

SESIÓN 1:

Working with primitives data types

Instructora: Almendra Paz Rodríguez

Correo:
javaocp@mitocodenetwork.com

www.mitocode.com





BIENVENIDOS!

COMENZAREMOS EN 2 MINUTOS

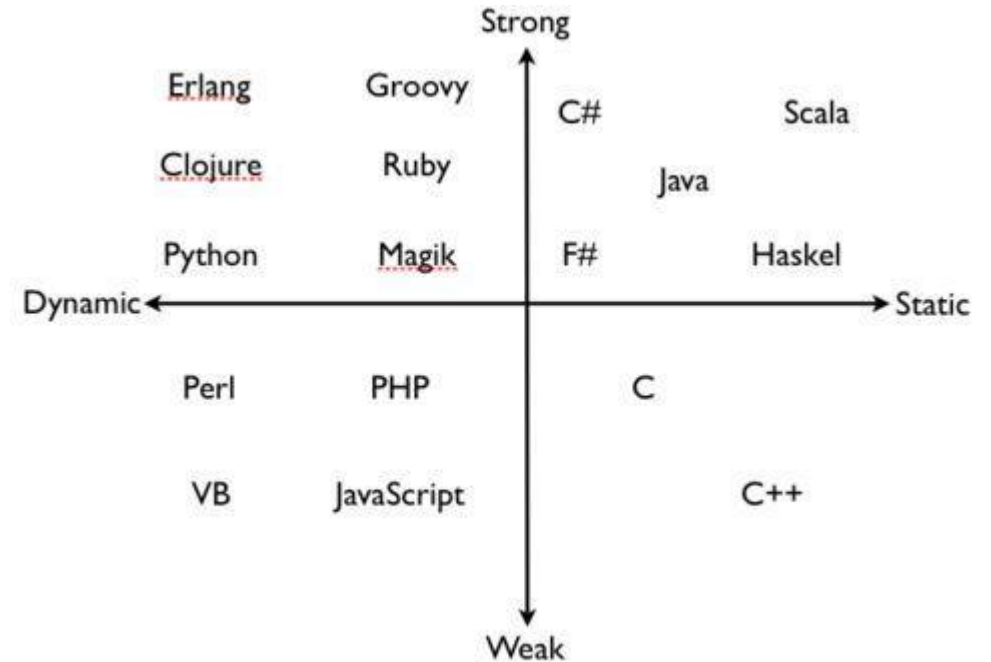


INTRODUCCIÓN

- ❑ Java es un lenguaje fuertemente tipado
- ❑ Ventajas:
 - ❑ disminuye errores porque el compilador puede detectarlo
 - ❑ Código fácil de entender
- ❑ Desventaja: es más verboso, curva de aprendizaje más elevada

```
int a = 1;  
String b = "hola";  
bool c = true;
```

```
let a = 1  
let b = "hola"  
let c = true
```



OBJETIVOS PARA EL EXAMEN

- **Handling date, time, text, numeric and boolean values**
 - **Use primitives and wrapper** classes including Math API, **parentheses, type promotion, and casting to evaluate arithmetic and boolean expressions**
 - Manipulate text, including text blocks, using String and StringBuilder classes
 - Manipulate date, time, duration, period, instant and time-zone objects using Date-Time API

AGENDA DE LA SESIÓN

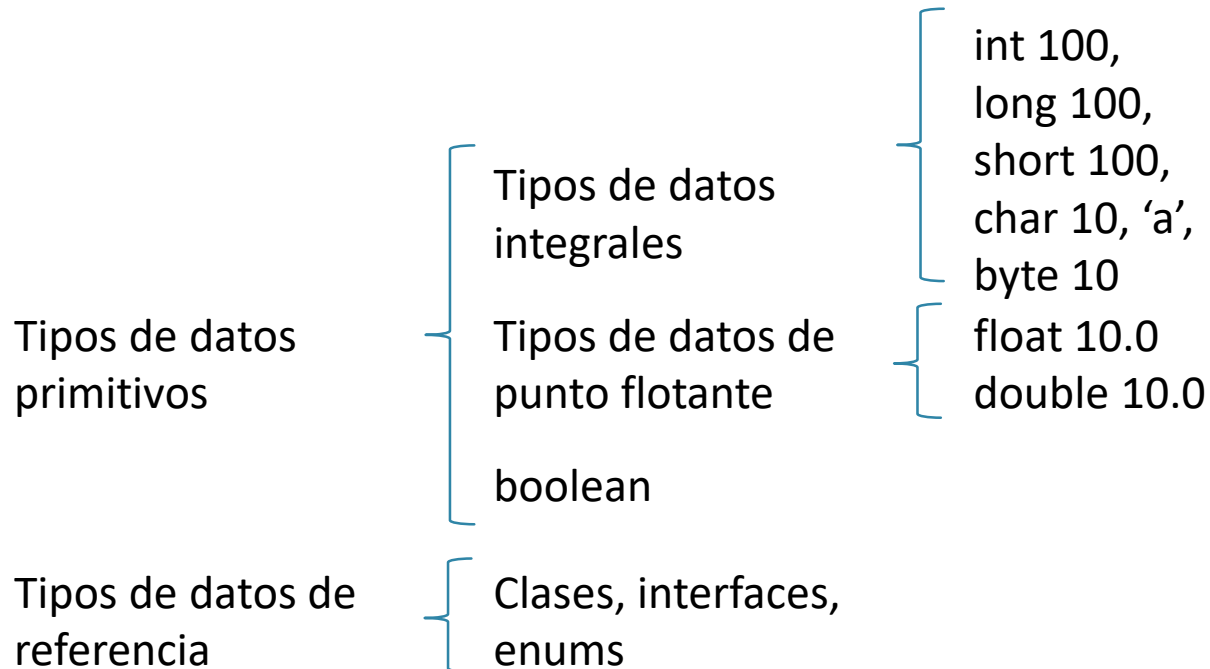
- ❑ Tipos de dato en Java
 - ❑ Diferencia entre variables de referencia y primitivas
- ❑ Declarar e inicializar variables
 - ❑ Declarar e inicializar variables
 - ❑ Variables no inicializadas y valores por default
 - ❑ Asignar valores variables
 - ❑ Variables finales
- ❑ Wrapper Classes
- ❑ Var

TIPOS DE DATOS

TIPOS DE DATO EN JAVA



■ Datos primitivos y datos de referencia



- Los tipos de datos son importantes, pues de esa forma indicamos al compilador los tipos de datos con los que queremos trabajar.
- Si trabajamos con tipos de datos primitivos, Java (compilador y JVM) se encargará de separar el espacio necesario y sabrá las operaciones permitidas.
- Los datos de referencia son estructuras diseñadas especialmente de las cuales Java no tendrá mayor información.

TIPOS DE DATO EN JAVA



□ Datos primitivos y datos de referencia

Tipos de datos
primitivos

Tipos de datos
integrales

Tipos de datos de
punto flotante

boolean

Tipos de datos de
referencia

Clases, interfaces,
enums

int 100,
long 100,
short 100,
char 10, 'a',
byte 10
float 10.0
double 10.0

Data Type	Size (in bits)	Range of values	Sample Values	Operations supported
byte	8	-2^7 to 2^7-1 , i.e., -128 to 127	-1, 0, 1	mathematical, bitwise
char	16	0 to $2^{16}-1$, i.e., 0 to 65,535	0, 1, 2, 'a', "u0061"	mathematical, bitwise
short	16	-2^{15} to $2^{15}-1$, i.e., -32,768 to 32,767	-1, 2, 3	Mathematical, bitwise
int	32	-2^{31} to $2^{31}-1$	-1, 2, 3	mathematical, bitwise
long	64	-2^{63} to $2^{63}-1$	-1, 2, 3	mathematical, bitwise
float	32	approximately $\pm 3.40282347E+38F$	1.1f, 2.0f	mathematical
double	64	approximately $\pm 1.79769313486231570E+308$	1.1, 2.0	mathematical
boolean	1	true or false	true, false	logical

