Soluciones Ejercicios Tema 1

Germán Moltó Martínez
gmolto@dsic.upv.es
Estructuras de Datos y Algoritmos
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

BonoMetro (I)

```
public class BonoMetro{
   private int saldo;
   private final static int SALDO_INICIAL = 10;
   public BonoMetro(int saldo){ this.saldo = saldo; }
   public BonoMetro(){
      this(SALDO_INICIAL);
   }
   public int getSaldo(){ return this.saldo;}
   public void setSaldo(int saldo){this.saldo = saldo;}
   public String toString(){return "BonoMetro de " + this.saldo + " viajes.";}
   public void fichar(){
      if (saldo > 0) saldo--;
      else System.err.println("BonoMetro agotado");
   }
}
```


BonoMetro (II)

```
public class MaquinaExpendedora{
   public static void actualiza(BonoMetro b){
      b.setSaldo(10);
   }
}
```


Solución Ejercicio de Clases: Persona (1/3)

```
public class Persona
{
   private int edad;
   private String nif;
   private int altura;

   private final static int DEFAULT_EDAD = 25;
   private final static String DEFAULT_NIF="11111111A";
   private final static int DEFAULT_ALTURA = 175;

   public Persona(int edad, String nif, int altura){
      this.edad = edad;
      this.nif = nif;
      this.altura = altura;
    }
```

Solución Ejercicio de Clases: Persona (2/3)

```
public Persona(){
   this (DEFAULT_EDAD, DEFAULT_NIF,  DEFAULT_ALTURA);
  }
   public int getEdad(){return edad;}
   public String getNif(){ return nif;}
   public int getAltura(){return altura;}

    public void setEdad(int edad){this.edad = edad;}
   public void setNif(String nif){this.nif = nif;}
   public void setAltura(int altura){this.altura = altura;}
   public String toString(){
     return "Mi NIF es " + nif + " tengo " + edad + " anyos y mido " +
  altura + " cm ";
     }}
```

Solución Ejercicio de Clases: Persona (3/3)

```
public class TestPersona
{
   public static void main(String args[]){
       Persona p1 = new Persona();
       System.out.println("Persona 1: " + p1);

       Persona p2 = new Persona(30,"58765786M",190);
       System.out.println("Persona 2: " + p2);

    }
}
```

Solución Ejercicio: Yogures (I)

```
public class Yogur
{
   private double calorias;

   public Yogur(){
     calorias = 120.5;
   }
   public double getCalorias(){
     return calorias;
   }
}
```


Solución Ejercicio: Yogures (II)

```
public class YogurDesnatado extends Yogur
{
  public YogurDesnatado(){
     super();
   }

   public double getCalorias(){
       return super.getCalorias() / 2;
    }
}
```

Solución Ejercicio: Yogures (III)

- Solución alternativa:

```
public class YogurDesnatado extends Yogur
{
  public YogurDesnatado(){
     super(); //No es necesaria, se hace automáticamente.
     this.calorias = this.calorias / 2;
   }
}
```

- Esta solución requiere que el atributo calorias en la clase Yogur sea protected.

Solución Ejercicio: Yogures (IV)

```
public class TestYogures
{
   public static void main(String args[]){
      Yogur y = new Yogur();
      YogurDesnatado yd = new YogurDesnatado();

      System.out.println("Calorias: " + y.getCalorias());
      System.out.println("Calorias Desnatado: " +
   yd.getCalorias());
    }
}
```


Solución Ejercicio Herencia: Felino

- Solución 1: Aprovechar la definición del atributo loQueHablo y del método habla.

```
public class Gato extends Felino{
   public Gato(){
         loQueHablo = "Miau";
   }
}
```

- Solución 2: Redefinir la implementación del método habla

```
public class Gato extends Felino{
    public String habla(){ return "Miau";}
}
```


Solución Ejercicio Herencia: Personas (I)

```
public class Persona
{
  public Persona(){
   }
  public void hablar(){
     System.out.println("Estoy hablando");
   }
  public void comer(){
     System.out.println("Yummy Yummy!");
    }
}
```

Solución Ejercicio Herencia: Personas (II)

```
public class Ingeniero extends Persona
{
   public void razonar(){
     System.out.println("Observo, reflexiono, concluyo.");
    }

   public void trabajarEnGrupo(){
     System.out.println("Colaboro con otras personas");
    }
}
```


Solución Ejercicio Herencia: Personas (III)

```
public class IngenieroInformatico extends Ingeniero
{
    public void crearPrograma(){
        hablar(); comer(); razonar(); trabajarEnGrupo();
    }
}
```

- Nótese que es posible invocar cualquier de los métodos definidos sobre un objeto de tipo IngenieroInformatico ya que se reciben por herencia.

