Ejercicio 1

```
public record Pedido(LocalDate fecha, String usuario,
		String pais, String ciudad,Envio envio, Set<String> categorias,
		String producto, Double precioUnitario, Integer unidades)
		implements Comparable<Pedido> {

	private static final Double IVA = 0.21;

	public Pedido{
		Checkers.checkNoNull(producto, precioUnitario,unidades);
		Checkers.check("Fecha de pedido no válida", LocalDate.now().isAfter(fecha));
		Checkers.check("Precio unitario no válido", precioUnitario >= 0);
		Checkers.check("Unidades no válidas", unidades >= 0);
	}

	public Double precioTotal() {
		Double precio = precioUnitario() * unidades();
		return precio + precio * IVA;
	}

	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((fecha == null) ? 0 : fecha.hashCode());
		result = prime * result + ((producto == null) ? 0 : producto.hashCode());
		result = prime * result + ((usuario == null) ? 0 : usuario.hashCode());
		return result;
	}

	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (!(obj instanceof Pedido))
			return false;
		Pedido other = (Pedido) obj;
		if (fecha == null) {
			if (other.fecha != null)
				return false;
		} else if (!fecha.equals(other.fecha))
			return false;
		if (producto == null) {
			if (other.producto != null)
				return false;
		} else if (!producto.equals(other.producto))
			return false;
		if (usuario == null) {
			if (other.usuario != null)
				return false;
		} else if (!usuario.equals(other.usuario))
			return false;
		return true;
	}
```

```
	public int compareTo(Pedido p) {
		int res = fecha().compareTo(p.fecha());
		if (res == 0) {
			res = usuario().compareTo(p.usuario());
			if (res == 0) {
				res = producto().compareTo(p.producto());
			}
		}
		return res;
	}

	public String toString() {
		return "Pedido [fecha=" + fecha + ", nombre=" + usuario + ", pais=" + pais +
			", ciudad=" + ciudad + ", tipoEnvio=" + envio + ", categorias=" +
			categorias + ", producto=" + producto + ", precioUnitario=" +
			precioUnitario + ", unidades=" + unidades + ", precioTotal=" +
			precioTotal() + "]";
	}
}
```

Ejercicio 2

```
public class EstudioPedidos {

	private Set<Pedido> pedidos;

	public EstudioPedidos(Stream<Pedido> sPedidos) {
		this.pedidos = sPedidos.collect(Collectors.toSet());
	}

	public Set<Pedido> getPedidos(){
		return new HashSet<Pedido> (pedidos);
	}

	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((pedidos == null) ? 0 : pedidos.hashCode());
		return result;
	}

	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		EstudioPedidos other = (EstudioPedidos) obj;
		if (pedidos == null) {
			if (other.pedidos != null)
				return false;
		} else if (!pedidos.equals(other.pedidos))
			return false;
		return true;
	}

	public String toString() {
		return "EstudioPedidos [pedidos=" + pedidos + "]";
	}
```

Ejercicio 3

```
public class FactoriaEstudioPedidos {

	public static EstudioPedidos leerEstudioPedidos(String rutaFichero) {
		EstudioPedidos res = null;

		try {
			Stream<Pedido> stPedidos=
					Files.lines(Paths.get(rutaFichero))
					     .skip(1)
					    .map(FactoriaEstudioPedidos::parsearPedido);
			res = new EstudioPedidos(stPedidos);
		} catch (IOException e) {
			e.printStackTrace();
		}

		return res;
	}

	private static Pedido parsearPedido(String cad) {
		Checkers.checkNoNull(cad);
		String [] trozos = cad.split(";");
		Checkers.check("Formato no válido", trozos.length == 9);
		LocalDate fecha = parsearFecha(trozos[0].trim());
		String usuario = trozos[1].trim();
		String pais = trozos[2].trim();
		String ciudad = trozos[3].trim();
		Envio envio = Envio.valueOf(trozos[4].trim());
		Set<String> categorias = parsearCategorias(trozos[5].trim());
		String producto = trozos[6].trim();
		Double precioUnitario = parsearPrecio(trozos[7].trim());
		Integer unidades = Integer.parseInt(trozos[8].trim());
		return new Pedido(fecha, usuario, pais, ciudad, envio, categorias,
				producto, precioUnitario, unidades);
	}

	private static Double parsearPrecio(String cad) {
		String cleaned = cad.replace("$", "").replace(".", "");
		return Double.parseDouble(cleaned);
	}

	private static Set<String> parsearCategorias(String cad) {
		String [] trozos = cad.split("and");
		return Arrays.stream(trozos)
			.map(String::trim)
			.collect(Collectors.toSet());
	}

	private static Set<String> parsearCategorias2(String cad) {
		String [] trozos = cad.split("and");
		Set<String> res = new HashSet<String> ();
		for (String elem:trozos) {
			res.add(elem.trim());
		}
		return res;
	}

	private static LocalDate parsearFecha(String cad) {
		return LocalDate.parse(cad, DateTimeFormatter.ofPattern("d/M/yy"));
	}
}
```

