Red Hat Decision Manager 7.11

Developing solvers with Red Hat build of OptaPlanner in Red Hat Decision Manager

Red Hat Decision Manager 7.11 Developing solvers with Red Hat build of OptaPlanner in Red Hat Decision Manager

Legal Notice

Abstract

This document describes how to develop solvers with Red Hat build of OptaPlanner in Red Hat Decision Manager to find the optimal solution to planning problems.

Table of Contents

PREFACE

As a developer of business decisions , you can use Red Hat build of OptaPlanner to develop solvers that determine the optimal solution to planning problems. OptaPlanner is a built-in component of Red Hat Decision Manager. You can use solvers as part of your services in Red Hat Decision Manager to optimize limited resources with specific constraints.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see *our CTO Chris Wright's message* .

PART I. UPGRADING YOUR RED HAT BUILD OF OPTAPLANNER PROJECTS TO OPTAPLANNER 8

If you have OptaPlanner projects that you created with the OptaPlanner 7 or earlier pubic API and you want to upgrade your project code to OptaPlanner 8, review the information in this guide. This guide also includes changes to implementation classes which are outside of the pubic API.

The OptaPlanner public API is a subset of the OptaPlanner source code that enables you to interact with OptaPlanner through Java code. So that you can upgrade to higher OptaPlanner versions within the same major release, OptaPlanner follows semantic versioning. This means that you can upgrade from OptaPlanner 7.44 to OptaPlanner 7.48 for example without breaking your code that uses the OptaPlanner public API. The OptaPlanner public API classes are compatible within the versions of a major OptaPlanner release. However, when Red Hat releases a new major release, disrupting changes are sometimes introduced to the public API.

OptaPlanner 8 is a new major release and some of the changes to the public API are not are not compatible with earlier versions of OptaPlanner. OptaPlanner 8 will be the foundation for the 8.x series for the next few years. The changes to the public API that are not compatible with earlier versions that were required for this release were made for the long term benefit of this project.

Table 1. Red Hat Decision Manager and Red Hat build of OptaPlanner versions

Decision Manager	OptaPlanner
7.7	7.33
7.8	7.39
7.9	7.44
7.10	7.48
7.11	8.5

Every upgrade note has a label that indicates how likely it is that your code will be affected by that change. The following table describes each label:

Table 2. Upgrade impact labels

Label	Impact
Major	Likely to affect your code.
Minor	Unlikely to affect your code, especially if you followed the examples, unless you have customized the code extensively.

Any changes that are not compatible with earlier versions of OptaPlanner are annotated with the **Public API** tag.

CHAPTER 1. CHANGES THAT ARE NOT COMPATIBLE WITH OPTAPLANNER 7.X OR EARLIER

The changes listed in the section are not compatible with OptaPlanner 7.x or earlier versions of OptaPlanner.

Java 11 or higher required
Major, Public API

If you are using JRE or JDK 8, upgrade to JDK 11 or higher.

- On Linux, get OpenJDK from your Linux software repository. On Fedora and Red Hat Enterprise Linux, enter the following command:

  ```
  sudo dnf install java-11-openjdk-devel
  ```

- On Windows and macOS, download OpenJDK from the AdoptOpenJDK website.

SolverFactory and PlannerBenchmarkFactory no longer support KIE containers
Major, Public API

Because OptaPlanner now aligns with Kogito, the KIE container concept no longer applies. Therefore, **SolverFactory** no longer allows you to create **Solver** instances from KIE containers. This also applies to **PlannerBenchmarkFactory** and benchmarks.

OSGi metadata removed
Major, Public API

Because of the limited usage of OSGi and the maintenance burden it brings, the OptaPlanner JAR files in the OptaPlanner 8.x series no longer include OSGi metadata in their **META-INF/MANIFEST.MF** file.

Refrain from using Java serialization
Minor, Public API

In OptaPlanner 8, most uses of the **Serializable** marker interface were removed from the public API. Consider serializing with JSON or another format.

SolverFactory.getScoreDirectorFactory() replaced with ScoreManager
Major, Public API

In version 7 of OptaPlanner, using **ScoreDirectorFactory** was necessary in order to explain the score. In version 8 of OptaPlanner, new functionality was added to the **ScoreManager** and as a result there is no longer any reason to create new instances of **ScoreDirector**.

An example from a ***.java** file in OptaPlanner 7:

```
ScoreDirectorFactory<CloudBalance> scoreDirectorFactory =
solverFactory.getScoreDirectorFactory();
try (ScoreDirector<CloudBalance> scoreDirector = scoreDirectorFactory.buildScoreDirector()) {
    scoreDirector.setWorkingSolution(solution);
    Score score = scoreDirector.calculateScore();
}
```

An example from a ***.java** file in OptaPlanner 8:

```
ScoreManager<CloudBalance> scoreManager = ScoreManager.create(solverFactory);
Score score = scoreManager.updateScore(solution);
```

Methods that allowed you to retrieve an instance of **ScoreDirector** and **ScoreDirectorFactory** have been removed from the public API without replacement. A reduced version of the **ScoreDirector** interface was promoted to the public API to promote the **ProblemFactChange** interface to the public API.

SolverFactory: getSolverConfig() removed

Minor, Public API

The **SolverFactory.getSolverConfig()** method has been deprecated and replaced with the **SolverFactory.create(SolverConfig)** method. A **SolverConfig** instance is now instantiated before a **SolverFactory** instance is instantiated, which is more natural. The previous order has been removed.

An example from a ***.java** file in OptaPlanner 7:

```
SolverFactory<MySolution> solverFactory =
SolverFactory.createFromXmlResource(".../mySolverConfig.xml");
SolverConfig solverConfig = solverFactory.getSolverConfig();
...
Solver<MySolution> solver = solverFactory.buildSolver();
```

An example from a ***.java** file in OptaPlanner 8:

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource(".../mySolverConfig.xml");
...
SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
Solver<MySolution> solver = solverFactory.buildSolver();
```

If you were also passing a **ClassLoader**, pass it to both **SolverConfig.createFromXmlResource()** and **SolverFactory.create()**.

SolverConfig: buildSolver() removed

Minor, Public API

The **SolverConfig.buildSolver()** method is an inner method that does not belong in the public API. Use the **SolverFactory.buildSolver()** method instead.

An example from a ***.java** file in OptaPlanner 7:

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource(".../mySolverConfig.xml");
...
Solver<MySolution> solver = solverConfig.buildSolver();
```

An example from a ***.java** file in OptaPlanner 8:

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource(".../mySolverConfig.xml");
...
SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
Solver<MySolution> solver = solverFactory.buildSolver();
```

PlannerBenchmarkConfig: buildPlannerBenchmark() removed

Minor, Public API

The **PlannerBenchmarkConfig.buildPlannerBenchmark()** method is an inner method that does not belong in the public API. Use the **PlannerBenchmarkFactory.buildPlannerBenchmark()** method instead.

An example from a ***.java** file in OptaPlanner 7:

```
PlannerBenchmarkConfig benchmarkConfig = PlannerBenchmarkConfig.createFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
...
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();
```

An example from a ***.java** file in OptaPlanner 8:

```
PlannerBenchmarkConfig benchmarkConfig = PlannerBenchmarkConfig.createFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
...
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.create(benchmarkConfig);
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();
```

SolverFactory: cloneSolverFactory() removed
Minor, Public API

The **SolverFactory.cloneSolverFactory()** method has been deprecated and replaced with the new **SolverConfig(SolverConfig)** copy constructors and the **SolverFactory.cloneSolverFactory()** method has been removed.

An example from a ***.java** file in OptaPlanner 7:

```
private SolverFactory<MySolution> base;

public void userRequest(..., long userInput) {
    SolverFactory<MySolution> solverFactory = base.cloneSolverFactory();
    solverFactory.getSolverConfig()
        .getTerminationConfig()
        .setMinutesSpentLimit(userInput);
    Solver<MySolution> solver = solverFactory.buildSolver();
    ...
}
```

An example from a ***.java** file in OptaPlanner 8:

```
private SolverConfig base;

public void userRequest(..., long userInput) {
    SolverConfig solverConfig = new SolverConfig(base); // Copy it
    solverConfig.getTerminationConfig()
        .setMinutesSpentLimit(userInput);
    SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
    Solver<MySolution> solver = solverFactory.buildSolver();
    ...
}
```

SolverFactory: createEmpty() removed

Minor, **Public API**

The **SolverFactory.createEmpty()** method has been deprecated and replaced with the **new SolverConfig()** method. The **SolverFactory.createEmpty()** method has been removed.

An example from a ***.java** file in OptaPlanner 7:

```
SolverFactory<MySolution> solverFactory = SolverFactory.createEmpty();
SolverConfig solverConfig = solverFactory.getSolverConfig()
...
Solver<MySolution> solver = solverFactory.buildSolver();
```

An example from a ***.java** file in OptaPlanner 8:

```
SolverConfig solverConfig = new SolverConfig();
...
SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
Solver<MySolution> solver = solverFactory.buildSolver();
```

XML <solver/> root element now belongs to the https://www.optaplanner.org/xsd/solver namespace
Major, **Public API**

OptaPlanner now provides an XML schema definition for the solver configuration. Although OptaPlanner retains compatibility with previous versions of the existing XML configuration, migrating to the XSD is strongly recommended because OptaPlanner might support only valid configuration XML in the future.

An example from the ***SolverConfig.xml** file in OptaPlanner 7:

```
<?xml version="1.0" encoding="UTF-8"?>
<solver>
  ...
</solver>
```

An example from the ***SolverConfig.xml** file in OptaPlanner 8:

```
<?xml version="1.0" encoding="UTF-8"?>
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  ...
</solver>
```

Using the XSD might require reordering some of the XML elements of the configuration. Use code completion in the IDE to migrate to a valid XML.

Property subPillarEnabled in move selector configuration has been removed
Minor, **Public API**

The **subPillarEnabled** property on **PillarSwapMoveSelector** and **PillarChangeMoveSelector** has been deprecated and replaced with a new property, **subPillarType**. The **subPillarEnabled** property has been removed.

An example from the ***SolverConfig.xml** and ***BenchmarkConfig.xml** files in OptaPlanner 7:

```
<pillar...MoveSelector>
  ...
  <pillarSelector>
    <subPillarEnabled>false</subPillarEnabled>
    ...
  </pillarSelector>
  ...
</pillar...MoveSelector>
```

An example from the ***SolverConfig.xml** and ***BenchmarkConfig.xml** files in OptaPlanner 8:

```
<pillar...MoveSelector>
  <subPillarType>NONE</subPillarType>
  <pillarSelector>
    ...
  </pillarSelector>
  ...
</pillar...MoveSelector>
```

Solver: getScoreDirectorFactory() removed

Major, **Public API**

The **getScoreDirectorFactory()** method has been deprecated and has now been removed from both **Solver** and **SolverFactory** classes.

You no longer need to create a **Solver** instance just to calculate or explain a score in the UI. Use the **ScoreManager** API instead.

An example from a ***.java** file in OptaPlanner 7:

```
SolverFactory<VehicleRoutingSolution> solverFactory = SolverFactory.createFromXmlResource(...);
Solver<VehicleRoutingSolution> solver = solverFactory.buildSolver();
uiScoreDirectorFactory = solver.getScoreDirectorFactory();
...
```

An example from a ***.java** file in OptaPlanner 8:

```
SolverFactory<VehicleRoutingSolution> solverFactory = SolverFactory.createFromXmlResource(...);
ScoreManager<VehicleRoutingSolution> scoreManager = ScoreManager.create(solverFactory);
...
```

ScoreDirectorFactory should not be used anymore because it has always been outside the public API and all of its functionality is exposed in various parts of the public API.

Solver.explainBestScore() has been removed

Major, **Public API**

The **explainBestScore()** method on the **Solver** interface was deprecated in 7.x and has now been removed. You can obtain the same information through the new **ScoreManager** API.

Red Hat recommends that you do not parse the results of this method call in any way.

An example from a ***.java** file in OptaPlanner 7:

```
solver = ...;
scoreExplanation = solver.explainBestScore();
```

An example from a ***.java** file in OptaPlanner 8:

```
MySolution solution = ...;
ScoreManager<MySolution> scoreManager = ...;
scoreExplanation = scoreManager.explainScore(solution);
```

The Solver interface methods **getBestSolution()**, **getBestScore()**, and **getTimeMillisSpent()** have been removed
Minor, Public API

Several methods on the **Solver** interface were deprecated in 7.x and have been removed. You can obtain the same information by registering an **EventListener** through the **Solver.addEventListener(…)**.

An example from a ***.java** file in OptaPlanner 7:

```
solver = ...;
solution = solver.getBestSolution();
score = solver.getBestScore();
timeMillisSpent = solver.getTimeMillisSpent();
```

An example from a ***.java** file in OptaPlanner 8:

```
solver = ...;
solver.addEventListener(event -> {
    solution = event.getNewBestSolution();
    score = event.getNewBestScore();
    timeMillisSpent = event.getTimeMillisSpent();
});
```

Annotation scanning has been removed
Major, Public API

The **<scanAnnotatedClasses/>** directive in the solver configuration was deprecated in 7.x and is now removed.

An example from the ***.xml** file in OptaPlanner 7:

```
<solver>
   ...
   <scanAnnotatedClasses/>
   ...
</solver>
```

An example from the ***.xml** file in OptaPlanner 8:

```
<solver>
   ...
```

```
    <solutionClass>...</solutionClass>
    <entityClass>...</entityClass>
    ...
</solver>
```

New package for @PlanningFactProperty and @PlanningFactCollectionProperty
Major, Public API

The **@PlanningFactProperty** and **@PlanningFactCollectionProperty** annotations now share the same package with other similar annotations, such as **@PlanningSolution**. The old annotations were deprecated in 7.x and removed.

An example from a ***.java** file in OptaPlanner 7:

```
import org.optaplanner.core.api.domain.solution.drools.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.solution.drools.ProblemFactProperty;
```

An example from a ***.java** file in OptaPlanner 8:

```
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.solution.ProblemFactProperty;
```

filterClassList replaced with a single filter class
Minor, Public API

The configuration of **EntitySelector**, **ValueSelector**, and **MoveSelector** now has a single filter class in both the configuration API and the solver configuration XML.

In practice, you do not need multiple selection filter classes often, and you can replace them with a single selection filter class that implements the logic of all of them. Passing a single selection class now requires less boilerplate code.

An example from a ***.java** file in OptaPlanner 7:

```
ValueSelectorConfig valueSelectorConfig = new ValueSelectorConfig();
valueSelectorConfig.setFilterClassList(Collections.singletonList(MySelectionFilterClass.class));
```

An example from a ***.java** file in OptaPlanner 8:

```
ValueSelectorConfig valueSelectorConfig = new ValueSelectorConfig();
valueSelectorConfig.setFilterClass(MySelectionFilterClass.class);
```

Replacing multiple selection filter classes with a single selection filter class

An example from the ***.xml** file in OptaPlanner 7:

```
<swapMoveSelector>
  <entitySelector>
    <filterClass>com.example.FilterA</filterClass>
    <filterClass>com.example.FilterB</filterClass>
  </entitySelector>
</swapMoveSelector>
```

An example from a ***.java** file in OptaPlanner 7:

```java
package com.example;
...
public class FilterA implements SelectionFilter<MySolution, MyPlanningEntity> {

    @Override
    public boolean accept(ScoreDirector<MySolution> scoreDirector, MyPlanningEntity selection) {
        return selection.getValue() < 500;
    }
}
```

```java
package com.example;
...
public class FilterB implements SelectionFilter<MySolution, MyPlanningEntity> {

    @Override
    public boolean accept(ScoreDirector<MySolution> scoreDirector, MyPlanningEntity selection) {
        return selection.getOrder() == Order.ASC;
    }
}
```

An example from the ***.xml** file in OptaPlanner 8:

```xml
<swapMoveSelector>
  <entitySelector>
    <filterClass>com.example.SingleEntityFilter</filterClass>
  </entitySelector>
</swapMoveSelector>
```

An example from a ***.java** file in OptaPlanner 8:

```java
package com.example;
...
public class SingleEntityFilter implements SelectionFilter<MySolution, MyPlanningEntity> {

    @Override
    public boolean accept(ScoreDirector<MySolution> scoreDirector, MyPlanningEntity selection) {
        return selection.getValue() < 500 && selection.getOrder() == Order.ASC;
    }
}
```

AcceptorConfig renamed to LocalSearchAcceptorConfig
Minor

This only impacts the configuration API. The solver configuration XML file remains intact.

Naming consistency with other local-search-specific configuration classes has been implemented.

An example from a ***.java** file in OptaPlanner 7:

```java
LocalSearchPhaseConfig localSearchPhaseConfig = new LocalSearchPhaseConfig()
        .withAcceptorConfig(new AcceptorConfig().withEntityTabuSize(5));
```

An example from a ***.java** file in OptaPlanner 8:

```
LocalSearchPhaseConfig localSearchPhaseConfig = new LocalSearchPhaseConfig()
        .withAcceptorConfig(new LocalSearchAcceptorConfig().withEntityTabuSize(5));
```

Custom properties XML configuration format changes
Minor, Public API

This issue only impacts the solver configuration XML, specifically **<scoreDirectorFactory/>**, **<moveIteratorFactory/>**, **<moveListFactory/>**, **<partitionedSearch/>** and **<customPhase/>**.

This change was made to enforce the structure of the configuration XML in build time.

An example from the ***.xml** file in OptaPlanner 7:

```
<partitionedSearch>
  <solutionPartitionerClass>com.example.MySolutionPartitioner</solutionPartitionerClass>
  <solutionPartitionerCustomProperties>
    <partCount>4</partCount> <!-- a custom property -->
    <minimumProcessListSize>300</minimumProcessListSize> <!-- a custom property -->
  </solutionPartitionerCustomProperties>
</partitionedSearch>
```

An example from the ***.xml** file in OptaPlanner 8:

```
<partitionedSearch>
  <solutionPartitionerClass>com.example.MySolutionPartitioner</solutionPartitionerClass>
  <solutionPartitionerCustomProperties>
    <property name="partCount" value="4"/> <!-- a custom property -->
    <property name="minimumProcessListSize" value="300"/> <!-- a custom property -->
  </solutionPartitionerCustomProperties>
</partitionedSearch>
```

<variableNameInclude/> elements are now wrapped by the <variableNameIncludes/> element
Minor, Public API

This update only impacts the solver configuration XML, specifically the **<swapMoveSelector/>** and **<pillarSwapMoveSelector/>**.

This change was made to enforce the structure of the configuration XML in build time.

An example from the ***.xml** file in OptaPlanner 7:

```
<swapMoveSelector>
  <variableNameInclude>variableA</variableNameInclude>
  <variableNameInclude>variableB</variableNameInclude>
</swapMoveSelector>
```

An example from the ***.xml** file in OptaPlanner 8:

```
<swapMoveSelector>
  <variableNameIncludes>
    <variableNameInclude>variableA</variableNameInclude>
```

```
    <variableNameInclude>variableB</variableNameInclude>
  </variableNameIncludes>
</swapMoveSelector>
```

Solution interface removed

Minor, **Public API**

The **Solution** interface was deprecated and removed. The **AbstractSolution** interface which is only used by Business Central has also been removed.

Remove the **Solution** interface, annotate the **getScore()** method with **@PlanningScore**, and replace the **getProblemFacts()** method with a **@ProblemFactCollectionProperty** annotation directly on every problem fact getter (or field).

An example from a ***.java** file in OptaPlanner 7:

```java
@PlanningSolution
public class CloudBalance implements Solution<HardSoftScore> {

    private List<CloudComputer> computerList;
    ...

    private HardSoftScore score;

    @ValueRangeProvider(id = "computerRange")
    public List<CloudComputer> getComputerList() {...}

    public HardSoftScore getScore() {...}
    public void setScore(HardSoftScore score) {...}

    public Collection<? extends Object> getProblemFacts() {
        List<Object> facts = new ArrayList<Object>();
        facts.addAll(computerList);
        ...
        return facts;
    }

}
```

An example from a ***.java** file in OptaPlanner 8:

```java
@PlanningSolution
public class CloudBalance {

    private List<CloudComputer> computerList;
    ...

    private HardSoftScore score;

    @ValueRangeProvider(id = "computerRange")
    @ProblemFactCollectionProperty
    public List<CloudComputer> getComputerList() {...}

    @PlanningScore
    public HardSoftScore getScore() {...}
```

```
    public void setScore(HardSoftScore score) {...}

}
```

For a single problem fact that is not wrapped in a **Collection**, use the **@ProblemFactProperty** annotation, as shown in the following example, with field annotations this time:

An example from a ***.java** file in OptaPlanner 7:

```
@PlanningSolution
public class CloudBalance implements Solution<HardSoftScore> {

    private CloudParametrization parametrization;
    private List<CloudBuilding> buildingList;
    @ValueRangeProvider(id = "computerRange")
    private List<CloudComputer> computerList;
    ...

    public Collection<? extends Object> getProblemFacts() {
        List<Object> facts = new ArrayList<Object>();
        facts.add(parametrization); // not a Collection
        facts.addAll(buildingList);
        facts.addAll(computerList);
        ...
        return facts;
    }

}
```

An example from a ***.java** file in OptaPlanner 8:

```
@PlanningSolution
public class CloudBalance {

    @ProblemFactProperty
    private CloudParametrization parametrization;
    @ProblemFactCollectionProperty
    private List<CloudBuilding> buildingList;
    @ValueRangeProvider(id = "computerRange")
    @ProblemFactCollectionProperty
    private List<CloudComputer> computerList;
    ...

