

**EJERCICIO 1 (1,5 puntos)**

```
public class Vivienda implements Comparable<Vivienda> {
    private Double superficie;
    private Double precio;
    private Integer numHabitaciones;
    private TipoVivienda tipo;
    private String direccion;
    private String distrito;

    public Vivienda(Double sp, Double pr, Integer nh, TipoVivienda tp,
                    String dr, String dt) {
        Checkers.check("Precio positivo", pr > 0.);
        Checkers.check("Superficie positiva", sp > 0.);
        Checkers.check("Número de habitaciones >= 0", nh >= 0);
        Checkers.check("Distrito cinco dígitos", dt.length() == 5);
        for (Character c: dt.toCharArray()) {
            Checkers.check("Distrito cinco dígitos", Character.isDigit(c));
        }
        superficie = sp;
        precio = pr;
        numHabitaciones = nh;
        tipo = tp;
        direccion = dr;
        distrito = dt;
    }
    public Double getSuperficie() {
        return superficie;
    }
    public Integer getNumHabitaciones() {
        return numHabitaciones;
    }
    public String getDireccion() {
        return direccion;
    }
    public Double getPrecio() {
        return precio;
    }
    public Double getPrecioM2() {
        return precio/superficie;
    }
    public Boolean getCapital() {
        return distrito.charAt(2) == '0';
        // También:
        // Integer valor = Integer.valueOf(distrito);
        // return (valor / 100) % 10 == 0;
    }
    public String getDistrito() {
        return distrito;
    }
    public TipoVivienda getTipo() {
        return tipo;
    }
}
```

```

// equals y toString generados por Eclipse
public int compareTo(Vivienda p) {
    int v;
    v = this.getDireccion().compareTo(p.getDireccion());
    if (v == 0){
        v = this.getDistrito().compareTo(p.getDistrito());
    }
    return v;
}

```

EJERCICIO 2 (1,5 puntos)

```

public class FactoriaCatalogo {

    public static Catalogo leerRegistros(String rutaFichero) {
        Catalogo res = null;
        try {
            Stream<Vivienda> s = Files.lines(Paths.get(rutaFichero))
                .skip(1)
                .map(FactoriaCatalogo::parsearRegistro);
            res = new Catalogo(s);
        } catch (IOException e) {
            System.out.println("No se ha encontrado el fichero "
                + rutaFichero);
            e.printStackTrace();
        }
        return res;
    }
    private static Vivienda parsearRegistro(String linea) {
        String [] trozos = linea.split(",");
        Checkers.check("Cadena con formato no válido", trozos.length!= 6);
        Double sp = Double.parseDouble(trozos[0].trim());
        Double pr = new Double(trozos[1].trim());
        Integer nh = Integer.parseInt(trozos[2].trim());
        TipoVivienda tp =
            TipoVivienda.valueOf(trozos[3].trim().toUpperCase());
        String dr = trozos[4].trim();
        String dt = trozos[5].trim();
        return new Vivienda(sp, pr, nh, tp, dr, dt);
    }
}

```

EJERCICIO 3 (7 puntos)

- a) Método getPrecioMedioPorDistritoDeTipos (1 punto)

```

public Map<String, Double> getMapVivXDistritoTipos(
    Set<TipoVivienda> conjTipos) {
    return lista.stream()
        .filter(v -> settipos.contains(v.getTipo()))
        .collect(Collectors.groupingBy(Vivienda::getDistrito,
            Collectors.averagingDouble(Vivienda::getPrecio)));
}

```

b) Método getDistritosMasNViviendas (1,5 puntos)

```
public Map<String, Long> getNumViviendasPorDistrito() {  
    return lista.stream()  
        .collect(Collectors.groupingBy(x -> x.getDistrito(),  
            Collectors.counting()));  
}  
  
public Set<String> getDistritosMasViviendas(Integer n) {  
    return getNumViviendasPorDistrito().entrySet().stream()  
        .filter(x -> x.getValue() >= n)  
        .map(x -> x.getKey())  
        .collect(Collectors.toSet());  
}
```

c) Método getTiposViviendasMasCarasCalle (1,5 puntos)

```
public List<TipoVivienda> getNumHabDistintasCalle(String calle,  
    Integer n) {  
    Comparator<Vivienda> cmp =  
        Comparator.comparing(x -> x.getPrecioM2());  
    return lista.stream()  
        .filter(x -> x.getDireccion().contains(calle))  
        .sorted(cmp.reversed())  
        .limit(n)  
        .map(x -> x.getTipo())  
        .distinct()  
        .collect(Collectors.toList());  
}
```

d) Método getViviendaMasBarataPorTipoEnCapital (1,5 puntos)

```
public Map<TipoVivienda, Vivienda>  
    getMapViviendaMasBarataPorTipoEnCapital() {  
    return lista.stream()  
        .filter(x -> x.getCapital())  
        .collect(Collectors.groupingBy(Vivienda::getTipo,  
            Collectors.collectingAndThen(  
                Collectors.minBy(  
                    Comparator.comparing(Vivienda::getPrecio)),  
                x -> x.get()))  
    );  
}
```

e) Método getPorcViviendasPreciosMayorUmbralPorDistrito (1,5 puntos)

```
public Map<String, Double> getPorcPreciosMayoresXDistrito(Double umbral) {  
    Map<String, List<Double>> preciosM2XDistrito =  
        lista.stream()  
        .collect(Collectors.groupingBy(Vivienda::getDistrito,  
            Collectors.mapping(x -> x.getPrecioM2(),  
                Collectors.toList()))  
    );
```

```

    return preciosM2XDistrito.entrySet().stream()
        .collect(Collectors.toMap(e -> e.getKey(),
                                  e -> porcentajePorEncima(e.getValue(), umbral)));
}

private Double porcentajePorEncima(List<Double> precios, Double umbral) {
    Integer total = precios.size();
    Long numPorEncima = precios.stream()
        .filter(x -> x > umbral)
        .count();
    return numPorEncima * 1.0 / total;
}

```

Alternativa:

```

public Map<String, Double> getPorcPreciosMayorXDistrito2(Double umbral) {
    Map<String, List<Double>> preciosM2XDistrito =
        lista.stream()
            .collect(Collectors.groupingBy(Vivienda::getDistrito,
                Collectors.mapping(x -> x.getPrecioM2(), Collectors.toList())));
    Map<String, Double> res = new HashMap<>();
    for (String s: preciosM2XDistrito.keySet()) {
        res.put(s, porcentajePorEncima(preciosM2XDistrito.get(s), umbral));
    }
    return res;
}

```