Tema 2: Lección 1

Ejercicio 1: Responde verdadero (V) o falso (F)

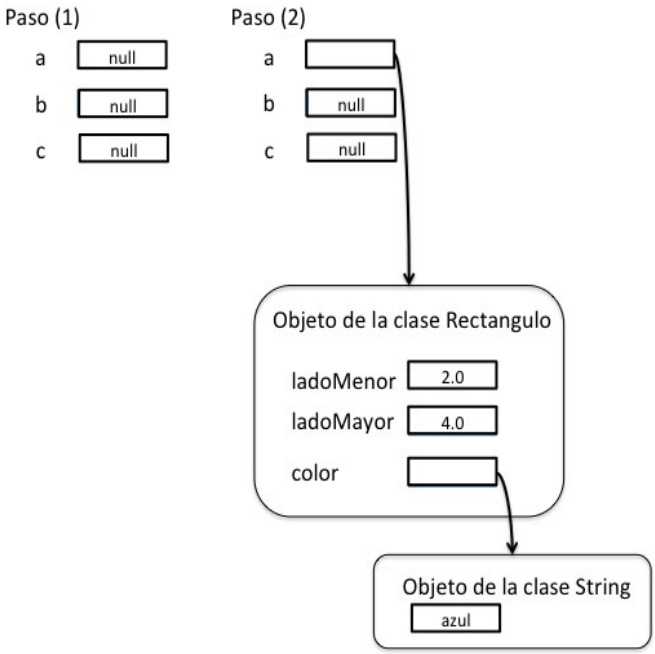
Dos objetos con el mismo estado pueden tener distinta identidad.	
Si hay mil objetos de una clase X habrá mil copias de su variable de clase x1.	
El código: **MuertoViviente vampiro;** crea en Java un objeto de la clase MuertoViviente.	
El código: **attr_writer :color** crea el consultor y el modificador del atributo color en Ruby.	
En Ruby los métodos de instancia son públicos y los atributos de instancia son privados, por defecto.	
La identidad de un objeto en programación orientada a objetos la da su dirección de memoria.	
Los constructores por defecto devuelven void.	
La encapsulación de un conjunto de elementos implica de forma implícita su ocultamiento para el resto de elementos del sistema.	
Cuando definimos una clase, definimos el estado y el comportamiento de un conjunto de objetos, y en algunas ocasiones también definimos estado y/o comportamiento de la propia clase.	
Todos los lenguajes de programación soportan los siguientes atributos de visibilidad de forma explícita para sus atributos: **private** , **package** y **public**	
Para invocar a los métodos de clase no es necesario que exista previamente una instancia de dicha clase en el sistema.	
Los paquetes son específicos de Java.	
En Java y Ruby las clases se tratan como objetos mientras que en C++ las clases constituyen un patrón.	

Ejercicio 2. Sea la clase java Rectángulo:

```
class  Rectangulo {
      float ladoMenor, ladoMayor;
      String color;
      Rectangulo(){
            ladoMenor=2;
            ladoMayor=4;
            color=”azul”;
      }
      void setColor(String unColor) {
            color = unColor;
      }
      float area(){
            return ladoMenor*ladoMayor;
      }
}
```

Representa lo que va ocurriendo con cada una de estas instrucciones. Se proporciona el resultado de los pasos (1) y (2), haz una figura para cada uno de los pasos (3), (4), (5) y (6).

```
(1)      Rectangulo a, b, c;
(2)      a = new Rectangulo();
```




```
(3)     b = new Rectángulo();
(4)     a.setColor("rojo");
(5)     float x = a.area();
(6)     c = a;
```

Ejercicio 3. Dada la clase Java:

```
public class Prueba {
      public static final int A = 1;
      static String s = "";
      private  int b= 2;
      int c;
}
```

a. ¿Cuántos atributos tiene? Indica si se trata de atributos de instancia o de clase. ¿Cuál es la visibilidad de cada uno de ellos?

b. A partir del siguiente código, suponiendo que está en otra clase del mismo paquete:

```
1. Prueba obj1 = new Prueba();
2. Prueba obj2 = new Prueba();
3. obj1.A = 3;
4. Prueba.s = "hola";
5. Prueba.A = 14;
6. obj1.b = 0;
7. obj2.b = 5;
8. obj1.c = 4;
9. obj2.c = 6;
```

b.1. ¿Se produciría algún error de compilación? ¿por qué? (Recomendación: ejecutar el código para comprobarlo).

b.2. Indica cuál es el estado de obj1 y obj2 después de su ejecución y eliminados los errores de compilación si los había.