TIPOS DE VARIABLES

TIPOS DE DATO EN JAVA



- Una *variable* es el nombre de una pieza de memoria que almacena datos.
- Las variables primitivas almacenan datos primitivos
- Las variables de referencia almacenan datos de referencia.

```
int i;  
// i es una variable primitiva del tipo de dato int  
  
String str;  
// str es una variable de referencia del tipo de dato  
java.lang.String
```

- Diferencias:
 - Las variables primitivas almacenan el valor del dato primitivo.
 - Las variables de referencia almacenan la dirección de memoria donde se encuentran almacenados los datos.
 - Ambos almacenan valores y en la asignación, asignan sus valores. La JVM interpreta el valor.

Orden tamaño:

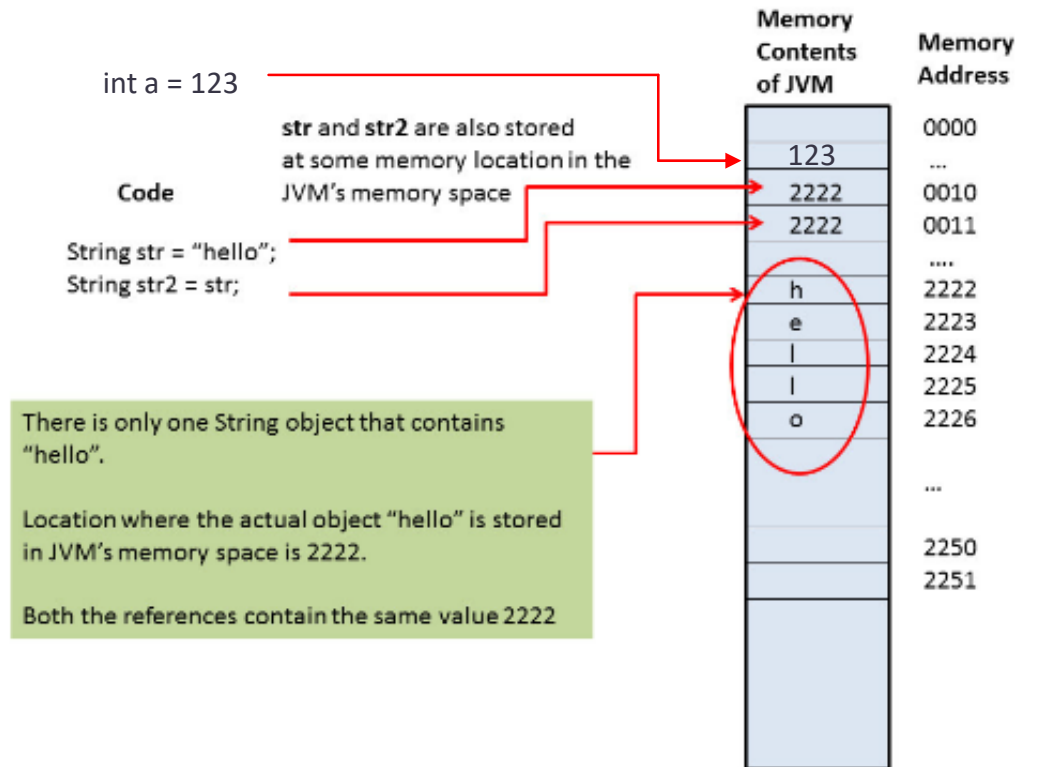
byte → short → int → long float → double
char → int

ASIGNACIÓN EN MEMORIA

TIPOS DE DATO EN JAVA



```
public class VariableTypes{  
    public static void main(String[] args){  
        int a = 123;  
        String str = "hello";  
        String str2 = str;  
        System.out.println(a+" "+str+" "+str2);  
    }  
}
```



DECLARAR E INICIALIZAR VARIABLES



```
int a;  
String nombre;  
int b, c, d;  
String s1, s2, s3;  
int e = 10; // con literal  
int f=1, g=2, h=3;  
int i = h; // con otra variable  
String str1="123";  
String str2=null;  
int j, k=10, l = e;  
String str3 = "admin", str4;
```

Declaración
sin inicialización

Declaración
+ inicialización

Combinados

- Declaración sin inicialización: solo es válido un tipo de dato por statement.
- Declaración e inicialización al mismo tiempo
- Mezclando declaraciones

¿Compilan correctamente?

```
int n = 20; int m = n = 10;  
int a = 10, int b = 3;  
int c = 10, String str2 = "hi";  
int x = y = 10;
```



- Los nombres de las variables siguen las reglas de los java identifiers.