Solución Ejercicio: Diseño de Constructores

```
public class Base{
  public int bPublico; int bProtegido;  private int bPrivado;

  public Base(int bPublico, int bProtegido, int bPrivado){
    this.bPublico = bPublico;  this.bProtegido = bProtegido;
     this.bPrivado = bPrivado;
 }
}
public class Derivada extends Base{
   public int dPublico; private int dPrivado;

public Derivada(int bPublico, int bProtegido, int bPrivado, int dPublico,
     int dPrivado){
        super(bPublico, bProtegido, bPrivado);
        this.dPublico = dPublico;  this.dPrivado = dPrivado;
      } }
```


Ejercicio Interfaces: Arrancar (I)

```
public interface Vehiculo{
   void arrancar();
}

public class Coche implements Vehiculo {
    public void arrancar(){System.out.println("Brruumm!");}
}

public class Moto implements Vehiculo{
  public void arrancar(){System.out.println("Ratatata!");}
}
```

Ejercicio Interfaces: Arrancar (II)

```
public abstract class Vehiculo {
  public abstract void arrancar();
}

public class Coche extends Vehiculo {
    public void arrancar(){System.out.println("Brruumm!");}
}

public class Moto extends Vehiculo {
  public void arrancar(){System.out.println("Ratatata!");}
}
```


Ejercicio Interfaces: Arrancar (IV)

```
public class Arrancador{
 public void arrancaVehiculos(Vehiculo[] vVehiculos){
   for (int i = 0; i < vVehiculos.length ; i++) {
    v[i].arrancar();
   }
 }
}
```


Solución Ejercicio: Clase Cilindro (I)

1. Modificador de visibilidad adecuado: protected, visible para la clase, las derivadas y las pertenecientes al mismo paquete.
2. Implementación incorrecta b). Se pretende acceder al atributo privado *tipo* definido en Circulo.
3. Implementaciones que favorecen la reutilización:
   - public double area(){return 2 * super.area() + super.perimetro() * altura;}
   - public double volumen(){return super.area() * altura;}

Solución Ejercicio: Clase Cilindro (II)

4. Redefinición de la clase Cilindro:

```
public class Cilindro {
    private double altura;
    private Circulo base;

    public Cilindro(double radioBase, double altura){
        base = new Circulo(radioBase);
        this.altura = altura;
    }
    public double area(){
        return 2 * base.area() + base.perimetro() * altura;
    }
    public double volumen(){
        return base.area() * altura;
    }
}
```

Solución Ejercicio: losAnimales

- adoptaAnimal(new Armadillo());
  - NO provoca error de compilación ya que hay una conversión de tipos por ampliación, de Armadillo a Animal.
- Object o = new Armadillo();
  - NO da error de compilación. El polimorfismo lo permite.
- Armadillo a1 = new Animal();
  - SI da error de compilación. Armadillo NO es subclase de Animal.
- Armadillo a2 = new Muflon();
  - SI da error de compilación. Armadillo y Muflon no tienen relación de herencia.
- Resultado de ejecución: Grunt \n MOO \n Groar!

Solución: Conversor de Divisas (I)

```
public interface ConversorDivisa
{
  double convierte(double cantidad);
}
```

```
public class EuroDolarConversorDivisa implements ConversorDivisa
{
  public double convierte(double cantidad){
    return cantidad * 1.50;
  }

  public String toString(){
    return "Conversor de Euros a Dolares";
  }
}
```


Solución: Conversos de Divisas (II)

1. Una interfaz debe ser pública.
2. Lo lógico es que el método *convierte* reciba un tipo primitivo en lugar de un tipo objeto.
3. El método *convierte* es público en la interfaz (por definición) y no se puede reducir su visibilidad (a private) en la clase que lo implementa.
4. El método *convierte* recibe un double. Al poner un float se está creando un método nuevo sin ninguna relación con el que queremos implementar (ya que su lista de argumentos es diferente).
5. El perfil del método toString devuelve un String.
6. Falta la e del nombre del método.
7. Una clase implementa una interfaz (no la extiende).

Solución La Lavadora

- Hay que respetar el número de argumentos de los constructores:

```
public class Lavadora extends ElectroDomestico
{
  public Lavadora(){
    super(TiposCorriente.TIPO_220V, 4500.0);
  }
}
```

Solución Actores y Películas: Errores (I)

```
public class Persona {
    private String nombre;
    public Persona(String nombre) {this.nombre = nombre;}
    public String toString(){return this.nombre;}
}
```

```
public class Actor extends Persona {
private String pelicula;
public Actor(String nombre, String pelicula) {
    super(nombre);
    this.pelicula = pelicula;
}
  public String getPelicula() { return this.pelicula};
  public String toString(){"Nombre: " + super.toString() + "Película: " + this.pelicula;}
}
```


Solución Actores y Películas: Errores (I)

```
public class Peliculas {
  public static void mostrarReparto(Actor lista[], String pelicula) {
   for (int i = 0; i < lista.length; i++)
      if (lista[i].getPelicula().equals(pelicula))
            System.out.println(lista[i].toString());
   }
}
```

- Ojo con los límites del vector y para obtener su longitud.
- Se debe utilizar equals para comparar el contenido de dos String.
  - El operador == compara únicamente las referencias.