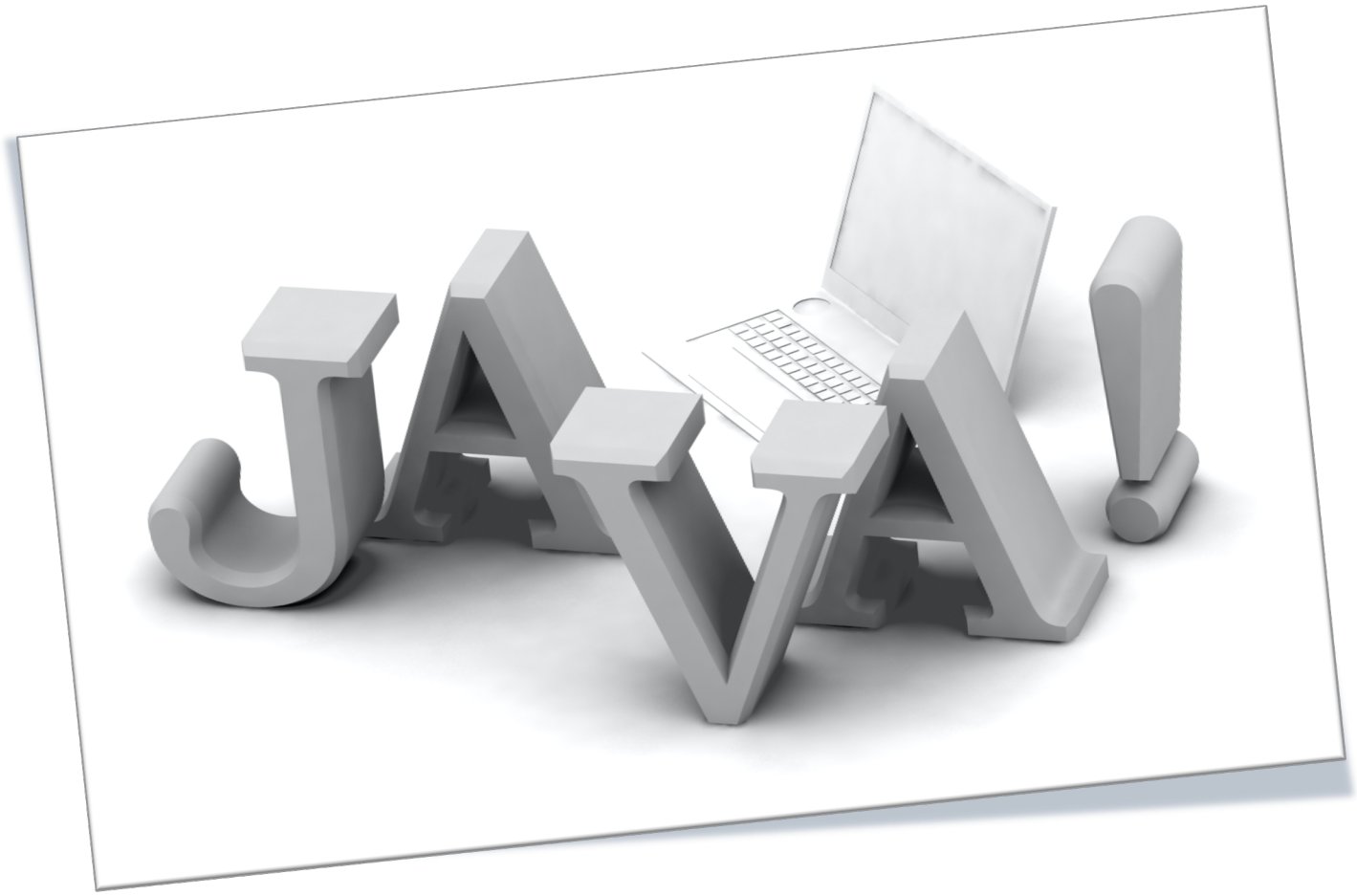
PRÁCTICAS

Capítulo 7: Colecciones



Unidad 7: Colecciones. Práctica 7.1

Crea un proyecto que se llame Capitulo7
El proyecto debe estar en el paquete org.iesjaroso.daw.capitulo7.
Organiza los ejercicios en paquetes si lo consideras necesario.

Para probar el ejercicio lo haremos una llamada desde el método principal de la clase Capitulo7

Ejercicio 1.: Crea una clase genérica NumerosAleatorios que permita generar números aleatorios. La clase debe poder crear cualquier número: byte, short, int, long, float y double. Pero no debe permitir otro tipo de dato que no sea número. Debe existir métodos que permita la generación de datos.
Forma de uso:
NumerosAleatorios<Double>.generar();
NumerosAleatorios<Short>.generar();

Genéricos:

- ✓ Crea un método public static T generar(<T extends Number> numero)
- ✓ Crea un método que permita crear una lista de números de cualquier tipo.
- ✓ Crea un método public static <T> T max(T x, T y) que devuelva el máximo de 2 objetos
- ✓ Crea un método public static <T> int contarCoincidencias (T[] list, T item): cuenta de una lista genérica, todos los elementos que sean iguales a item

Ejercicio 2.: Crea una clase llamada Persona con los siguientes atributos

- String nombre
- String apellido1
- String apellido2
- LocalDate fechaNacimiento

Y los métodos:

- Constructor
- Getters y setters
- Equals. Dos personas son iguales si tienen el mismo nombre apellido1 y apellido2
- Establece orden por defecto apellido1, apellido2 y nombre
- Persona generador()// Genera una persona a partir de los siguientes datos:
  Nombres:

```
final String[] nombres= {"Hugo", "Lucas", "Martin", "Daniel", "Pablo", "Mateo",
"Alejandro", "Leo", "Alvaro", "Manuel", "Lucia", "Sofia", "Martina", "Maria",
"Paula", "Julia", "Emma", "Valeria", "Nora", "Alba"};
```

  Apellidos:

```
final String[] apellidos = {"García", "González", "Rodríguez", "Fernández",
"López", "Martínez", "Sánchez", "Pérez", "Gómez", "Martin", "Jiménez",
"Ruiz", "Hernández", "Diaz", "Moreno", "Muñoz", "Álvarez", "Romero",
"Alonso", "Gutiérrez", "Navarro", "Torres", "Domínguez", "Vázquez", "Ramos",
"Gil", "Ramírez", "Serrano", "Blanco", "Molina", "Morales", "Suarez",
"Ortega", "Delgado", "Castro", "Ortiz", "Rubio", "Marín", "Sanz", "Núñez",
"Iglesias", "Medina", "Garrido", "Cortes", "Castillo", "Santos", "Lozano",
"Guerrero", "Cano", "Prieto", "Méndez", "Cruz", "Calvo", "Gallego", "Vidal",
"León", "Márquez", "Herrera", "Peña", "Flores", "Cabrera", "Campos", "Vega",
"Fuentes", "Carrasco", "Diez", "Caballero", "Reyes", "Nieto", "Aguilar",
"Pascual", "Santana", "Herrero", "Lorenzo", "Montero", "Hidalgo", "Giménez",
"Ibáñez", "Ferrer", "Duran", "Santiago", "Benítez", "Mora", "Vicente",
"Vargas", "Arias", "Carmona", "Crespo", "Román", "Pastor", "Soto", "Sáez",
"Velasco", "Moya", "Soler", "Parra", "Esteban", "Bravo", "Gallardo",
"Rojas"};
```

Para generar una persona se genera un nombre, 2 apellidos al azar, y una fecha en un intervalo. Utiliza los métodos privados:
- ✓ String generaNombre()
- ✓ String generaApellido()
- ✓ LocalDate generaFecha(LocalDate ini, LocalDate fin)

- Método toString() tiene el formato: [apellido1] [apellido2] [nombre], [edad] "años"

Un poco de historia…

Por todos es conocida la famosa frase de Tony Hoare llamando a **null como el error del billón de dólares**. Muchos de nosotros hemos sufrido alguna Null Pointer Exception, a veces en partes del código donde parece imposible que sucedan, posiblemente ese NPE se ha estado fraguando desde otras partes lejanas de la aplicación.

Para corregir estos errores, algunos lenguajes han decidido eliminar por completo los temidos null, pero para aquellos lenguajes que en su momento lo incluyeron, no es tan fácil. De ahí la existencia de alternativas cómo el patrón **Option**, el cual nos permite mostrar de manera explicita (mediante el sistema de tipos) la posibilidad de que un método pueda no devolver el valor deseado. Esto nos obliga a controlar la posible ausencia de valor de manera explicita, permitiéndonos elegir un valor alternativo en caso de dicha ausencia o simplemente realizar otra acción.

