obj = new ______();
        obj.m();
    }
}
```

You have to select 1 option

- XXX and YYY
- XXX and XXX
- YYY and YYY
- YYY and XXX
- None of the options will make the code compile.
    - Remember that an overriding method can only throw a subset of checked exceptions declared in the throws clause of the overridden method. Here, method m in XXX does not declare any checked exception in its throws clause, therefore, method m in YYY cannot declare any checked exception in its throws clause either. Class YYY will, therefore, not compile.

Problema 5

Consider the following code:

```
public class MyClass {

  protected int value = 10;

}
```

Which of the following statements are correct regarding the field value?
You have to select 1 option:

- It cannot be accessed from any other class.
- It can be read but cannot be modified from any other class.
- It can be modified but only from a subclass of MyClass.
    - It can also be modified from any class defined in the same package.
- It can be read and modified from any class within the same package.
    - Note that since value is protected, a class in another package which extends MyClass will only inherit this variable, but it cannot read or modify the value of a variable of a MyClass instance. For example:
    - `//in different package`
    - `class X extends MyClass{`
    - `    public static void main(String[] args){`
    - `        int a = new MyClass().value; //This will not compile`
    - `//because X does not own MyClass's value.`
    - `        a = new X().value; //This will compile fine because X inherits value.`
    - `    }`
    - `}`

Pregunta 6

What should be the return type of the following method?

```
public RETURNTYPE methodX( byte by){
    double d = 10.0;
    return (long) by/d*3;
}
```

You have to select 1 option:

- int
- long
- double
- float
- byte

Explanattion:
Note that the cast (long) applies to 'by' not to the whole expression.

```
( (long) by ) / d * 3;
```

Now, division operation on long gives you a double. So the return type should be double.

Problema 7

What would be the result of attempting to compile and run the following program?

```
class TestClass{
   static TestClass ref;
   String[] arguments;
   public static void main(String args[]){
      ref = new TestClass();
      ref.func(args);
   }
   public void func(String[] args){
      ref.arguments = args;
   }
}
```

You had to select 1 option

- The program will fail to compile, since the static method main is trying to call the non-static method `func`.
    - The concept here is that a non-static method (i.e. an instance method) can only be called on an instance of that class. Whether the caller itself is a static method or not, is immaterial.
    - The main method is calling ref.func(); - this means the main method is calling a non-static method on an actual instance of the class TestClass (referred to by 'ref'). Hence, it is valid. It is not trying calling it directly such as func() or this.func(), in which case, it would have been invalid.

- The program will fail to compile, since the non-static method func cannot access the static member variable `ref`.
    - Non static methods can access static as well as non static methods of a class.

- The program will fail to compile, since the argument `args` passed to the static method `main` cannot be passed on to the non-static method `func`.
- It certainly can be.

- The program will fail to compile, since method `func` is trying to assign to the non-static member variable `'arguments'` through the static member variable ref.

- The program will compile and run successfully.

Problema 8

Identify correct option(s)
You had to select 2 options

- Multiple inheritance of state includes ability to inherit instance methods from multiple classes.
    - Methods do not have state. Ability to inherit instance methods from multiple classes is called multiple inheritance of implementation. Default methods introduce one form of multiple inheritance of implementation. A class can implement more than one interface, which can contain default methods that have the same name. However, such a class cannot be compiled. In this case, the implementing class is required to provide its own implementation of the common method to avoid ambiguity.

- Multiple inheritance of state includes ability to inherit instance fields from multiple classes.

- Multiple inheritance of type includes ability to inherit instance fields as well as instance methods from multiple classes.

- Multiple inheritance of type includes ability to implement multiple interfaces and ability to inherit static or instance fields from interfaces and/or classes.

- Multiple inheritance of type includes ability to implement multiple interfaces and/or ability to extend from multiple classes.

Explanation:
Interfaces, classes, and enums are all "types". Java allows a class to implement multiple interfaces. In this way, Java supports multiple inheritance of types.
"State", on the other hand, is represented by instance fields. Only a class can have instance fields and therefore, only a class can have a state. (Fields defined in an interface are always implicitly static, even if you don't specify the keyword static explicitly. Therefore, an interface does not have any state.) Since a class is allowed to extend only from one class at the most, it can inherit only one state. Thus, Java does not support multiple inheritance of state.

Problema 9

Given the following code, which method declarations can be inserted at line 1 without any problems?