DECLARAR E INICIALIZAR VARIABLES



```
public class Inicializacion{

    int i;
    double sueldo;
    String name;
    boolean isPremium;

    public static void main(String args[]){
        Inicializacion nueva = new Inicializacion();
        int localVariable;

        System.out.println("int i:" + nueva.i);
        System.out.println("double sueldo:" + nueva.sueldo);
        System.out.println("String name:" + nueva.name);
        System.out.println("boolean isPremium:" + nueva.isPremium);
    }
}
```

- Variables no inicializadas y por defecto
 - Variables de clase e instancia se inicializan con 0 o 0.0 en el caso de valores primitivos numéricos, false para booleanos y null para referencias.
 - Las variables locales de método o bloque no se inicializan por defecto
 - Java no inicializa las variables locales
 - Arrojará error de compilación en caso quiera usarse sin haberse inicializado.

Si agregamos al método main la siguiente línea:
System.out.println(localVariable);

¿compila correctamente? *No compila porque la variable "localVariable" no se ha inicializado.*

DECLARAR E INICIALIZAR VARIABLES



```
public class Inicializacion{  
    int i;  
    double sueldo;  
    String name;  
    boolean isPremium;  
  
    public static void main(String args[]){  
        int localVariable;  
        localVariable = 20;  
        System.out.println("localVar:" + localVariable);  
    }  
}
```

- Para el caso de variables locales el compilador nos previene de errores. Realiza análisis de rutas para evitar que se “use” una variables sin inicializar.


Si hacemos la siguiente modificaciones, compila correctamente?

```
int localVariable, x = 10;  
if(x == 10){  
    localVariable = 20;  
}  
System.out.println("localVar:" + localVariable);
```

- El compilador hace evaluación solo con constantes en tiempo de compilación (variables final)

ASIGNANDO VALORES A VARIABLES



- **Literales:** valores fijos que no van a cambiar:
 - Ejemplo: true, false, 10, 'a'
- Reglas acerca de literales:
 - Se pueden usar guiones bajos en literales numéricos:
 - 100_000.00
 - 100_000_._00_  No usar el _ en las siguientes posiciones
 - Por defecto los literales numéricos son int. Si tienen decimales serán double.
 - Se pueden escribir literales long y float añadiendo una L o F (mayúscula o minúscula). Ejem: **123L**, **12.0f**
- null también es un literal que sirve para que la variable de referencia no apunte a ningún objeto.
- Java también permite escribir número en formatos hexadecimal, octal y binario.
 - Hexadecimal: 0xF (15 en base decimal)
 - Octal: 017 (15 en base decimal)
 - Binario: 0B10 0b10
- **Asignar usando otra variable:** int j = x;
- **Asignar usando el retorno de un método**
 - Ejem: int score = evaluate();

DE DIFERENTES TIPOS ASIGNANDO VALORES A VARIABLES



- 1. Cuando el tipo de dato destino es mayor que tipo de dato origen. (Conversión implícita por ensanchamiento)



Orden tamaño:

byte → short → int → long float → double
char → int

- 2. Cuando el destino tiene menor tamaño que el origen



- Si el valor es una constante, puede “caber” en el dato más pequeño. Solo para byte, char, short o int (estrechamiento implícito)

```
byte b = 10; // 10 es un int pero “cabe” en un byte
char c1;
final char c2 = 10; final int a2 = 20;
b = c2;
c1 = a2;
```