}
```

Do not add the **@ProblemFactCollectionProperty** annotation on getters (or fields) that have a **@PlanningEntityCollectionProperty** annotation.

BestSolutionChangedEvent: isNewBestSolutionInitialized() removed
Minor, Public API

The **BestSolutionChangedEvent.isNewBestSolutionInitialized()** method has been deprecated and replaced with the
BestSolutionChangedEvent.getNewBestSolution().getScore().isSolutionInitialized() method. The
BestSolutionChangedEvent.isNewBestSolutionInitialized() method has been removed.

An example from a ***.java** file in OptaPlanner 7:

```
public void bestSolutionChanged(BestSolutionChangedEvent<CloudBalance> event) {
    if (event.isEveryProblemFactChangeProcessed()
            && event.isNewBestSolutionInitialized()) {
        ...
    }
}
```

An example from a ***.java** file in OptaPlanner 8:

```
public void bestSolutionChanged(BestSolutionChangedEvent<CloudBalance> event) {
    if (event.isEveryProblemFactChangeProcessed()
            && event.getNewBestSolution().getScore().isSolutionInitialized()) {
        ...
    }
}
```

If you check **isFeasible()**, it checks if the solution is initialized.

An example from a ***.java** file in OptaPlanner 8:

```
public void bestSolutionChanged(BestSolutionChangedEvent<CloudBalance> event) {
    if (event.isEveryProblemFactChangeProcessed()
            // isFeasible() checks isSolutionInitialized() too
            && event.getNewBestSolution().getScore().isFeasible()) {
        ...
    }
}
```

\<valueSelector>: variableName is now an attribute
Minor, **Public API**

When power-tweaking move selectors, such as **\<changeMoveSelector>**, in a use case with multiple planning variables, the **\<variableName>** XML element has been replaced with a **variableName="…"** XML attribute. This change reduces the solver configuration verbosity. After being deprecated for the entire 7.x series, the old way has now been removed.

An example from the ***SolverConfig.xml** and ***BenchmarkConfig.xml** files in OptaPlanner 7:

```
<valueSelector>
  <variableName>room</variableName>
</valueSelector>
```

An example from the ***SolverConfig.xml** and ***BenchmarkConfig.xml** files in OptaPlanner 8:

```
<valueSelector variableName="room"/>
```

Partitioned Search: threadFactoryClass removed
Minor, **Public API**

Because **\<solver>** has supported a **\<threadFactoryClass>** element for some time, the **\<threadFactoryClass>** element under **\<partitionedSearch>** has been removed.

An example from the ***SolverConfig.xml** and ***BenchmarkConfig.xml** files in OptaPlanner 7:

```xml
<solver>
  ...
  <partitionedSearch>
    <threadFactoryClass>...MyAppServerThreadFactory</threadFactoryClass>
    ...
  </partitionedSearch>
</solver>
```

An example from the ***SolverConfig.xml** and ***BenchmarkConfig.xml** files in OptaPlanner 8:

```xml
<solver>
  <threadFactoryClass>...MyAppServerThreadFactory</threadFactoryClass>
  ...
  <partitionedSearch>
    ...
  </partitionedSearch>
</solver>
```

SimpleDoubleScore and HardSoftDoubleScore removed
Minor, **Public API**

The use of double-based score types is not recommended because they can cause score corruption. They have been removed.

An example from a ***.java** file in OptaPlanner 7:

```java
@PlanningSolution
public class MyPlanningSolution {

    private SimpleDoubleScore score;

    ...

}
```

An example from a ***.java** file in OptaPlanner 8:

```java
@PlanningSolution
public class MyPlanningSolution {

    private SimpleLongScore score;

    ...

}
```

Score.toInitializedScore() removed
Minor, **Public API**

The **Score.toInitializedScore()** method was deprecated and replaced with the **Score.withInitScore(int)** method in 7.x and is now removed.

An example from a ***.java** file in OptaPlanner 7:

```
score = score.toInitializedScore();
```

An example from a ***.java** file in OptaPlanner 8:

```
score = score.withInitScore(0);
```

Various justification **Comparators** removed
Minor, **Public API**

The following **Comparator** implementations were deprecated in 7.x and now removed:

- **org.optaplanner.core.api.score.comparator.NaturalScoreComparator**

- **org.optaplanner.core.api.score.constraint.ConstraintMatchScoreComparator**

- **org.optaplanner.core.api.score.constraint.ConstraintMatchTotalScoreComparator**

- **org.optaplanner.core.api.score.constraint.IndictmentScoreComparator**

An example from a ***.java** file in OptaPlanner 7:

```
NaturalScoreComparator comparator = new NaturalScoreComparator();
ConstraintMatchScoreComparator comparator2 = new ConstraintMatchScoreComparator();
```

An example from a ***.java** file in OptaPlanner 8:

```
Comparator<Score> comparator = Comparable::compareTo;
Comparator<ConstraintMatch> comparator2 = Comparator.comparing(ConstraintMatch::getScore);
```

FeasibilityScore removed
Minor, **Public API**

The **FeasibilityScore** interface was deprecated in 7.x and its only method **isFeasible()** moved to the **Score** supertype. The interface has now been removed.

You should refer to **Score**s by their ultimate type, for example **HardSoftScore** instead of to **Score**.

@PlanningEntity.movableEntitySelectionFilter removed
Minor, **Public API**

The **movableEntitySelectionFilter** field on the **@PlanningEntity** annotation was deprecated in 7.x and a new field **pinningFilter** has been introduced with a name that shows the relation to the **@PlanningPin** annotation. This filter implements a new **PinningFilter** interface, returning true if the entity is pinned, and false if movable. The logic of this new filter is therefore inverted as compared to the old filter.

You should update your **@PlanningEntity** annotations by supplying the new filter instead of the old filter. The old filter has now been removed.

An example from a ***.java** file in OptaPlanner 7:

```
@PlanningEntity(movableEntitySelectionFilter = MyMovableEntitySelectionFilter.class)
```

An example from a ***.java** file in OptaPlanner 8:

```
@PlanningEntity(pinningFilter = MyPinningFilter.class)
```

@PlanningVariable.reinitializeVariableEntityFilter removed
Minor, **Public API**

The **reinitializeVariableEntityFilter** field on the **@PlanningVariable** annotation was deprecated in 7.x and now removed.

*ScoreHolder classes turned into interfaces
Minor, **Public API**

In OptaPlanner 7, **ScoreHolder** classes, used exclusively for Drools score calculation, exposed a number of public methods which, if used, allowed the user to unintentionally corrupt or otherwise negatively affect their scores.

In OptaPlanner 8, these methods have been removed and the classes have been turned into interfaces. Most users do not use the removed and potentially harmful methods.

However, if you do use these methods, you will find suitable replacements in the public API in areas of score explanation and constraint configuration.

ValueRangeFactory class now final
Minor

ValueRangeFactory class is a factory class that has only static methods. There is no need for you to extend this class, and it has therefore been made **final**.

An example from a ***.java** file in OptaPlanner 7:

```
class MyValueRangeFactory extends ValueRangeFactory {
    ...
}
```

An example from a ***.java** file in OptaPlanner 8:

```
class MyValueRangeFactory {
    ...
}
```

ConstraintMatchTotal and Indictment are now interfaces
Minor, **Public API**

ConstraintMatchTotal and **Indictment** classes have been converted into interfaces. As a result, their implementations were moved out of the public API, together with methods that allowed them to mutate their state. These methods were never intended for the public API, and therefore there is no replacement for them.

You might still need the instances themselves if you choose to implement **ConstraintMatchAwareIncrementalScoreCalculator**:

```
ConstraintMatchTotal maximumCapacityMatchTotal = new ConstraintMatchTotal(...);
```

An example from a ***.java** file in OptaPlanner 8:

```
ConstraintMatchTotal maximumCapacityMatchTotal = new DefaultConstraintMatchTotal(...);
```

ScoreManager: generic type Score added
Major, **Public API**

The **ScoreManager** and **ScoreExplanation** APIs now have the generic type **Score** to avoid downcasts in your code, for example from **Score** to **HardSoftScore**.

An example from a ***.java** file in OptaPlanner 7:

```
@Inject // or @Autowired
ScoreManager<TimeTable> scoreManager;
```

An example from a ***.java** file in OptaPlanner 8:

```
@Inject // or @Autowired
ScoreManager<TimeTable, HardSoftScore> scoreManager;
```

An example from a ***.java** file in OptaPlanner 7:

```
ScoreExplanation<TimeTable> explanation = scoreManager.explainScore(timeTable);
HardSoftScore score = (HardSoftScore) explanation.getScore();
```

An example from a ***.java** file in OptaPlanner 8:

```
ScoreExplanation<TimeTable, HardSoftScore> explanation =
scoreManager.explainScore(timeTable);
HardSoftScore score = explanation.getScore();
```

ConstraintMatchTotal, ConstraintMatch, and Indictment: generic type Score added
Major

Similar to **ScoreManager** and **ScoreExplanation**, the **ConstraintMatchTotal**, **ConstraintMatch**, and **Indictment** APIs now have a generic type **Score** to avoid downcasts in your code, for example from **Score** to **HardSoftScore**.

An example from a ***.java** file in OptaPlanner 7:

```
ScoreExplanation<TimeTable> explanation = scoreManager.explainScore(timeTable);
Map<String, ConstraintMatchTotal> constraintMatchTotalMap =
scoreExplanation.getConstraintMatchTotalMap();
ConstraintMatchTotal constraintMatchTotal = constraintMatchTotalMap.get(contraintId);
HardSoftScore totalScore = (HardSoftScore) constraintMatchTotal.getScore();
```

An example from a ***.java** file in OptaPlanner 8:

```
ScoreExplanation<TimeTable, HardSoftScore> explanation =
scoreManager.explainScore(timeTable);
Map<String, ConstraintMatchTotal<HardSoftScore>> constraintMatchTotalMap =
scoreExplanation.getConstraintMatchTotalMap();
```

```
    ConstraintMatchTotal<HardSoftScore> constraintMatchTotal =
constraintMatchTotalMap.get(contraintId);
    HardSoftScore totalScore = constraintMatchTotal.getScore();
```

An example from a ***.java** file in OptaPlanner 7:

```
    ScoreExplanation<TimeTable> explanation = scoreManager.explainScore(timeTable);
    Map<Object, Indictment> indictmentMap = scoreExplanation.getIndictmentMap();
    Indictment indictment = indictmentMap.get(lesson);
    HardSoftScore totalScore = (HardSoftScore) indictment.getScore();
```

An example from a ***.java** file in OptaPlanner 8:

```
    ScoreExplanation<TimeTable, HardSoftScore> explanation =
scoreManager.explainScore(timeTable);
    Map<Object, Indictment<HardSoftScore>> indictmentMap = scoreExplanation.getIndictmentMap();
    Indictment<HardSoftScore> indictment = indictmentMap.get(lesson);
    HardSoftScore totalScore = indictment.getScore();
```

ConstraintMatchAwareIncrementalScoreCalculator: generic type Score added

Minor

The interface **ConstraintMatchAwareIncrementalScoreCalculator** now also has a generic type parameter for **Score** to avoid raw type usages of **ConstraintMatchTotal** and **Indictment**.

An example from a ***.java** file in OptaPlanner 7:

```
public class MachineReassignmentIncrementalScoreCalculator
        implements ConstraintMatchAwareIncrementalScoreCalculator<MachineReassignment> {

    @Override
    public Collection<ConstraintMatchTotal> getConstraintMatchTotals() {
        ...
    }


    @Override
    public Map<Object, Indictment> getIndictmentMap() {
        ...
    }

}
```

An example from a ***.java** file in OptaPlanner 8:

```
public class MachineReassignmentIncrementalScoreCalculator
        implements ConstraintMatchAwareIncrementalScoreCalculator<MachineReassignment,
HardSoftLongScore> {

    @Override
    public Collection<ConstraintMatchTotal<HardSoftLongScore>> getConstraintMatchTotals() {
        ...
    }
```

```
   @Override
   public Map<Object, Indictment<HardSoftLongScore>> getIndictmentMap() {
      ...
   }

}
```

AbstractCustomPhaseCommand was removed
Minor, **Public API**

The abstract class **AbstractCustomPhaseCommand** was removed. Any class that extends it should directly implement the **CustomPhaseCommand** interface.

An example from a ***.java** file in OptaPlanner 7:

```
public class DinnerPartySolutionInitializer extends AbstractCustomPhaseCommand<DinnerParty> {

   @Override
   public void changeWorkingSolution(ScoreDirector<DinnerParty> scoreDirector) {
      ...
   }

}
```

An example from a ***.java** file in OptaPlanner 8:

```
public class DinnerPartySolutionInitializer implements CustomPhaseCommand<DinnerParty> {

   @Override
   public void changeWorkingSolution(ScoreDirector<DinnerParty> scoreDirector) {
      ...
   }

}
```

Score calculators moved to the public API
Major

The interfaces **EasyScoreCalculator**, **IncrementalScoreCalculator**, and **ConstraintMatchAwareIncrementalScoreCalculator** have moved to a new package in the public API. Their deprecated counterparts have been removed. The deprecated class **org.optaplanner.core.impl.score.director.incremental.AbstractIncrementalScoreCalculator** has also been removed. Replace the use of the removed interfaces and classes with their counterparts in the public API.

An example from the **EasyScoreCalculator.java** file in OptaPlanner 7:

```
...
import org.optaplanner.core.impl.score.director.easy.EasyScoreCalculator;
...

public class CloudBalancingEasyScoreCalculator implements EasyScoreCalculator<CloudBalance>
```

```
{
    ...
}
```

An example from the **EasyScoreCalculator.java** file in OptaPlanner 8:

```
...
import org.optaplanner.core.api.score.calculator.EasyScoreCalculator;
...

public class CloudBalancingEasyScoreCalculator implements EasyScoreCalculator<CloudBalance,
HardSoftScore> {
    ...
}
```

An example from the **IncrementalScoreCalculator.java** file in OptaPlanner 7:

```
...
import org.optaplanner.core.impl.score.director.incremental.AbstractIncrementalScoreCalculator;
...

public class CloudBalancingIncrementalScoreCalculator extends
AbstractIncrementalScoreCalculator<CloudBalance> {
    ...
}
```

An example from the **IncrementalScoreCalculator.java** file in OptaPlanner 8:

```
...
import org.optaplanner.core.api.score.calculator.IncrementalScoreCalculator;
...

public class CloudBalancingIncrementalScoreCalculator implements
IncrementalScoreCalculator<CloudBalance, HardSoftScore> {
    ...
}
```

An example from the **ConstraintMatchAwareIncrementalScoreCalculator.java** file in OptaPlanner 7:

```
...
import org.optaplanner.core.impl.score.director.incremental.AbstractIncrementalScoreCalculator;
import
org.optaplanner.core.impl.score.director.incremental.ConstraintMatchAwareIncrementalScoreCalculator
;
...

public class CheapTimeConstraintMatchAwareIncrementalScoreCalculator
        extends AbstractIncrementalScoreCalculator<CheapTimeSolution>
        implements ConstraintMatchAwareIncrementalScoreCalculator<CheapTimeSolution> {
    ...
}
```

An example from the **ConstraintMatchAwareIncrementalScoreCalculator.java** file in OptaPlanner 8:

```
...
import org.optaplanner.core.api.score.calculator.ConstraintMatchAwareIncrementalScoreCalculator;
...

public class CheapTimeConstraintMatchAwareIncrementalScoreCalculator
    implements ConstraintMatchAwareIncrementalScoreCalculator<CheapTimeSolution,
HardMediumSoftLongScore> {
  ...
  }
```

PlannerBenchmarkFactory: createFromSolverFactory() removed
Major, Public API

The **PlannerBenchmarkFactory.createFromSolverFactory()** method has been deprecated and replaced with the **PlannerBenchmarkFactory.createFromSolverConfigXmlResource(String)** method. The **PlannerBenchmarkFactory.createFromSolverFactory()** method has been removed.

An example from a ***.java** file in OptaPlanner 7:

```
SolverFactory<CloudBalance> solverFactory = SolverFactory.createFromXmlResource(
    ".../cloudBalancingSolverConfig.xml");
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.createFromSolverFactory(solverFactory);
```

An example from a ***.java** file in OptaPlanner 8:

```
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.createFromSolverConfigXmlResource(
    ".../cloudBalancingSolverConfig.xml");
```

If you programmatically adjust the solver configuration, you can use **PlannerBenchmarkConfig.createFromSolverConfig(SolverConfig)** and then **PlannerBenchmarkFactory.create(PlannerBenchmarkConfig)** instead.

PlannerBenchmarkFactory: getPlannerBenchmarkConfig() removed
Minor, Public API

The **PlannerBenchmarkFactory.getPlannerBenchmarkConfig()** method has been deprecated and replaced with the **PlannerBenchmarkFactory.create(PlannerBenchmarkConfig)** method. A **PlannerBenchmarkConfig** instance is now instantiated before a **PlannerBenchmarkFactory** instance is instantiated. This order is more logical. **PlannerBenchmarkFactory.getPlannerBenchmarkConfig()** has been removed.

An example from a ***.java** file in OptaPlanner 7:

```
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.createFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
PlannerBenchmarkConfig benchmarkConfig = benchmarkFactory.getPlannerBenchmarkConfig();
...
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();
```

An example from a ***.java** file in OptaPlanner 8:

```
PlannerBenchmarkConfig benchmarkConfig = PlannerBenchmarkConfig.createFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
...
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.create(benchmarkConfig);
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();
```

XML <plannerBenchmark/> root element now belongs to the https://www.optaplanner.org/xsd/benchmark namespace

Minor, **Public API**

OptaPlanner now provides an XML Schema Definition (XSD) for the benchmark configuration. Although OptaPlanner keeps compatibility with earlier versions of the existing XML configuration, migrating to the XSD is strongly recommended because OptaPlanner might support only valid configuration XML in the future.

An example from the ***BenchmarkConfig.xml** file in OptaPlanner 7:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<plannerBenchmark>
  ...
</plannerBenchmark>
```

An example from the ***BenchmarkConfig.xml** file in OptaPlanner 8:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<plannerBenchmark xmlns="https://www.optaplanner.org/xsd/benchmark"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.optaplanner.org/xsd/benchmark
https://www.optaplanner.org/xsd/benchmark/benchmark.xsd">
  ...
</plannerBenchmark>
```

Using the XSD might require reordering some of the XML elements of the configuration. Use code completion in the IDE to migrate to a valid XML.

ProblemBenchmarksConfig: xStreamAnnotatedClass removed

Major, **Public API**

The **<xStreamAnnotatedClass/>** has been removed from the **<problemBenchmarks/>** configuration together with the corresponding **getXStreamAnnotatedClassList()** and **setXStreamAnnotatedClassList()** methods in the **ProblemBenchmarksConfig** class.

An example from a ***.java** file in OptaPlanner 7:

```java
ProblemBenchmarksConfig problemBenchmarksConfig = new ProblemBenchmarksConfig();
problemBenchmarksConfig.setXStreamAnnotatedClassList(MySolution.class);
```

An example from a ***.java** file in OptaPlanner 8:

```java
package com.example;
...
public class MySolutionFileIO extends XStreamSolutionFileIO<MySolution> {
  public MySolutionFileIO() {
```

```
        super(MySolution.class);
    }
}

...

ProblemBenchmarksConfig problemBenchmarksConfig = new ProblemBenchmarksConfig();
problemBenchmarksConfig.setSolutionFileIOClass(MySolutionFileIO.class);
```

An example from the ***BenchmarkConfig.xml** file in OptaPlanner 7:

```xml
<plannerBenchmark>
...
  <solverBenchmark>
    <problemBenchmarks>
      <xStreamAnnotatedClass>com.example.MySolution</xStreamAnnotatedClass>

      ...
    </problemBenchmarks>
    ...
  </solverBenchmark>
...
</plannerBenchmark>
```

An example from the ***BenchmarkConfig.xml** file in OptaPlanner 8:

```xml
<plannerBenchmark>
...
  <solverBenchmark>
    <problemBenchmarks>
      <!-- See the "After in *.java" section to create the MySolutionFileIO. -->
      <solutionFileIOClass>com.example.MySolutionFileIO</solutionFileIOClass>

      ...
    </problemBenchmarks>
    ...
  </solverBenchmark>
...
</plannerBenchmark>
```

BenchmarkAggregatorFrame: createAndDisplay(PlannerBenchmarkFactory) removed
Minor

The **BenchmarkAggregatorFrame.createAndDisplay(PlannerBenchmarkFactory)** method has been deprecated and replaced with the
BenchmarkAggregatorFrame.createAndDisplayFromXmlResource(String) method. The
BenchmarkAggregatorFrame.createAndDisplay(PlannerBenchmarkFactory) method has been removed.

An example from a ***.java** file in OptaPlanner 7:

```java
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.createFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
BenchmarkAggregatorFrame.createAndDisplay(benchmarkFactory);
```

An example from a ***.java** file in OptaPlanner 8:

```
BenchmarkAggregatorFrame.createAndDisplayFromXmlResource(
    ".../cloudBalancingBenchmarkConfig.xml");
```

If you programmatically adjust the benchmark configuration, you can use
BenchmarkAggregatorFrame.createAndDisplay(PlannerBenchmarkConfig) instead.

Removed JavaScript expression support in configuration
Minor

Various elements of both the solver configuration and benchmark configuration no longer support
nested JavaScript expressions. You must replace these with either auto-configuration or with integer
constants.

An example from the **solverConfig.xml** file in OptaPlanner 7:

```
<solver>
    ...
    <moveThreadCount>availableProcessorCount - 1</moveThreadCount>
    ...
</solver>
```

An example from the `solverConfig.xml`file in OptaPlanner 8:

```
<solver>
    ...
    <moveThreadCount>1</moveThreadCount> <!-- Alternatively, use "AUTO" or omit entirely. -->
    ...
</solver>
```

An example from the **benchmarkConfig.xml** file in OptaPlanner 7:

```
<plannerBenchmark>
    ...
    <parallelBenchmarkCount>availableProcessorCount - 1</parallelBenchmarkCount>
    ...
</plannerBenchmark>
```

An example from the **benchmarkConfig.xml** file in OptaPlanner 8:

```
<plannerBenchmark>
    ...
    <parallelBenchmarkCount>1</parallelBenchmarkCount> <!-- Alternatively, use "AUTO" or omit
entirely. -->
    ...
</plannerBenchmark>
```

Removed the deprecated variable listeners
Major, Public API

The deprecated interface **VariableListener** from package
org.optaplanner.core.impl.domain.variable.listener has ben removed, along with the deprecated
interface **StatefulVariableListener** and the deprecated class **VariableListenerAdapter** in that same
package. Use an interface **VariableListener** from package **org.optaplanner.core.api.domain.variable**
instead.

An example of a **VariableListener.java** file in OptaPlanner 7:

```
...
import org.optaplanner.core.impl.domain.variable.listener.VariableListenerAdapter;
...

public class MyVariableListener extends VariableListenerAdapter<Object> {

  ...

  @Override
  void afterEntityRemoved(ScoreDirector scoreDirector, Object entity);
    ...
  }

  ...
}
```

An example from a **VariableListener.java** file in OptaPlanner 8:

```
...
import org.optaplanner.core.api.domain.variable.VariableListener;
...

public class MyVariableListener extends VariableListener<MySolution, Object> {

  ...

  @Override
  void afterEntityRemoved(ScoreDirector<MySolution> scoreDirector, Object entity);
    ...
  }

  ...
}
```

An example of a **StatefulVariableListener.java** file in OptaPlanner 7:

```
...
import org.optaplanner.core.impl.domain.variable.listener.StatefulVariableListener;
...

public class MyStatefulVariableListener implements StatefulVariableListener<Object> {

  ...

  @Override
  public void clearWorkingSolution(ScoreDirector scoreDirector) {
    ...
  }

  ...
}
```

An example from the **StatefulVariableListener.java** file in OptaPlanner 8:

```
...
import org.optaplanner.core.api.domain.variable.VariableListener;
...

public class MyStatefulVariableListener implements VariableListener<MySolution, Object> {

  ...