Ejercicio 3.: Crea una clase llamada Grupo con los siguientes atributos:
- String nombre //Nombre del grupo. GrupoA
- List<Persona> grupo

Crea los siguientes métodos:
- Contructor
- Métodos:
  - ✓ public int buscar(Persona persona)
  - ✓ public boolean insertar(Persona persona, int pos)
  - ✓ public boolean insertar(Persona persona) inserta al principio
  - ✓ public Optional<Persona> obtener(int pos)
  - ✓ public Optional<Persona> Persona(persona): Borra una persona
  - ✓ public Optional<Persona> borrar(int pos): borra de una posición
  - ✓ public boolean actualizar(Persona persona, int pos)
  - ✓ public int numPersona(). Número de personas que hay en el grupo.
- public static List<Persona> generarGrupo(int cantidad). Genera una cierta cantidad de personas. Pueden existir repetidos.
- public static Set<Persona> generarGrupoConjunto(int cantidad). Genera una cierta cantidad de personas. Pueden existir repetidos.
- Crea los métodos:
  - ✓ static Set<Persona> union(List<Persona> listA, List<Persona> listB)
  - ✓ static Set<Persona> interseccion(List<Persona> listA, List<Persona> listB)
- Crea un método ordenar().
- Crea un método desordenar().
- Crea el método eliminarRepetidos()
- Crea un método String listar(): Lista cada persona en una línea.
- Sobrescribe equals de grupo. Un grupo es igual que otro si tienen exactamente las mismas personas. No cuentan los repetidos.

- Crea un método List<Persona> subGrupo(int inicio, int final) Extrae una sublista de personas desde una posición inicial a otra.
- Crea un método boolean soyYo(Persona persona). Determina que son la misma persona el objeto actual y otro pasado por parámetro
- Crea un método boolean esHermano(Persona persona) determina si dos personas son hermanos/as. Hay que mirar los apellidos.
- Crea un método boolean esPadreDe(Persona persona) determina si el objeto actual es padre de otro objeto pasado por parámetro.
- Crea un método boolean esHijoDe(Persona persona) determina si el objeto actual es hijo de otro objeto pasado por parámetro.
- Crea un método boolean esFamiliaDe(Persona persona) determina si el objeto actual es familia de otro objeto pasado por parámetro, es decir, si es hermano, padre o hijo.
- public String agruparApellido(): Muestra todas las personas agrupadas por apellido1 contabilizando el número de ocurrencias. Es decir, contabiliza todos los apellidos iguales
- public String agruparNombre(): Muestra todas las personas agrupadas por nombre contabilizando el número de ocurrencias.
- public List<Persona> buscarPorEdad(int min, int max).
- public String mostrarPrimeros(int n) muestra las n primeras personas
- public String mostrarUltimas(int n) muestra las n primeras personas
- public Persona azar(). Muestra una persona al azar. La persona no se borra del grupo

En el método main:

a) Crea un 1 grupo llamado “GrupoA” y genera y almacena 100 personas.
b) Imprime los repetidos.
c) Crea 2 grupos GrupoB y GrupoC con la mitad de personas del GrupoA. Imprime ambos grupos
d) Añade 5 personas (usando el método azar()) del grupoA al grupoB y otras 5 del grupoB al grupoA
e) Imprime las personas que están en ambos grupos.
f) Imprime las personas que están en el grupoA que no estén en el grupoB.
g) Ordena el grupo A por edad e imprime en pantalla.
h) Elimina los repetidos de ambos grupos.
i) Busca si hay algún familiar en el grupoA en el grupoB
j) Suma un año a todas las personas del grupoA
k) Muestra las 5 personas de más edad
l) Borra todos las personas cuyo apellido comience por R del grupoA
m) Actualiza cambiando los 5 últimas personas del grupoA por las 5 primeras del grupoB. No deben permanecer en las listas originales.
n) Genera 5 grupos (con el generador) con 10 personas por grupo. Se llamarán grupo1, grupo2, … Introduce los 5 grupos en un mapa. Map<String grupo, List<Persona>> mapaGrupo
   Determina usando expresiones lambda:
   - La suma de años de todos los miembros del grupo1
   - Cual de los 5 grupos tiene más edad acumulada

