



2020

Profesores:

Martín Domínguez
Barragán, Pablo.

Diseño e Implementación de Estructuras de Datos

Guía 04 - Programar en JAVA

Contenido

Guía 04 - Programar en JAVA	2
Ejercicio 01	2
Clase Punto	3
Clase Recta	3
Clase App	3
Subir el proyecto actualizado al GitHub	4
Ejercicio 02	5
Creación de Clases 1	6
Crear clase de prueba	7
Ejercicio 03	7
Pasos para la creación de la clase	8
Clase de prueba	8

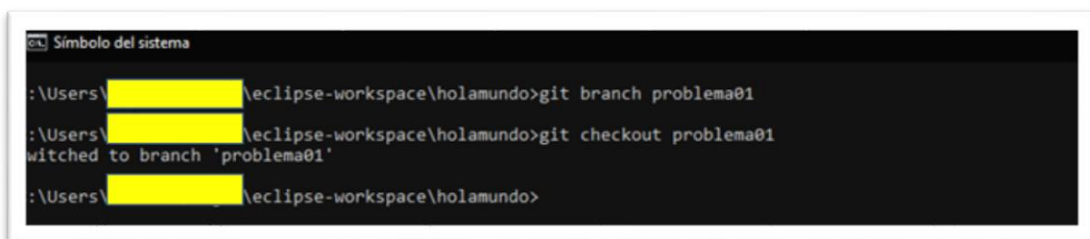
Guía 04 - Programar en JAVA

Problema 01

Antes de todo esto, vamos a crear una nueva rama en el git, para poder trabajar con nuestro proyecto de forma independiente a cada uno de los problemas:

git branch problema01

git checkout problema01



```
Símbolo del sistema
C:\Users\[usuario]\eclipse-workspace\holamundo>git branch problema01
C:\Users\[usuario]\eclipse-workspace\holamundo>git checkout problema01
Switched to branch 'problema01'
C:\Users\[usuario]\eclipse-workspace\holamundo>
```

O mediante una simple línea de comando:

git checkout -b problema01

Se desea modelar ecuaciones lineales de una variable independiente (rectas). Una recta se define por dos puntos. Cada punto es un par de la forma (x,y) donde los valores del par son del tipo primitivo float.

Crear el paquete "died.lab01.problema01" Dentro de dicho paquete se crearán 2 clases Recta y Punto Se le solicita que defina la clase Punto y la clase Recta

Clase Punto

- Constructor:
 - Punto(float x,float y) Crea una instancia de un punto con las coordenadas argumentos.
- Métodos (puede crear los métodos get y set, con el IDE automáticamente, seleccionando el menu "Source" --> "Generate Getters and Setters Method...")
 - getX() : float Retornar el valor de abcisa.
 - getY() : float Retornar el valor de ordenada.
 - setX(float nuevoValor) : void Establece un nuevo valor de abcisa.
 - setY(float nuevoValor) : void Establece un nuevo valor de ordenada.
- equals(Object otroPunto) : boolean Retorna true si el argumento es instancia de la clase Punto y los puntos tienen las mismas coordenadas.

Clase Recta

- Constructores
 - Recta(Punto p1, Punto p2): Crea una instancia de Recta con los puntos argumentos.
 - Recta(): Crea una instancia de la recta identidad ($y(x) = x$)
- Métodos
 - pendiente() : float Retornar la pendiente de la recta ($m = (y1 - y0) / (x1 - x0)$)
 - paralelas(Recta otraRecta) : boolean Recta true en caso que la recta argumento sea paralela a la recta receptora del mensaje, false caso contrario. (Dos rectas son paralelas si tienen la misma pendiente.)
 - equals(Object otraRecta) : boolean Retorna true si el argumento es instancia de la clase Recta y dichos objetos representan la misma recta. (Pueden estar definidas por distintos puntos y representar la misma recta; para verificar este caso se puede tomar un punto de cada recta, para determinar una tercera recta: si la pendiente de esta coincide con la pendiente de las dos anteriores, se trata de la misma recta.)

Clase App

Crear la clase App para realizar una aplicación de consola. Esta clase App tendrá el método main con la siguiente secuencia de instrucciones:

1. Crear el punto P1(1,1)
2. Crear el punto P2(2,2).
3. Crear la recta R1 con los puntos P1 y P2.
4. Mostrar por pantalla la pendiente de la recta R1.
5. Crear el punto P3(3,3).

