Práctica Nº 15

Palabras Claves: static y final

1. ¿Cuál de los siguientes fragmentos de código compila? En aquellos casos en los que no compila. ¿Qué cambios habría que hacer para que compile?

```
public class Foo{
    static int x;
    public void go(){
        System.out.println(x);
    }
}
```

```
public class Foo2{
    static int x;
    public static void go(){
        System.out.println(x);
    }
}
```

```
public class Foo3{
    final int x;
    public void go(){
        System.out.println(x);
    }
}
```

```
public class Foo4{
    static final int x = 12;
    public void go(){
        System.out.println(x);
    }
}
```

```
public class Foo5{
    static final int x = 12;
    public void go(final int x){
        System.out.println(x);
    }
}
```

```
public class Foo6{
    int x = 12;
    public static void go(final int x){
        System.out.println(x);
    }
}
```

```
public class Foo{
    static int x;
    public void go(){
        System.out.println(x);
    }
}
```

Compila. Los métodos no estáticos pueden acceder a los atributos estáticos.

```
public class Foo2{
    static int x;
    public static void go(){
        System.out.println(x);
    }
}
```

Compila. Los métodos estáticos pueden acceder a los atributos estáticos.

```
public class Foo4{
    static final int x = 12;
    public void go(){
        System.out.println(x);
    }
}
```

Compila. Los métodos no estáticos pueden acceder a las constantes.

```
public class Foo5{
    static final int x = 12;
    public void go(final int x){
        System.out.println(x);
    }
}
```

Compila. No se estaría imprimiendo la constante sino el valor del parámetro (el parámetro oculta a la constante).

```
public class Foo6{
    int x = 12;
    public static void go(final int x){
        System.out.println(x);
    }
}
```

Compila. El método estático no está accediendo al atributo no estático sino al parámetro (el parámetro oculta al atributo no estático).

2. ¿Es posible crear una clase llamada **CadenaDeCaracteres** como subclase de la clase `String` del paquete `java.lang`? ¿Por qué? ¿Qué otras clases de la API de java tienen la misma característica?

No, no es posible dado que la clase String es final. Otras clases final son, por ejemplo: Integer, Double, Float, Long.

3. Para el resto de esta práctica cree un proyecto llamado Practico15.

   a. Cree una clase llamada **ComoAccederAMetodosDeClase** en la que se declaran una variable de instancia y una variable de clase, y un método de instancia y uno de clase. Intente desde ambos métodos acceder a las variables declaradas ¿Es posible? Escriba un método `main()` e intente invocar a ambos métodos sin crear ningún objeto. ¿Es posible? ¿Por qué?

```
package practico15;

public class ComoAccederAMetodosDeClase {

	static int x;
	int y;

	public static void accediendoDesdeStatic(){
		System.out.println("AccediendoDesdeStatic a X: "+x);
		System.out.println("AccediendoDesdeStatic a Y: "+y);
	}

	public void accediendoDesdeNoStatic(){
		System.out.println("AccediendoDesdeStatic a X: "+x);
		System.out.println("AccediendoDesdeStatic a Y: "+y);
	}

	public static void main(String[] args) {
		accediendoDesdeStatic();
		accediendoDesdeNoStatic();
	}

}
```

Las líneas marcadas en rojo resultan en error. Desde un método estático NO es posible acceder a atributos no estáticos. Por dicho motivo desde "accediendoDesdeStatic" no es posible acceder al valor de y. El segundo error ocurre debido a que no es posible acceder a un método de instancia sin haber definido una instancia, el compilador no sabe sobre que instancia invocar dicho método. Por el contrario, los métodos estáticos (como "accediendoDesdeStatic"), como son métodos de clase no se encuentran asociados a ninguna instancia, es por ello que pueden ser accedidos sin necesidad de crear una instancia.

   b. Cree un paquete llamado **ej1** y codifique en él una clase llamada **Temperatura** con dos métodos de clase: **calcularCelsius(double temp)** y **calcularFarenheit(double temp).** El primero recibe como parámetro una temperatura en grados Fahrenheit, la convierte a grados Celsius y devuelve el valor convertido. El segundo recibe una temperatura en grados Celsius, la convierte a grados Fahrenheit y retorna el nuevo valor.

