Ejercicio 1

```
public Boolean getHayJugadorFechaPorcentajeMayor(LocalDate fecha, Double r) {
	return jugadores.stream()
		.filter(x -> x.getFecha().equals(fecha) && x.getTiros1() > 0)
		.anyMatch(x -> x.getPuntos1() * 100. / x.getTiros1() > r);
}
```

Ejercicio 2

```
public Double getMediaJugadoresInicial(Character c) {
	return jugadores.stream()
		.filter(x -> x.getNombre().charAt(0) == c)
		.mapToInt(x -> x.getPuntos1() + x.getPuntos2() + x.getPuntos3())
		.average()
		.getAsDouble();
}
```

Solución alternativa:

```
public Double getMediaJugadoresInicial(Character c) {
	Double res = jugadores.stream()
		.filter(x -> x.getNombre().charAt(0) == c)
		.collect(Collectors.averagingInt(
			x -> x.getPuntos1() + x.getPuntos2() + x.getPuntos3());

	if (Double.isNaN(res))
		throw new NoSuchElementException();
	return res;
}
```

Ejercicio 3

Apartado a

```
public Map <String, SortedSet<Jugador>> getMapConjOrdenadoPorPais2() {

	Comparator<Jugador> cmp = Comparator.comparing(Jugador::getTiempo)
		.reversed();

	Map <String, SortedSet<Jugador>> res = new HashMap<>();
	for (Jugador j: jugadores) {
		String clave = j.getPais();
		if (res.containsKey(clave)) {
			res.get(clave).add(j);
		} else {
			SortedSet<Jugador> sj = new TreeSet<>(cmp);
			sj.add(j);
			res.put(clave, sj);
		}
	}
	return res;
}
```

Apartado b

```
public Map <String, SortedSet<Jugador>> getMapConjOrdenadoPorPais() {

	Comparator<Jugador> cmp = Comparator.comparing(Jugador::getTiempo)
		.reversed();

	return jugadores.stream()
		.collect(Collectors.groupingBy(Jugador::getPais,
			Collectors.toCollection(() -> new TreeSet<>(cmp))));
}
```

Ejercicio 4

Solución considerando los tiros libres de un partido del jugador en ese año:

```
public List<String> getListaJugadoresMenosPuntos1Año(Integer año, Integer n) {
	return jugadores.stream()
		.filter(x -> a.equals(x.getFecha().getYear()))
		.sorted(Comparator.comparing(Jugador::getPuntos1))
		.limit(n)
		.map(Jugador::getNombre)
		.distinct()
		.collect(Collectors.toList());
}
```

Solución considerando la suma de todos los tiros libres del jugador en ese año:

```
public List<String> getListaJugadoresMenosPuntos1Año(Integer a, Integer n) {

	Map<String, Integer> mpaux = jugadores.stream()
		.filter(x -> a.equals(x.getFecha().getYear()))
		.collect(Collectors.groupingBy(Jugador::getNombre,
			Collectors.summingInt(Jugador::getPuntos1)));

	//También se puede usar Entry.comparingByValue() en sorted
	return mpaux.entrySet().stream()
		.sorted(Comparator.comparing(x -> x.getValue()))
		.limit(n)
		.map(x -> x.getKey())
		.collect(Collectors.toList());
}
```

Ejercicio 5

```
public Map<String, List<String>> getMapListaJugadoresOrdenadaTiempoPorPais() {
	Map<String, List<Jugador>> mpaux=jugadores.stream()
		.collect(Collectors.groupingBy(Jugador::getPais));

	return mpaux.entrySet().stream()
		.collect(Collectors.toMap(
			x -> x.getKey(),
			x -> ordenaNombres(x.getValue())));
}
```

Solución alternativa:

```
public Map<String, List<String>> getMapListaJugadoresOrdenadaTiempoPorPais2() {
	return jugadores.stream()
		.collect(Collectors.groupingBy(Jugador::getPais,
			Collectors.collectingAndThen(Collectors.toList(),
				lista -> ordenaNombres(lista))));
}

private List<String> ordenaNombres(List<Jugador> lj) {
	return lj.stream()
		.sorted(Comparator.comparing(Jugador::getTiempo).reversed())
		.map(Jugador::getNombre).
		.collect(Collectors.toList());
}
```

Ejercicio 6

```
public Long getNumeroJugadoresMasPuntos(Integer n) {
	return getMapPuntosTotalesPorJugador().entrySet().stream()
		.filter(x -> x.getValue() > n)
		.count();
}

private Map<String, Integer> getMapPuntosTotalesPorJugador() {
	return jugadores.stream()
		.collect(Collectors.groupingBy(
			x -> x.getNombre(),
			Collectors.summingInt(
				x -> x.getPuntos1() + x.getPuntos2() + x.getPuntos3())
		));
}
```

Solución alternativa:

```
public Long getNumeroJugadoresMasPuntos(Integer n) {
	return getMapPuntosTotalesPorJugador().values().stream()
		.filter(totalPuntos -> totalPuntos > n)
		.count();
}
```