DE DIFERENTES TIPOS ASIGNANDO VALORES A VARIABLES



```
//Ensanchamiento
byte b1= 10; char c1; int a1 = 20; long l1= 30;double d1;
d1 = l1;
l1 = a1;
a1 = b1;
System.out.println(d1+" "+l1+" "+a1);

//Estrechamiento implícito
final char c2= 10; final int a2 = 20;
b1 = c2;
c1 = a2;
System.out.println(b1+" "+c1);
```

- Cuando no es una constante, usamos “cast” para prometer que el dato entrará en el destino.
- No se pueden castear booleanos.
- Y si no se cumple la promesa? Solo se asignará el valor que pueda entrar.

```
//Estrechamiento explícito
byte b3= 10; int a3 = 20; long l3= 30; double d3 = 40.00;
b3 = (byte) a3;
a3 = (int) l3;
l3 = (long) d3;
System.out.println(b3+" "+a3+" "+l3);
```

DE DIFERENTES TIPOS

ASIGNANDO VALORES A VARIABLES



- **Asignaciones entre byte, short → char**
- Para asignaciones entre (byte y short) hacia char, y viceversa, siempre es necesario usar cast, a menos que usemos constantes.
- **Asignaciones de int y long → float y double**
- Hay un ensanchamiento implícito de int hacia float; y de long hacia double.
- **Asignaciones de float y double → int y long**
- Se requiere un cast cuando se asigna de float hacia int; y de double hacia long.

Orden tamaño:

byte → short → int → long float → double
char → int

EJERCICIO 1

Given:

```
char a = 'a', b = 98; //1
```

```
int a1 = a; //2
```

```
int b1 = (int) b; //3
```

```
System.out.println((char)a1+(char)b1); //4
```

- What is the output?
 - Compilation error at //1
 - Compilation error at //2
 - Compilation error at //3
 - Compilation error at //4
 - ab
 - 195



EJERCICIO 2

What will be the result of compiling and running the following code:

```
float foo = 2, bar = 3, baz = 4; //1
float mod1 = foo % baz, mod2 = baz % foo; //2
float val = mod1 > mod2 ? bar : baz; //3
System.out.println(val);
```

- What is the output?
 - Compilation error at //1
 - Compilation error at //2
 - Compilation error at //3
 - 3
 - 3.0 ✓
 - 3.0f
 - 4
 - 4.0
 - 4.0f

WRAPPER CLASES



- Cuando definimos métodos para recibir objetos, no se permite recibir datos primitivos y viceversa.

```
public void multipleTypes(Object o){  
    System.out.println("");  
}
```

- ArrayList por ejemplo, solo recibe objetos.
 - ¿Cómo le agregamos datos primitivos?
- Pensando en este problema, los diseñadores de Java crearon los wrapper clases, las cuales se encuentran en el paquete java.lang y son clases inmutables.

- Byte, Short, Integer, Float, Double, Long
 - Extienden de Number
- Character, Number, Boolean
 - Extienden de Object
- Tenemos 3 formas de crear los wrapper clases
 - Usando el constructor: siempre crea un nuevo objeto.
 - Usando el método valueOf: es más eficiente, **puede** retornar un cached object. (número de -128 a 127, true y false, '\u0000' al '\u007f')
 - A través de auto-boxing (a partir de Java 5)

CREACIÓN MEDIANTE MÉTODO VALUEOF WRAPPER CLASES



- Todos los wrapper clases tienen dos firmas estáticas del método `valueOf()`, uno recibe el dato primitivo y el otro un `String`. La excepción es la clase `Character` que no tiene una firma del método para recibir un `String`.
- Si pasamos como parámetro un `null` o un `String` que no se pueda convertir, encontraremos un `NumberFormatException`.
- En el caso de `Boolean` arroja `false` si se pasa un valor diferente a `true`.

