**PDOO. Examen Febrero 13/14.Segundo parcial**

**Nombre:**_____ **Grupo:** _____

**1. (2 puntos)** Implementa en Java y Ruby Deportista y Corredor sin incluir nada que no esté en el diagrama de clases. **NOTA**: En Ruby implementa Corredor como una clase.

**JAVA**

```
public interface Deportista {
    public void entrenar();
    public void entrenar(String entrenador);
    public void comerSano();
}
public abstract class Corredor implements Deportista{
    private String nombre;
    private String licencia;
    @Override
    public void comerSano() { }
    protected abstract void correr();
}
```

**RUBY**

```
module Deportista
  def entrenar(nombreEntrenador)
     raise NotImplementedError.new('metodo no implementado')
  end
  def comerSano
     raise NotImplementedError.new('metodo no implementado')
  end
end


 class Corredor
   include Deportista
   @nombre
   @licencia
   def correr
```

**PDOO. Examen Febrero 13/14.Segundo parcial**

**Nombre:**_____ **Grupo:** _____

```
    raise NotImplementedError.new('metodo no implementado')
  end
  def comerSano
  end
  private_class_method :new
end
```

**2. (1 punto)** Implementa en Java el método comerSano() de Maratonista teniendo en cuenta que los maratonistas tienen que comerConcentradoProteinicoMaratoniano() antes de comerSano() como Corredor.

```
@Override
    public void comerSano(){
      this.comerConcentradoProteinicoMaratoniano();
      super.comerSano();
    }
```

**3. (1 punto)** Implementa en Java y en Ruby el caso de que Maratonista, además de heredar de Corredor, también heredara de Caminante. Define sólo lo que necesites en la clase Maratonista.

JAVA

```
public class Maratonista extends Corredor implements Caminante {}
```

RUBY

```
class Maratonista<Corredor
  include Caminante
end
```

**4. (0,5 puntos)** Qué tipo de relación hay entre Corredor y Deportista y cuál es su significado.

**Al menos había que decir:** Es una relación de realización y significa que la clase Corredor o alguna de sus subclases deben implementar los métodos definidos en la interfaz Deportista

**5. (0,5 puntos)** Qué tipo de relación hay entre Corredor y Velocista y cuál es su significado.

**Al menos había que decir:** Es una relación de Herencia y significa que todo lo definido en Corredor también está definido en Velocista.

**PDOO. Examen Febrero 13/14.Segundo parcial**

**Nombre:**_____ **Grupo:** _____

**6.** (**0,5 puntos)** Marca de los siguientes cuáles son métodos abstractos en Corredor:

| | |
|---|---|
| correr() | X |
| comerSano() | |
| entrenar() | X |
| entrenar(Entrenador entrenador) | X |

**7.(0,5 puntos)** Marca qué métodos están redefinidos y/o sobrecargados en la clase Maratonista.

| | redefinido | sobrecargado |
|---|---|---|
| correr() | X | |
| comerSano() | X | |
| entrenar() | X | X |

**8. (0,5 puntos)** Marca las palabras que describen el tipo de herencia que hay entre la clase Corredor y Velocista:

| | |
|---|---|
| Simple | X |
| Múltiple | |
| Especialización | X |
| Especificación | X |
| Construcción | |
| Limitación | |

**9. (0,5 puntos)** La clase Marchista en el diagrama de clases es incorrecta. Complétala, modificando lo mínimo para que lo sea.

| Marchista |
|---|
| +correr():void<br>+entrenar(nombreEntrenador:String):void<br>+entrenar():void |

**10. (0,5 puntos)** Proporciona el tipo estático y dinámico de la variable que se indica en las siguientes sentencias Java

| | Variable | Tipo Estático | Tipo Dinámico |
|---|---|---|---|
| Corredor c= new Maratonista(); | c | Corredor | Maratonista |
| Deportista d= new Velocista(); | d | Deportista | Velocista |
| c=new Marchista(); | c | Corredor | Marchista |
| d = c; | d | Deportista | Marchista |

**PDOO. Examen Febrero 13/14.Segundo parcial**

**Nombre:**_____ **Grupo:** _____

**11. (1,5 puntos)** En el siguiente código Java:

1. Corrige los posibles errores de compilación.
2. Una vez corregidos los errores de compilación, indica las líneas de código en las que habría error de ejecución.

| | Corrección error en Compilación | Error en ejecución |
|---|---|---|
| Corredor c= new Velocista(); | | |
| Deportista d= new Marchista(); | | |
| d.correr(); | ((Corredor)d).correr()<br>o<br>((Marchista)d).correr() | |
| c.comerSano(); | | |
| c.correr(150); | ((Velocista)c).correr(150); | |
| Velocista v= c; | Velocista v= (Velocista) c; | |
| ArrayList<Corredor> corredores= new ArrayList(); | | |
| corredores.add(c); | | |
| corredores.add(d); | corredores.add((Corredor)d);<br>o<br>corredores.add((Marchista)d); | |
| corredores.get(0).correr(10); | ((Velocista)corredores.get(0)).correr(10); | |
| corredores.get(1).correr(10); | ((Velocista)corredores.get(1)).correr(10); | Un Marchista no puede comportarse como un Velocista |
| c= new Corredor(); | La clase corredor es abstracta y no se puede instanciar. | |

**12. (0,5 punto)** A partir de la clase paramétrica proporcionada, Equipo, la cual representa a equipos de cualquier tipo de corredores con un capitán al frente,

```
public class Equipo<Tipo> {
        ArrayList<Tipo> componentes;
```

**PDOO. Examen Febrero 13/14.Segundo parcial**

**Nombre:**_____ **Grupo:** _____

```
        Tipo capitan;

        public Equipo(Tipo capi) {
                componentes= new ArrayList();
                 this.setCapitan(capi);
        }
        public Tipo getComponente(int n){
                  return (Tipo)componentes.get(n);
        }
        public void setComponente(Tipo elemento){
                componentes.add(elemento);
        }
        public void setCapitan(Tipo capi) {
                        capitan=capi;
        }
        public Tipo getCapitan(){
                return capitan;
        }
}
```

**completa el siguiente main() para definir un equipo de Velocistas con su capitán, usando para ello los Velocistas que se proporcionan.**

**// NOTA:** usa los métodos definidos en Equipo

```
        public static void main(String[] args){

                 Velocista v1 = new Velocista();
                 Velocista v2 = new Velocista();
                 Velocista capitan= new Velocista();
                // Continuar con el código
                Equipo<Velocista> equipo = new Equipo(capitan);
                equipo.setComponente(v1);
                equipo.setComponente(v2);
                Velocista v3 = equipo.getComponente(0);
                equipo.setCapitan(v3);
                Velocista v4 = equipo.getCapitan();

}
```

**13. (0,5 puntos)**¿Cuál es el objetivo principal del patrón MVC? Indica qué elementos intervienen y cuál es su función principal.

**Al menos había que decir:**

Objetivo: Obtener sistemas modulares poco acoplados, manteniendo la separación entre el modelo de negocio y la interfaz de usuario.

Elementos:

Modelo: modela la lógica del modelo de negocio.

Vista: Una de las posibles IU correspondiente al modelo de negocio.

Controlador: establece la correspondencia entre la vista y el modelo.