Guia-Examen-Practico-FP-Java.pdf

Misco_Jones

Fundamentos de Programación

1º Grado en Ingeniería Informática - Tecnologías Informáticas

Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla

Apuntes Java

1.Clase con atributos

●Crear atributos

```
private LocalDate fecha;
private String ciudad;

private Tipo nombre;
```

●Constructor a partir de los atributos (Source → Generate constructor using fields)

```
public Registro(LocalDate fecha, String ciudad, Double direccion) {
            *****
    this.fecha = fecha;
    this.ciudad = ciudad; this.direccion = direccion;
```

►Comprobar que alguna propiedad cumpla alguna condición

En los ** del constructor, se coloca:

```
Checkers.check("Texto de condicion", condicion);
```

●Getters/Setters
(Source → Generate Getters & Setters)

►Get de propiedad derivada

●ToString
(Source → To String)

●Equals/HashCode
(Source → Equals/HashCode)

●Compare To
Para ordenar de forma natural

```
public int compareTo(Registro o) {
            int res=atributo1.compareTo(o.atributo1);
            if(res==0) {
                  res=atributo2.compareTo(o.atributo2);
                  if(res==0) {
                        res=atributo3.compareTo(o.atributo3);
                        y asi encadenando para establecer el orden
            }
            return res;
      }
```

2.Clase con los metodos

- Atributo con un (conjunto, lista...) de objetos ....

```
private List ó Set nombre;
```

- constructor vacío

```
public Clase metodos() {
      nombre(igual al anterior) = new HashSet<Clase atributos>();
                                        ArrayList<>
}
```

- constructor de la clase a partir de un Stream

```
public Clase metodos(Stream<Clase atributos> r) {
      this.nombre = nombre.collect(Collectors.toSet());
                                                  .toList()
}
```

- Añadir

```
public void añadir(Registro r) {
      registros.add(r);
}
```

3.Metodos (Tipos y codigos)

- Ordenar (menor a mayor)

```
.sorted(Comparator.comparing(Registro::getVelocidadMaxima)
```

- Ordenar a la inversa (mayor a menor)

```
.sorted(Comparator.comparing(Registro::getVelocidadMaxima).reversed())
```

- Hacer la media

```
.collect(Collectors.averagingDouble(Registro::getVelocidadMedia));
```

- Hacer un diccionario

```
.collect(Collectors.groupingBy(Clave, Collectors.--[valor]);
```

- Stream de claves

```
d.keySet().stream()
```

- Stream de valores

```
d.valueSet().stream()
```

- Stream de Claves/Valores

```
d.EntrySet().stream
```

- Valor maximo/minimo (max/min)

```
.max(Comparator.comparing(r->r.getPipo()));
```

●Valor maximo/minimo (maxBy/minBy)
AndThen corrige el tipo Optional que **maxBy**/**minBy** devuelven, mediante la ultima linea, que es siempre igual.

```
Collectors.collectingAndThen(
                Collectors.maxBy( Comparator.comparing  (
                     Registro::getVelocidadMaxima)),
                     o -> o.get() )));
```

●Clave con mayor valor

```
return apariciones.entrySet().stream()
                          .max(Entry.comparingByValue())
                          .get()
                          .getKey();
```

●Añadir valor a un diccionario

```
public void añadirAlbum(Album a) {
            if(albumesPorCuartil.containsKey(a.getCuartil())) {
                  albumesPorCuartil.get(a.getCuartil()).add(a);

             }else {
                  SortedSet<Album>ss=new TreeSet<>();
                  ss.add(a);
                  albumesPorCuartil.put(a.getCuartil(), ss);
            }
      }
```

●Sacar los valores a un nuevo contenedor(lista, set)

```
public Set<Album> getAlbumes(){
//Set resultado Set<Album>res=new HashSet<>();
            //Coleccion de Sorted Sets formada por los valores del diccionario
            Collection<SortedSet<Album>>sets=albumesPorCuartil.values();

            //Para cada Sorted Set en la coleccion
            for(SortedSet<Album>ss:sets) {
                  //Se añaden al set resultado res.addAll(ss);
            }
            return res;
      }
```

●Diccionario de .counting, pero de tipo Integer

```
Map<Distrito, Integer> di = inspecciones.stream()
                          .filter(i->i.getEsCritica() == true)
                          .collect(Collectors.groupingBy(Inspeccion::getDistrito,
                          Collectors.reducing(0, e -> 1, Integer::sum)));
```

●Lanzar excepciones

```
if(condicion) {
      throw new excepcion;
      }
```

4.Factoria

```
public static Inspecciones leerInspecciones (String ruta) {
      Inspecciones res = null;
      try {
            Stream<Inspeccion> i =Files.lines(Paths.get(ruta))
            .skip(1)
            .map(FactoriaInspecciones::parsearInspeccion);
          res = new Inspecciones(i);
      }catch(IOException e) { System.out.println("Ruta
            erronea"+ruta); e.printStackTrace();
      }
      return res;
}
```

```
private static Registro parsearRegistro(String lineaCSV) { String[] sp =
      lineaCSV.split(","); ← Marca la separacion
       Checkers.check("Cadena con formato no válido", sp.length == 5);
                                                                  n· de atri

      LocalDate fecha = LocalDate.parse(sp[0].trim(),
                        DateTimeFormatter.ofPattern("y-M-d"));
      String ciudad = sp[1].trim();
      Double direccion = Double.valueOf(sp[2].trim()); Double
      velocidadMedia = Double.valueOf(sp[3].trim()); Double
      velocidadMaxima = Double.valueOf(sp[4].trim());
      return new Registro(fecha, ciudad, direccion,
                          velocidadMedia,velocidadMaxima);
}
```

► .trim() para String

```
String ciudad = sp[1].trim();
```

► .trim() para LocalDate

```
LocalDate fecha = LocalDate.parse(sp[0].trim(),
                  DateTimeFormatter.ofPattern("y-M-d"));
```

► .trim() para Integer, Double y Enum

```
Double direccion = Double.valueOf(sp[2].trim()); Integer.
                   Distrito. ← (Nombre del enumerado)
```

► .trim() para Boolean

```
Boolean esCritico=sp[6].trim().equals("Critical");
```

^---Este texto marca si es true, abre el csv para verlo