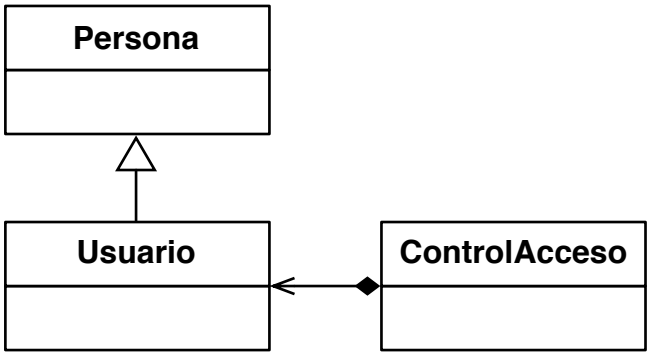
Clase Usuario

Ejercicio 4.: Login. Se desea gestionar el control de acceso de usuarios a un sistema. Vamos a usar la clase Persona del ejercicio 2, y además disponemos de las siguientes clases con sus respectivos atributos y métodos:

Clase Usuario:

Atributos:

- String usuario. El nombre de usuario se **genera automáticamente** usando la primera letra del nombre, las 3 primeras letras del apellido1 y las 3 primeras letras del apellido2. Por ejemplo: Zoe Molina Gracia, tiene el nombre de usuario= zmolgra
- String contraseña. Contraseña de entrada al sistema
- LocalDateTime fechaEntrada: Cuando hace un login correcto se registra la fecha y hora del sistema
- int MAXNUMINTENTOS = 3. Número de intentos máximos permitidos antes de que se bloquea la cuenta.
- int intentos. Indica los intentos fallidos
- LocalDateTime bloqueadoHasta



Constructor:

- public Usuario(Persona usuario, String contraseña)
- public Usuario(String nombre, String apellido1, String apellido2, String contraseña)

Métodos:

- public String getUsuario()
- public String getPass()
- public boolean isBloqueado(): devuelve true si está bloqueado
- public String msgBloqueado(). Muestra el mensaje: “Usuario <usuario> bloqueado hasta <bloqueadoHasta> ” Nota poner fecha en el formato: 12-ABR-2020 a las 18:30
- public int getIntentos()
- public void setPass(String pass): Ajusta la contraseña
- private void setFechaEntrada(LocalDateTime fecha)
- public void incrementarIntentos()
- public void resetIntentos()
- toString. Mostrará el nombre, el número de intentos y si está bloqueado, hasta cuando
- equals. Dos usuarios son iguales si tienen el mismo nombre de usuario

Otra clase llamada **ControlAcceso** contiene los siguientes atributos:

- Map<String, Usuario> usuarios
- private final static int NUMEROINTENTOSPERMITIDOS //Máx 3
- private final static int MAXMINUTOSBLOQUEADO = 1

Constructor:

- public ControlAcceso()

Métodos:

- public boolean addUsuario(Usuario usuario): añade un usuario si no se encuentra en la lista de usuarios. Ten en cuenta que la estructura que usas es un mapa
- public void generateUser(int cantidad). Usando el método Persona generador() se debe añadir la 'cantidad' de personas al azar. Fíjate en el constructor de la clase Usuario
- public boolean login(String user, String pass) que permita verificar la identidad de un usuario. Al método se le pasará una cadena con el nombre del usuario (user), y otra con la contraseña (pass). Si el usuario no existe. Se escribirá un mensaje "Usuario o contraseña no correcto" y se incrementará el número de intentosAccesoSistema
  Si el usuario existe, pero la contraseña es incorrecta se escribirá el mensaje "Usuario o contraseña no correcto" y se incrementará el número de intentos permitidos antes de ser bloqueado. Si el número de intentos es mayor a 3, al usuario se le tomará la fecha y hora del sistema y se le bloqueará durante 1 minuto y se reiniciarán el número de intentos.
  Si el usuario introduce correctamente los datos de acceso se actualiza la fechaEntrada y se inicializan el número de intentos a 0
- public boolean logout(String user): Solo funciona con usuarios dentro del sistema. Si el usuario es incorrecto se lanzará una excepción que diga "Usuario incorrecto". Si es correcto pero no ha iniciado sesión (logeado), se escribirá el mensaje "El usuario no ha iniciado sesión". Para verificar si ha iniciado sesión se comprueba que la fechaEntrada sea distinta del null. Si se realiza el logout adecuadamente este usuario cambiará fechaEntrada a null e imprimirá el mensaje: "Adios <nombreUsuario>, has estado <hora> horas <minuto>minutos y <segundo>segundos en el sistema."
- toString(): muestra todos los usuarios del sistema. Se mostrará con el siguiente formato:

Usuario:	Nombre:	Conectado:	Bloqueado:
lmarroj	Lola Martin Rojas	No	No

- public void verUsuariosBloqueados(): lista los usuarios que han sido bloqueados. Nota: Mira en los apuntes o en internet como se recorre un mapa usando un iterador.
- public void usuariosConectados(): lista los usuarios que tienen iniciada una sesión en el sistema. Con el formato:

Usuario:	Nombre:	Fecha entrada:	
lmarroj	Lola Martin Rojas	14/03/2020 – 18:33:21	- 1h 10m 30s conectada

En el programa principal se pedirán por teclado los siguientes datos:

- ❖ usuario, nombre usuario
- ❖ contras, contraseña de usuario

Después se realizará una llamada al método login(usuario, contras) pasando como argumento el nombre del usuario y la contraseña

- ❖ Prueba el resto de métodos
- ❖ En la clase principal prueba un método que permita ordenar los usuarios por otros criterios.

```
//Sorting using Anonymous Inner class.
Collections.sort(usuarios, new Comparator<Usuario>(){
  public int compare(Usuario u1, Usuario u2){
    return u1.usuario.compareTo(u2.usuario);
  }
});
Collections.sort(usuarios, (Usuario u1, Usuario u2) ->
u1.usuario.compareTo(u2.usuario));
```

Ejercicio 5.: Lotería navidad. En este sorteo hay dos bombos:

- Un bombo con las 100.000 bolas numeradas (desde la número 00000 a la 99999)
- Un bombo con los premios con exactamente 1807 bolas distribuidas de la siguiente manera:

Cantidad	Premio al billete	Descripción
1	4 000 000 €	*El Gordo*. Primer Premio
1	1 250 000 €	Segundo Premio
1	500 000 €	Tercer Premio
2	200 000 €	Cuarto Premio
8	60 000 €	Quinto Premio
1 794	1 000 €	*La Pedrea*

Se pide:

- Crear una aplicación que simule el sorteo, debe crear los dos bombos con las bolas e ir asociando el premio con un número
- Dado un número de lotería con una apuesta (cantidad jugada) determine si el número ha sido premiado y la cantidad obtenida de acuerdo a la siguiente tabla:

Cantidad	Premio billete	Descripción	Total
1	4 000 000 €	El Gordo. Primer Premio	4 000 000 €
1	1 250 000 €	Segundo Premio	1 250 000 €
1	500 000 €	Tercer Premio	500 000 €
2	200 000 €	Cuarto Premio	400 000 €
8	60 000 €	Quinto Premio	480 000 €
1 794	1 000 €	*La Pedrea*	1 794 000 €
2	20 000 €	A los números anterior y posterior al primer premio (aproximación)	40 000 €
2	12 500 €	A los números anterior y posterior al segundo premio (aproximación)	25 000 €
2	9 600 €	A los números anterior y posterior al tercer premio (aproximación)	19 200 €
99	1 000 €	A la centena (tres primeras cifras) del primero premio	99 000 €
99	1 000 €	A la centena (tres primeras cifras) del segundo premio	99 000 €
99	1 000 €	A la centena (tres primeras cifras) del tercer premio	99 000 €
198	1 000 €	A la centena (tres primeras cifras) de los cuartos premios	198 000 €
999	1 000 €	A los números cuyas dos últimas cifras coincidan con las del primer premio	999 000 €
999	1 000 €	A los números cuyas dos últimas cifras coincidan con las del segundo premio	999 000 €
999	1 000 €	A los números cuyas dos últimas cifras coincidan con las del tercer premio	999 000 €
9 999	200 €	Reintegro a los números cuya última cifra coincida con la del primer premio	1 999 800 €

Nota: recuerda que un billete es un décimo (hay que dividir entre diez para calcular el premio por cada décimo)
USAR COLECCIONES: Listas o conjuntos