# Example 1

## JSprit – Single Depot

```java
public class SmallWetterau1 {

   public static final int UNITS = 0;

   public static void main(String[] args) {

      VehicleRoutingProblem.Builder vrpBuilder
         = VehicleRoutingProblem.Builder.newInstance();

      Location n0 = Location.newInstance(3, -4);

      Service n1 = Service.Builder.newInstance("n1")
            .setLocation(Location.newInstance(-6, 8))
            .addSizeDimension(UNITS, 15)
            .setTimeWindow(TimeWindow.newInstance(30, 45))
            .setServiceTime(3)
            .build();

      Service n2 = Service.Builder.newInstance("n2")
            .setLocation(Location.newInstance(-8, -8))
            .addSizeDimension(UNITS, 10)
            .setTimeWindow(TimeWindow.newInstance(30, 60))
            .setServiceTime(7)
            .build();

      Service n3 = Service.Builder.newInstance("n3")
            .setLocation(Location.newInstance(11, -3))
            .addSizeDimension(UNITS, 15)
            .setTimeWindow(TimeWindow.newInstance(15, 30))
            .setServiceTime(12)
            .build();

      Location n4 = Location.newInstance(-12, 8);

      vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3));

      IncompleteCostMatrix.Builder costMatrixBuilder =
            IncompleteCostMatrix.Builder.newInstance();

      // First variant used to add distances
      costMatrixBuilder.addTransportTime(n0,n1.getLocation(),
            slowWayFunction(n0,n1.getLocation()));

      // Second (perhaps more easy variant)
      Set<RelationKey> normalWays = new HashSet<>();
      normalWays.add(RelationKey.newKey(n1, n4));
      normalWays.add(RelationKey.newKey(n1, n2));
      normalWays.add(RelationKey.newKey(n1, n3));

      for(RelationKey key : normalWays)
         costMatrixBuilder.addTransportTime(key.from, key.to, normalWayFunction(key.from, key.to));

      Set<RelationKey> fastWays = new HashSet<>();
      fastWays.add(RelationKey.newKey(n0,n2));
      fastWays.add(RelationKey.newKey(n0, n3));
      fastWays.add(RelationKey.newKey(n2, n4));

      for(RelationKey key : fastWays)
         costMatrixBuilder.addTransportTime(key.from, key.to, fastWayFunction(key.from, key.to));

      costMatrixBuilder.completeTransportTimeMatrix();
      IncompleteCostMatrix cm = costMatrixBuilder.build();
      vrpBuilder.setRoutingCost(cm);

      // Vehicle type definition
      VehicleType vehicleType = VehicleTypeImpl.Builder.newInstance("unitVehicleType")
            .addCapacityDimension(UNITS, 200).build();

      // Vehicle instance defintion
      VehicleImpl vehicleInstance = VehicleImpl.Builder.newInstance("unitVehicleInstance")
            .setType(vehicleType)
            .setLatestArrival(300)
            .setStartLocation(n0)
            .build();

      // Adding vehicle instance to the problem
      vrpBuilder.addVehicle(vehicleInstance);

      VehicleRoutingProblem vrp = vrpBuilder.build();
   }

   public static double fastWayFunction(Location end1, Location end2) {
      return 0.5 * EuclideanDistanceCalculator
         .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
   }

   public static double normalWayFunction(Location end1, Location end2) {
      return  EuclideanDistanceCalculator
         .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
   }

   public static double slowWayFunction(Location end1, Location end2) {
      return 2 * EuclideanDistanceCalculator
            .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 1;
   }
}
```