6. Crear la recta R2 con los puntos P2 y P3.
7. Mostrar por pantalla (true o false) si R1 y R2 representan la misma recta.

Subir el proyecto actualizado al GitHub

Debe estar en la rama **problema01** antes de subirlo:

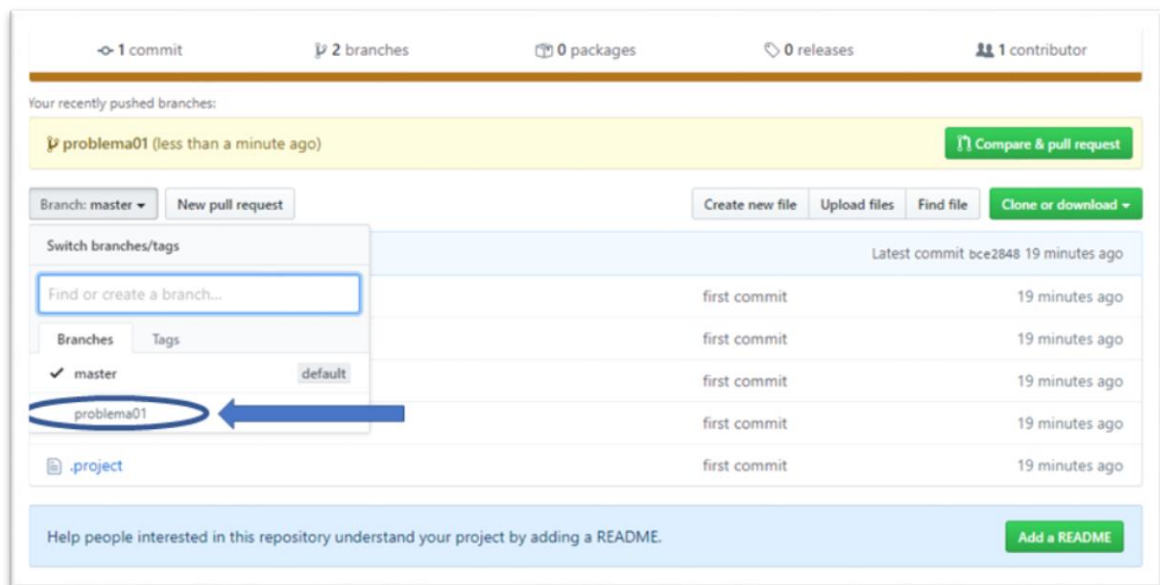
git checkout problema01

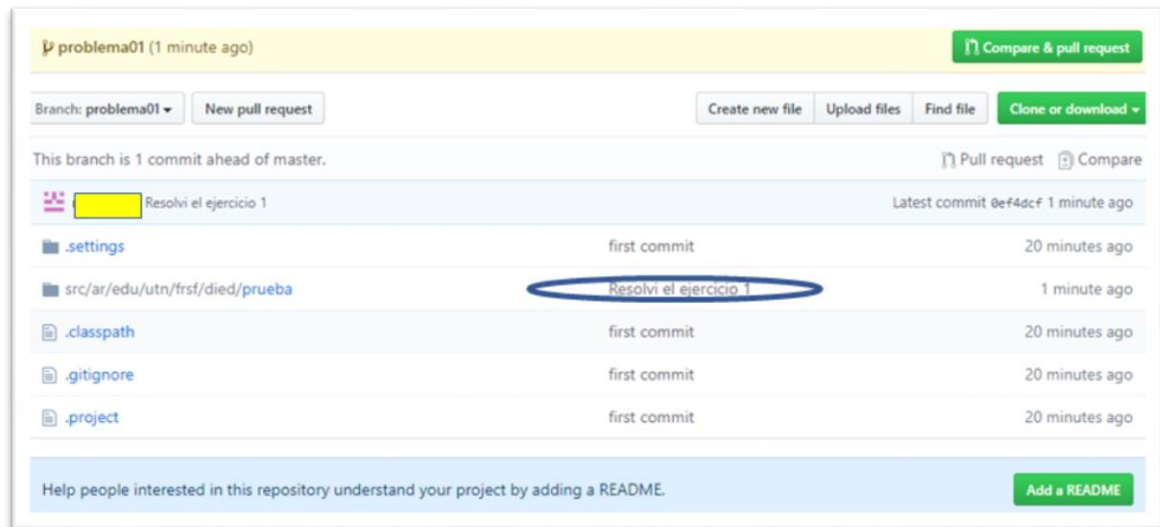
git add .

