FUNDAMENTOS DE PROGRAMACIÓN. Curso 2024/25

SEGUNDO EXAMEN PARCIAL. 27 de mayo de 2025. Soluciones

Ejercicio 1: tipo Producto (0,5 ptos)

```
public record Producto (String nombre, Double precio, String categoria) {
	public Producto {
		Checkers.check("El precio no puede ser negativo", precio >=0.);
		Checkers.check("La categoría no puede tener más de 100
			caracteres", categoria.length() <= 100);
	}
}
```

Ejercicio 2: tipo Factura (1,5 ptos)

```
public class Factura implements Comparable<Factura> {
	private String numero;
	private LocalDate fecha;
	private String cliente;
	private String pais;
	private Double impuestos;
	private TipoCliente tipoCliente;
	private List<Producto> productos;

	public Factura(String numero, LocalDate fecha, String cliente,
			String pais, Double impuestos,
		TipoCliente tipoCliente, List<Producto> productos) {
		this.numero = numero;
		this.fecha = fecha;
		setCliente(cliente);
		setPais(pais);
		setImpuestos(impuestos);
		setTipoCliente(tipoCliente);
		Checkers.check("La lista de productos no puede estar vacía",
			productos.size() >= 0);
		this.productos = new ArrayList<>(productos);
	}

	public String getCliente() {
		return cliente;
	}

	public void setCliente(String cliente) {
		this.cliente = cliente;
	}

	public String getPais() {
		return pais;
	}
	public void setPais(String pais) {
		this.pais = pais;
	}
	public Double getImpuestos() {
		return impuestos;
	}

	public void setImpuestos(Double impuestos) {
		Checkers.check("El porcentaje de impuestos debe estar entre 0
			y 100", impuestos >= 0. && impuestos <=100.);
		this.impuestos = impuestos;
	}
```

```java
public TipoCliente getTipoCliente() {
	return tipoCliente;
}
public void setTipoCliente(TipoCliente tipoCliente) {
	this.tipoCliente = tipoCliente;
}
public String getNumero() {
	return numero;
}
public LocalDate getFecha() {
	return fecha;
}
public List<Producto> getProductos() {
	return new ArrayList<>(productos);
}

public Double getPrecioTotal() {
	Double precio = productos.stream()
				.mapToDouble(Producto::precio)
				.sum();
	return precio + precio * getImpuestos() / 100.;
}

//También con bucles
public Double getPrecioTotal() {
	Double precio = 0.0;
	for (Producto p: productos) {
		precio += p.precio();
	}
	return precio + precio * getImpuestos() / 100.;
}

public Integer getNumeroProductos () {
	return productos.size();
}

public Integer getNumeroCategorias() {
	Long res= productos.stream()
			.map(Producto::categoria)
			.distinct()
			.count();
	return res.intValue();
}

// También con bucles
public Integer getNumeroCategorias2() {
	Set<String> res = new HashSet<>();
	for (Producto p: productos) {
		res.add(p.categoria());
	}
	return res.size();
}

public String toString() {
	return "Factura [numero=" + numero + ", fecha=" + fecha
		+ ", cliente=" + cliente + ", pais=" + pais
		+ ", impuestos=" + impuestos + ", tipoCliente="
		+ tipoCliente + ", productos=" + productos + "]";
}
```

```
public int hashCode() {
	return Objects.hash(fecha, numero);
}

public boolean equals(Object obj) {
	if (this == obj)
		return true;
	if (obj == null)
		return false;
	if (getClass() != obj.getClass())
		return false;
	Factura other = (Factura) obj;
	return Objects.equals(fecha, other.fecha)
		&& Objects.equals(numero, other.numero);
}

public int compareTo(Factura f) {
	int res = getFecha().compareTo(f.getFecha());
	if (res == 0) {
		res = getNumero().compareTo(f.getNumero());
	}
	return res;
}
```

Ejercicio 3: Factoría (1 pto)

```
private static final String SEPARADOR_PRIMARIO = ",";
private static final String SEPARADOR_PRODUCTOS = ";";
private static final String SEPARADOR_PRODUCTO = "-";

public static Factura parsearFactura(String cadena) {
	Checkers.checkNoNull(cadena);
	String[] trozos = cadena.split(SEPARADOR_PRIMARIO);
	Checkers.check("Formato no espearado<" + cadena + ">",
		trozos.length == 7);

	String numero = trozos[0].trim();
	LocalDate fecha = LocalDate.parse(trozos[1].trim());
	String cliente = trozos[2].trim();
	String pais = trozos[3].trim();
	Double impuestos = Double.parseDouble(trozos[4].trim());
	TipoCliente tipo = TipoCliente.valueOf(trozos[5].trim());
	List<Producto> productos = parseaProductos(trozos[6].trim());

	return new Factura(numero, fecha, cliente, pais, impuestos,
			tipo, productos);
}

private static List<Producto> parseaProductos(String cadProductos) {
	String clean = cadProductos.replace("\'", "");
	String[] trozos = clean.split(SEPARADOR_PRODUCTOS);

	List<Producto> res = new ArrayList<>();
	for (String trozo: trozos) {
		res.add(parseaProducto(trozo));
	}
	return res;
}
```

```
	private static Producto parseaProducto(String cadProducto) {
		String clean = cadProducto.replace("[", "").replace("]","");
		String[] trozos = clean.split(SEPARADOR_PRODUCTO);
		String nombre = trozos[0].trim();
		Double precio = Double.parseDouble(trozos[1].trim());
		String categoria = trozos[2].trim();
		return new Producto(nombre,precio, categoria);
	}
```

Ejercicio 4: Tratamientos secuenciales (7 ptos)

```
public class EstadisticasFacturas extends Facturas {

	public EstadisticasFacturas(Stream<Factura> facturas) {
		super(facturas);
	}

	public String toString() {
		return super.toString() + " Numero facturas: "
			+ getFacturas().size();
	}
}
```

Apartado 1 – 1 Pto

```
public List<Double> getPrecioTotalNFacturasMasRecientesCliente
		(String cliente, Integer n) {

	Comparator<Factura> cmp = Comparator.comparing(Factura::getFecha);

	return getFacturas().stream()
			.filter(f -> f.getCliente().equals(cliente))
			.sorted(cmp.reversed())
			.limit(n)
			.map(Factura::getPrecioTotal)
			.collect(Collectors.toList());
}
```

Apartado 2 – 1 Pto

```
public Producto getProductoMasCaroPosteriorFecha(LocalDate fecha) {

	return getFacturas().stream()
			.filter(f -> f.getFecha().isAfter(fecha))
			.flatMap(f -> f.getProductos().stream())
			.max(Comparator.comparing(Producto::precio))
			.get();
}
```

Apartado 3 – 1,5 Ptos

```
public SortedMap<String, Integer>
            getNumeroDeCategoriasDistintasPorCliente() {
      Map<String, Set<String>> m = new HashMap<>() ;

      for (Factura f: getFacturas()) {
            for (Producto p: f.getProductos()) {
            String clave = f.getCliente();
                  Set<String> valor = m.getOrDefault(clave,
                        new HashSet<>());
                  valor.add(p.categoria());
                  m.put(clave, valor);
            }
      }

      SortedMap<String, Integer> res= new TreeMap<String,Integer>();
      for (Map.Entry<String, Set<String>> e: m.entrySet()){
            res.put(e.getKey(), e.getValue().size());
      }

      return res;
}

//También
public SortedMap<String, Integer>
         getNumeroDeCategoriasDistintasPorCliente() {
      Map<String, Set<String>> m = new HashMap<>();
      for (Factura f:getFacturas()) {
            for (Producto p: f.getProductos()) {
                  String clave = f.getCliente();
                  if (m.containsKey(clave)) {
                        m.get(clave).add(p.categoria());
                  } else {
                        Set<String> valor = new HashSet<>();
                        valor.add(p.categoria());
                        m.put(clave, valor);
                  }
            }
      }

      SortedMap<String, Integer> res= new TreeMap<String, Integer>();
      for (Map.Entry<String, Set<String>> e: m.entrySet()){
            res.put(e.getKey(), e.getValue().size());
      }

      return res;
}
```

Apartado 4 – 1,5 Ptos

```
public SortedMap<TipoCliente, Double>
            getPorcentajeFacturasDePaisPorTipoCliente (String pais) {
      Long numFacturasPais = getFacturas().stream()
                              .filter(f -> f.getPais().equals(pais))
                              .count();

      return getFacturas().stream()
                  .filter(f -> f.getPais().equals(pais))
                  .collect(Collectors.groupingBy(
                        Factura::getTipoCliente,
                        TreeMap::new,
                        Collectors.collectingAndThen(
                              Collectors.counting(),
                              numFactTipo >
                                    100.0 * numFactTipo/numFacturasPais)));
}
```

Apartado 5 – 1,5 Ptos

```
public Map<String, Double>
           getGastoTotalSuperiorUmbralPorClienteConCategoria (
            String categoria, Double umbral) {

      Map<String, Double> res = getFacturas().stream()
            .filter(f -> tieneProductoCategoria(f, categoria))
            .collect(Collectors.groupingBy(
                  Factura::getCliente,
                  Collectors.summingDouble(Factura::getPrecioTotal)));

      return res.entrySet().stream()
            .filter(e -> e.getValue()>umbral)
                  .collect(Collectors.toMap(Map.Entry::getKey,
                        Map.Entry::getValue));
}

private Boolean tieneProductoCategoria(Factura f, String categoria) {
      return f.getProductos().stream()
            .anyMatch(p -> p.categoria().equals(categoria));
}
```

