Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2012/2013)

Contesta en esta misma hoja

Apellidos:
Nombre:

Apartado 1 (2 puntos)

Señala los errores de compilación del siguiente fragmento de código Java (si los hubiera) e indica cómo corregirlos de la manera más sencilla posible.

```java
/* Archivo centro/Espacio.java */
package centro;
public class Espacio {
        String nombre;
        protected int capacidad;
        public Espacio(int c) { capacidad = c; }

        public String reservar(String usuario) { return usuario + " reserva:" + nombre; }
}
```

```java
/* Archivo reservas/Aula.java*/
package reservas;
import centro.Espacio;

class Aula extends Espacio {
        private boolean audiovisuales;
        public Aula(String nombre, int capacidad) {
                super(capacidad);
                audiovisuales = true;
        }

        public void anular() {
             System.out.println(super.nombre + " con capacidad " + capacidad);
        }

        protected boolean reservar() { return true; }
}

public class Apartado1 {
        public static void main(String[] args) {
                Espacio e1, e2;
                e1 = new Espacio();
                e2 = new Aula("Aula2", 26);
                System.out.println(e2.audiovisuales);
                System.out.println(e1.reservar("Jose"));
                e1.anular();
                System.out.println(((Aula) e2).reservar());
        }
}
```

Errores y Corrección:

(puedes seguir al dorso)

Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2011/2012)

Contesta en esta misma hoja

Apellidos:
Nombre:

Apartado 2 (3.5 puntos)

Completa el siguiente código para que la salida del programa sea la de más abajo. Téngase en cuenta que todos los vehículos a motor se crean con un consumo inicial dado en fábrica (que no se cambiará nunca) y con kilometraje inicial 0 (modificable según se muestra en el `main()` abajo). El consumo actual de los vehículos de gasolina se mantiene igual al dado en fábrica hasta que superen los 75000 kilómetros, y a partir de ese momento se incrementa en 0.4 para el resto de la vida del vehículo. Para los vehículos de gasoil el consumo actual se calcula incrementando el dado en fábrica en un 1% por cada 5000 kilómetros acumulados (ver ejemplo en el `main()`: 49500 km equivale a un 9% de incremento).

```
public abstract class VehiculoMotor {
        // completar



        public VehiculoMotor(String nombre, double consumoFabrica) { // completar contructor


        }

        public abstract double consumoActual();

        public String toString() { return nombre + " : " + consumoFabrica + " : " + consumoActual(); }

        // añadir métodos nuevos









}
// añadir clases nuevas









public class Apartado2 {
        public static void main(String[] args) {
                VehiculoMotor c1 = new CocheGasolina("Ford Focus 1.9", 5.0);
                CocheGasoil c2 = new CocheGasoil("Toyota Yaris 1.4D", 4.0);
                CocheGasolina c3 = new CocheGasolina("Citroen C3 1.6", 5.2);

                c1.sumarKilometros( 30000 );
                c1.sumarKilometros( 55700 );  // acumula 85700 km, es decir, más de 75000
                c2.sumarKilometros( 49500 );
                c3.sumarKilometros( 35200 );

                System.out.println(c1); //consumo actual 5.4 = 5.0 + 0.4 (por más de 75000 km)
                System.out.println(c2); //consumo actual 4.36 = 4.0 * (1 + 1% * 9) ver enunciado
                System.out.println(VehiculoMotor.get("Citroen C3 1.6"));
                System.out.println(VehiculoMotor.get("Ford Focus"));
        }
}
```

Salida:

```
Ford Focus 1.9 : 5.0 : 5.4
Toyota Yaris 1.4D : 4.0 : 4.36
Citroen C3 1.6 : 5.2 : 5.2
null
```

Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2011/2012)

Contesta AL DORSO, usa hojas adicionales si fuese necesario

Apellidos:
Nombre:

Apartado 3 (4.5 puntos)

Se quiere programar una clase de utilidad *PrettyPrinter*, que imprima por consola cualquier conjunto de objetos con estructura en forma de árbol, como por ejemplo un directorio de un sistema de ficheros, un árbol genealógico, o la estructura de paquetes y clases de un diagrama UML.

Para abstraerte de las clases concretas que PrettyPrinter debe imprimir, crea una interfaz *ArbolIterable* con un método **`int numHijos()`** que devuelva el número de hijos de un objeto, y otro método **`ArbolIterable getHijo(int i)`** que devuelva el hijo i-ésimo. El método "*imprime"* de *PrettyPrinter* debe aceptar como segundo parámetro una cadena que se usará para indentar los hijos.

Implementa las clases necesarias para que el siguiente listado dé el resultado mostrado más abajo. El programa imprime la estructura de un diagrama UML formado por dos paquetes y tres clases. Por simplicidad asume que un paquete UML puede contener paquetes y clases UML, y una clase UML no puede contener ni paquetes ni clases.

```
package apartado3;

public class MainArbol {
	public static void main(String[] args) {
		UMLPackage j  = new UMLPackage("uam");
		UMLPackage u  = new UMLPackage("docencia");
		UMLClass   al = new UMLClass("Profesor", Privacidad.PUBLIC);
		UMLClass   ve = new UMLClass("Asignatura", Privacidad.PUBLIC);
		UMLClass   rt = new UMLClass("Utilidades", Privacidad.PACKAGE);
		j.add(u);
		u.add(al);
		u.add(ve);
		u.add(rt);

		PrettyPrinter.imprime(j, "  ");
	}
}
```

```
Salida:
uam
  docencia
    + Profesor
    + Asignatura
    ~ Utilidades
```