  @Override
  public void close() {
    ...
  }

  ...
}
```

CHAPTER 2. CHANGES BETWEEN OPTAPLANNER 8.2.0 AND OPTAPLANNER 8.3.0

The changes listed in this section were made between OptaPlanner 8.2.0 and OptaPlanner 8.3.0.

ConstraintMatch.compareTo() inconsistent with equals()
Minor

The **equals()** override in **ConstraintMatch** has been removed. As a result, two different **ConstraintMatch** instances are never considered equal. This contrasts with the **compareTo()** method, which continues to consider two instances equal if all their field values are equal.

PART II. GETTING STARTED WITH RED HAT BUILD OF OPTAPLANNER

As a business rules developer, you can use Red Hat build of OptaPlanner to find the optimal solution to planning problems based on a set of limited resources and under specific constraints.

Use this document to start developing solvers with OptaPlanner.

CHAPTER 3. INTRODUCTION TO RED HAT BUILD OF OPTAPLANNER

OptaPlanner is a lightweight, embeddable planning engine that optimizes planning problems. It helps normal Java programmers solve planning problems efficiently, and it combines optimization heuristics and metaheuristics with very efficient score calculations.

For example, OptaPlanner helps solve various use cases:

- *Employee/Patient Rosters*: It helps create time tables for nurses and keeps track of patient bed management.

- *Educational Timetables*: It helps schedule lessons, courses, exams, and conference presentations.

- *Shop Schedules*: It tracks car assembly lines, machine queue planning, and workforce task planning.

- *Cutting Stock*: It minimizes waste by reducing the consumption of resources such as paper and steel.

Every organization faces planning problems; that is, they provide products and services with a limited set of constrained resources (employees, assets, time, and money).

OptaPlanner is open source software under the Apache Software License 2.0. It is 100% pure Java and runs on most Java virtual machines.

3.1. PLANNING PROBLEMS

A *planning problem* has an optimal goal, based on limited resources and under specific constraints. Optimal goals can be any number of things, such as:

- Maximized profits – the optimal goal results in the highest possible profit.

- Minimized ecological footprint – the optimal goal has the least amount of environmental impact.

- Maximized satisfaction for employees or customers – the optimal goal prioritizes the needs of employees or customers.

The ability to achieve these goals relies on the number of resources available. For example, the following resources might be limited:

- The number of people

- Amount of time

- Budget

- Physical assets, for example, machinery, vehicles, computers, buildings, and so on

You must also take into account the specific constraints related to these resources, such as the number of hours a person works, their ability to use certain machines, or compatibility between pieces of equipment.

Red Hat build of OptaPlanner helps Java programmers solve constraint satisfaction problems efficiently. It combines optimization heuristics and metaheuristics with efficient score calculation.

3.2. NP-COMPLETENESS IN PLANNING PROBLEMS

The provided use cases are *probably* NP-complete or NP-hard, which means the following statements apply:

- It is easy to verify a given solution to a problem in reasonable time.

- There is no simple way to find the optimal solution of a problem in reasonable time.

The implication is that solving your problem is probably harder than you anticipated, because the two common techniques do not suffice:

- A brute force algorithm (even a more advanced variant) takes too long.

- A quick algorithm, for example in the bin packing problem, *putting in the largest items first* returns a solution that is far from optimal.

By using advanced optimization algorithms, OptaPlanner finds a good solution in reasonable time for such planning problems.

3.3. SOLUTIONS TO PLANNING PROBLEMS

A planning problem has a number of solutions.

Several categories of solutions are:

Possible solution

A possible solution is any solution, whether or not it breaks any number of constraints. Planning problems often have an incredibly large number of possible solutions. Many of those solutions are not useful.

Feasible solution

A feasible solution is a solution that does not break any (negative) hard constraints. The number of feasible solutions are relative to the number of possible solutions. Sometimes there are no feasible solutions. Every feasible solution is a possible solution.

Optimal solution

An optimal solution is a solution with the highest score. Planning problems usually have a few optimal solutions. They always have at least one optimal solution, even in the case that there are no feasible solutions and the optimal solution is not feasible.

Best solution found

The best solution is the solution with the highest score found by an implementation in a given amount of time. The best solution found is likely to be feasible and, given enough time, it's an optimal solution.

Counterintuitively, the number of possible solutions is huge (if calculated correctly), even with a small data set.

In the examples provided in the **planner-engine** distribution folder, most instances have a large number of possible solutions. As there is no guaranteed way to find the optimal solution, any implementation is forced to evaluate at least a subset of all those possible solutions.

OptaPlanner supports several optimization algorithms to efficiently wade through that incredibly large number of possible solutions.

Depending on the use case, some optimization algorithms perform better than others, but it is impossible to know in advance. Using OptaPlanner, you can switch the optimization algorithm by changing the solver configuration in a few lines of XML or code.

3.4. CONSTRAINTS ON PLANNING PROBLEMS

Usually, a planning problem has minimum two levels of constraints:

- A *(negative) hard constraint* must not be broken.
  For example, one teacher can not teach two different lessons at the same time.

- A *(negative) soft constraint* should not be broken if it can be avoided.
  For example, Teacher A does not like to teach on Friday afternoons.

Some problems also have positive constraints:

- A *positive soft constraint (or reward)* should be fulfilled if possible.
  For example, Teacher B likes to teach on Monday mornings.

Some basic problems only have hard constraints. Some problems have three or more levels of constraints, for example, hard, medium, and soft constraints.

These constraints define the *score calculation* (otherwise known as the *fitness function*) of a planning problem. Each solution of a planning problem is graded with a score. With OptaPlanner, score constraints are written in an object oriented language such as Java, or in Drools rules.

This type of code is flexible and scalable.

CHAPTER 4. EXAMPLES PROVIDED WITH RED HAT BUILD OF OPTAPLANNER

Several Red Hat build of OptaPlanner examples are shipped with Red Hat Decision Manager. You can review the code for examples and modify it as necessary to suit your needs.

> **NOTE**
>
> Red Hat does not provide support for the example code included in the Red Hat Decision Manager distribution.

Some of the OptaPlanner examples solve problems that are presented in academic contests. The **Contest** column in the following table lists the contests. It also identifies an example as being either *realistic* or *unrealistic* for the purpose of a contest. A *realistic contest* is an official, independent contest that meets the following standards:

- Clearly defined real-world use cases

- Real-world constraints

- Multiple real-world datasets

- Reproducible results within a specific time limit on specific hardware

- Serious participation from the academic and/or enterprise Operations Research community.

Realistic contests provide an objective comparison of OptaPlanner with competitive software and academic research.

Table 4.1. Examples overview

Example	Domain	Size	Contest	Directory name
N queens	1 entity class (1 variable)	Entity ⇐ **256** Value ⇐ **256** Search space ⇐ **10^616**	Pointless (cheatable)	**nqueens**
Cloud balancing	1 entity class (1 variable)	Entity ⇐ **2400** Value ⇐ **800** Search space ⇐ **10^6967**	No (Defined by us)	**cloudbalancing**
Traveling salesman	1 entity class (1 chained variable)	Entity ⇐ **980** Value ⇐ **980** Search space ⇐ **10^2504**	Unrealistic TSP web	**tsp**

Example	Domain	Size	Contest	Directory name
Tennis club scheduling	1 entity class (1 variable)	Entity ⇐ **72** Value ⇐ **7** Search space ⇐ **10^60**	No (Defined by us)	**tennis**
Meeting scheduling	1 entity class (2 variables)	Entity ⇐ **10** Value ⇐ **320** and ⇐ **5** Search space ⇐ **10^320**	No (Defined by us)	**meetingscheduling**
Course timetabling	1 entity class (2 variables)	Entity ⇐ **434** Value ⇐ **25** and ⇐ **20** Search space ⇐ **10^1171**	Realistic ITC 2007 track 3	**curriculumCourse**
Machine reassignment	1 entity class (1 variable)	Entity ⇐ **50000** Value ⇐ **5000** Search space ⇐ **10^184948**	Nearly realistic ROADEF 2012	**machineReassignment**
Vehicle routing	1 entity class (1 chained variable) 1 shadow entity class (1 automatic shadow variable)	Entity ⇐ **2740** Value ⇐ **2795** Search space ⇐ **10^8380**	Unrealistic VRP web	**vehiclerouting**
Vehicle routing with time windows	All of Vehicle routing (1 shadow variable)	Entity ⇐ **2740** Value ⇐ **2795** Search space ⇐ **10^8380**	Unrealistic VRP web	**vehiclerouting**
Project job scheduling	1 entity class (2 variables) (1 shadow variable)	Entity ⇐ **640** Value ⇐ **?** and ⇐ **?** Search space ⇐ **?**	Nearly realistic MISTA 2013	**projectjobscheduling**

Example	Domain	Size	Contest	Directory name
Task assigning	1 entity class (1 chained variable) (1 shadow variable) 1 shadow entity class (1 automatic shadow variable)	Entity ⇐ **500** Value ⇐ **520** Search space ⇐ **10^1168**	No Defined by us	**taskassigning**
Exam timetabling	2 entity classes (same hierarchy) (2 variables)	Entity ⇐ **1096** Value ⇐ **80** and ⇐ **49** Search space ⇐ **10^3374**	Realistic ITC 2007 track 1	**examination**
Nurse rostering	1 entity class (1 variable)	Entity ⇐ **752** Value ⇐ **50** Search space ⇐ **10^1277**	Realistic INRC 2010	**nurserostering**
Traveling tournament	1 entity class (1 variable)	Entity ⇐ **1560** Value ⇐ **78** Search space ⇐ **10^2301**	Unrealistic TTP	**travelingtournament**
Cheap time scheduling	1 entity class (2 variables)	Entity ⇐ **500** Value ⇐ **100** and ⇐ **288** Search space ⇐ **10^20078**	Nearly realistic ICON Energy	**cheaptimescheduling**
Investment	1 entity class (1 variable)	Entity ⇐ **11** Value = **1000** Search space ⇐ **10^4**	No Defined by us	**investment**
Conference scheduling	1 entity class (2 variables)	Entity ⇐ **216** Value ⇐ **18** and ⇐ **20** Search space ⇐ **10^552**	No Defined by us	**conferencescheduling**

Example	Domain	Size	Contest	Directory name
Rock tour	1 entity class (1 chained variable) (4 shadow variables) 1 shadow entity class (1 automatic shadow variable)	Entity ⇐ **47** Value ⇐ **48** Search space ⇐ **10^59**	No Defined by us	**rocktour**
Flight crew scheduling	1 entity class (1 variable) 1 shadow entity class (1 automatic shadow variable)	Entity ⇐ **4375** Value ⇐ **750** Search space ⇐ **10^12578**	No Defined by us	**flightcrewscheduling**

4.1. N QUEENS

Place *n* queens on a *n* sized chessboard so that no two queens can attack each other. The most common *n* queens puzzle is the eight queens puzzle, with *n = 8*:
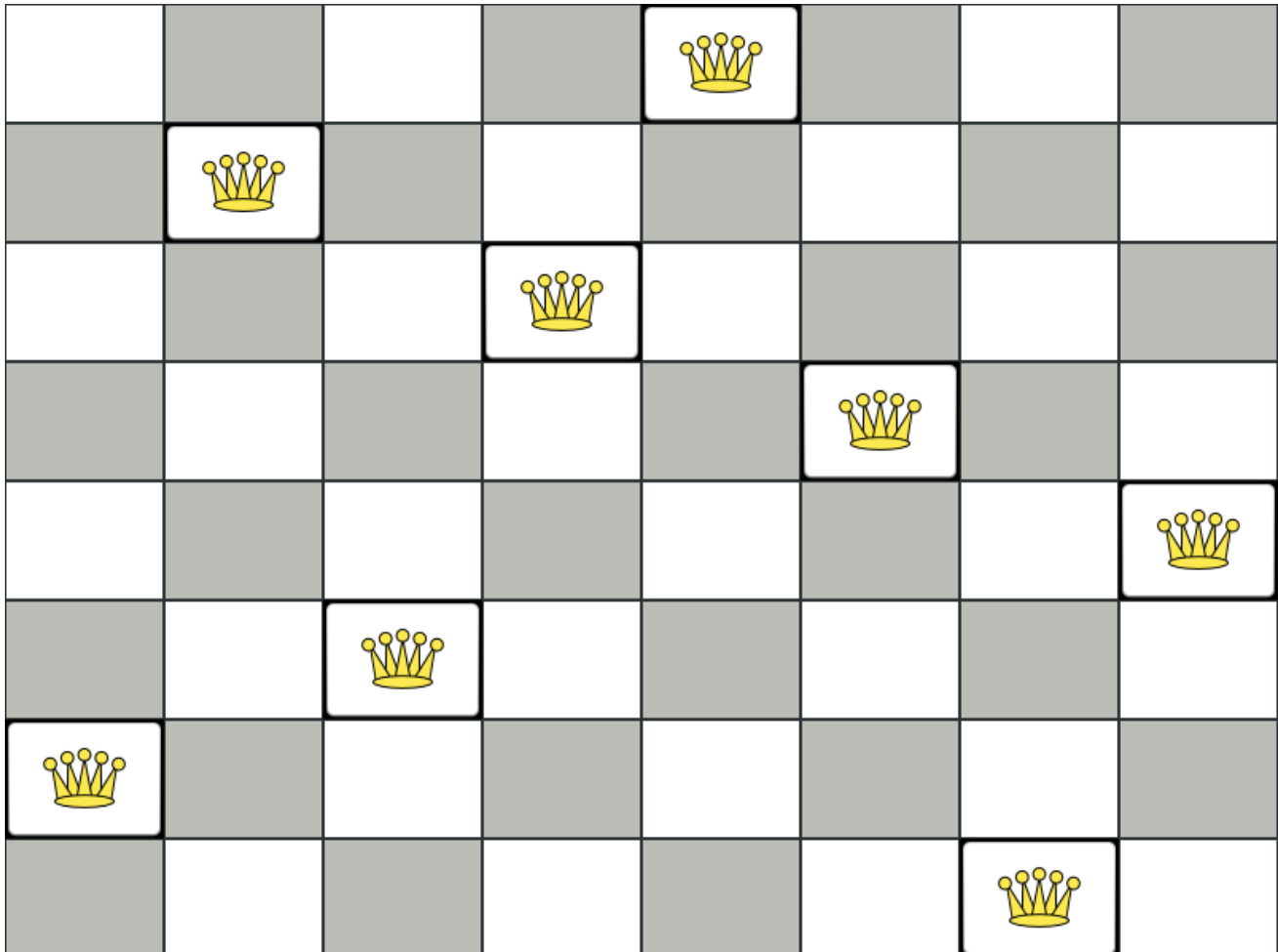
Constraints:

- Use a chessboard of n columns and n rows.

- Place n queens on the chessboard.

- No two queens can attack each other. A queen can attack any other queen on the same horizontal, vertical, or diagonal line.

This documentation heavily uses the four queens puzzle as the primary example.

A proposed solution could be:

Figure 4.1. A wrong solution for the Four queens puzzle
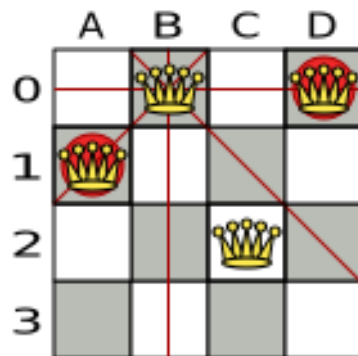
The above solution is wrong because queens **A1** and **B0** can attack each other (so can queens **B0** and **D0**). Removing queen **B0** would respect the "no two queens can attack each other" constraint, but would break the "place n queens" constraint.

Below is a correct solution:

Figure 4.2. A correct solution for the Four queens puzzle
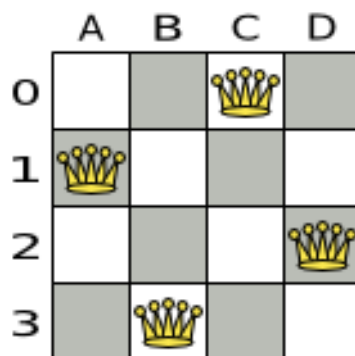
All the constraints have been met, so the solution is correct.

Note that most n queens puzzles have multiple correct solutions. We will focus on finding a single correct solution for a given n, not on finding the number of possible correct solutions for a given n.

Problem size