TESTS

```
public class TestEstadisticasFacturas {

      public static void mostrarFacturas(Collection<Factura> col) {
            col.stream()
                  .forEach(System.out::println);
      }

      public static void main (String [] args) {
            EstadisticasFacturas est =
                  FactoriaFacturas.creaFacturas("data/facturas.csv");
            mostrarFacturas(est.getFacturas());
```

```
        System.out.println("EJERCICIO 1" + "=".repeat(70));
        testGetPrecioTotalNFacturasMasRecientesCliente(est,
                "Gary Santiago", 3);
        testGetPrecioTotalNFacturasMasRecientesCliente(est,
                "Lisa Byrd", 2);

        System.out.println("EJERCICIO 2" + "=".repeat(70));
        LocalDate fecha = LocalDate.of(2025, 2, 1);
        testGetProductoMasCaroPosteriorFecha(est, fecha);

        System.out.println("EJERCICIO 3" + "=".repeat(70));
        testGetNumeroDeCategoriasDistintasPorCliente(est);

        System.out.println("EJERCICIO 4" + "=".repeat(70));
        String pais = "Estados Unidos";
        testGetPorcentajeFacturasDePaisPorTipoCliente(est, pais);
        pais = "España";
        testGetPorcentajeFacturasDePaisPorTipoCliente(est, pais);

        System.out.println("EJERCICIO 5" + "=".repeat(70));
        String categoria = "Muebles";
        Double umbral = 8000. ;
        testGetGastoTotalSuperiorUmbralPorClienteConCategoria (est,
                categoria, umbral);
        categoria = "Electrónica";
        umbral = 8000. ;
        testGetGastoTotalSuperiorUmbralPorClienteConCategoria (est,
                categoria, umbral);
    }

    private static void testGetNumeroDeCategoriasDistintasPorCliente
                (EstadisticasFacturas est) {
        try {
                Map<String, Integer> res =
                        est.getNumeroDeCategoriasDistintasPorCliente();
                String msg = String.format(
                "El número de categorías distintas por cliente es\n%s",
                res);
                System.out.println(msg);
        } catch (Exception e) {
                System.out.println(">>>Capturada excepcion inesperada" +
                        e.getMessage());
        }
    }
    private static void
                testGetGastoTotalSuperiorUmbralPorClienteConCategoria
                (EstadisticasFacturas est,
                String categoria, Double umbral) {
        try {
                Map<String, Double> res = est
                      .getGastoTotalSuperiorUmbralPorClienteConCategoria(
                      categoria, umbral);
                String msg = String.format("El gasto total por cliente
                      con categoria %s y superior a %f es\n%s",
                      categoria, umbral, res);
                System.out.println(msg);
        } catch (Exception e) {
                System.out.println(">>>Capturada excepcion inesperada" +
                      e.getMessage());
        }
    }
```

```
	private static void testGetPorcentajeFacturasDePaisPorTipoCliente
			(EstadisticasFacturas est, String pais) {
		try {
			SortedMap<TipoCliente, Double> res = est
				.getPorcentajeFacturasDePaisPorTipoCliente(pais);
			String msg = String.format("El porcentaje de cada pais
				por tipo de cliente es \n%s", res);
			System.out.println(msg);
		} catch (Exception e) {
			System.out.println(">>>Capturada excepcion inesperada" +
				e.getMessage());
		}
	}

	private static void testGetProductoMasCaroPosteriorFecha
			(EstadisticasFacturas est, LocalDate fecha) {
		try {
			Producto res = est
				.getProductoMasCaroPosteriorFecha(fecha);
			String msg = String.format("El producto más caro con
				fecha posterior a %s es \n%s", fecha,
				res.toString());
			System.out.println(msg);
		} catch (Exception e) {
			System.out.println(">>>Capturada excepcion inesperada" +
				e.getMessage());
		}
	}

	private static void testGetPrecioTotalNFacturasMasRecientesCliente
			(EstadisticasFacturas est, String cliente, Integer n) {
		try {
			List<Double> res = est
				.getPrecioTotalNFacturasMasRecientesCliente
					(cliente, n);
			String msg = String.format("Los precios de las %d
				facturas más recientes del cliente %s son:\n %s",
				n, cliente, res.toString());
			System.out.println(msg);
		} catch (Exception e) {
			System.out.println(">>>Capturada excepcion inesperada" +
				e.getMessage());
		}
	}
}
```