```
public class OverloadTest{
    public int sum(int i1, int i2) { return i1 + i2; }
    // 1
}
```

You had to select 3 options

- `public int sum(int a, int b) { return a + b; }`
    - Will cause duplicate method. Variable names don't matter. Only their types.

- `public int sum(long i1, long i2) { return (int) i1; }`

- `public int sum(int i1, long i2) { return (int) i2; }`

- `public long sum(long i1, int i2) { return i1 + i2; }`

- `public long sum(int i1, int i2) { return i1 + i2; }`
    - Only the return type is different so the compiler will complain about having duplicate method sum.

Explanation:

The rule is that you cannot have methods that create ambiguity for the compiler in a class. It is illegal for a class to have two methods having same name and having same type of input parameters in the same order.
Name of the input variables and return type of the method are not looked into.
1. Option 1 is wrong because, then both the methods will be same (as their method name and the class/type and order of the input parameters will be same). So this amounts to duplicate method which is not allowed. As mentioned, name of the input parameters does not matter. Only the type of parameters and their order matters.
2. 2 is valid because the type of input parameters are different. So this is a different method and does not amount to duplication.
3 and 4 are valid for the same reason
5 is not valid because it leads to duplicate method(as their method name and the class/type and order of the input parameters will be same). Note that as mentioned in the comments, return type does not matter.

Problema 10

Which of the following methods does not return any value?
You had to select 1 option

- ```
  public doStuff() throws FileNotFoundException, IllegalArgumentException{
      //valid code not shown
  }
  ```
  - It is missing the return type. Every method must have a return type specified in its declaration. It could be a valid constructor though if the class is named doStuff because the constructors don't return anything, not even void.

- ```
  public null doStuff() throws FileNotFoundException, IllegalArgumentException{
      //valid code not shown
  }
  ```
  - null can be a return value not a return type because null is not a type.

- ```
  public doStuff() {
      //valid code not shown
  }
  ```
  - This is not a valid method because there is no return type declared. Although it can be a valid constructor if the name of the class is doStuff.

- ```
  public void doStuff() throws FileNotFoundException, IllegalArgumentException{
      //valid code not shown
  }
  ```
  - A method that does not return anything should declare its return type as void. Note that this is different from constructors. A constructor also doesn't return anything but for a constructor, you don't specify any return type at all. That is how a constructor is differentiated from a regular method.

- ```
  private doStuff() {
      //valid code not shown
  }
  ```
  - This is not a valid method because there is no return type declared. Although it can be a valid constructor if the name of the class is doStuff.

Problema 11

Consider the following code:

```
import java.util.ArrayList;

public class Student{

    ArrayList<Integer> scores;
    private double average;

    public ArrayList<Integer> getScores(){ return scores; }

    public double getAverage(){ return average; }

    private void computeAverage(){
        //valid code to compute average
        average =//update average value
    }

    public Student(){
        computeAverage();
    }

    //other code irrelavant to this question not shown

}
```

What can be done to improve the encapsulation of this class?
You had to select 2 options

- Make the class private.
- Make the scores instance field private.
  - An important aspect of encapsulation is that other classes should not be able to modify the state fields of a class directly. Therefore, the data members should be private (or protected if you want to allow subclasses to inherit the field) and if the class wants to allow access to these fields, it should provide appropriate setters and getters with public access.
- Make getScores() protected.
- Make computeAverage() public.
- Change getScores to return a copy of the scores list:
  ```
  public ArrayList<Integer> getScores(){
      return new ArrayList(scores);
  }
  ```
  - If you return the same scores list, the caller would be able to add or remove elements from it, thereby rendering the average incorrect. This can be prevented by returning a copy of the list.
  ```

Problema 12

What will be the result of attempting to compile the following program?

```
public class TestClass{
   long l1;
   public void TestClass(long pLong) { l1 = pLong ; }  //(1)
   public static void main(String args[]){
      TestClass a, b ;
      a = new TestClass();  //(2)
      b = new TestClass(5);  //(3)
   }
}
```

You had to select 1 option

- A compilation error will be encountered at (1), since constructors should not specify a return value.
  - But it becomes a valid method if you give a return type.
- A compilation error will be encountered at (2), since the class does not have a default constructor.
  - The class has an implicit default constructor since the class doesn't have any constructor defined.
- A compilation error will be encountered at (3).
  - Because (1) is a method and not a constructor. So there is no constructor that take a parameter.
- The program will compile correctly.
- It will not compile because parameter type of the constructor is different than the type of value passed to it.
  - If (1) was a valid constructor 'int' would be promoted to long at the time of passing.

Explanation:
// Yopienso que la idea es engañar que es un constructor, peor al tener void, es un metodo

The declaration at (1) declares a method, not a constructor because it has a return value.
The method happens to have the same name as the class, but that is ok.

The class has an implicit default constructor since the class contains no constructor declarations.
This allows the instantiation at (2) to work.

Problema 13

Given the following code, which of the constructors shown in the options can be added to class B without causing a compilation to fail?

```
class A{
   int i;
   public A(int x) { this.i = x; }
}
class B extends A{
   int j;
   public B(int x, int y) { super(x);  this.j = y; }
}
```

You had to select 2 options

- `B( ) { }`
- `B(int y ) { j = y; }`
- `B(int y ) { super(y*2 ); j = y; }`
- `B(int y ) { i = y; j = y*2; }`
- `B(int z ) { this(z, z); }`

Explanation:
1. Remember that an instance of a class is also an instance of its parent class. Therefore, as a part of constructing an instance of a subclass, the JVM has to initialize those parts of the instance that are inherited from the super class as well. Further, the parts inherited from the super class need to be initialized first because the subclass may depend on them. Since it is the job of a constructor to initialize an instance, a constructor of the super class has to be invoked before the constructor of the subclass can proceed. The compiler ensures that at least one constructor of the super class is invoked if you do not explicitly call a super class's constructor by adding super(); (i.e. a call to the no-args constructor) as the first line of the sub class constructor. It automatically adds this call IF and ONLY IF the subclass's constructor does not explicitly call any of the super class's constructor in the first line of its code.

Now, if the super class (which means class A in this question) does not have a no-args constructor, the call to `super()`; will fail. Hence, choices `B( ) { }`, `B(int y ) { j = y; }` and `B(int y ) { i = y; j = y*2; }` are not valid because these constructors do not explicitly invoke the super class's construct and the compiler automatically inserts a call to `super()` in them but A does not have a no-args constructor.

Choice `B(int y ) { super(y*2 ); j = y; }` is valid because it explicitly calls `super( int )`, which is available in A.

2. Instead of calling a super class's constructor using super(<args>), you can also call another constructor of the sub class in the first line (as given in choice `B(int z ) { this(z, z); }`). The first line of this constructor is a call to `this(int, int)`, which causes `B(int, int)` to be invoked and `B(int, int)` calls super(int). So the super class A is correctly instantiated before the sub class B begins initialization.

Problema 14

Consider the following class:

```
class TestClass{
    void probe(int... x) { System.out.println("In ..."); }  //1

    void probe(Integer x) { System.out.println("In Integer"); } //2

    void probe(long x) { System.out.println("In long"); } //3

    void probe(Long x) { System.out.println("In LONG"); } //4

    public static void main(String[] args){
        Integer a = 4; new TestClass().probe(a); //5
        int b = 4; new TestClass().probe(b); //6
    }
}
```

What will it print when compiled and run?
You had to select 2 options

- In Integer and In long
- In ... and In LONG, if //2 and //3 are commented out.
- In Integer and In ..., if //4 is commented out.
- It will not compile, if //1, //2, and //3 are commented out.
- In LONG and In long, if //1 and //2 are commented out.

Explanation:

To answer this type of questions, you need to know the following rules:

1. The compiler always tries to choose the most specific method available with least number of modifications to the arguments.
2. Java designers have decided that old code should work exactly as it used to work before boxing-unboxing functionality became available.
3. Widening is preferred to boxing/unboxing (because of rule 2), which in turn, is preferred over var-args.

Thus,
1.
`probe(Integer)` will be bound to `probe(Integer)` (exact match). If that is not available, it will be bound to `probe(long)`, and then with `probe(int...)` in that order of preference.
`probe(long)` is preferred over `probe(int...)` because unboxing an Integer gives an int and in pre 1.5 code `probe(long)` is compatible with an int (Rule 2).

It is never bound to `probe(Long )` because `Integer` and `Long` are different object types and there is no IS-A relation between them. (This holds true for any two wrapper classes).
It could, however, be bound to `probe(Object )` (if it existed), because Integer `IS-A` Object.

2.
`probe(int)` is bound to probe(long) (because of Rule 2) , then to `probe(Integer )` because boxing an int qives you an Integer, which matches exactly to `probe(Integer)`, and then to `probe(int...)`.

It is never bound to `probe(Long )` because int is not compatible with Long.

We advise you to run this program and try out various combinations. The exam has questions on this pattern but they are not this tough. If you have a basic understanding, you should be ok.

Problema 15

Consider the following code:

```
public class TestClass{
   public void method(Object o){
      System.out.println("Object Version");
   }
   public void method(java.io.FileNotFoundException s){
      System.out.println("java.io.FileNotFoundException Version");
   }
   public void method(java.io.IOException s){
      System.out.println("IOException Version");
   }
   public static void main(String args[]){
      TestClass tc = new TestClass();
      tc.method(null);
   }
}
```

What would be the output when the above program is compiled and run? (Assume that `FileNotFoundException` is a subclass of `IOException`, which in turn is a subclass of Exception)

- It will `print Objec`t Version
- It will `print java.io.IOException` Version
- It will `print java.io.FileNotFoundException` Version
- It will not compile.
- It will throw an exception at runtime.

Explanation:

The reason is quite simple, the most specific method depending upon the argument is called. Here, null can be passed to all the 3 methods but `FileNotFoundException` class is the subclass of `IOException` which in turn is the subclass of Object. So, `FileNotFoundException` class is the most specific class. So, this method is called. Had there been two most specific methods, it would not even compile as the compiler will not be able to determine which method to call. For example:

```
public class TestClass{
    public void method(Object o){
       System.out.println("Object Version");
    }
    public void method(String s){
       System.out.println("String Version");
    }
    public void method(StringBuffer s){
       System.out.println("StringBuffer Version");
    }
    public static void main(String args[]){
       TestClass tc = new TestClass();
       tc.method(null);
    }}
```

Here, `null` can be passed as both `StringBuffer` and `String` and none is more specific than the other. So, it will not compile.

Problema 16

Complete the following code by filling the two blanks -

```
class XXX{
    public void m() throws Exception{
        throw new Exception();
    }
}
class YYY extends XXX{
    public void m(){ }

    public static void main(String[] args) {
        ________  obj = new ______();
        obj.m();
    }
}
```

You had to select 1 option

- XXX XXX
- XXX YYY
- YYY XXX
- YYY YYY

Explanation:

This question is based on two concepts -

1. The overriding method may choose to have no `throws` clause even if the overridden method has a throws clause. Thus, the method m in YYY is valid.

2. Whether a call needs to be wrapped in a try/catch or whether the enclosing method requires a throws clause depends on the class of the reference and not the class of the actual object. This is because it is the compiler that checks whether a call needs to have exception handling and the compiler knows only about the declared class of a variable. It doesn't know about the actual object referred to by that variable (which is known only to JVM at run time).

Here, if you define `obj` of type XXX, the call `obj.m()` will have to be wrapped into a try/catch because `main()` doesn't have a throws clause. But if you define `obj` of class `YYY`, there is no need of try catch because `YYY's m()` does not throw an exception.

Now, if the declared class of obj is YYY, you cannot assign it an object of class XXX because XXX is a superclass of YYY. So the only option left is to do:

YYY `obj = new YYY();`

Problema 17

Given the following definition of class, which member variables are accessible from OUTSIDE the package com.enthu.qb?

```
package com.enthu.qb;
public class TestClass{
   int i;
   public int j;
   protected int k;
   private int l;
}
```

You have to select 2 Options:

Member variable i.
No modifier means package(or default access) and is only accessible inside the package.

Member variable j.
public things (classes, methods and fields) are accessible from anywhere.

Member variable k.
Only if the accessing class is a subclass of TestClass.

Member variable k, but only for subclasses.
protected things (methods and fields) can be accessed from within the package and from subclasses

Member variable l.
private things are accessible only from the class that has it.

Explanation:
`public > protected > package (i.e. no modifier) > private`
where `public` is least restrictive and `private` is most restrictive.
Remember:
`protected` is less restrictive than package access. So a method(or field) declared as protected will be accessible from a subclass even if the subclass is not in the same package.
The same is not true for package access.
A top level class can only have either `public` or no access modifier but a method or field can have all the four. Note that `static`, `final`, `native` and `synchronized` are not considered as access modifiers.

Problema 18

Which of the following are true regarding overloading of a method?
You had to select 1 option

- An overloading method must have a different parameter list and same return type as that of the overloaded method.
    - There is no restriction on the return type. If the parameters are different then the methods are totally different (other than the name) so their return types can be anything.

- If there is another method with the same name but with a different number of arguments in a class then that method can be called as overloaded.

- If there is another method with the same name and same number and type of arguments but with a different return type in a class then that method can be called as overloaded.
    - For overloading a method, the "signature" of the overloaded methods must be different. In simple terms, a method signature includes method name and the number and type of arguments that it takes. So if the parameter list of the two methods with the same name are different either in terms of number or in terms of the types of the parameters, then they are overloaded.
    - For example: Method m1 is overloaded if you have two methods : void m1(int k); and void m1(double d); or if you have: void m1(int k); and void m1(int k, double d);
    - Note that return type is not considered a part of the method signature.

- An overloaded method means a method with the same name and same number and type of arguments exists in the super class and sub class.
    - This is called overriding and not overloading.

Problema 19

Consider the following class...

```
class TestClass{
   int x;
   public static void main(String[] args){
      // lot of code.
   }
}
```

You had to select 1 option

- By declaring x as static, main can access this.x
- By declaring x as public, main can access this.x
- By declaring x as protected, main can access this.x
- main cannot access this.x as it is declared now.
    - Because main() is a static method. It does not have 'this'!
- By declaring x as private, main can access this.x

Explanation
// Fijate que las opciones tienen `this.x`

It is not possible to access x from main without making it static. Because main is a static method and only static members are accessible from static methods. There is no 'this' available in main so none of the this.x are valid.

Problema 20

What can be added to the following Person class so that it is properly encapsulated and the code prints 29?

```
class Person{
    //Insert code here
}
public class Employee extends Person{
        public static void main(String[] args) {
            Employee e = new Employee();
            e.setAge(29);
            System.out.println(e.getAge());
        }
}
```

You had to select 2 options

```
private int age;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
```

```
protected int age;
   public int getAge() {
       return age;
   }
   public void setAge(int age) {
       this.age = age;
   }
```

protected is not a valid way to encapsulate a field because any class in a package can access the field.

```
int age;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
```

No access modifer to age means it has default access i.e. all the members of the package can access it. This breaks encapsulation.

```
private int age;
   private int getAge() {
       return age;
   }
   private void setAge(int age) {
       this.age = age;
   }
```

If you make getAge and setAge private, you cannot call them from Employee class.

```
    private int age;
    public int getAge() {
        return age;
    }
    protected void setAge(int age) {
        this.age = age;
    }
```

This is a ambiguous question because it doesn't give all the information. It really depends on the business logic of the class and the whole application whether the accessor methods (and specially the setter) should be public or protected or even private. The field should be private.

Problema 21

Consider the following class...

```
public class ParamTest {
  public static void printSum(int a, int b){
      System.out.println("In int "+(a+b));
  }

  public static void printSum(Integer a, Integer b){
      System.out.println("In Integer "+(a+b));
  }

  public static void printSum(double a, double b){
      System.out.println("In double "+(a+b));
  }

  public static void main(String[] args) {
      printSum(1, 2);
  }
}
```

What will be printed?

- In int 3
- In Integer 3
- In double 3.0
- In double 3
- It will not compile.

Explanation:
// se va con los primitivos

The call to printSum(1, 2) will be bound to printSum(int, int) because 1 and 2 are ints, which are exact match to int, int.

Note that if printSum(int, int) method were not there in the code, printSum(double, double) would have been invoked instead of printSum(Integer, Integer) because widening is preferred over boxing/unboxing.

Problema 22

Which lines contain a valid constructor in the following code?

```
public class TestClass{
   public TestClass(int a, int b) { } // 1
   public void TestClass(int a) { }   // 2
   public TestClass(String s); // 3
   private TestClass(String s, int a) { }     //4
   public TestClass(String s1, String s2) { }; //5
}
```

You had to select 3 options

- Line // 1
- Line // 2
  - Constructors cannot return anything. Not even void.
- Line // 3
  - Constructors cannot have empty bodies (i.e. they cannot be abstract)
- Line // 4
  - You may apply public, private, and protected modifiers to a constructor. But not static, final, synchronized, native and abstract. It can also be package private i.e. without any access modifier.
- Line // 5
  - The compiler ignores the extra semi-colon.

Problema 23

What will the following program print when run?

```
public class ChangeTest {
    private int myValue = 0;

    public void showOne(int myValue){
        myValue = myValue;
    }

    public void showTwo(int myValue){
        this.myValue = myValue;
    }
    public static void main(String[] args) {
        ChangeTest ct = new ChangeTest();
        ct.showTwo(200);
        System.out.println(ct.myValue);
        ct.showOne(100);
        System.out.println(ct.myValue);
    }
}
```

You had to select 1 option

- 0 followed by 100.
- 100 followed by 100.
- 0 followed by 200.
- 100 followed by 200.
- 200 followed by 200.
- 200 followed by 100

Explanation

// Las cosas claras, asigna al local, no al bueno.

There are a couple of important concepts in this question:

1. Within an instance method, you can access the current object of the same class using 'this'. Therefore, when you access this.myValue, you are accessing the instance member myValue of the ChangeTest instance.

2. If you declare a local variable (or a method parameter) with the same name as the instance field name, the local variable "shadows" the member field. Ideally, you should be able to access the member field in the method directly by using the name of the member (in this example, myValue). However, because of shadowing, when you use myValue, it refers to the local variable instead of the instance field.

In showTwo method, there are two variables accessible with the same name myValue. One is the method parameter and another is the member field of ChangeTest object. Ideally, you should be able to access the member field in the method directly by using the name myValue but in this case, the method parameter shadows the member field because it has the same name. So by doing this.myValue, you are changing the instance variable myValue by assigning it the value contained in local variable myValue, which is 200. So in the next line when you print ct.myValue, it prints 200.

Now, in the showOne method also, there are two variables accessible with the same name myValue. One is the method parameter and another is the member field of ChangeTest object. So when you use myValue, you are actually using the method parameter instead of the member field.

Therefore, when you do : myValue = myValue; you are actually assigning the value contained in method parameter myValue to itself. You are not changing the member field myValue. Hence, when you do System.out.println(ct.myValue); in the next line, it still prints 200.