```
4queens   has   4 queens with a search space of    256.
8queens   has   8 queens with a search space of   10^7.
16queens  has  16 queens with a search space of  10^19.
```

> 32queens has 32 queens with a search space of 10^48.
> 64queens has 64 queens with a search space of 10^115.
> 256queens has 256 queens with a search space of 10^616.

The implementation of the *n* queens example has not been optimized because it functions as a beginner example. Nevertheless, it can easily handle 64 queens. With a few changes it has been shown to easily handle 5000 queens and more.

4.1.1. Domain model for N queens

This example uses the domain model to solve the four queens problem.

- **Creating a Domain Model**
  A good domain model will make it easier to understand and solve your planning problem.

  This is the domain model for the *n* queens example:

  ```java
  public class Column {

      private int index;

      // ... getters and setters
  }
  ```

  ```java
  public class Row {

      private int index;

      // ... getters and setters
  }
  ```

  ```java
  public class Queen {

      private Column column;
      private Row row;

      public int getAscendingDiagonalIndex() {...}
      public int getDescendingDiagonalIndex() {...}

      // ... getters and setters
  }
  ```

- **Calculating the Search Space.**
  A **Queen** instance has a **Column** (for example: 0 is column A, 1 is column B, ...) and a **Row** (its row, for example: 0 is row 0, 1 is row 1, ...).

  The ascending diagonal line and the descending diagonal line can be calculated based on the column and the row.

  The column and row indexes start from the upper left corner of the chessboard.

  ```java
  public class NQueens {

      private int n;
  ```

```
        private List<Column> columnList;
        private List<Row> rowList;

        private List<Queen> queenList;

        private SimpleScore score;

        // ... getters and setters
}
```

- **Finding the Solution**
  A single **NQueens** instance contains a list of all **Queen** instances. It is the **Solution** implementation which will be supplied to, solved by, and retrieved from the Solver.

Notice that in the four queens example, NQueens's **getN()** method will always return four.

Figure 4.3. A solution for Four Queens
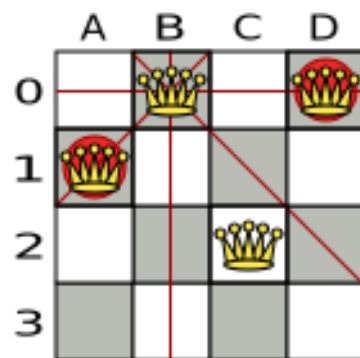
Table 4.2. Details of the solution in the domain model

	columnIndex	rowIndex	ascendingDiagonalIndex (columnIndex + rowIndex)	descendingDiagonalIndex (columnIndex - rowIndex)
A1	0	1	1 (**)	-1
B0	1	0 (*)	1 (**)	1
C2	2	2	4	0
D0	3	0 (*)	3	3

When two queens share the same column, row or diagonal line, such as (*) and (**), they can attack each other.

4.2. CLOUD BALANCING

For information about this example, see *Red Hat build of OptaPlanner quick start guides* .

4.3. TRAVELING SALESMAN (TSP – TRAVELING SALESMAN PROBLEM)

Given a list of cities, find the shortest tour for a salesman that visits each city exactly once.

The problem is defined by Wikipedia. It is one of the most intensively studied problems in computational mathematics. Yet, in the real world, it is often only part of a planning problem, along with other constraints, such as employee shift rostering constraints.

Problem size

```
dj38     has  38 cities with a search space of   10^43.
europe40 has  40 cities with a search space of   10^46.
st70     has  70 cities with a search space of   10^98.
pcb442   has 442 cities with a search space of  10^976.
lu980    has 980 cities with a search space of 10^2504.
```

Problem difficulty

Despite TSP's simple definition, the problem is surprisingly hard to solve. Because it is an NP-hard problem (like most planning problems), the optimal solution for a specific problem dataset can change a lot when that problem dataset is slightly altered:
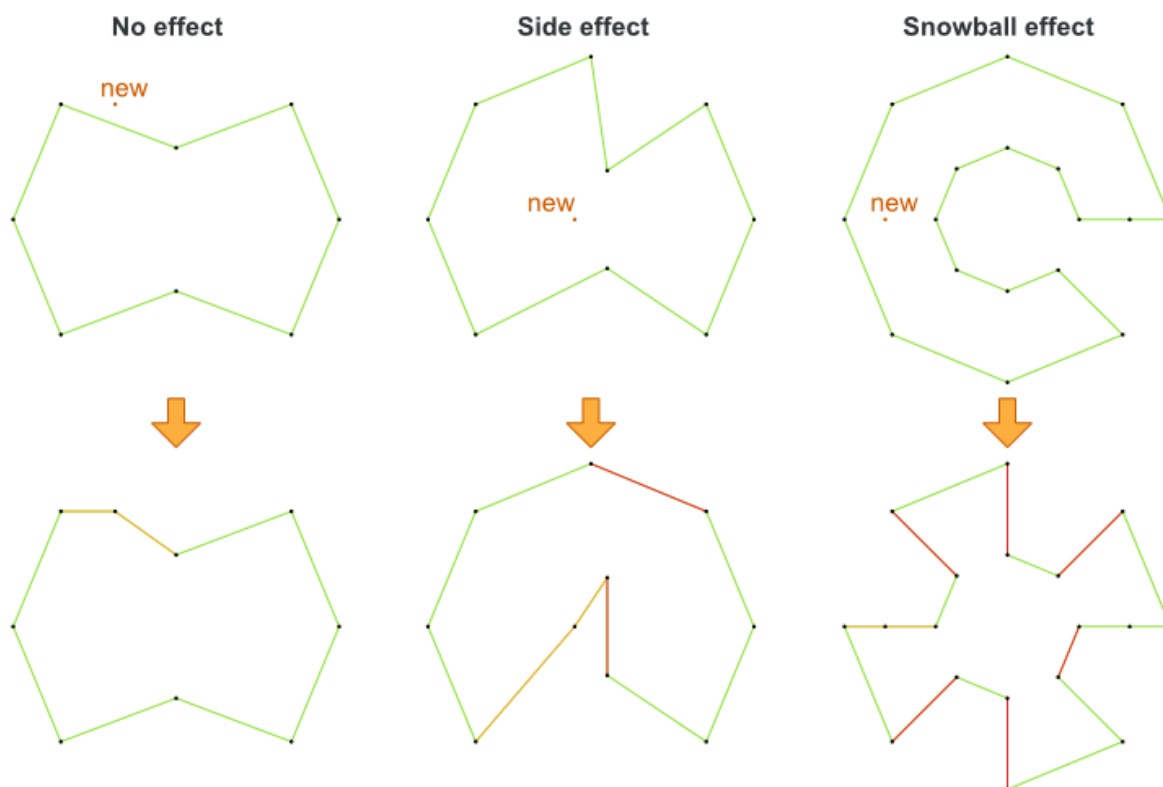
4.4. TENNIS CLUB SCHEDULING

Every week the tennis club has four teams playing round robin against each other. Assign those four spots to the teams fairly.

Hard constraints:

- Conflict: A team can only play once per day.

- Unavailability: Some teams are unavailable on some dates.

Medium constraints:

- Fair assignment: All teams should play an (almost) equal number of times.

Soft constraints:

- Evenly confrontation: Each team should play against every other team an equal number of times.

Problem size

> munich-7teams has 7 teams, 18 days, 12 unavailabilityPenalties and 72 teamAssignments with a search space of 10^60.

Figure 4.4. Domain model
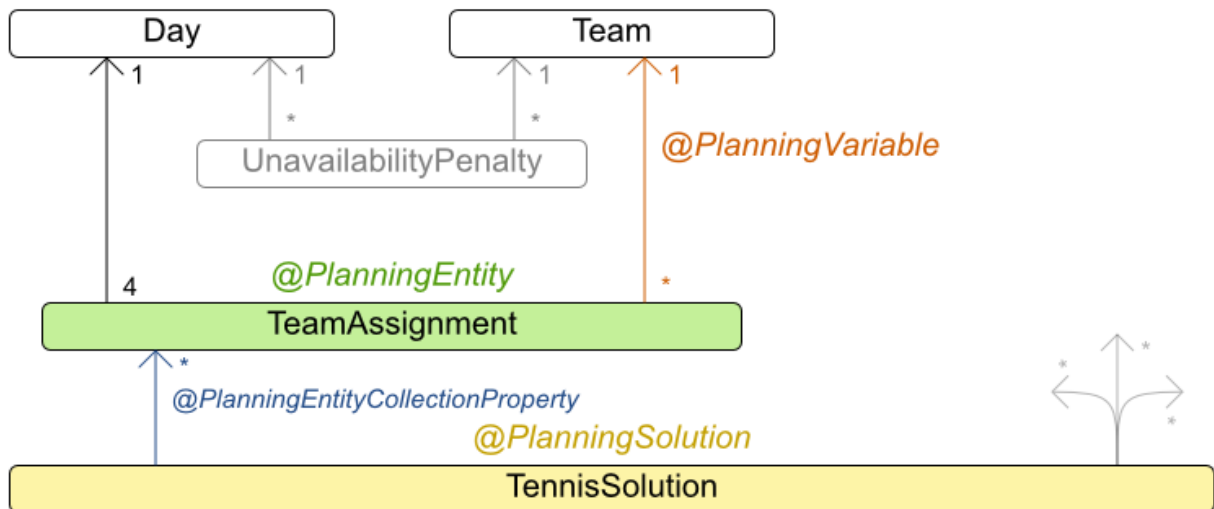
4.5. MEETING SCHEDULING

Assign each meeting to a starting time and a room. Meetings have different durations.

Hard constraints:

- Room conflict: two meetings must not use the same room at the same time.

- Required attendance: A person cannot have two required meetings at the same time.

- Required room capacity: A meeting must not be in a room that doesn't fit all of the meeting's attendees.

- Start and end on same day: A meeting shouldn't be scheduled over multiple days.

Medium constraints:

- Preferred attendance: A person cannot have two preferred meetings at the same time, nor a preferred and a required meeting at the same time.

Soft constraints:

- Sooner rather than later: Schedule all meetings as soon as possible.

- A break between meetings: Any two meetings should have at least one time grain break between them.

- Overlapping meetings: To minimize the number of meetings in parallel so people don't have to choose one meeting over the other.

- Assign larger rooms first: If a larger room is available any meeting should be assigned to that room in order to accommodate as many people as possible even if they haven't signed up to that meeting.

- Room stability: If a person has two consecutive meetings with two or less time grains break between them they better be in the same room.

Problem size