git commit -m "Resolvi el ejercicio 1"

git push origin problema01

[Observación] En el repositorio no encontrará nada actualizado, ya que está ubicado en la rama master por defecto. Para cambiar de rama y ver los cambios, cambie de rama:





La rama “problema01” es una marca que permite guardar por separado todas las modificaciones que hicimos en el problema 01. Cuando realizamos un push guardamos toda esa información en el servidor remoto. El siguiente paso es copiar la solución del problema 01, a la rama master que es la que tendrá todas las versiones integradas de todos los problemas de las guías. Para ello, realizar lo siguiente

git checkout master

git merge problema01

git push origin master

De esta manera con el comando “merge” le indicamos a git que nos copie a la rama actual (“master”) todo los archivos nuevos o actualizados que están en la rama problema01.

Luego con “push origin master” enviamos esta actualización a la rama master y en dicha rama vamos a ir sumando todo el código fuente de los problemas, **una vez que los tenemos resueltos. (verificar en github que los archivos también están en la rama master)**

Cuando trabajamos parcialmente en un problema, trabajaremos en la rama del problema, por lo tanto podremos ir actualizando en el sistema git, todas las versiones de la solución parcial al mismo, hasta que finalicemos la resolución completa. Una vez que el problema fue resuelto lo integramos a la rama master, con el comando merge.

Problema 02

[Importante] Regresar a la rama master y, a partir de ahí, crear una nueva rama en el git: “problema02”.

git checkout master

git checkout – b problema02

Se desea modelar la clase Temperatura que permita registrar los grados de una temperatura y la escala (F° o C°), y que permita realizar conversiones entre dichos valores.

Para la conversión usar la siguiente formula:

- De F° a C° $\Rightarrow C^{\circ} = (F^{\circ} - 32) * \frac{5}{9}$
- De C° a F° $\Rightarrow ^{\circ}F = C^{\circ} * \frac{9}{5} + 32$

Creación de Clases 1.

Crear el paquete "died.lab01.problema02".

1. Dentro del paquete crear el tipo de datos enum, de nombre Escala, que tenga dos valores "FAHRENHEIT" y "CELCIUS".
2. Crear la clase Temperatura
3. Atributos de la clase Temperatura (defina el modificador de visibilidad que considere apropiado)
 - a. Double grados;
 - b. Escala escala;
4. Constructores
 - a. El constructor por defecto que inicializa en 0 la temperatura y en CELCIUS la escala.
 - b. El constructor con dos argumentos, la temperatura y la escala que asignará a la Temperatura un valor inicial y una escala predefinida.
5. Métodos
 - a. Crear el método toString() que retorna un string representando la temperatura en formato . Por ejemplo toString() retorna "14 C°" para representar la temperatura de 14 grados celcius
 - b. Crear el método "Double asCelcius()" que retorna la temperatura actual en grados Celcius.
 - c. Crear el método "Double asFahrenheit()" que retorna la temperatura actual en grados Fahrenheit.
 - d. Crear un método aumentar(Temperatura temperatura) y un método disminuir(Temperatura). Estos métodos aumentan o disminuyen la temperatura según el valor del parámetro recibido.
 - i. Verificar que el valor sea siempre mayor que 0.0.
 - ii. Tener en cuenta que si la temperatura recibida como parámetro está en una escala, y el objeto que ejecuta el método está en otra escala hay que realizar la conversión o invocar al método que realiza la conversión

Crear clase de prueba

Crear una clase "App" que contenga un método main con la siguiente secuencia

```
Temperatura dia1 = new Temperatura(30.0, Escala.CELCIUS);
Temperatura dia2 = new Temperatura(55.0, Escala.FAHRENHEIT);
System.out.println("T1 en C°: "+dia1.asCelcius());
System.out.println("T1 en F°: "+dia1.asFahrenheit());
System.out.println("T2 en C°: "+dia2.asCelcius());
System.out.println("T2 en F°: "+dia2.asFahrenheit());
dia2.aumentar(dia1);
System.out.println("T1+T2 en C°: "+dia2.asCelcius());
System.out.println("T1+T2 en F°: "+dia2.asFahrenheit());
dia1.disminuir(new Temperatura(10.0, Escala.CELCIUS));
System.out.println("T1 en C°: "+dia1.asCelcius());
System.out.println("T1 en F°: "+dia1.asFahrenheit());
```