Problema 24

Consider the following code:

```
public class MyClass {

  protected int value = 10;

}
```

Which of the following statements are correct regarding the field value?

You had to select 1 option

- It cannot be accessed from any other class.
- It can be read but cannot be modified from any other class.
- It can be modified but only from a subclass of MyClass.
    - It can also be modified from any class defined in the same package.
- It can be read and modified from any class within the same package.
    - Note that since value is protected, a class in another package which extends MyClass will only inherit this variable, but it cannot read or modify the value of a variable of a MyClass instance. For example:

```
//in different package
class X extends MyClass{
   public static void main(String[] args){
      int a = new MyClass().value; //This will not compile
//because X does not own MyClass's value.
       a = new X().value; //This will compile fine because X inherits value.
   }
}
```

Problema 25

Consider the following code appearing in the same file:

```
class Data {
    int x = 0, y = 0;
    public Data(int x, int y){
        this.x = x; this.y = y;
    }
}
public class TestClass {
    public static void main(String[] args) throws Exception {
        Data d = new Data(1, 1);
        //add code here
    }
}
```

Which of the following options when applied individually will change the Data object currently referred to by the variable d to contain 2, 2 as values for its data fields?

You had to select 2 options

- Add the following two statements:
  d.x = 2;
  d.y = 2;

- Add the following statement:
  d = new Data(2, 2);
  - This will create a new Data object and will not change the original Data object referred to be d.

- Add the following two statements:
  d.x += 1;
  d.y += 1;

- Add the following statement:
  d = d + 1;
  - This will not compile because Java does not allow operator overloading for user defined classes.

Problema 26

What will the following program print?

```
public class TestClass{
  static int someInt = 10;
  public static void changeIt(int a){
    a = 20;
  }
  public static void main(String[] args){
    changeIt(someInt);
    System.out.println(someInt);
  }
}
```

You had to select 1 option

- 10
- 20
- It will not compile.
- It will throw an exception at runtime.
- None of the above.

Explanation:
Theoretically, java supports Pass by Value for everything (i.e. primitives as well as Objects). Remember that:
1. Primitives are always passed by value.
2. Object "references" are passed by value. So it looks like the object is passed by reference but actually it is the value of the reference that is passed.

An example:

```
Object o1 = new Object();
```

Let us say, this object is stored at memory location 15000. Since o1 actually stores just the address of the memory location where the object is stored, it contains 15000.

Now, when you call someMethod(o1); the value 15000 is passed to the method. Therefore, this is what happens inside the method someMethod():

```
someMethod( Object localVar) {
```

localVar now contains 15000, which means localVar also points to the same memory location where the object is stored. Therefore, when you call a method on localVar, it will be executed on the same object.

However, when you change the value of localVar itself, for example, if you do localVar=null, then localVar starts pointing to a different memory location. But the original variable o1 still contains 15000 so it still points to the same object.
}

Problema 27

What will the following code print when compiled and run?

```
class X{
    public X(){
        System.out.println("In X");
    }
}
class Y extends X{
    public Y(){
        super();
        System.out.println("In Y");
    }
}
class Z extends Y{
    public Z(){
        System.out.println("In Z");
    }
}
public class Test {
    public static void main(String[] args) {
        Y y = new Z();
    }
}
```

You had to select 1 option

It will not compile.	In X	In Z	In Y	In Z
	In Y	In Y	In X	In X
	In Z	In X	In Z	In Y

There is no problem with the code. // Remember that before the fields of a subclass can be initialized by a constructor, the fields of superclass have to be initialized. Therefore, a super class constructor must first execute before a subclass constructor can execute. This order of invocation of constructors goes up the chain up from the subclass that is being created up to the top most super class, which is java.lang.Object. // The explicit call to super(); in class Y is not required because the compiler puts a call to super(); anyway if you don't explicitly call either any super class constructor using super(...) or another constructor of the same class using this(...) first ("..." refers to appropriate arguments as required for a given constructor). // The declared type of a variable is immaterial here. Only the actual class of object that is being instantiated is important. Therefore, if you instantiate class Z, Z's constructor will be invoked, but internally, that constructor will first invoke Y's constructor before executing the rest of Z's constructor. Similarly, Y's constructor will first invoke X's constructor before executing the rest of Y's constructor. Finally, X's constructor will first invoke Objects's constructor before executing the rest of X's constructor. Object class's constructor doesn't print anything so no output is generated because of that. But once that is finished, the remaining code of X constructor's is executed, which prints "In X", then the control goes back to Y's constructor, which prints, "In Y". Finally, the control goes back to Z's constructor, which prints, "In Z".

