SESIÓN 2: Herencia. Clases Abstractas. Interfaces

Objetivos

- Saber definir subclases a partir de superclases mediante herencia.
- Saber usar el polimorfismo y el enlace dinámico.
- Saber utilizar el modificador **protected**.
- Saber usar genéricos: clases genéricas, interfaces, métodos.
- Diseñar y usar clases abstractas.
- Saber usar interfaces de las API.

Material proporcionado

- La clase **Fraccion.**
- Los test de pruebas de los distintos ejercicios propuestos, **TestEstudianteEmpleado**, **TestIteradorPrimos , TestBusqueda** y **TestOctogono.**

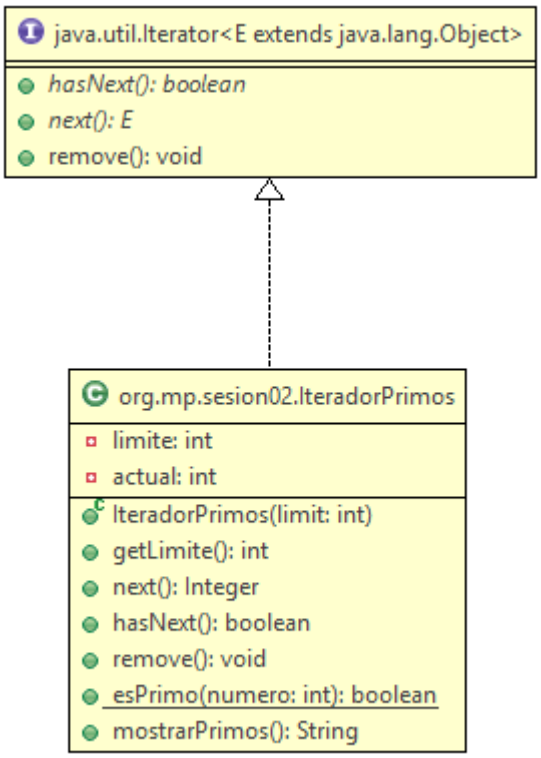
Requisitos

- Seguir el esquema de nombrado de paquetes, es decir: **org.mp.sesion02**. En ese paquete se crearán todos los programas que se proponen en la sesión dándoles un nombre alusivo a lo que realiza el programa y que se indica en cada ejercicio entre paréntesis y en negrita.
- Documentar todas las clases.
- Generar de todas las clases los diagramas **UML**.
- Todas las clases deberán superar con éxito los test de pruebas.
- Al final de la sesión, el alumno deberá cargar el trabajo realizado a su repositorio indicando la clave correspondiente a la sesión.
- Las sesiones se han diseñado para cubrir una semana de trabajo.

Ejercicios propuestos

1. Diseñar una clase llamada **Persona** y dos subclases de ella llamadas **Estudiante** y **Empleado**. Diseñar también las clases **Administrativo** y **Profesor** como subclases de **Empleado**. Una persona tiene nombre, dirección, teléfono y correo electrónico. Un estudiante tiene un nivel que es de grado, máster o doctorado. Un empleado tiene un despacho, salario y fecha de contratación. Para la fecha de contratación, utiliza la clase **GregorianCalendar**. Un profesor tiene un horario de tutorías y una categoría. Un administrativo tiene asignada una unidad de destino.
   Sobre-escribe el método **toString( )** en cada clase.

2. Define una clase llamada **IteradorPrimos** para iterar números primos. El constructor recibe un argumento que especifica el límite del máximo número primo. Por ejemplo, IteradorPrimos(100) crea un objeto que itera números primos menores o iguales que 100.
   El diagrama de clases es el que sigue.



Para la implementación del método **mostrarPrimos**, debes de usar los métodos **next** y **hasNext** aun siendo posible otras soluciones.

Trabajo autónomo

3. A partir de la clase **Fraccion** que se proporciona, haz una clase genérica.
Diseña la clase **Busqueda**. Esta clase debe tener un método estático genérico **BusquedaBinaria**.

4. Un octógono es una figura plana de ocho lados y ocho vértices. Supongamos que se trata de un octógono regular, es decir, con los lados y ángulos iguales. Se muestra en la figura.

Se puede calcular el perímetro y el área a partir de las fórmulas:

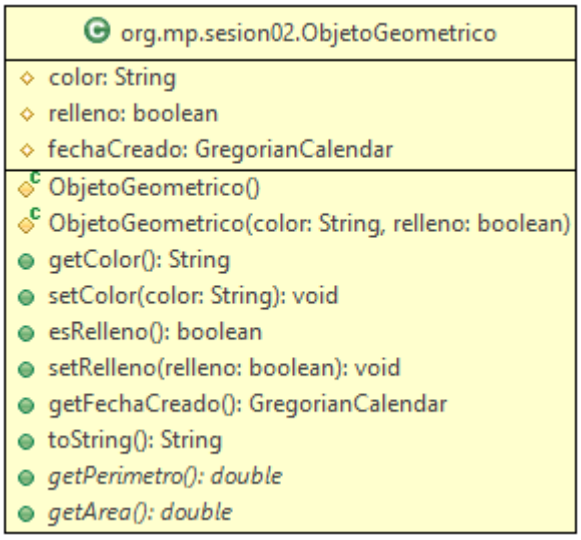
$$perímetro = 8 * lado$$
$$área = \left(2 + {}^{4}/_{\sqrt{2}}\right) * lado * lado$$

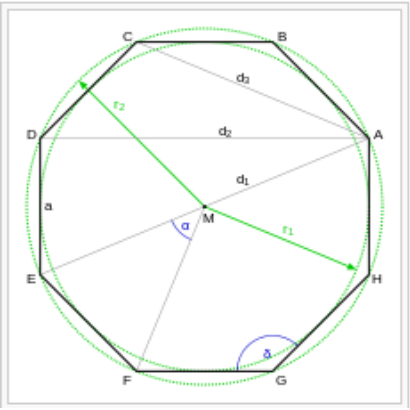
Diseña la clase **Octogono** que herede de la clase **ObjetoGeometrico** y que implemente las interfaces **Comparable**, **Cloneable** y **DeColor**.

De la clase **ObjetoGeometrico** se proporciona el diagrama UML.

Las interfaces **Comparable** y **Cloneable** están descritas en el tema 2.

La interface **DeColor** tiene un único método llamado **comoColorea**r. Este método devuelve un **String**.


org.mp.sesion02.ObjetoGeometrico
color: String
relleno: boolean
fechaCreado: GregorianCalendar
ObjetoGeometrico()
ObjetoGeometrico(color: String, relleno: boolean)
getColor(): String
setColor(color: String): void
esRelleno(): boolean
setRelleno(relleno: boolean): void
getFechaCreado(): GregorianCalendar
toString(): String
getPerimetro(): double
getArea(): double


C
B
D
A
E
H
F
G
M
a
α
δ