La salida esperada por consola para la secuencia anterior será:

```
T1 en C°: 30.0
T1 en F°: 86.0
T2 en C°: 12.777777777777779
T2 en F°: 55.0
T1+T2 en C°: 60.555555555555556
T1+T2 en F°: 141.0
T1 en C°: 20.0
T1 en F°: 68.0
```

Una vez realizado el problema 02, realizar un push con la rama problema02

Finalmente integrar la solución a la rama master.

git checkout master

git merge problema02

git push origin master

Problema 03

[Importante] NO REGRESAR A LA RAMA MASTER, desde la rama "problema02", crear una nueva rama en el git "problema03"

Se desean registrar las temperaturas históricas para distintas ciudades de las que se obtuvieron muestras. Para ello crearemos la clase Registro, que permite registrar hasta 30 temperaturas históricas

Pasos para la creación de la clase

1. Crear el paquete "died.lab01.problema03"
2. Crear la clase Registro.
3. Atributos (mínimos):
 - a. private String ciudad: ciudad a la que pertenece el registro de temperaturas
 - b. private Temperatura[] historico: listado de las temperaturas históricas
 - c. Podrá crear otros atributos privados que considere necesarios, como por ejemplo el índice actual en el que se encuentra el último elemento agregado al arreglo, o la capacidad disponible, o la cantidad de elementos insertados, o una constante que indique la capacidad máxima del arreglo
 - d. Importante: deberá importar las clases que necesite del paquete "ar.edu.utn.frsf.died.guia02.ejercicio02"
4. Constructores
 - a. Registro() : asigna como nombre de ciudad "-" y crea el arreglo con la capacidad máxima.
 - b. Registro(String ciudad) : asigna como nombre de ciudad el valor del argumento ciudad y crea el arreglo con la capacidad máxima.
5. Métodos (defina la visibilidad de cada método que considere apropiada)
 - a. String getCiudad() : retorna el nombre de la ciudad
 - b. void imprimir() : imprime por consola la información de la ciudad y cada una de las temperaturas registradas. Ejemplo
TEMPERATURAS REGISTRADAS EN : DENVER
1 24.130797130990256 C°
2 18.65927617973209 C°
 - c. agregar(Temperatura t) : agrega una temperatura al arreglo, si el arreglo no está lleno. Si está lleno, simplemente termina el método sin agregar el elemento.
 - d. MediaAsCelcius() y mediaAsFahrenheit(): retorna la temperatura promedio en C° Y F° respectivamente.
 - e. Temperatura maximo(): busca la temperatura máxima del listado. Deberá implementar este método de manera recursiva. Para ello puede crear los métodos auxiliares que necesite. (Ayuda: este método puede ser utilizado como rutina guía y crear el método recursivo que busca el máximo como un método privado que recibe los argumentos que considere necesarios).

Clase de prueba

Crearemos una clase de prueba App, con un método main que creará el registro histórico de 2 ciudad y agregará en cada ciudad temperaturas aleatorias. Para ello se usará la clase Random, del API de java, que posee un método "nextDouble()", que retorna un valor de tipo Double entre 0.0 y 1.0.

- [https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Random.html#nextDouble\(\)](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Random.html#nextDouble())
- Para simular temperaturas entre 0 °C y 50 °C multiplicaremos el valor random por 50
- Para simular temperaturas entre 20°F y 105 °C, multiplicaremos el valor random por 85 y luego le sumaremos 20.0

```
public static void main(String[] args) {
    Registro r1 = new Registro("DENVER");
    Registro r2 = new Registro("NAIROBI");
    Random generadorAleatorio = new Random();

    for(int i=0;i<10;i++) {
        r1.agregar(new
Temperatura(generadorAleatorio.nextDouble()*50,Escala.CELCIUS));
    }

    for(int i=0;i<10;i++) {
        r2.agregar(new
Temperatura(generadorAleatorio.nextDouble()*105,Escala.FARENHEIT));
    }
    r1.imprimir();
    System.out.println("Promedio en "+r1.getCiudad()+" :"+r1.mediaAsCelcius()
+" C°");
    System.out.println("Maximo en "+r1.getCiudad()+" :"+r1.maximo().asCelcius()
+" C°");
    r2.imprimir();
    System.out.println("Promedio en "+r2.getCiudad()+"
:"+r2.mediaAsFahrenheit() +" F°");
    System.out.println("Maximo en "+r2.getCiudad()+"
:"+r2.maximo().asFahrenheit() +"F°");
}
```