Problema 28

Consider the following class:

```
public class Test{
    public int id;
}
```

Which of the following is the correct way to make the variable 'id' read only for any other class?
You had to select 1 option

- Make 'id' private.
    - This will not allow others to read or write.
- Make 'id' private and provide a public method getId() which will return its value.
- Make 'id' static and provide a public static method getId() which will return its value.
- Make id 'protected'

Explanation:
This is a standard way of providing read only access to internal variables.

Problema 29

When a class, whose members should be accessible only to members of that class, is coded such a way that its members are accessible to other classes as well, this is called …
You had to select 1 option

- strong coupling
- weak coupling
- strong typing
- weak encapsulation
- weak polymorphism
- high cohesion
- low cohesion

Explanation:
When a class is properly encapsulated, only the members that are part of its public API are publicly accessible to other classes. Rest is all private.

Problema 30

What will the following program print?

```
public class InitTest{
   public InitTest(){
      s1 = sM1("1");
   }
   static String s1 = sM1("a");
   String s3 = sM1("2");{
      s1 = sM1("3");
   }
   static{
      s1 = sM1("b");
   }
   static String s2 = sM1("c");
   String s4 = sM1("4");
    public static void main(String args[]){
        InitTest it = new InitTest();
    }
    private static String sM1(String s){
       System.out.println(s);  return s;
    }
}
```

You had to select 1 option

- The program will not compile.
- It will print : a b c 2 3 4 1
- It will print : 2 3 4 1 a b c
- It will print : 1 a 2 3 b c 4
- It will print : 1 a b c 2 3 4

Explanation
First, static statements/blocks are called IN THE ORDER they are defined. Next, instance initializer statements/blocks are called IN THE ORDER they are defined. Finally, the constructor is called. So, it prints a b c 2 3 4 1.

Problema 31

What will the following class print when compiled and run?

```
class Holder{
   int value = 1;
   Holder link;
   public Holder(int val){ this.value = val; }
   public static void main(String[] args){
      final Holder a = new Holder(5);
      Holder b = new Holder(10);
      a.link = b;
      b.link = setIt(a, b);
      System.out.println(a.link.value+" "+b.link.value);
   }

   public static Holder setIt(final Holder x, final Holder y){
       x.link = y.link;
       return x;
   }

}
```

You had to select 1 option

- It will not compile because 'a' is final.
    - 'a' is final is true, but that only means that a will keep pointing to the same object for the entire life of the program. The object's internal fields, however, can change.
- It will not compile because method setIt() cannot change x.link.
    - Since x and y are final, the method cannot change x and y to point to some other object but it can change the objects' internal fields.
- It will print 5, 10.
- It will print 10, 10.
- It will throw an exception when run.
    - When method setIt() executes, x.link = y.link, x.link becomes null because y.link is null so a.link.value throws NullPointerException.

Problema 32

What will be the result of attempting to compile and run the following class?

```
public class InitTest{
   static String s1 = sM1("a");{
      s1 = sM1("b");
   }
   static{
      s1 = sM1("c");
   }
   public static void main(String args[]){
      InitTest it = new InitTest();
   }
   private static String sM1(String s){
      System.out.println(s);  return s;
   }
}
```

You had to select 1 option

- The program will fail to compile.
- The program will compile without error and will print a, c and b in that order when run.
- The program will compile without error and will print a, b and c in that order when run.
- The program will compile without error and will print c, a and b in that order when run.
- The program will compile without error and will print b, c and a in that order when run.

Explanation:
First, static statements/blocks are called IN THE ORDER they are defined. (Hence, a and c will be printed.) Next, instance initializer statements/blocks are called IN THE ORDER they are defined. Finally, the constructor is called. So, then it prints b.

Problema 33

How can you declare 'i' so that it is not visible outside the package test.

```
package test;
public class Test{
   XXX int i;
   /*  irrelevant code */
}
```

You had to select 2 options

- private
    - Note that the question does not require that 'i' should be accessible from test package. So private is fine.
- public
    - Marking it public will make it accessible from all classes in all packages.
- protected
    - It will make it available to a subclass even if the subclass is in a different package.
- No access modifier
- friend
    - There is no such modifier in Java.

Problema 34

What will be the output when the following program is run?

```
public class TestClass{
    char c;
    public void m1(){
        char[ ] cA = { 'a' , 'b'};
        m2(c, cA);
        System.out.println( ( (int)c)  + "," + cA[1] );
    }
    public void m2(char c, char[ ] cA){
        c = 'b';
        cA[1] = cA[0] = 'm';
    }
    public static void main(String args[]){
        new TestClass().m1();
    }
}
```

You had to select 1 option

- Compile time error.
    - c is an instance variable of numeric type so it will be given a default value of 0, which prints as empty space.
- ,m
    - Without the cast to int, c would be printed as empty space and cA[1] is 'm'
- 0,m
    - Because of the explicit cast to int in the println() call, c will be printed as 0.
- b,b
- b,m

Explanation:

Note that Arrays are Objects (i.e. cA instanceof Object is true) so are effectively passed by reference. So in m1() the change in cA[1] done by m2() is reflected everywhere the array is used.

c is a primitive type and is passed by value. In method m2() the passed parameter c is different from the instance variable 'c' because local variables (and method parameters) shadow instance variables with same name. So instance member 'c' keeps its default (i.e. 0) value.