Ejercicio 4. ¿Quién tiene la responsabilidad de responder a los mensajes que se corresponden con los métodos de clase?

Ejercicio 5. Razona si las siguientes afirmaciones son ciertas o falsas, suponiendo que estamos en la clase citada:

1. Los atributos de clase son accesibles sólo desde métodos de clase (no desde métodos de instancia)
2. Los atributos de instancia son accesibles sólo desde métodos de instancia (no desde métodos de clase).
3. La palabra reservada "this" (Java) puede emplearse tanto en métodos de clase como de instancia.

Ejercicio 6. ¿Qué mecanismos tiene Java para la ocultación de información? Y ¿Ruby?

Ejercicio 7. Atributos de instancia y de clase

Sea la siguiente clase en Java:

```
Class C {
   private static int contador2 = 0;
   private int contador1= 0;
   public void incrementarContador1() {contador1++;}
   public void incrementarContador2() {contador2++;}
   public int getContador1() {return contador1;}
   public int getContador2() {return contador2;}
}
```

Tras ejecutar el siguiente trozo de código:

```
C objeto1 = new C();
C objeto2 = new C();
objeto1.incrementaContador1();
objeto1.incrementaContador2();
objeto2.incrementaContador1();
objeto2.incrementaContador2();
int valor1 = objeto1.getContador1();
int valor2 = objeto1.getContador2();
int valor3 = objeto2.getContador1();
int valor4 = objeto2.getContador2();
```

1. ¿Qué valores tienen valor1, valor2, valor3 y valor4?

2. Traduce el código Java proporcionado a Ruby

Ejercicio 8. Variables de instancia y de instancia de la clase en Ruby

Sea el siguiente código en Ruby:

```
class A
  @variable = 50
  def variable
     puts "valor de la variable: #{@variable}"
  end
  def self.variable
     puts "valor de la variable:  #{@variable}"
  end
end
```

1. Indica cuál es el resultado de ejecutar las siguientes líneas de código	
`A.new.variable`	
`A.variable`	
2. Explica el porqué de estos resultados	

Ejercicio 9. Variables en Ruby

Sea el siguiente código en Ruby:

```
class A
  CONSTANT= 0
  $Global= 10
  @@a1=20
  @a2=7

  def initialize
    @a3=40
  end
  def ver_a1
    puts "valor del atributo a1 #{@@a1}"
  end
  def self.ver_a1
    puts "valor del atributo a1 #{@@a1}"
  end
  def ver_a2
    puts "valor del atributo a2 #{@@a2}"
  end
  def self.ver_a2
    puts "valor del atributo a2 #{@@a2}"
  end
 def ver_a3
    puts "valor del atributo a3 #{@a3}"
  end
end
```

Indica cuál es el resultado de ejecutar las siguientes líneas de código en una clase main del mismo módulo:

1. puts A::CONSTANT 2. A::CONSTANT = 2 3. puts A::CONSTANT	
4. puts "valor de la global #{$Global}" 5. $Global = 30 6. puts "valor de la global #{$Global}"	
7. A.new.ver_a1	
8. A.ver_a1	
9. A.new.ver_a2	
10. A.ver_a2	
11. A.new.ver_a3	
12. A.ver_a3	
Explica el porqué de estos resultados e indica qué tipo de variables son a1, a2 y a3	

Ejercicio 10. Paquetes y módulos

Sean los ficheros A.java y B.java cuyo contenido es:

A.java

```
package p1;
import p2.B;
class A { (...) }
```

B.java

```
package p2;
public class B {
     String atributo1;
     public String atributo2;
    (…) }
```

a) Responde Verdadero (V) o Falso (F)	
1. La clase A pertenece al paquete p1	
2. Como se ha hecho un import de la clase B, atributo1 es accesible desde A	
3. Como se ha hecho un import de la clase B, atributo2 es accesible desde A	
b) ¿Es posible traducir exactamente a Ruby el código anterior? En caso negativo, ¿cuál sería la forma más próxima de implementar una estructura similar?	

