Nombre y Apellidos:

Instrucciones. La tarea constará de diferentes puntos que crearan un modelo de clases y un programa que desarrollaremos en la unidad siguiente. Las instrucciones de entrega están al final

RAE 6. Escribe programas que manipulen información, seleccionando y utilizando tipos avanzados de datos.

Criterios de evaluación

c) Se han utilizado listas para almacenar y procesar información.

d) Se han utilizado iteradores para recorrer los elementos de las listas. Indicador 57,65

f) Se han creado clases y métodos genéricos. 59, 65

Indicador 56, 65

Detalles de la tarea de esta unidad.

Enunciado.

Tarea Unidad 11

En esta **tarea vamos a aplicar lo aprendido en el unidad 11** de la **API Stream**, sobre **nuevas clases y modelos de datos**. **Continuamos la tarea de la unidad 10**

1.1 Condiciones de Entrega

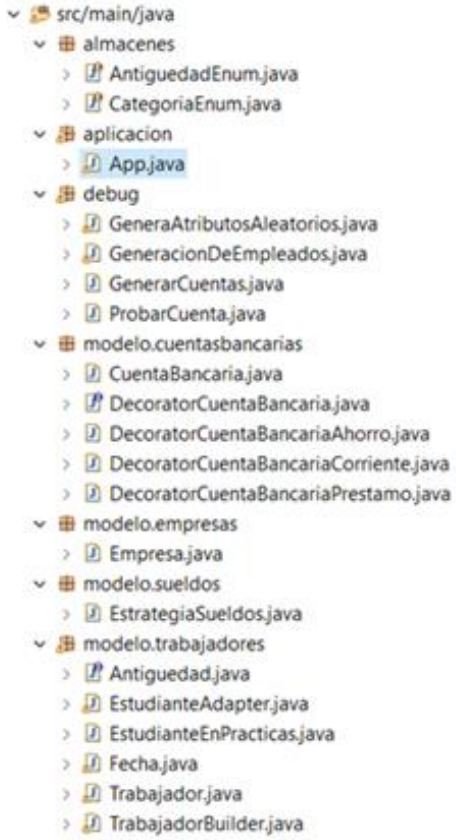
Entregamos un zip nombreApellidosTareaUnidad11 que contendrá:

1. El proyecto **final**. El proyecto completo funcionando.

2. **Pantallazos probando las seis nuevas funcionalidades del menú**, de manera que se vea al menos **un trabajador completo y una empresa completa**. Podéis **añadirme el texto de la ejecución también al pdf**. Lo entregamos en un fichero pdf de nombre TareaTema8NombreApellidos.pdf.

Nombre y Apellidos:

Estructura de proyecto



Migrando a colecciones y refactorizando. 10 puntos

Migrando

Indicadores 56, 57, 59, 65

1. En esta primera parte de la tarea vamos a migrar nuestra aplicación de arrays a colecciones. Debeis cambiar **todos los arrays** que tiene vuestro código **por colecciones**. El **array de trabajadores** lo podéis **cambiar por un ArrayList**, de manera que **generaTrabajadores** devuelva un **List<Trabajador>.** Devolvemos un **interfaz porque programamos para las abstracciones**, pero crear un ArrayList. **(1,25 puntos)**

```
    public static List<Trabajador> generaEmpleados(int numero) {
```

Nombre y Apellidos:

2. Igualmente, en la **clase GeneraAtributosAleatorios cambiad** todos los **arrays por tipos List**. Eliminamos todos los arrays de nuestra aplicación. **(1,25 puntos)**

```
public class GeneraAtributosAleatorios {



    public static final List<String>nombres
```

3. Finalmente, en **Main usaremos listas** para **trabajadores.** (1 punto)

```
public class MainTrabajador {

    public static void main(String[] args) {

    List<Trabajador> trabajadores;
```

4. Modificamos un método y **añadimos dos nuevos métodos** a GeneracionEmpleados, **usando las funciones de agregación de Collectors. (1,25 puntos)**

a. Cambiamos:

```
    public static double mediaSueldo(List<Trabajador> trabajadores)
```

b. Añadimos

Devuelve el trabajador con **Maxímo sueldo**

Nombre y Apellidos:

```
public static Trabajador trabajadorMaximoSueldo(List<Trabajador> trabajadores)
{
```

Devuelve el trabajador con **mínimo sueldo**

```
public static Trabajador trabajadorMinimoSueldo(List<Trabajador> trabajadores)
{
```

Devueve un **HasMap agrupando** por el **nombre de la categoria**

```
public static HashMap<String,List<Trabajador>>
agrupaPorCategoria(List<Trabajador> trabajadores) {
```

Devueve un **HasMap agrupando por el nombre de la antigüedad**

```
public                  static                  HashMap<String,List<Trabajador>>
agrupaPorAntiguedad(List<Trabajador> trabajadores) {
```