Problema 35

Consider the following classes...

```
class Teacher{
      void teach(String student){
          /* lots of code */
      }
}
class Prof extends Teacher{
        //1
}
```

Which of the following methods can be inserted at line //1 ?
You had to select 4 options

- public void teach() throws Exception
    - It overloads the teach() method instead of overriding it.
- private void teach(int i) throws Exception
    - It overloads the teach() method instead of overriding it.
- protected void teach(String s)
    - This overrides Teacher's teach method. The overriding method can have more visibility. (It cannot have less. Here, it cannot be private.)
- public final void teach(String s)
    - Overriding method may be made final.
- public abstract void teach(String s)
    - This is wrong because class Prof has not been declared as abstract. Note that otherwise it is OK to override a method by an abstract method.

Explanation:
Note that 'protected' is less restrictive than default 'no modifier'. So choice 3 is valid.
"public abstract void teach(String s)" would have been valid if class Prof had been declared abstract.

Problema 36

What will the following code print when compiled and run?

```
interface Pow{
    static void wow(){
        System.out.println("In Pow.wow");
    }
}

abstract class Wow{

   static void wow(){
       System.out.println("In Wow.wow");
  }
}

public class Powwow extends Wow implements Pow {
    public static void main(String[] args) {
        Powwow f = new Powwow();
        f.wow();
    }
}
```

You had to select 1 option

- In Wow.wow
    - Observe that f is reference variable of type Powwow. Since Powwow extends Wow, Powwow inherits the static method wow() from Wow. Java allows a static method of a class to be invoked using a reference variable and so f.wow() invoke's Wow's wow().
    - Static methods of a interface are not inherited in the same way by an implementing class. Therefore, the static method wow() defined in the interface Pow, cannot be accessed through a reference variable. It can only be accessed using the name of the interface i.e. using Pow.wow().
- In Pow.wow
- It will print In Pow.wow if f.wow() is replaced with ((Pow)f).wow();
    - It will not compile if you make this change because it is illegal to invoke a static method of an interface using a reference variable. If you want to invoke a static method of an interface, you need to use the name of the interface. For example, Pow.wow();
    - You can use a reference variable to invoke a static method of a class or super class though. So this is valid: ((Wow)f).wow(); even though it is not a good practice to use a reference to invoke a static method.
- It will not compile.

Problema 37

What will the following code print when run?

```
class A{
    String value = "test";
    A(String val){
        this.value = val;
    }
}
public class TestClass {
    public static void main(String[] args) throws Exception {
        new A("new test").print();
    }
}
```

You had to select 1 option

- test
- new test
- It will not compile.
    - There is no method named print() defined in class A. Further, there is no such method in class Object either. To print the contents of an object you can use toString() method that returns a String: System.out.println(a.toString());
    - However, for this to print a meaningful value, class A should override the Object class's toString() method to return a meaningful String.
- It will throw an exception at run time.

Problema 38

Consider the following code appearing in the same file:

```
class Data {
    private int x = 0, y = 0;
    public Data(int x, int y){
        this.x = x; this.y = y;
    }
}
public class TestClass {
    public static void main(String[] args) throws Exception {
        Data d = new Data(1, 1);
        //add code here
    }
}
```

Which of the following options when applied individually will change the Data object currently referred to by the variable d to contain 2, 2 as values for its data fields?

You had to select 1 option

- Add the following two statements :
  d.x = 2;
  d.y = 2;
  - Note that x and y are private in class Data. Therefore, you cannot access these members from any other class.

- Add the following statement:
  d = new Data(2, 2);
  - This will create a new Data object and will not change the original Data object referred to be d.

- Add the following two statements:
  d.x += 1;
  d.y += 1;
  - Note that x and y are private in class Data. Therefore, you cannot access these members from any other class.

- Add the following method to Data class:
  public void setValues(int x, int y){
  this.x.setInt(x); this.y.setInt(y);
  }
  Then add the following statement:
  d.setValues(2, 2);
  - x is primitive int. You cannot call any methods on a primitive. so this.x.setInt(...) or this.y.setInt(...) don't make any sense.

- Add the following method to Data class:
  public void setValues(int x, int y){
  this.x = x; this.y = y;
  }
  Then add the following statement:
  d.setValues(2, 2);
  - This is a good example of encapsulation where the data members of Data class are private and there is a method in Data class to manipulate its data. Compare this approach to making x and y as public and letting other classes directly modify the values.