# Example 2

## JSprit – Multiple Depot

```java
public class SmallWetterau2 {

  public static final int UNITS_FOOD = 0;
  public static final int UNITS_BEVERAGE = 1;

   public static void main(String[] args) {

      VehicleRoutingProblem.Builder vrpBuilder
         = VehicleRoutingProblem.Builder.newInstance();

      Location n0 = Location.newInstance(3, -4);

      Delivery n1 = Delivery.Builder.newInstance("n1")
            .setLocation(Location.newInstance(-6, 8)).addSizeDimension(UNITS_BEVERAGE, 15)
            .setTimeWindow(TimeWindow.newInstance(30, 45)).setServiceTime(3)
            .build();

      Delivery n2 = Delivery.Builder.newInstance("n2")
            .setLocation(Location.newInstance(-8, -8)).addSizeDimension(UNITS_FOOD, 10)
            .setTimeWindow(TimeWindow.newInstance(30, 60)).setServiceTime(7).build();

      Delivery n3 = Delivery.Builder.newInstance("n3")
            .setLocation(Location.newInstance(11, -3)).addSizeDimension(UNITS_FOOD, 15)
            .setTimeWindow(TimeWindow.newInstance(15, 30)).setServiceTime(12).build();

      Location n4 = Location.newInstance(-12, 8);

      vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3));

      IncompleteCostMatrix.Builder costMatrixBuilder =
            IncompleteCostMatrix.Builder.newInstance();

      // First variant used to add distances
      costMatrixBuilder.addTransportTime(n0,n1.getLocation(),
            slowWayFunction(n0,n1.getLocation()));

      // Second (perhaps more easy variant)
      Set<RelationKey> normalWays = new HashSet<>();
      normalWays.add(RelationKey.newKey(n1, n4));
      normalWays.add(RelationKey.newKey(n1, n2));
      normalWays.add(RelationKey.newKey(n1, n3));

      for(RelationKey key : normalWays)
         costMatrixBuilder.addTransportTime(key.from, key.to, normalWayFunction(key.from, key.to));

      // Second (perhaps more easy variant)
      Set<RelationKey> fastWays = new HashSet<>();
      fastWays.add(RelationKey.newKey(n0,n2));
      fastWays.add(RelationKey.newKey(n0, n3));
      fastWays.add(RelationKey.newKey(n2, n4));

      for(RelationKey key : fastWays)
         costMatrixBuilder.addTransportTime(key.from, key.to, fastWayFunction(key.from, key.to));

      costMatrixBuilder.completeTransportTimeMatrix();
      IncompleteCostMatrix cm = costMatrixBuilder.build();
      vrpBuilder.setRoutingCost(cm);

      // Vehicle food type definition
      VehicleType vehicleTypeF = VehicleTypeImpl.Builder.newInstance("unitVehicleTypeF")
            .addCapacityDimension(UNITS_FOOD, 200).build();
      // Vehicle food instance defintion
      VehicleImpl vehicleInstanceF = VehicleImpl.Builder.newInstance("unitVehicleInstanceF")
            .setType(vehicleType).setLatestArrival(300).setStartLocation(n0).build();
      // Adding vehicle food instance to the problem
      vrpBuilder.addVehicle(vehicleInstanceF);

        // Vehicle beverage type definition
      VehicleType vehicleTypeB = VehicleTypeImpl.Builder.newInstance("unitVehicleTypeB")
            .addCapacityDimension(UNITS_BEVERAGE, 150).build();
      // Vehicle food instance defintion
      VehicleImpl vehicleInstanceB = VehicleImpl.Builder.newInstance("unitVehicleInstanceB")
            .setType(vehicleType).setLatestArrival(300).setStartLocation(n4).build();
      // Adding vehicle beverage instance to the problem
      vrpBuilder.addVehicle(vehicleInstanceB);

      VehicleRoutingProblem vrp = vrpBuilder.build();
   }

   public static double fastWayFunction(Location end1, Location end2) {
      return 0.5 * EuclideanDistanceCalculator
         .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
   }

   public static double normalWayFunction(Location end1, Location end2) {
      return  EuclideanDistanceCalculator
         .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
   }

   public static double slowWayFunction(Location end1, Location end2) {
      return 2 * EuclideanDistanceCalculator
            .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 1;
   }
}
```

# Example 3

## Athos – Single Depot

```
model WetterauOrders1
products
   unit weight 1.0
functions
   durationFunction slowWayFunction 2 * length + 1
   durationFunction normalWayFunction  length
   durationFunction fastWayFunction  0.5 * length
network
   nodes
      n0 at (3, -4) isDepot unit sprouts vehicles customers n1, n2, n3 at 0 latestTime 300
      n1 at (-6, 8) hasDemand unit units 15 earliestTime 30 latestTime 45 serviceTime 3
      n2 at (-8, -8) hasDemand unit units 10 earliestTime 30 latestTime 60 serviceTime 7
      n3 at (11, -3) hasDemand unit units 15 earliestTime 15 latestTime 30 serviceTime 12
      n4 at (-12, 8)
   edges
      s1 from n0 to n1 function slowWayFunction
      group normalWayGroup function normalWayFunction members
         w1 from n1 to n4
         w2 from n1 to n2
         w3 from n1 to n3
      group fastWayGroup function fastWayFunction members
         f1 from n0 to n2
         f2 from n0 to n3
         f3 from n2 to n4
agentTypes
   agentType vehicles maxWeight 200
      behaviour awt awaitTour when finished do die
      behaviour die vanish
```

# Example 4

## Athos – Multiple Depot

```
model WetterauOrders2
products
  unitsFood weight 1.0
  unitsBeverage weight 1.0
functions
  durationFunction slowWayFunction 2 * length + 1
  durationFunction normalWayFunction  length
  durationFunction fastWayFunction  0.5 * length
network
  nodes
    n0 at (3, -4) isDepot unitsFood sprouts vehiclesFood customers n2, n3 at 0 latestTime 300
    n1 at (-6, 8) hasDemand unitsBeverage units 15 earliestTime 30 latestTime 45 serviceTime 3
    n2 at (-8, -8) hasDemand unitsFood units 10 earliestTime 30 latestTime 60 serviceTime 7
    n3 at (11, -3) hasDemand unitsFood units 15 earliestTime 15 latestTime 30 serviceTime 12
    n4 at (-12, 8) isDepot unitsBeverage sprouts vehiclesBeverage customers n1 at 0 latestTime 300
  edges
    s1 from n0 to n1 function slowWayFunction
      group normalWayGroup function normalWayFunction members
        w1 from n1 to n4
        w2 from n1 to n2
        w3 from n1 to n3
      group fastWayGroup function fastWayFunction members
        f1 from n0 to n2
        f2 from n0 to n3
        f3 from n2 to n4
agentTypes
  agentType vehiclesFood maxWeight 200
    behaviour awt awaitTour when finished do die
    behaviour die vanish
  agentType vehiclesBeverage maxWeight 150
    behaviour awt awaitTour when finished do die
    behaviour die vanish
```