Primitive type	Wrapper class	Example of creating
<code>boolean</code>	<code>Boolean</code>	<code>Boolean.valueOf(true)</code>
<code>byte</code>	<code>Byte</code>	<code>Byte.valueOf((byte) 1)</code>
<code>short</code>	<code>Short</code>	<code>Short.valueOf((short) 1)</code>
<code>int</code>	<code>Integer</code>	<code>Integer.valueOf(1)</code>
<code>long</code>	<code>Long</code>	<code>Long.valueOf(1)</code>
<code>float</code>	<code>Float</code>	<code>Float.valueOf((float) 1.0)</code>
<code>double</code>	<code>Double</code>	<code>Double.valueOf(1.0)</code>
<code>char</code>	<code>Character</code>	<code>Character.valueOf('c')</code>

UTILIZANDO CONVERSIÓN DESDE UN STRING WRAPPER CLASES



- Para obtener los primitivos a partir de un Wrapper tenemos dos formas:
 - Utilizando el método xxxValue(). Ejemplo: intValue(), booleanValue(), charValue().
 - Utilizando el unboxing
- Podemos obtener primitivos a partir de un String utilizando el método parseXX(). De acuerdo al argumento que pasemos, podemos obtener un NumberFormatException.

Wrapper class	Converting String to a primitive	Converting String to a wrapper class
Boolean	<code>Boolean.parseBoolean("true")</code>	<code>Boolean.valueOf("TRUE")</code>
Byte	<code>Byte.parseByte("1")</code>	<code>Byte.valueOf("2")</code>
Short	<code>Short.parseShort("1")</code>	<code>Short.valueOf("2")</code>
Integer	<code>Integer.parseInt("1")</code>	<code>Integer.valueOf("2")</code>
Long	<code>Long.parseLong("1")</code>	<code>Long.valueOf("2")</code>
Float	<code>Float.parseFloat("1")</code>	<code>Float.valueOf("2.2")</code>
Double	<code>Double.parseDouble("1")</code>	<code>Double.valueOf("2.2")</code>
Character	None	None

AUTOBOXING Y UNBOXING

WRAPPER CLASES



- A partir de Java 5 tenemos procesos automática de auto-boxing y unboxing.

```
//auto-boxing
//estas asignaciones son equivalente
Integer i = Integer.valueOf(100);
Integer i = 100;
// el autoboxing también utilizará cached object
// i1 e i2 apuntan al mismo objeto
Integer i1 = 100;
Integer i2 = 100;
```

```
//Unboxing
Integer i1 = 10;
int i2= i1;
//tener cuidado con que la variable destino pueda soportar
el tipo de wrapper
byte b = i1;
float f = i1;
// Utilizando parseInt
int i3 = Integer.parseInt("10",2); //devolverá 2
double d1 = Double.parseDouble("10.0"); //devolverá 10.0
```

RESUMEN WRAPPER CLASES



Método creación	Ejemplo	
Constructores	<code>Long l2 = new Long(10); Long l3 = new Long("10"); // deprecada</code> <code>Float f1 = new Float(10.2f);</code> <code>Float f2 = new Float(10.2); //Uso de Double</code>	
Método valueOf	<code>Long l4 = Long.valueOf(10);</code> <code>Long l5 = Long.valueOf(10L);</code> <code>Long l6 = Long.valueOf("10");</code>	
	<code>Long l7 = Long.valueOf("10.2");</code> <code>Long l8 = Long.valueOf(null);</code>	NumberFormatException
	<code>Boolean b1 = Boolean.valueOf(null);</code>	Siempre será false
Auto boxing	<code>Long l2 = 10; // no compila es int</code> <code>Long l2 = 10L;</code>	Usa caché

EJERCICIO 3

Given:

```
public static float parseFloat1(String s1){  
    try{  
        return Float.parseFloat(s1);  
    }catch(NumberFormatException e){  
        return 0.0f;  
    }catch(IllegalArgumentException e){  
        return Float.NaN;  
    }  
}
```

Identify correct statements.

- a) Calling `parseFloat1(""+Float.NEGATIVE_INFINITY);` will return `0.0f`.
- b) Calling `parseFloat1(""+Float.POSITIVE_INFINITY);` will return `0.0f`.
- c) Calling `parseFloat1("junk");` will return `0.0f`. ✓
- d) Calling `parseFloat1("-Infinity");` will return `NaN`.
- e) Calling `parseFloat1("NaN");` will return `0.0f`.

