Universidad Carlos III de Madrid

Programación de Sistemas
Grado en Ingeniería de Sistemas de Comunicaciones y Grado en Ingeniería Telemática

Leganés, 20 de marzo de 2015 — Examen parcial 1 (teoría)
Duración de la prueba: 20 min — Puntuación: 3 puntos sobre 10 del examen

Sólo una opción es correcta en cada pregunta. Cada respuesta correcta suma 0.3 puntos. Cada respuesta incorrecta resta 0.1 puntos. Las preguntas no contestadas no suman ni restan puntos.

Marca:  Anula:  No uses:   

- Marca la respuesta a cada pregunta con una equis ("X") en la tabla de abajo.
- Si marcas más de una opción o ninguna opción, la pregunta se considera no contestada.
- Rellena **tus datos personales** antes de comenzar a realizar el examen.

Nombre:	Grupo:

Firma:

NIA:

	A	B	C	D		A	B	C	D
1					6				
2					7				
3					8				
4					9				
5					10				

1.- Dada la clase `Animal` y la clase `Mamifero` que deriva de ella, y el siguiente fragmento de código, indica cuál de las líneas de código es correcta:

```
Animal a = new Animal("Mamífero");
Mamifero m = new Mamífero();
```

(a) `a = (Mamifero) m("Mamífero");`

(b) `m = a;`

(c) *** `a = m;`

(d) `m = (Mamifero) a;`

2.- Indica cuál de las siguientes afirmaciones es *incorrecta*:

(a) *** Dos objetos distintos de la clase `JButton` no pueden tener registrado el mismo escuchador de eventos `ActionListener`.

(b) Cualquier objeto de la clase `JButton` puede estar asociado a varios escuchadores de eventos diferentes.

(c) Cualquier objeto de la clase `JButton` genera un evento `ActionEvent` cuando el usuario pulsa en el botón.

(d) La interfaz `ActionListener` sólo tiene un método `actionPerformed` que es público y abstracto.

3.- Dada la clase `Color`, con un único constructor sin parámetros, y la clase `ColorRGB` que deriva de ella, para acceder a dicho constructor:

(a) Se usa `Color()` desde la primera línea del constructor de `ColorRGB`.

(b) Se usa `this.Color()` desde la primera línea del constructor de `ColorRGB`.

(c) Se usa `super()` en cualquier línea, pero siempre dentro del constructor de `ColorRGB`.

(d) *** Se usa `super()` desde la primera línea del constructor de `ColorRGB`.

4.- Si la clase `MedioDeTransporte` no implementa todos los métodos de la interfaz `Tarificable`, indica cuál sería la declaración correcta de la clase:

(a) *** `public abstract class MedioDeTransporte implements Tarificable`

(b) `public abstract interface MedioDeTransporte implements Tarificable`

(c) `public MedioDeTransporte implements Tarificable`

(d) `public abstract class MedioDeTransporte extends Tarificable`

5.- Dada la clase `Persona` y la clase `Alumno` que deriva de ella, para obtener el valor del atributo privado `nombre` de la clase padre desde la clase hija:

(a) Se puede usar `super(nombre)` desde la primera línea del constructor de `Alumno`.

(b) *** Como el atributo es privado, no es posible acceder a su valor sin un método de acceso público.

(c) Se puede usar `super.nombre`.

(d) Se puede usar `(Persona) this.nombre`.

6.- Dado el siguiente código, la última línea:

```
public interface I1 { ... }
public interface I2 { ... }
public class C1 implements I1 { ... }
public class C2 extends C1 implements I2 { ... }
I1 obj = new C2();
```

(a) Es correcta porque `C2` implementa `I2`.

(b) Es incorrecta porque `C2` no implementa `I1` porque ya implementa `I2`.

(c) Es incorrecta porque `I2` no hereda de `I1`.

(d) *** Es correcta porque `C2` implementa `I1`.

7.- Dada la clase `Aplicacion`, con un método abstracto `abrir()`, las clases no abstractas `Word` y `Firefox` que derivan de ella, y el siguiente código, indica cuál de las afirmaciones es correcta.

```
Aplicacion[] aplicaciones = { new Word(), new Firefox(), new Firefox() };
for(int i=0; i<aplicaciones.length; i++) {
  aplicaciones[i].abrir();
}
```

(a) El código compila correctamente pero salta una excepción en ejecución porque la aplicación *Firefox* está duplicada en el array.

(b) El código no es correcto porque habría que usar `instance of` para saber de qué clase se trata para ejecutar el método adecuadamente.

(c) El código no es correcto y habría que usar un *casting* (en concreto *downcasting*) para saber de qué clase se trata y así ejecutar el método adecuadamente.

(d) *** El código es perfectamente válido y cada aplicación se abre correctamente con la implementación del método propia de cada una de ellas.

8.- Dada la clase `Vehiculo` que implementa la interfaz `AccionesVehiculo` y la clase `Motocicleta` que deriva de `Vehiculo`:

(a) *** La clase `Motocicleta` implementa la interfaz `AccionesVehiculo` aunque no lo declare explícitamente.

(b) La clase `Motocicleta` no puede implementar la interfaz `AccionesVehiculo` porque no existe la herencia múltiple en Java.

(c) La interfaz `AccionesVehiculo` sólo puede tener métodos implementados (con código) si tiene al menos un método abstracto.

(d) La interfaz `AccionesVehiculo` sólo puede tener métodos implementados (con código) si son públicos.

9.- Para añadir la etiqueta `label` a la ventana `window` se usaría:

(a) *** `window.getContentPane().add(label)`

(b) `getContentPane(window).add(label)`

(c) Ninguna de las otras opciones es correcta.

(d) `window.getContentPane().add(new label())`

10.- Dada la clase `Figura` y las clases `Cuadrado` y `Triangulo` que derivan de la primera, si todas tienen su propio método `double getArea()`, ¿cómo se puede invocar al método de la clase padre desde las clases hijas?

(a) *** super.getArea()

(b) No se puede porque es un método oculto abstracto.

(c) this.getArea()

(d) super(getArea())