Grado en Ingeniería Informática y Doble Grado en Informática y Administración de Empresas	***MODELO A*** Asignatura Estructura de Datos y Algoritmos **24 de Febrero de 2014.** **PRIMER EXAMEN PARCIAL**

Nombre:

.....

Apellidos:

.....

Grupo: .....

LEA ATENTAMENTE ESTAS INSTRUCCIONES ANTES DE COMENZAR LA PRUEBA

1. Es necesario poner todos los datos del alumno en el cuadernillo de preguntas (este documento). Use un bolígrafo para rellenarlos.
2. El examen de la parte de teoría está compuesto por 10 Preguntas con cuatro posibles respuestas cada una (5 puntos en total).
3. Para que la pregunta multirespuesta se considere contestada correctamente, se deben marcar todas las casillas que sean correctas **(pueden existir una o más contestaciones correctas a cada pregunta, pero nunca ninguna).**
4. Las contestaciones incorrectas **no restan** puntos.
5. Solamente se evaluará la contestación en este cuadernillo de preguntas.
6. Cuando finalice la prueba, se deben entregar el enunciado del examen y cualquier hoja que haya empleado.
7. No está permitido salir del aula por ningún motivo hasta la finalización del examen.
8. Desconecten los móviles durante el examen.
9. La duración del examen es de **30 minutos**.

NO PASE DE ESTA HOJA hasta que se le indique el comienzo del examen

1. Teniendo en cuenta el siguiente código y respecto a los constructores de una clase:

```
public class A {

	float atr1;

	public A(float atr1) {
		this.atr1=atr1;
	}

	public float A() {
		return atr1;
	}

	public static void main(String[] args) {
		A objA=new A(3);
	}
}
```

   a) **El constructor de la clase A se invoca automáticamente cuando se crea un objeto de la clase.**

   b) La clase A tendrá tres constructores: los dos constructores definidos explícitamente en el código anterior y un constructor definido implícitamente por Java sin parámetros.

   c) El segundo constructor es correcto porque un constructor puede devolver un tipo de datos.

   d) **El primer constructor es correcto porque un constructor puede recibir parámetros.**

2. Respecto a la Programación Orientada a Objetos (POO):

   a) Permite empaquetar los datos junto con sus operaciones y separar además la definición de la implementación.

   b) Un objeto es una combinación de datos y las operaciones que permiten manipularlos.

   c) Un programa en ejecución que sigue los principios de la POO crea y manipula (mediante llamadas a métodos) objetos concretos (instancias).

   d) Un programa que sigue los principios de la POO no está orientado a su reutilización debido al encapsulamiento de los datos y operaciones.

3. Respecto al siguiente código:

```
public class A extends B {

   public A() {
   }

   protected long metodo1(long atriB) {
         this.atriB=atriB;
         return atriB;
   }

   protected long metodo1() {
         return 0;
   }

   public static void main(String[] args) {
         A objA=new A();
   }
}
```

a) El objeto A se almacena en una dirección de memoria una vez se ha instanciado la clase A.

b) El objeto A no está asociado a una posición de memoria concreta.

c) El método metodo1 es un método sobrecargado.

d) El método metodo1 no puede ser un método sobreescrito de la superclase B porque está sobrecargado en la clase A.

4. Respecto al encapsulamiento:

a) El modificador de acceso más restrictivo es protected.

b) Un buen encapsulamiento se consigue definiendo los atributos de una clase como privados y facilitando métodos que permitan consultar y modificar dichos atributos.

c) En una clase los métodos de acceso a los atributos (set y get) deben ser declarados como privados.

d) En Java, para facilitar el encapsulamiento, si no se indica explícitamente el modificador de acceso por defecto para los elementos de una clase es private.

5. Sobre clases abstractas e interfaces:

a) La instancia de una interfaz es un objeto al que no se puede acceder a sus métodos.

b) Un método abstracto solo proporciona la definición o cabecera y puede ser implementado de diferentes maneras en las subclases de la clase abstracta.

c) Una misma clase puede implementar más de una interfaz pero debe implementar todos los métodos definidos en las mismas.

d) Las clases abstractas permiten en Java la herencia múltiple (varios padres para una misma subclase).

6. Teniendo en cuenta el siguiente código y los conceptos relacionados con el acceso a los métodos y atributos de una clase:

```
public class B {
	long atriB;
	static long atriB1;

	public B() {
	}

	protected long metodo1() {
		atriB=0;
		return atriB;
	}

	public static void main(String[] args) {
		atriB=0;
	}
}
```

a) Todos los elementos de instancia de la clase B sólo pueden ser utilizados cuando se ha instanciado un objeto de una clase dada.

b) Existe un error de en el método main porque para acceder al atributo atriB necesitamos crear una instancia de la clase u objeto.

c) El atributo atriB1 no pertenece a las instancias de la clase si no a la propia clase.

d) Todas las respuestas son correctas.

7. Teniendo en cuenta el siguiente código:

```
public class Primera extends Segunda implements Tercera {…}
```

a) Tercera es una interfaz, Primera y Segunda son clases.

b) Existen una relación de herencia entre Primera y Segunda. Segunda es la superclase de Primera.

c) Existen una relación de herencia entre Primera y Segunda. Primera es la superclase de Segunda.

d) Es necesario que Segunda sea una clase abstracta y Tercera puede ser una interfaz.

8. Teniendo en cuenta el siguiente código:

```
public class A extends B {
	protected long atr;

	public A(long a) {
		atr=a;
	}

	public A() {
		this(12);
	}

	protected long metodo1(long a, float b) {
		atr=a;
		super.atr2 = b;
		return atr;
	}

	protected long metodo1(float b, long a) {
		atr=a;
		super.atr2*= b;
		return atr;
	}
}
```

a) Atr2 es un atributo definido en la superclase B.

b) La sobrecarga del método metodo1 es incorrecta porque la signatura (lista de argumentos) es igual en las dos definiciones.

c) Una de las dos definiciones del método método1 tiene que ser estática (static) para que la sobrecarga sea correcta.

d) Desde un constructor no se puede llamar a otro constructor de la misma clase.

9. Teniendo en cuenta el siguiente código:

```
public class B {
	long atriB;
	public B() {

	}

	protected long metodo1() {
		atriB=0;
		return atriB;
	}
}
public class A extends B {
	public long atrA;

	public A() {
	}
	protected int metodo1() {
		return 0;
	}

	public static void main(String[] args) {
		B obj = new B();
		obj.atriB=obj.metodo1();

		obj=new A();
		obj.metodo1();
    }
```

a) La sobreescritura del método método1 es incorrecta porque las dos definiciones devuelven diferentes tipos de datos.

b) La clase B no puede ser instanciada dentro del *main* de la clase A porque existe una relación de herencia entre ambas.

c) El objeto obj no puede ser definido e instanciado de la superclase B (instrucción `B obj = new B()`) y luego instanciado como de la subclase A (instrucción obj=new A()).

d) Todas las respuestas son incorrectas.

10. Teniendo en cuenta el siguiente código y respecto al concepto de herencia:

```
public class A {
   private long atriA;
   public int atriA2;

   public A() {
   }

   protected long metodoA() {
          atriA=0;
          return atriA;
      }

   public long getAtriA() {
          return atriA;
   }

   public void setAtriA(long atriA) {
          this.atriA = atriA;
   }
}
public class B extends A {
   long atriB;
   public B() {
   }

   protected long metodoB() {
          atriB=0;
          return atriB;
      }
}
public class C extends A{
   public long atrC;
   public C() {

   }
   protected int metodoC() {
          return 0;
   }

}
```

a) La superclase A contiene los atributos y métodos comunes a las subclases B y C, mientras que las subclases B y C sólo definen aquellos atributos y métodos propios no definidos en la superclase.

b) Desde las subclases B y C, el acceso a los atributos privados de la superclase A no es posible en ningún caso debido al principio de encapsulamiento de la POO.

c) Las subclases B y C heredan todos los métodos (excepto los privados) de su superclase A, pero sólo podrían reescribir el método A si este fuera abstracto.

d) Si se incluyera el siguiente código:

```
public class D extends B, C
```

sería correcto porque representa un ejemplo de herencia múltiple.