EJERCICIO 4

What will the following code print when run?

```
public class TestClass{
    public static Integer wiggler(Integer x){
        Integer y = x + 10;
        x++;
        System.out.println(x);
        return y;
    }

    public static void main(String[] args){
        Integer dataWrapper = new Integer(5);
        Integer value = wiggler(dataWrapper);
        System.out.println(dataWrapper+value);
    }
}
```

- 5 and 20
- 6 and 515
- 6 and 20
- 6 and 615
- It will not compile.



VAR (LOCAL VARIABLE TYPE INFERENCE)



¿Cuánto errores tiene el presente código?

```
public class VarClass{
    var i;
    static var getHighest(var students){
        int highest = 0;
        var k = "nuevo", o = "temporal";
        for(var student : students){
            if(highest < student.marks)
                highest = student.marks;
        }
        return highest;
    }
}
```

- Solo usarlo como variable local (en un método, bloque estático o constructor).
- No se puede usar como tipo de parámetro o tipo de retorno
- No se puede usar sin inicializar. La inicialización debe ser diferente de null.
- Siempre toma el tipo con que se inicializa
- Su valor puede cambiar en tiempo de ejecución pero el tipo de dato no.
- No es permitida para una declaración compuesta.

EJERCICIO 5

Identify correct statements about the above code.

- a) It will compile if the *getHighest* method declaration is changed to:

static var getHighest(ArrayList<Student> students)

- a) It will compile if the *getHighest* method declaration is changed to: *static int getHighest(var students)*
- b) It will compile if *allStudents* type is changed from var to ArrayList<Student>.
- c) It will compile if the type of *h* is changed from var to int.
- d) None of the above.



Given:

```
import java.util.*;

class Student{
    int marks;
}

public class TestClass {
    static var getHighest(var students){
        int highest = 0;
        for(var student : students){
            if(highest < student.marks) highest = student.marks;
        }
        return highest;
    }

    public static void main(String[] args) {
        var student = new Student();
        var allStudents = new ArrayList<Student>();
        allStudents.add(student);
        var h = getHighest(allStudents);
        System.out.println(h);
    }
}
```

EJERCICIO 6

Which of the following can be valid declarations of an integer variable? Select 3 options

a) `private var x = 10`

b) `final int x = 10`



c) `public Int x = 10`

d) `int x = 10`




e) `static int x = 10`



f) `global int x = 10`

EJERCICIO 7

```
public class TestClass {  
    public void myMethod(String... params) {  
        var a = params;  
        var b = params[0];  
    }  
}
```

- What are the types of the variables a and b?
 - a) Object[] and Object
 - b) String[] and Object
 - c) String[] and String 
 - d) List and Object
 - e) List<String> and String

ACTIVIDAD 1

- Indicaciones
 - Desarrolle el ejercicio dentro de un método main
 - Declara en una sola línea las variables decimales: sueldoNeto, sueldoBruto. Utilice guiones como separador de miles.
 - Declare e inicialice en una sola línea las variables enteras con sus datos: edad, numHijos
 - Declare e inicialice en una sola línea las variables de referencia con sus datos: nombre, apellidoPaterno, apellidoMaterno
- En caso su edad sea mayor de 0 años, imprima su edad en pantalla.
- Cree una variable de tipo byte y asígnele su número de hijos.
- Cree una variable de tipo float y asígnele la resta de sus sueldos
- Cree una nueva variable entera y asigne a ésta la suma de sus sueldos

ACTIVIDAD 2

- Indicaciones
 - Declara 3 variables de tipo var.
 - Inicialice las variables con su nombre, apellido materno y edad.
 - Convierta su variable edad a un wrapper mediante autoboxing
 - Cree una variable de tipo Float con su sueldo.
 - Cree una variable de tipo Double con su edad. Utilice valueOf.
 - Intente construir un Boolean con su nombre.



GRACIAS

ENLACE PARA FEEDBACK

