Exercise 1

You and your team are doing professional data consulting on a wide spectrum of projects. Last week, you've been given a call from a company called We Really Care Inc (short WRC Inc.) This company is one of the oldest health insurance companies in the world and they've aggregated insane amount of data about their activity all across the globe.

After countless months of trying to collect all of the bits and pieces into uniform data schema, you've managed to minimize all the relevant logs into a single well-formed data set that consists of four distributed collections:

```
case class Broker(brokerId: Int, fullName: String, ...)
case class Client(clientId: Int, fullName: String, ...)
case class Contract(contractId: Int, brokerId: Int, clientId:
Int, planId: Int, ...)
case class Plan(planId: Int, yearIntroducedIn: Int, price:
Double, ...)

val brokers: RDD[Broker] = ...
val contracts: RDD[Contract] = ...
val clients: RDD[Client] = ...
val plans: RDD[Plan] = ...
```

The company employs brokers whose job is to seek and arrange contracts with clients. Contracts are tied to specific plans in the company's product line-up.

Question 1.1

In the past year WRC has been having hard time financially. Due to this, upper management decided to let some people go. To make an educated decision about whom to fire they need to analyze the performance of the current employees.

Part 1

Compute an RDD that maps broker full names to their total revenue (i.e. total income from all of their contracts). Try to compute this as efficiently as possible.

```
 val brokerNameRevenue: RDD[(String, Double)] = ???
```

Answer

```
val planContracts: RDD[(Int, Int)] =
  contracts.map(c => (c.planId, c.contractId))
val planPrices: RDD[(Int, Double)] =
  plans.map(p => (p.planId, p.price))
val contractPrices: RDD[(Int, Double)] =
  planContracts.join(planPrices).values
val contractBrokers: RDD[(Int, Int)] =
  contracts.map(c => (c.contractId, c.brokerId))
val brokerContractPrices: RDD[(Int, Double)] =
  contractBrokers.join(contractPrices).values
val brokerIdRevenues: RDD[(Int, Double)] =
  brokerContractPrices.reduceByKey(_ + _)
val brokerNames: RDD[(Int, String)] =
  brokers.map(b => (b.brokerId, b.fullName))
val brokerNameRevenues: RDD[(String, Double)] =
  brokerNames.leftOuterJoin(brokerIdRevenues)
            .values
            .mapValues(_.getOrElse(0.0))
```

Part 2

Efficiently compute an RDD that outputs the names of the 20% lowest-performing employees.

```
 val low20: Array[String] = ???
```

Answer

```
val low20 = brokerNameRevenue
 .top((brokers.count * 0.2).toInt)(Ordering.by(_._2).reverse)
```

Question 1.2

WRC is quite an old company and has a large list of life-long clients who managed to get a contract a long time time, when everything was cheaper. Naturally, WRC doesn't benefit as much as it could from those clients.

To optimize its revenue, WRC decided to do a proactive outreach to such clients and try to upsell them a newer, fancier plan that is actually just the same thing but more expensive.

Part 1

Compute an RDD of client's full names whose latest contract is older than 10 years old.

```
val upsellClientNames: RDD[String] = ???
```

Answer

```
val planYear: RDD[(Int, Int)] =
  plans.map(p => (p.planId, p.yearIntroducedIn))
vla planClient: RDD[(Int, Int)] =
  contracts.map(c => (c.planId, p.clientId))
val clientYear: RDD[(Int, Int)] =
  planClient.join(planYear).values
val clientLatestYear: RDD[(Int, Int)] =
  clientYear.reduceByKey(_ max _)
Val clientNames: RDD[(Int, String)] =
  clients.map(c => (c.clientId, c.fullName))
val upsellClientNames: RDD[String] =
  clientLatestYear.filter(p => 2017 - p._2 > 10)
                  .join(clientNames)
                  .values
```

Part 2

For every upsell client, find the cheapest plan of 2017 that's still more expensive than the current one they have at the moment, if any.

```
val upsellSuggestion: RDD[(String, Plan)] = ???
```

Answer

```
val contractClient: RDD[(Int, Int)] = ... // similar to above
val contractPlan: RDD[(Int, Plan)] =
  contracts.map(c => (c.planId, c.contractId))
          .join(plans.map(p => (p.planId, p)))
          .values
val clientPlan: RDD[(Int, Plan)] =
  contractClient.join(contractPlan).values
val clientUpsells: RDD[(Int, Plan)] =
  clientLatestPlan.cartesian(plans).filter {
    case ((clientId, clientPlan), newPlan) =>
      newPlan.price > clientPlan.price
  }.map {
    case ((clientId, clientPlan), newPlan) =>
      (clientId, newPlan)
  }
val clientCheapeastUpsell: RDD[(Int, Plan)] =
  clientUpsells.reduceByKey {
    case (p1, p2) if p1.price < p2.price => p1
    case (_, p2)                         => p2
  }
val clientNames: RDD[(Int, String)] =
  clients.map(c => (c.clientId, c.fullName))
val upsellSuggestion: RDD[(String, Plan)] =
  clientCheapeastUpsell.join(clientNames).values
```