Ejercicio 11. Acceso a métodos en Ruby

Sea el siguiente código en Ruby:

```
class Ejemplo
      def publico_implicito_protegido
           metodo_protegido
      end
      def publico_explicito_protegido
           self.metodo_protegido
      end
      def publico_implicito_privado
           metodo_privado
      end
      def publico_explicito_privado
           self.metodo_privado
      end


      protected
      def metodo_protegido
           puts "metodo protegido"
      end

      private
      def metodo_privado
           puts "método privado"
      end
end
```

1. Indica cuál es el resultado de ejecutar las siguientes líneas de código	
a) Ejemplo.new.publico_implicito_protegido	
b) Ejemplo.new.publico_explicito_protegido	
c) Ejemplo.new.metodo_protegido	
d) Ejemplo.new.publico_implicito_privado	
e) Ejemplo.new.publico_explicito_privado	
f) Ejemplo.new.metodo_privado	
2. Explica el resultado obtenido en cada uno de estos puntos	

Ejercicio 12. Tipo de variables en Ruby

Sea el siguiente código en Ruby:

```
class Planeta
  NOMBRES=['Mercurio','Venus','Tierra','Marte','Júpiter',
                            'Saturno','Urano','Neptuno'].freeze
  @@planetas_instanciados = 0

  def initialize
    if @@planetas_instanciados >= NAMES.size
      raise ArgumentError, 'Lo siento, pero ya se han creados todos
                             los planetas, ahora a descansar'
    end
    @nombre = NOMBRES[@@planetas_instanciados]
    @@planetas_instanciados += 1
    puts "Se ha creado el planeta #{@nombre}!"
  end
end
```

1. Indica cuál es el resultado de ejecutar la siguiente línea de código	
1.upto (9) {Planeta.new}	
2. Intenta entender el código proporcionado y explica el resultado obtenido	
3. Traduce el código anterior a Java	

Ejercicio 13. Visibilidad de métodos en Java

Sea el siguiente código en Java:

```
public class Ejercicio13 {

    public void publico(){
        System.out.println("método público");
    }
    void paquete(){
        System.out.println("método paquete");
    }
    protected void protegido(){
        System.out.println("método protegido");
    }
```

```
    private void privado(){
        System.out.println("método privado");
    }
}
```

Prueba el siguiente código en la misma clase, en otra clase del mismo paquete y en otra clase de distinto paquete:

```
 public static void main(String[] args) {
     Ejercicio13 e13 = new Ejercicio13();
     e13.publico();
     e13.protegido();
     e13.paquete();
     e13.privado();
}
```

Plantéate qué ocurre en cada una de las situaciones indiadas.

Ejercicio 14. Visibilidad de variables en Java

Sea el siguiente código en Java:

```
public class Ejercicio14 {
    private int i=0;
    int j= 1;
    protected int k=2;
    public int l=3;
}
```

Prueba el siguiente código en la misma clase, en otra clase del mismo paquete y en otra clase de distinto paquete:

```
 public static void main(String[] args) {
      Ejercicio14 e14 = new Ejercicio14();
      System.out.println("privada : " + e14.i);
      System.out.println("paquete : " + e14.j);
      System.out.println("protected : " + e14.k);
      System.out.println("public : " + e14.l);
}
```

Plantéate qué ocurre en cada una de las situaciones indicadas.

Ejercicio 15. Ámbito de variables y métodos en Java

Sea el siguiente código en Java:

```
public class Ejercicio15 {
    private static int j = 0;
    private int k = 2;

    public static void muestra_jk(){
        System.out.println("variable clase " + j);
        System.out.println("variable instancia " + k);
    }
    public void muestra_kj(){
        System.out.println("variable instancia " + k);
        System.out.println("variable clase " + j);
    }
```

```
    public static void main(String[] args){
         Ejercicio15.muestra_jk();
         Ejercicio15.j = 10;
         Ejercicio15.muestra_jk();
         Ejercicio15.muestra_kj();

         Ejercicio15 e15 = new Ejercicio15();
         e15.muestra_kj();
         e15.muestra_jk();
    }
}
```

1. Trata de identificar los errores de compilación que hay y elimínalos

2. Ejecútalo y trata de entender qué ocurre

Ejercicio 16. Crea en Java la clase Representante con atributos privados *nombre, puesto* y *lista_clientes*, siendo los dos primeros cadenas de texto y el último una lista que contiene objetos de la clase Cliente. Crea un constructor que inicialice los atributos considerando que al crearlo el Representante no tiene clientes.

Crea un nuevo constructor para la misma clase Representante, suponiendo que recibe una lista inicial de clientes para que la inicialice.

Haz el mismo ejercicio en Ruby. Recuerda que no puede haber métodos con igual nombre. ¿Qué soluciones habría para esto?

