



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

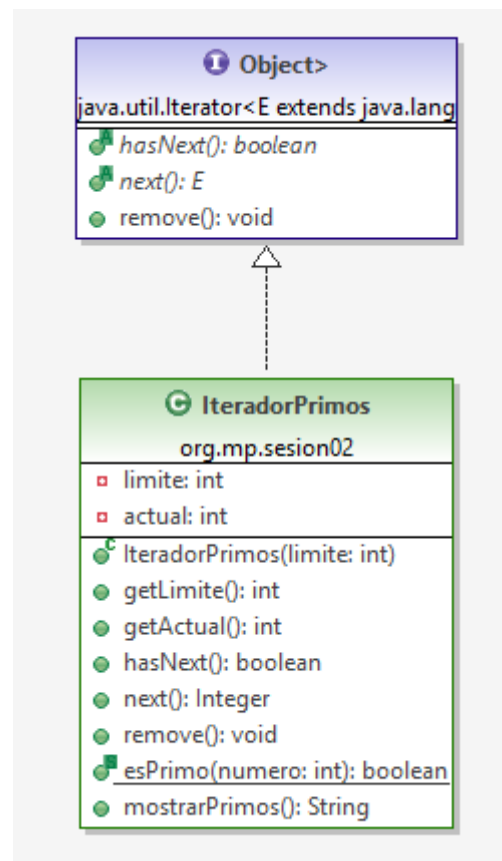
- Los test de pruebas de los distintos ejercicios propuestos, **TestEstudianteEmpleado**, **TestIteradorPrimos**, **TestBusqueda** y **TestOctogono**, **TestMatrizInteger**, **TestMatrizFraccion**

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.

3. Diseña la clase **Busqueda**. Esta clase debe tener un método estático genérico **busquedaBinaria**. Implementa la clase **Fraccion** para poder ser utilizada en este método.
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:

$$\text{perímetro} = 8 * \text{lado}$$

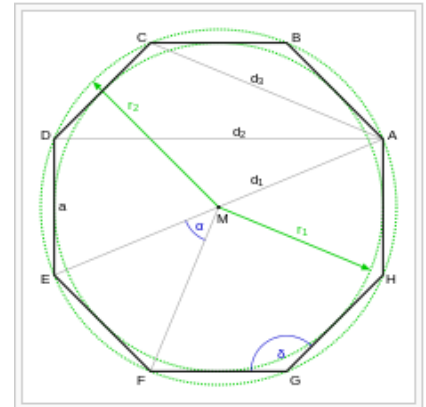
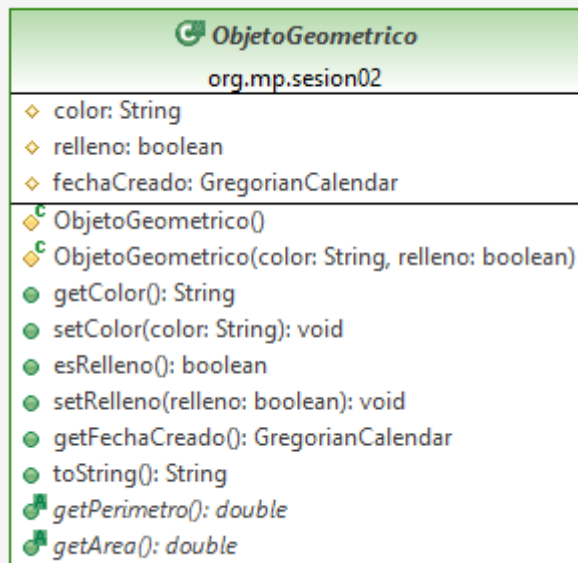
$$\text{área} = \left(2 + \frac{4}{\sqrt{2}}\right) * \text{lado} * \text{lado}$$

