Examen Segunda Evaluación Programación 1ºK Curso 2018/2019
Nombre y Apellidos:

_____

TODO EL EXAMEN ESTÁ A DOBLE CARA. Hojas entregadas aparte: _____

1. **(1 punto)** Marcar con una X la respuesta correcta (solo hay una). Las respuestas en blanco no quitan puntuación. Las respuestas incorrectas quitan un tercio de la puntuación de las correctas.

Dada la siguiente definición:
int[][] tabla=new int[10][8];
¿Cuánto vale tabla[2].length?

	Error de null
	10
X	8
	9

Si queremos que un atributo solo sea accesible por las clases derivadas, utilizaremos

	private
X	protected
	package
	herencia múltiple con interfaces

¿Qué modificadores incluyen implícitamente los métodos de una interfaz en Java y, por tanto, no es necesario indicarlos?

	protected y final.
X	public y abstract.
	public y final.
	void y abstract.

Supongamos que disponemos de las clases Persona, Profesor y Alumno, donde las dos últimas son subclases de la primera. Indicar cuál es la afirmación correcta

	La clase Persona tiene que ser abstracta obligatoriamente
	La clase Persona es final
X	Las clases Profesor y Alumno utilizan la palabra reservada extends en su declaración.
	Las clases Profesor y Alumno utilizan la palabra reservada implements en su declaración.

Examen Segunda Evaluación Programación 1ºK Curso 2018/2019
Nombre y Apellidos:

---

2. **(0’5 puntos)** Según el código siguiente ¿Qué se visualizará en pantalla? Escribir solo lo que se ve en pantalla, sin explicaciones.

```
class ClaseA{
      ClaseA ( int x ){
            System.out.println("AA”);
      }
}
class ClaseB extends ClaseA{
      ClaseB( ){
            super(6);
            System.out.println("BB”);
      }
}
public class Ejemplo{
      public static void main(String[] args) {
            ClaseB objB1=new ClaseB();
            ClaseB objB2;
            System.out.println("EXAMEN”);
      }
}
```

```
AA
BB
EXAMEN
```

Ejercicios Prácticos. Ignorar los import. Programa quiere decir un conjunto de clases (una o más) una de ellas con un método main. Como criterios de calificación de los ejercicios, se seguirán los mismos que se han tratado en clase (la ejecución correcta de un código sólo implica un 5 en la nota.
Calidad de los comentarios (si éstos fueran necesarios). Que los nombres de Clases, atributos y métodos sean adecuados. Que los modificadores de acceso sean adecuados. La claridad del código.
El uso de objetos y clases La extensibilidad del código ante cambios en las especificaciones. Que la salida del programa sea clara, etc). Se supone que la entrada del usuario es “amigable”. Si algún aspecto no está especificado en el enunciado, queda a criterio del alumno.

Abreviaturas **admitidas**:

psvm – public static void main(String[] args)
sout(“ggg”) - System.out.println(“ggg”)
Se permite abreviar los getter y los setters indicando cuáles se implementarían

Examen Segunda Evaluación Programación 1ºK Curso 2018/2019
Nombre y Apellidos:

Se permite usar nombres de atributos y de variables cortos (sin exagerar). Solo para los atributos y los nombres de variables.

3. a **(1,5 punto)** Escribir un método al que se le pasan dos tablas cuadradas de enteros del mismo tamaño y devuelve una tabla nueva con el mayor número de cada posición

Ejemplo:
Se le pasa

```
1 2 3          6 0 0
1 1 1          6 0 0
8 6 0          6 0 0
```

Y devuelve:

```
6 2 3
6 1 1
8 6 0
```

```
private static int[][] mayorIguales(int[][] a, int[][] b) {
      int[][] salida=new int[a.length][a.length];
      for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                  if(a[i][j]>b[i][j]){
                  salida[i][j]=a[i][j];
                  } else {
                  salida[i][j]=b[i][j];
                  }
            }
      }
      return salida;
}
```

3. b **(1 punto)** Escribir otro método que recibe otras dos tablas cuadradas pero que pueden ser de tamaños distintos y devuelve una tabla nueva con el mayor número de cada posición. (Pista: Si la más pequeña la pasamos a una del tamaño de la grande, podemos utilizar el método anterior)

Ejemplo:
Se le pasa

```
1 2 3          9 9
1 1 1          9 9
8 6 0
```

Y devuelve:

```
9 9 3
9 9 1
8 6 0
```

```
private static int[][] mayorCuadradas(int[][] a, int[][] b) {
	int[][] salida;
	int[][] grande;
	int[][] pequenya;
	if(a.length>b.length){
		grande=a;
		pequenya=b;
	} else {
		grande=b;
		pequenya=a;

	}

	salida=new int[grande.length][grande.length];
	for (int i = 0; i < pequenya.length; i++) {
		for (int j = 0; j < pequenya.length; j++) {
			salida[i][j]=pequenya[i][j];
		}
	}


	return mayorIguales(grande,salida);
	}
```

4. Queremos crear una aplicación sobre vehículos. Para ello implementaremos en java las siguientes clases:

- **(0,75 puntos)** Vehículo. Los vehículos tendrán un método pasaRevision() que dependerá de cada vehículo (es abstracto). Este método devuelve true si pasa la revisión y false si no. De los vehículos se guarda la matrícula (un String) y el año de fabricación (un entero). Ambos son obligatorios para crear vehículos.

```
public abstract class Vehiculo {
	private final String matricula;
	private final int anyo;

	public String getMatricula() {
	return matricula;
	}

	public int getAnyo() {
	return anyo;
	}
```

```
    public Vehiculo(String matricula, int anyo) {
    this.matricula = matricula;
    this.anyo = anyo;
    }

    public abstract boolean revisar();
}
```