Ejercicio 4

Ejercicio 4.1

```
public Map<Envio, Integer> getTotalPedidosPorEnvio(Set<Envio> envios, Integer mes) {
	Map<Envio, Integer> res = new HashMap<Envio, Integer>();

	for (Pedido pedido: pedidos) {
		Envio e = pedido.envio();
		if (pedido.fecha().getMonthValue() == mes && envios.contains(e)) {
			if (res.containsKey(e)) {
				res.put(e, res.get(e) + 1 );
			} else {
				res.put(e, 1);
			}
		}
	}
	return res;
}
```

Ejercicio 4.2

```
public Double getMediaPrecioPorPedidoUsuarioAlemania(String usuario, String categoria) {
	return pedidos.stream()
		.filter(p -> p.usuario().equals(usuario) && p.categorias().contains(categoria))
		.mapToDouble(p -> p.precioUnitario()*p.unidades() * 1.19)
		.average()
		.getAsDouble();
}
```

Ejercicio 4.3

Solución 1

```
public Map<String, List<Double>> getResumenPedidosUsuario() {
	return pedidos.stream()
		.collect(Collectors.groupingBy(Pedido::usuario,
			Collectors.collectingAndThen(
				Collectors.summarizingDouble(Pedido::precioUnitario),
				summary->crearLista(summary))));
	}

	private List<Double> crearLista (DoubleSummaryStatistics summary) {
		return List.of(summary.getMin(), summary.getAverage(), summary.getMax());
	}
```

Solución 2

```
public Map<String, List<Double>> getResumenPedidosUsuario() {
	return pedidosPorUsuario().entrySet().stream()
		.collect(Collectors.toMap(e -> e.getKey(),
					e -> getResumen(e.getValue())));
}

private Map<String, List<Pedido>> pedidosPorUsuario() {
	return pedidos.stream()
		.collect(Collectors.groupingBy(Pedido::usuario));
}
```

```
private List<Double> getResumen(List<Pedido> pedidos) {
	Double min = pedidos.stream()
		.mapToDouble(Pedido::precioUnitario)
		.min()
		.getAsDouble();
	Double max = pedidos.stream()
		.mapToDouble(Pedido::precioUnitario)
		.max()
		.getAsDouble();
	Double media = pedidos.stream()
		.mapToDouble(Pedido::precioUnitario)
		.average()
		.getAsDouble();
	return List.of(min, media, max);
}
```

Ejercicio 4.4

Solución 1

```
public String getUsuarioMasDerrochadorPosicion(LocalDate fecha, Integer n) {
	Map<String, Double> gastosPorUsuario= getTotalGastadoPorUsuario(fecha);

	Comparator<Map.Entry<String, Double>> c = Map.Entry.comparingByValue();
	return gastosPorUsuario.entrySet().stream()
		.sorted(c)
		.skip(n)
		.findFirst()
		.get()
		.getKey();
}

private Map<String, Double> getTotalGastadoPorUsuario(LocalDate fecha) {
	return this.pedidos.stream()
		.filter(p -> p.fecha().isAfter(fecha))
		.collect(Collectors.groupingBy(Pedido::usuario,
			Collectors.summingDouble(Pedido::precioTotal)));
}
```

Solución 2:

```
public String getUsuarioMasDerrochadorPosicion2(LocalDate fecha, Integer n) {
	Map<String, Double> gastosPorUsuario = getTotalGastadoPorUsuario(fecha);

	List<Map.Entry<String, Double>> l = gastosPorUsuario.entrySet().stream()
		.sorted(Comparator.comparing(e -> e.getValue()))
		.collect(Collectors.toList());
	return l.get(n).getKey();
}
```

Ejercicio 4.5

Solución 1

```
public Map<String, String> getProductoMayorPrecioPorPais(LocalDate fecha) {
	Map<String, List<Pedido>> pedidosPorPais = pedidos.stream()
			.filter(p -> p.fecha().equals(fecha))
			.collect(Collectors.groupingBy(Pedido::pais));
	return pedidosPorPais.entrySet().stream()
			.collect(Collectors.toMap(Map.Entry::getKey,
				e -> productoMasCaro(e.getValue())));
}

private String productoMasCaro(Collection<Pedido> pedidos) {
	return pedidos.stream()
		.max(Comparator.comparing(Pedido::precioTotal))
		.get()
		.producto();
}
```