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

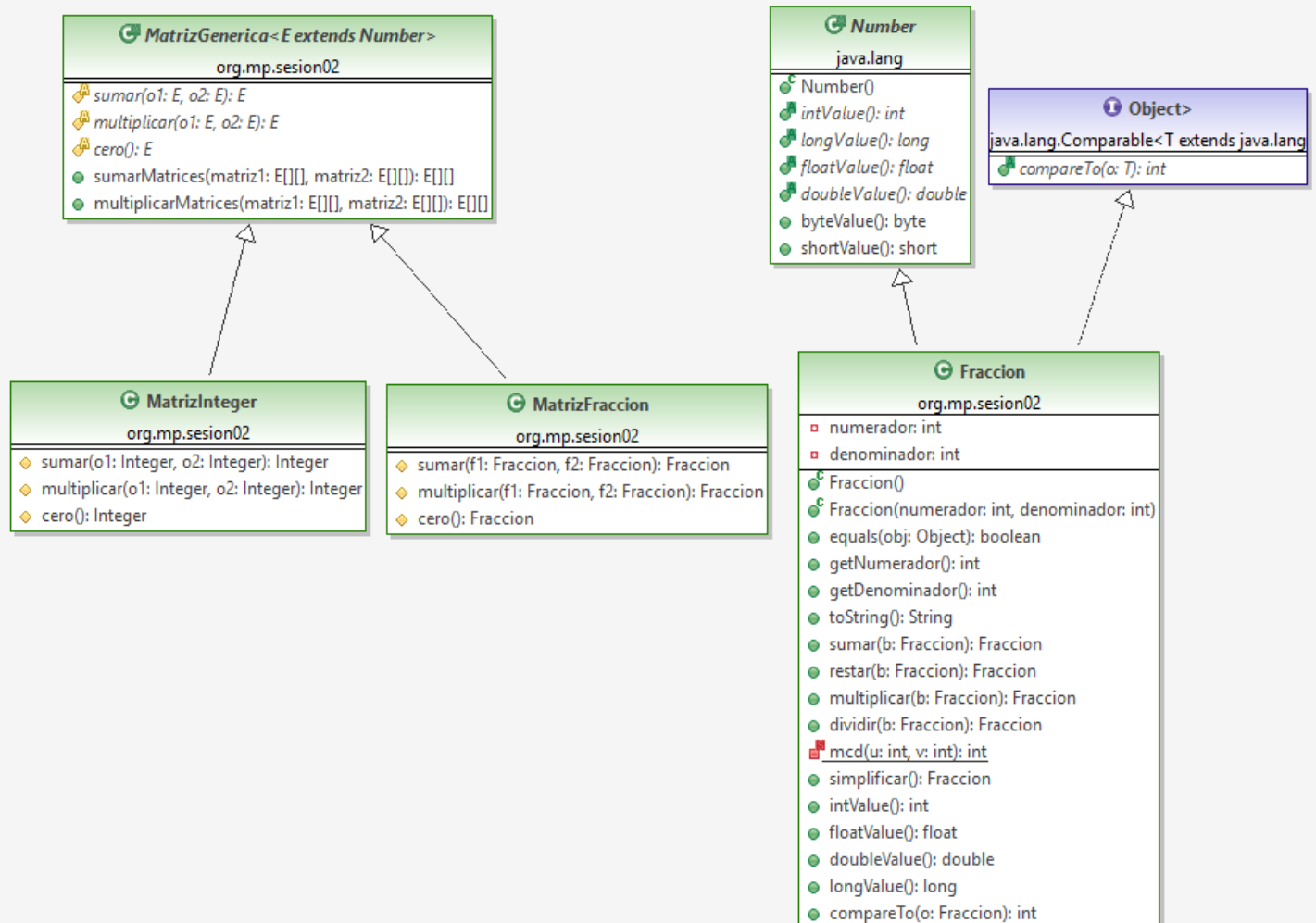
De la clase **ObjetoGeometrico** se proporciona el diagrama UML.

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

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



5. Implementa las clases que se muestran en el diagrama y que permitirán hacer operaciones con matrices utilizando tipos genéricos.



La clase **MatrizGenerica** tiene los métodos **sumarMatrices** y **multiplicarMatrices** que permitirán sumar y multiplicar dos matrices de tipo genérico $E[][]$. Los métodos sumar, multiplicar y cero son abstractos porque su implementación va a depender del tipo específico de los elementos del array.