No debería daros **problemas el groupingby**, si **alguna vez os diera problemas castear** el HashMap

```
(HashMap<String,List<Trabajador>>)
```

Refactorizando

Refactorizar: La **refactorización** (del inglés refactoring) **es una técnica de la ingeniería de software para reestructurar un código fuente**, **alterando su estructura interna** sin cambiar **su comportamiento externo.**

Muchas de las **técnicas de refactorización se basan en los SOLID principles**, y **los conceptos básicos de diseño orientado a objetos**. De todas maneras, en el sitio web **refactoring guru** tenéis **mucha información al respecto**, por si

Nombre y Apellidos:

alguien esta interesado en ver mas contenidos. Se proporcionarán **las pautas de refactorización en cualquier caso**.

https://refactoring.guru/

1. Lo **primero será quitar plusdeantiguedad, y porcentajecategoria** de **TrabajadorBuilder y Trabajador**. Quitamos las propiedades y sus métodos asociados. **(1 punto)**

```
private double plusAntiguedad = 0;
private double porcentajeCategoria = 0;
```

2. **Modificamos los enumerados** para guardar dentro los **valores de porcentaje y plus según categoría y antigüedad. (1 punto)**

```
public enum CategoriaEnum {
```

```
CATEGORIA_EMPLEADO("EMPLEADO", 0, 0.15),
```

```
public enum AntiguedadEnum {
```

```
ANTIGUEDAD_NOVATO("NOVATO", 0, 150.0),
```

Recordamos **los valores de la tarea 3**,más los nuevos. En este **apartado vamos a volver también a calcular los sueldos** basados en **categoría y antigüedad**.

Nombre y Apellidos:

Sueldo base	607 €
EMPLEADO **ENCARGADO** **DIRECTIVO** **TÉCNICO** FREELANCE ANONIMO ESTUDIANTE	+15% sueldo base +35% sueldo base +60% sueldo base +40% sueldo base +0% sueldo base +0% sueldo base +0% sueldo base
NOVATO **MADURO** **EXPERTO**	+150 € +300 € +600 €

3. **Vamos a modificar sueldoFunction en trabajadorBuilder y trabajador como sigue. Lo hacemos un interfaz Function que reciba un trabajador y devuelva un Double.**
   (1 punto)

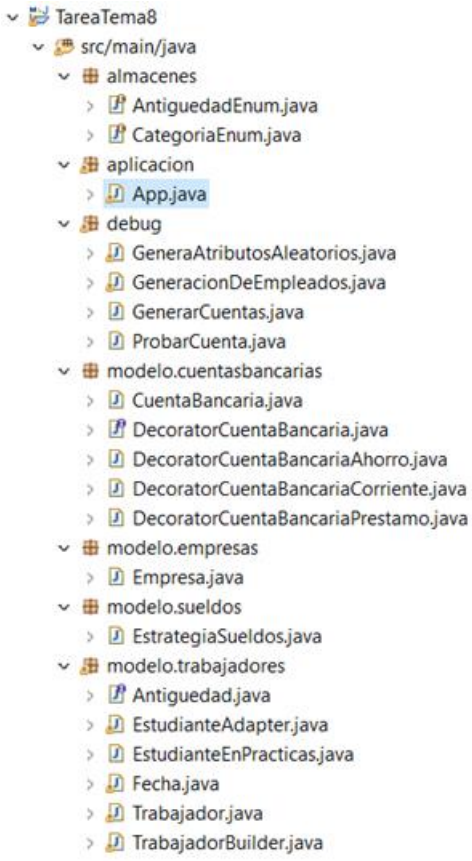
```
private Function <Trabajador, Double> sueldoFunction=
```

4. **Vamos a asegurarnos de crear todos los empleados con el builder incluso anónimo en generaEmpleados. (1 punto)**

5. **Vamos a reorganizar nuestros paquetes y nuestras clases:**
   (1 punto)
   a. Cambiando **el paquete datos**, por **modelo**, y añadiendo un subpaquete para cada Entidad de nuestra aplicación: **Trabajador, CuentaBancaria y Empresa**. Paquete **modelo.trabajadores**, **modelo.cuentasbancarias** y **modelo.empresas**.
   b. Añadimos un paquete **modelo.sueldos** para la nueva clase **EstrategiaSueldos** para **la tarea de la unidad siguiente**.
   c. Añadimos un paquete **modelo.empresas** para la **nueva clase Empresa q**ue introduciremos en la tarea de la unidad siguiente.
   d. Renombramos **la clase MainTrabajador.java** como **App.java** y creamos un **nuevo paquete aplicación** para esta clase

Nombre y Apellidos:

De esta manera que la nueva estructura de proyecto quedará como podéis ver en la siguiente imagen.



Nombre y Apellidos: