OBJETIVOS

- Diseño de tipos: herencia.
- Test de tipos.

DEFINICIÓN E IMPLEMENTACIÓN MEDIANTE HERENCIA DE LAS ENTIDADES ALUMNO, BECARIO, PROFESOR Y DESPACHO

Defina los tipos Alumno y Profesor, que heredan del tipo Persona; el tipo Becario, que hereda del tipo Alumno, y el tipo Despacho, que hereda del tipo Espacio. Para ello se le proporciona una plantilla con las propiedades de cada uno de estos tipos.

En los constructores, utilice como orden de los parámetros el mismo orden que se sigue en el boletín al definir las propiedades del tipo.

Nota: algunas propiedades no se van a implementar en este momento. Añada dichas propiedades con comentarios y coloque un *TODO* en los lugares adecuados para volver sobre ellos más adelante.

Tipo **Alumno**: extiende a Persona y añade las siguientes propiedades, restricciones y operaciones.

Propiedades:

- **Asignaturas**. Propiedad básica, de tipo `Set<Asignatura>`. Consultable. Sin restricciones. Representa las asignaturas en las que está matriculado un alumno en cada momento.
- **Curso**. Propiedad derivada, de tipo `Integer`. Consultable. El curso de un alumno es el mayor de los cursos de las asignaturas en que está matriculado, o 0 si no está matriculado en ninguna asignatura. **Esta propiedad no se implementa de momento**.

Restricciones:

- El email de un alumno no puede ser la cadena vacía y debe acabar en "@alum.us.es". Si no se cumple esta restricción, lance la excepción `ExcepcionAlumnoNoValido`.

Operaciones:

- `void matriculaAsignatura(Asignatura asig)`. Añade la asignatura `asig` al alumno. Si el alumno ya está matriculado en esa asignatura, se lanzará la excepción `ExcepcionAlumnoOperacionNoPermitida`.
- `void eliminaAsignatura(Asignatura asig)`. Elimina la asignatura `asig` del alumno. Si el alumno no está matriculado en esa asignatura, se lanzará la excepción `ExcepcionAlumnoOperacionNoPermitida`.
- `Boolean estaMatriculadoEn(Asignatura asig)`. Devuelve `true` si el alumno está matriculado en la asignatura `asig`, y `false` en caso contrario.
- Constructor: recibe como parámetros todas las propiedades básicas de una persona. Las asignaturas se inicializan con un conjunto vacío.
- Criterio de igualdad: el mismo que las personas.
- Criterio de orden natural: el mismo que las personas.
- Representación como cadena: el número del curso seguido del carácter 'º', entre paréntesis, seguido de un espacio y la representación como cadena de la persona. Por ejemplo, "(1º)

28864657W – García Vaquero, Pascual – 15/09/1998". Nota: hasta que no se implemente la propiedad curso, use un interrogante (?) para representar el curso del alumno; coloque un *TODO* para añadirlo más adelante.

Tipo **Becario**: extiende a Alumno y añade las siguientes propiedades, restricciones y operaciones:

Propiedades:
- **Beca**. Propiedad básica, de tipo Beca. Consultable. Sin restricciones.
- **Fecha de comienzo**. Propiedad básica, de tipo `LocalDate`. Consultable y modificable.
- **Fecha de fin**. Propiedad derivada, de tipo `LocalDate`. Consultable. Se calcula sumando a la fecha de comienzo el número de meses que dura la beca.

Restricciones:
- La fecha de comienzo debe ser posterior a la fecha actual (en el momento de crear el objeto o cambiar el valor de la propiedad). Si no se cumple esta restricción, lance la excepción `ExcepcionBecarioNoValido`.
- El email de un becario no se puede modificar. Si se utiliza la operación `setEmail()` heredada del tipo Alumno, se debe lanzar la excepción `UnsupportedOperationException`.

Operaciones:
- Constructor 1: recibe los mismos parámetros que el constructor del tipo Alumno, más un objeto de tipo Beca y la fecha de comienzo de la misma.
- Constructor 2: recibe los mismos parámetros que el constructor del tipo Alumno, más todos los parámetros necesarios para especificar completamente los datos de una beca y la fecha de comienzo de la beca. El constructor debe crear el objeto de tipo Beca correspondiente.
- Criterio de igualdad: el mismo que los alumnos.
- Criterio de orden natural: el mismo que los alumnos.
- Representación como cadena: la misma que los alumnos, seguida de un espacio y la representación como cadena de la beca. Por ejemplo: "(1º) 28864657W – García Vaquero, Pascual – 15/09/1998 [ABB2024, movilidad]".

Tipo **Profesor**: extiende a Persona y añade las siguientes propiedades, restricciones y operaciones:

Propiedades:
- **Categoría**. Propiedad básica, de tipo Categoria, que es un tipo enumerado con los siguientes valores: CATEDRATICO, TITULAR, CONTRATADO_DOCTOR, COLABORADOR, AYUDANTE_DOCTOR, AYUDANTE e INTERINO. Consultable y modificable.
- **Tutorías**. Propiedad básica, de tipo `SortedSet<Tutoria>`. Consultable. Sin restricciones.

Restricciones:
- Un profesor debe ser una persona mayor de edad (tener al menos 18 años). Si no se cumple esta restricción, lance la excepción `ExcepcionProfesorNoValido`.

Operaciones:
- `void nuevaTutoria(LocalTime horaComienzo, Integer duracionMinutos, DayOfWeek dia)`. Añade una nueva tutoría; si ya existía una tutoría igual (mismo día de la semana y misma hora de comienzo), la operación no tiene efecto.

- `void borraTutoria(LocalTime horaComienzo, DayOfWeek dia)`. Elimina la tutoría con el día y hora de comienzo indicados; si el profesor no tenía esa tutoría, la operación no tiene efecto.
- `void borraTutorias()`. Elimina todas las tutorías del profesor.
- Constructor: recibe como parámetros todas las propiedades básicas de una persona, más la categoría. Las tutorías se inicializan con un conjunto vacío.
- Criterio de igualdad: el mismo que las personas.
- Criterio de orden natural: el mismo que las personas.
- Representación como cadena: la misma que las personas, seguida de un espacio y la categoría profesional entre paréntesis. Por ejemplo: "28200400P – Martín Oviedo, María – 21/05/1985 (TITULAR)".

Tipo **Despacho**: extiende a Espacio y añade las siguientes propiedades, restricciones y operaciones:

Propiedades:
- **Profesores**. Propiedad básica, de tipo `Set<Profesor>`. Consultable y modificable. Representa los profesores que ocupan el despacho.

Restricciones:
- El número de profesores que ocupan un despacho no puede superar la capacidad del mismo. Si no se cumple esta restricción, lance la excepción `ExcepcionDespachoNoValido`. Tenga en cuenta que esta restricción no sólo afecta a la propiedad profesores, sino también a la propiedad capacidad heredada del tipo Espacio.
- El tipo de espacio asignado a un despacho siempre tiene que ser el correspondiente a 'otro tipo'. Si se utiliza la operación `setTipo()` heredada del tipo Espacio, se debe lanzar la excepción `UnsupportedOperationException`.

Operaciones:
- Constructor 1: recibe los mismos parámetros que el constructor del tipo Espacio salvo el tipo, que se inicializará con el valor 'OTRO'. Además, recibirá un conjunto de profesores.
- Constructor 2: recibe los mismos parámetros que el constructor del tipo Espacio salvo el tipo, que se inicializará con el valor 'OTRO'. Además, recibirá un profesor.
- Constructor 3: recibe los mismos parámetros que el constructor del tipo Espacio salvo el tipo, que se inicializará con el valor 'OTRO'. El conjunto de profesores se inicializará con un conjunto vacío.
- Criterio de igualdad: el mismo que los espacios.
- Criterio de orden natural: el mismo que los espacios.
- Representación como cadena: la misma que los espacios, seguida de un espacio y los profesores que ocupan el despacho (utilice la representación como cadena de la propiedad 'Profesores'). Por ejemplo: "M2.25 (planta 2) [28200400P – Martín Oviedo, María – 21/05/1985 (TITULAR), 33123210J – Vegarredonda Ordiales, Jorge – 25/11/1990 (CONTRATADO_DOCTOR)]".

TEST DE LOS TIPOS

Implemente un test para cada uno de los nuevos tipos. Realice con ellos las mismas pruebas que ya ha realizado con los tipos definidos en los boletines anteriores.