Una vez realizado el problema 03, realizar un push con la rama problema03

git checkout master

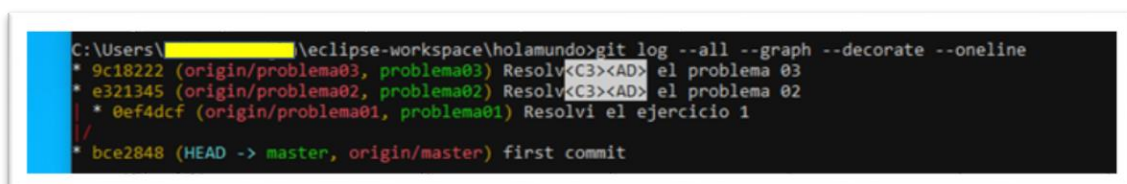
*git merge **problema03***

git push origin master

ANEXO: Visualización del árbol de las ramas creadas:

Ejecutar desde el Símbolo del sistema (cmd):

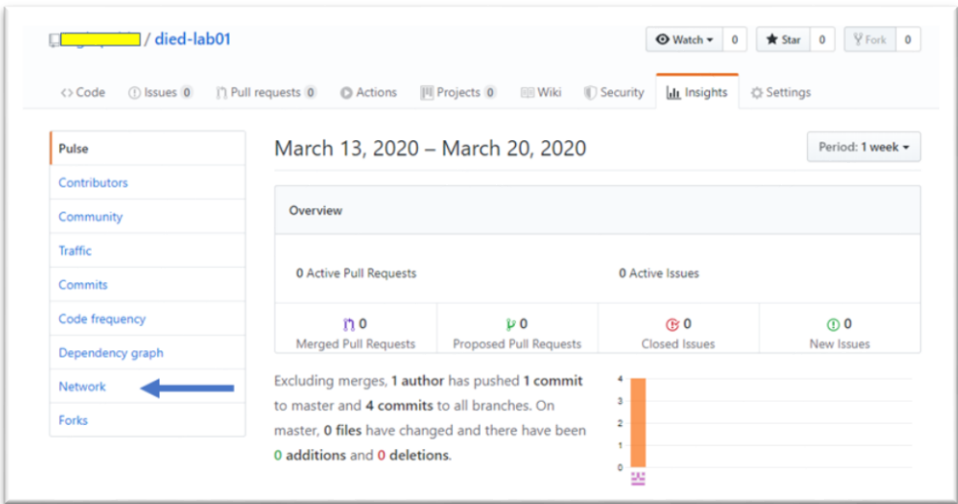
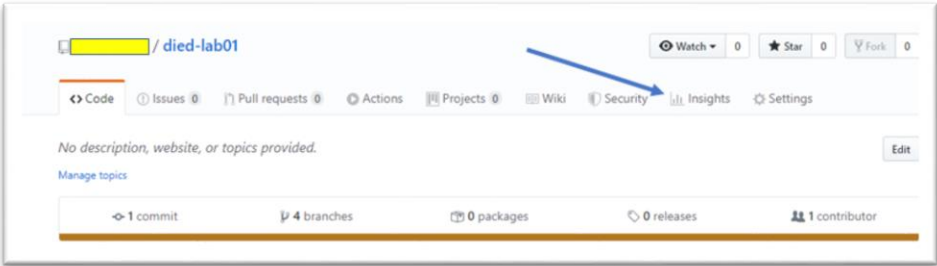
git log --all --graph --decorate --oneline



```
C:\Users\...>git log --all --graph --decorate --oneline
* 9c18222 (origin/problema03, problema03) Resolv<C3><AD> el problema 03
* e321345 (origin/problema02, problema02) Resolv<C3><AD> el problema 02
* 0ef4dcf (origin/problema01, problema01) Resolvi el ejercicio 1
* bce2848 (HEAD -> master, origin/master) first commit
```

```
* 9c18222 (origin/problema03, problema03) R
* e321345 (origin/problema02, problema02) R
| * 0ef4dcf (origin/problema01, problema01)
|/
* bce2848 (HEAD -> master, origin/master) f
```

O pueden verlo desde la página de GitHub:



Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.