Pruebe la clase **Temperatura** escribiendo la clase **TestTemperatura** que en su método main() verifica que sus métodos realizan la conversión correctamente. Para ello, invoque al método **calcularCelsius()** con el valor 72, ¿Qué valor se obtiene como respuesta?. Invoque al método **calcularFarenheirt()** con el valor 30,¿Qué valor se obtiene?

Sugerencia: para convertir valores **Farenheit** a valores **Celsius**:

- reste 32 al valor Fahrenheit, multiplique por 5 y divida entre 9.
- asegúrese que el resultado obtenido es correcto (por ejemplo realizando los cálculos en forma manual o con una máquina calculadora).

```
package ej1;

public class Temperatura {

    public static double calcularCelsius(double temp){
       return (temp-32)*5/9.0;
    }

    public static double calcularFarenheit(double temp){
       return (temp*9/5.0)+32;
    }

    public static void main(String[] args) {
        System.out.println("Farenheit: 72 "+"Celsius: "+calcularCelsius(72));
        System.out.println("Celsius: 30 "+"Farenheit: "+calcularFarenheit(30));
    }

}
```

```
Farenheit: 72 Celsius: 22.22222222222222
Celsius: 30 Farenheit: 86.0
```

c. Escriba una clase llamada **ObjetosFelices** que declare un método de clase llamado **instancias()** que devuelva la cantidad de objetos que se crean. ¿Cómo mantiene la cantidad de objetos creados?

```
public class ObjetosFelices {

    static int cantObjetos = 0;
    public ObjetosFelices(){
       cantObjetos++;
    }

    public static void main(String[] args) {
       ObjetosFelices o1 = new ObjetosFelices();
       ObjetosFelices o2 = new ObjetosFelices();
       ObjetosFelices o3 = new ObjetosFelices();
       ObjetosFelices o4 = new ObjetosFelices();

       System.out.println("Cantidad de Objetos creados: +ObjetosFelices.cantObjetos);
    }
}
```

La cantidad de objetos creados puede mantenerse declarando un atributo estático, al cual se le incrementa el valor en 1 cada vez que se invoca el constructor.

d. Codifique las siguientes clases:

```java
public class Circulo{

    public static final double PI = 3.14159;
    protected double r;

    public Circulo(double r){
        this.r = r;
    }

    public static double radianesAgrados(double rads){
        return (rads * 180 / PI);
    }

    public double area() {
             return (PI * r * r);
    }

    public double circunferencia(){
        return (2 * PI * r);
    }

}

public class CirculoPlano  extends  Circulo {
    private double cx, cy;
    public static final double PI = 3;

    public CirculoPlano(double r, double x, double y){
        super(r);
        this.cx = x;
        this.cy = y;
    }

    public boolean pertenece(double x, double y){
        double dx, dy;
        dx = x - cx;
        dy = y - cy;
        double distancia = Math.sqrt(dx*dx + dy*dy);
        return (distancia < r);
    }

}
```

Use la siguiente clase de prueba y responda las preguntas que están a continuación:

```java
public class TestOcultamiento {

   public static void main(String args[]){
       CirculoPlano cp=new CirculoPlano(10, 20, 10);
       System.out.println("Area : " + cp.area());
       System.out.println("Circunferencia: " + cp.circunferencia());
   }
}
```

i. ¿A qué PI hacen referencia los métodos **area**() y circunferencia() de `TestOcultamiento`?

Hacen referencia a los métodos de Circulo. Nótese que CirculoPlano no reimplementa los métodos de Circulo. En consecuencia, la única implementación de los métodos es provista por Circulo.

ii. Agregue un método estático llamado **descripción()** a las clases Circulo y `CirculoPlano` que retorne un `String` con una descripción general.

En la clase Circulo:

```
public static String descripcion(){
	return "Este es un círculo";
}
```

En la clase CirculoPlano:

```
public static String descripcion(){
	return "Este es un círculo plano";
}
```

iii. Luego, agréguele a la clase **TestOcultamiento** las siguientes líneas de código:

```
Circulo c= cp;
System.out.println("Descripción : " + c.descripcion());
```

¿Es posible sobrescribir métodos de clase?

Descripción : Este es un círculo

Los métodos de clase no pueden sobreescribirse.

```
System.out.println("Descripción : " + Circulo.descripcion());
System.out.println("Descripción : " + CirculoPlano.descripcion());
```

```
Descripción : Este es un círculo
Descripción : Este es un círculo plano
```