- **(0,5 puntos)** Barco. Hereda de Vehículo y tiene como atributos si las velas están en buen estado o no y si el casco del barco está en buen estado o no. Inicialmente, las velas y el casco están en buen estado. Para pasar la revisión, las velas y el barco deben estar en buen estado.

```
class Barco extends Vehiculo{

    private boolean velasBien=true;
    private boolean cascoBien=true;

    public void setVelasBien(boolean velasBien) {
    this.velasBien = velasBien;
    }

    public void setCascoBien(boolean cascoBien) {
    this.cascoBien = cascoBien;
    }

    public Barco(String matricula, int anyo) {
    super(matricula, anyo);
    }


    @Override
    public boolean revisar() {
    return velasBien && cascoBien;
    }
}
```

- **(1 punto)** Coche. Hereda de Vehículo y tiene como atributo el estado general del coche (NUEVO, PASABLE Y COCHAMBROSO, no hay ni habrá más estados que estos tres) que se le pasa para crearlo. Un coche pasará la revisión si es nuevo o pasable.

```
class Coche extends Vehiculo{
	private EstadosCoche estado;

	public EstadosCoche getEstado() {
	return estado;
	}

	public void setEstado(EstadosCoche estado) {
	this.estado = estado;
	}

	public Coche(String matricula, int anyo, EstadosCoche estado) {
	super(matricula, anyo);
	this.estado=estado;
	}

	public enum EstadosCoche {
	NUEVO, PASABLE, COCHAMBROSO;

	}

	@Override
	public boolean revisar() {
	return this.estado==EstadosCoche.NUEVO
			|| this.estado==EstadosCoche.PASABLE;
	}
}
```

- **(1'5 puntos)** TestVehiculos. El método main() de la clase TestVehiculos creará una lista (ArrayList) de cuatro Vehículos distintos (dos coches y dos barcos) y **los modificará** para que dos pasen la revisión y dos no la pasen. Después imprimirá las matrículas de los vehículos que no pasen la revisión.

```
public class TestVehiculos {
	public static void main(String[] args) {
	ArrayList<Vehiculo> lista=new ArrayList<>();
	lista.add(new Coche("TTTT",20012, Coche.EstadosCoche.NUEVO));
	lista.add(new Coche("RREE",2019, Coche.EstadosCoche.NUEVO));
	lista.add(new Barco("iuhufvd",1999));
	lista.add(new Barco("rRTR",2001));
	((Coche)lista.get(0)).setEstado(Coche.EstadosCoche.COCHAMBROSO);
	((Barco)lista.get(3)).setCascoBien(false);
	for (Vehiculo v :
```

```
                lista) {
            if (!v.revisar()) {
                System.out.println(v.getMatricula());
            }
      }
      }
}
```

- **(1 punto, extra)** Se creará un método cuentaBarcos que use instanceof al que se le pasa un ArrayList de Vehiculos y devuelve el número de Barcos que tiene ese ArrayList.

```
public static int cuentaBarcos(ArrayList<Vehiculo> l){
      int cuenta=0;
      for (Vehiculo v:
            l) {
            if(v instanceof Barco){
                cuenta++;
            }
      }
      return cuenta;
}
```

5. Se dispone del siguiente programa que lee los tiempos en que 10 corredores han acabado una carrera. El programa debe determinar qué corredores tienen el primer, segundo y último puesto.

(0'5 puntos) Completar el compareTo.
(1 punto) Completar el Compare
(1 punto) Mostrar el primero, el segundo y el último corredor

```
public class Corredor implements Comparable<Corredor>{
      private final String nombre;
      private final int dorsal;
      private final int tiempo;

      public String getNombre() { return nombre; }

      public int getDorsal() { return dorsal;}

      public int getTiempo() { return tiempo;}

      public Corredor(String nombre, int dorsal, int tiempo) {
      this.nombre = nombre;
      this.dorsal = dorsal;
      this.tiempo = tiempo;
      }

      @Override
      public int compareTo(Corredor o) {
```

```
            //ORDENA POR DORSAL
        return this.dorsal-o.dorsal;
    }

    public String info() {
    return this.dorsal+" - "+this.tiempo+" : "+this.nombre;
    }
}

import java.util.ArrayList;
import java.util.Comparator;

public class Carrera {
    public static void main(String[] args) {
    ArrayList<Corredor> participantes=new ArrayList<>();
    //Se rellena la lista con 10 participantes (NO HAY QUE HACERLO)

    participantes.sort(new Comparator<Corredor>() {
        @Override
        public int compare(Corredor o1, Corredor o2) {
                    //Ordena por tiempo y, si son iguales, por nombre
                if(o1.getTiempo()==o2.getTiempo()){
                return o1.getNombre().compareTo(o2.getNombre());
                }
                return o1.getTiempo()-o2.getTiempo();
        }
    });

    //Muestra la información del primero, del segundo y del último
        System.out.println(participantes.get(0).info());
        System.out.println(participantes.get(1).info());
        System.out.println(participantes.get(participantes.size() - 1).info());
    }
}
```