Solución 2

```
public Map<String, String> getProductoMayorPrecioPorPais2(LocalDate fecha) {
	return pedidos.stream()
			.filter(p->p.fecha().equals(fecha))
			.collect(Collectors.groupingBy(Pedido::pais,
				Collectors.collectingAndThen(Collectors.toList(),
					listaPedidos->productoMasCaro(listaPedidos))));
}
```

Tests

```
   public class TestEstudioPedidos {

	public static void main(String[] args) {
		EstudioPedidos estPed =
			FactoriaEstudioPedidos.leerEstudioPedidos("data/sales.csv");
		testLeerEstudioPedidos(estPed);
		testGetTotalPedidosPorEnvio(estPed, Set.of(Envio.PLUS, Envio.PRIORITY), 1);
		testGetMediaPrecioPorPedidoUsuarioAlemania(estPed, "Fredrick Beveridge",
			"office supplies");
		testGetUsuarioMasDerrochadorPosicion(estPed, LocalDate.of(2011, 1, 7), 3);
		testGetProductoMayorPrecioPorPais(estPed, LocalDate.of(2011, 1, 4));
		testGetResumenPedidosUsuario(estPed);
	}

	private static void testLeerEstudioPedidos(EstudioPedidos estPed) {
		String msg = String.format("Número de pedidos:     %d\n",
			estPed.getPedidos().size());
		System.out.println(msg);
		estPed.getPedidos().stream()
			.forEach(System.out::println);
	}

	private static void testGetTotalPedidosPorEnvio(EstudioPedidos ep, Set<Envio> envios,
			Integer mes) {
		System.out.println("\nTEST de getTotalPedidosPorEnvio");
		try {
			String msg = String.format("El total de pedidos para los envíos %s y el mes
				%d es", envios.toString(), mes);
			System.out.println(msg);
			imprimeMap(ep.getTotalPedidosPorEnvio(envios, mes));
		} catch (Exception e) {
```

```
                System.out.println("Excepción inesperada capturada:\n    " + e);
        }
    }

    private static void testGetMediaPrecioPorPedidoUsuarioAlemania(EstudioPedidos ep,
                String usuario, String categoria) {
        System.out.println("\nTEST de getMediaPrecioPorPedidoUsuarioAlemania");
        try {
                Double res = ep.getMediaPrecioPorPedidoUsuarioAlemania(usuario, categoria);
                String msg = String.format("La media de precio por producto en Alemania
                        para el usuario %s y la categoría %s hubiese sido de : %.2f euros",
                         usuario, categoria, res);
                System.out.println(msg);
        } catch (Exception e) {
                System.out.println("Excepción inesperada capturada:\n    " + e);
        }
    }

    private static void testGetUsuarioMasDerrochadorPosicion(EstudioPedidos ep,
                LocalDate fecha, Integer n) {
        System.out.println("\nTEST de getUsuarioMasDerrochador");
        try {
                String res = ep.getUsuarioMasDerrochadorPosicion(fecha, n);
                System.out.println(String.format("El usuario más derrochador en la posicion
                        %d de compras posteriores a %s es %s", n, fecha.toString(), res));
        } catch (Exception e) {
                System.out.println("Excepción inesperada capturada:\n    " + e);
        }
    }

    private static void testGetProductoMayorPrecioPorPais(EstudioPedidos ep,
                LocalDate fecha) {
        System.out.println("\nTEST de getProductoMayorPrecioPorPais");
        try {
                String msg = String.format("Producto mayor precio por pais en %s", fecha);
                System.out.println(msg);
                Map<String, String> res = ep.getProductoMayorPrecioPorPais(fecha);
                imprimeMap(res);
        } catch (Exception e) {
                System.out.println("Excepción inesperada capturada:\n    " + e);
        }
    }

    private static void testGetResumenPedidosUsuario(EstudioPedidos ep) {
        System.out.println("\nTEST de getResumenPedidosUsuario");
        try {
                Map<String, List<Double>> res = ep.getResumenPedidosUsuario();
                imprimeMap(res);
        } catch (Exception e) {
                System.out.println("Excepción inesperada capturada:\n    " + e);
        }
    }

    private static <K, V> void imprimeMap(Map<K, V> map) {
        map.entrySet().stream()
           .forEach(entry->System.out.println( entry.getKey() + " --> "
              + entry.getValue() ));
    }

}
```