Ejercicio 17. Sobre el ejercicio 16, supón, que por defecto, el puesto de un representante es el "base" a no ser que se indique en el constructor que es el puesto "avanzado". Escribe el código del constructor en Java y Ruby con la solución a este problema.

Ejercicio 18. Comparación de estado e identidad

Dada la siguiente clase:

```
class Comercial{
   private String nombre;
   private Empresa empresa;

   Comercial(String nombre){
      this.nombre=nombre;
      this.empresa=new Empresa("ACME"); //Constructor de Empresa con su nombre
   }

   public void setEmpresa(Empresa empresa){
      this.empresa=empresa;
   }
}
```

Y suponiendo que en otra clase se han creado:

```
Comercial c1 = new Comercial("Antonio");
Comercial c2 = new Comercial("Antonio");
Comercial c3 = c1;
Comercial c4 = c1;
```

Indica cuál sería el resultado de las siguientes comparaciones:	
`c1 == c2;`	
`c1 != c2;`	
`c1 == c3;`	
`c2 == c3;`	
`c1.equals(c2);`	
`c1.equals(c3);`	
`c2.equals(c3);`	

`Empresa Refrescos=new Empresa("Refrescos SA");`
`c1.setEmpresa(Refrescos);`
`c1 == c3;`	

`c1.equals(c3);`	
`c4=null;`	
`c1 == c4;`	

Ejercicio 19. Redefine el operador *equals* de la clase *Comercial* en el ejemplo 18 para que devuelva true si los dos comerciales que se comparan tienen el mismo estado, aunque no tengan la misma identidad.

Ejercicio 20. Implementa en Ruby el ejemplo del ejercicio 18 y redefine el método == para que se compare el estado y no la identidad de los objetos.

Ejercicio 21. Comparación de objetos en Java y el Ruby.

Sean la siguientes clases en Java:

```
public class Matricula {
    private int cursoAcademico;
    private ArrayList<Asignatura> asignaturas = new ArrayList();

    public Matricula(int curso, ArrayList<Asignatura> asig){
        this.cursoAcademico = curso;
        this.asignaturas = new ArrayList(asig);
    }
    ...
}
public class Asignatura {
    private String nombre;
    private String codigo;

    public Asignatura(String nombre, String codigo){
        this.nombre =nombre;
        this.codigo = codigo;
    }
    ...
}
```

1. Redefine el método equals() para estas dos clases teniendo en cuenta que dos objetos Asignatura son iguales cuando tienen el mismo valor del atributo codigo y que dos objetos Matricula son iguales cuando tiene iguales asignaturas y el mismo valor del atributo cursoAcademico
2. Prueba el resultado comparando objetos Asignatura y objetos Matricula

3. Realiza el mismo ejercicio en Ruby, teniendo en cuenta que en Ruby, para la comparación de igualdad de objetos, se redefine ==obj

Ejercicio 22. Las siguientes instrucciones en Java pretenden declarar Arrays. Haz pruebas en Java para entender cómo funcionan. Indica qué errores ves y escríbelas correctamente en Java.

	Motivo de error, si lo hay	Solución al error, en su caso
`int [][] c;`		
`int [5] e;`		
`int d[];`		
`int f[]=new int[3];`		
`int[] x=new int[10];`		
`Array.newInstance(int, 5);`		
`int [] dims={2,4};`		
`Array.newInstance(Alumno, dims);`		

Ejercicio 23. Define en Java y en Ruby una clase cuyas instancias representen atletas y otra clase cuyas instancias sean un equipo de atletas y un entrenador al frente. Incluye los atributos que consideres necesarios. Además, ten en cuenta que necesitamos saber cuántos equipos de atletas hay. Escribe un programa sencillo que cree un equipo y muestre:

a) Los atletas que corren en el mismo y su entrenador.
b) El número de equipos que tenemos

Ejercicio 24. Manejo de excepciones en Java y en Ruby

1. Declara en Java y Ruby una clase Empleado con atributos nombre y sueldo y un constructor que inicialice ambos empleando sus métodos modificadores.

2. Incluye el código que consideres necesario para que el método modificador del sueldo lance una excepción si la cantidad que se trata de asignar es inferior a 600€ o superior a 10.000€.

3. Escribe el código necesario en Java y Ruby para crear empleados y cambiar sus sueldos probando sueldos que cumplan las restricciones y otros que no las cumplan por ser demasiado bajos o altos. Incluye en dicho código la lógica necesaria para capturar y tratar excepciones de forma que no sucedan finales abruptos de la ejecución.