```
50meetings-160timegrains-5rooms  has  50 meetings, 160 timeGrains and 5 rooms with a search
space of 10^145.
100meetings-320timegrains-5rooms has 100 meetings, 320 timeGrains and 5 rooms with a search
space of 10^320.
200meetings-640timegrains-5rooms has 200 meetings, 640 timeGrains and 5 rooms with a search
space of 10^701.
400meetings-1280timegrains-5rooms has 400 meetings, 1280 timeGrains and 5 rooms with a search
space of 10^1522.
800meetings-2560timegrains-5rooms has 800 meetings, 2560 timeGrains and 5 rooms with a search
space of 10^3285.
```

4.6. COURSE TIMETABLING (ITC 2007 TRACK 3 – CURRICULUM COURSE SCHEDULING)

Schedule each lecture into a timeslot and into a room.

Hard constraints:

- Teacher conflict: A teacher must not have two lectures in the same period.

- Curriculum conflict: A curriculum must not have two lectures in the same period.

- Room occupancy: Two lectures must not be in the same room in the same period.

- Unavailable period (specified per dataset): A specific lecture must not be assigned to a specific period.

Soft constraints:

- Room capacity: A room's capacity should not be less than the number of students in its lecture.

- Minimum working days: Lectures of the same course should be spread out into a minimum number of days.

- Curriculum compactness: Lectures belonging to the same curriculum should be adjacent to each other (so in consecutive periods).

- Room stability: Lectures of the same course should be assigned to the same room.

The problem is defined by the International Timetabling Competition 2007 track 3 .

Problem size

```
comp01 has 24 teachers,  14 curricula,  30 courses, 160 lectures, 30 periods,  6 rooms and   53
unavailable period constraints with a search space of  10^360.
comp02 has 71 teachers,  70 curricula,  82 courses, 283 lectures, 25 periods, 16 rooms and  513
unavailable period constraints with a search space of  10^736.
comp03 has 61 teachers,  68 curricula,  72 courses, 251 lectures, 25 periods, 16 rooms and  382
unavailable period constraints with a search space of  10^653.
comp04 has 70 teachers,  57 curricula,  79 courses, 286 lectures, 25 periods, 18 rooms and  396
unavailable period constraints with a search space of  10^758.
comp05 has 47 teachers, 139 curricula,  54 courses, 152 lectures, 36 periods,  9 rooms and  771
unavailable period constraints with a search space of  10^381.
comp06 has 87 teachers,  70 curricula, 108 courses, 361 lectures, 25 periods, 18 rooms and  632
unavailable period constraints with a search space of  10^957.
comp07 has 99 teachers,  77 curricula, 131 courses, 434 lectures, 25 periods, 20 rooms and  667
unavailable period constraints with a search space of 10^1171.
comp08 has 76 teachers,  61 curricula,  86 courses, 324 lectures, 25 periods, 18 rooms and  478
unavailable period constraints with a search space of  10^859.
comp09 has 68 teachers,  75 curricula,  76 courses, 279 lectures, 25 periods, 18 rooms and  405
unavailable period constraints with a search space of  10^740.
comp10 has 88 teachers,  67 curricula, 115 courses, 370 lectures, 25 periods, 18 rooms and  694
unavailable period constraints with a search space of  10^981.
comp11 has 24 teachers,  13 curricula,  30 courses, 162 lectures, 45 periods,  5 rooms and   94
unavailable period constraints with a search space of  10^381.
comp12 has 74 teachers, 150 curricula,  88 courses, 218 lectures, 36 periods, 11 rooms and 1368
unavailable period constraints with a search space of  10^566.
comp13 has 77 teachers,  66 curricula,  82 courses, 308 lectures, 25 periods, 19 rooms and  468
unavailable period constraints with a search space of  10^824.
comp14 has 68 teachers,  60 curricula,  85 courses, 275 lectures, 25 periods, 17 rooms and  486
unavailable period constraints with a search space of  10^722.
```

Figure 4.5. Domain model

Curriculum course class diagram
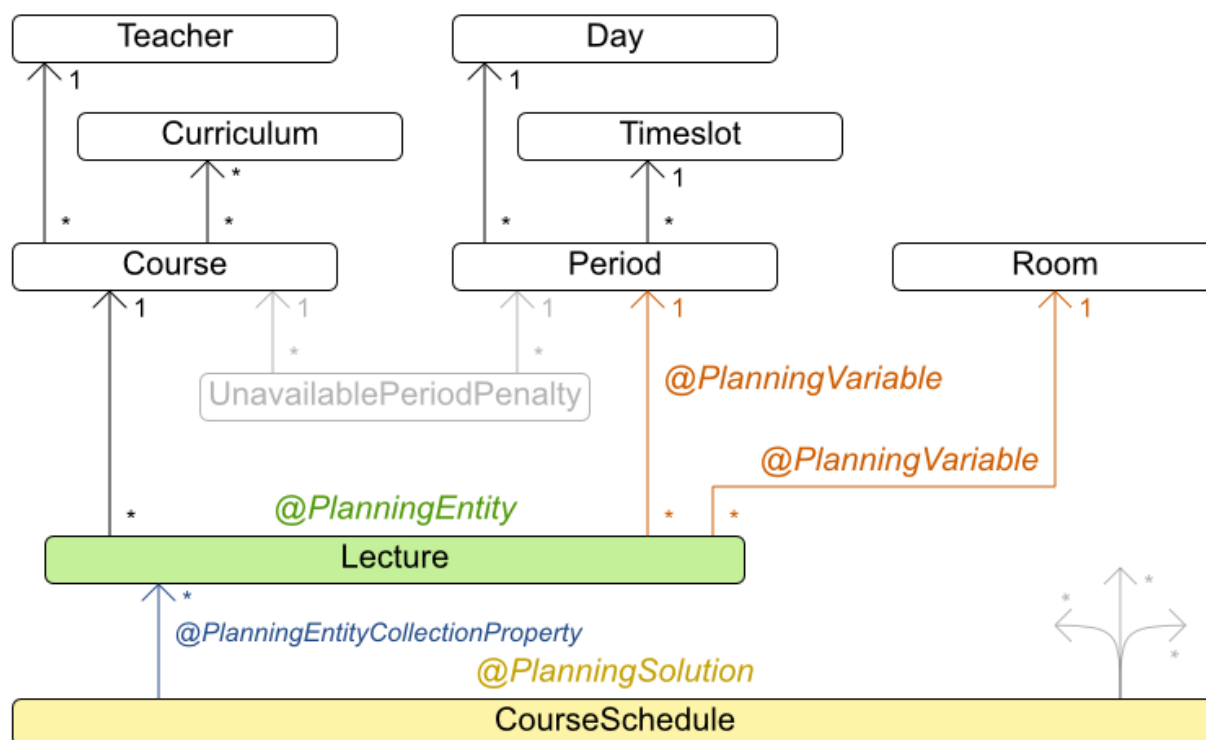
4.7. MACHINE REASSIGNMENT (GOOGLE ROADEF 2012)

Assign each process to a machine. All processes already have an original (unoptimized) assignment. Each process requires an amount of each resource (such as CPU or RAM). This is a more complex version of the Cloud Balancing example.

Hard constraints:

- Maximum capacity: The maximum capacity for each resource for each machine must not be exceeded.

- Conflict: Processes of the same service must run on distinct machines.

- Spread: Processes of the same service must be spread out across locations.

- Dependency: The processes of a service depending on another service must run in the neighborhood of a process of the other service.

- Transient usage: Some resources are transient and count towards the maximum capacity of both the original machine as the newly assigned machine.

Soft constraints:

- Load: The safety capacity for each resource for each machine should not be exceeded.

- Balance: Leave room for future assignments by balancing the available resources on each machine.

- Process move cost: A process has a move cost.

- Service move cost: A service has a move cost.

- Machine move cost: Moving a process from machine A to machine B has another A-B specific move cost.

The problem is defined by the Google ROADEF/EURO Challenge 2012 .
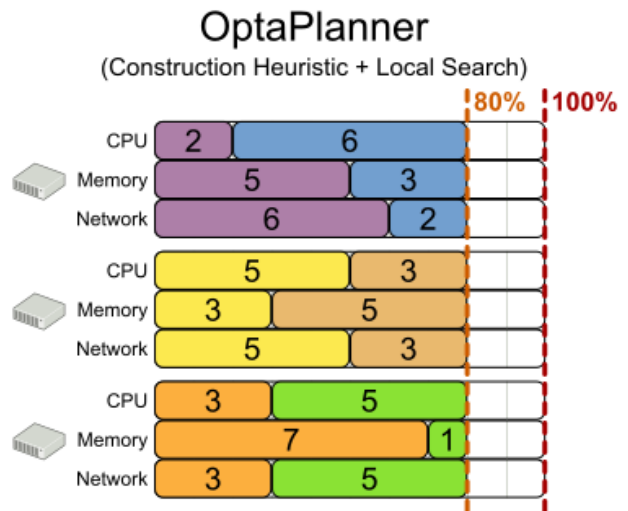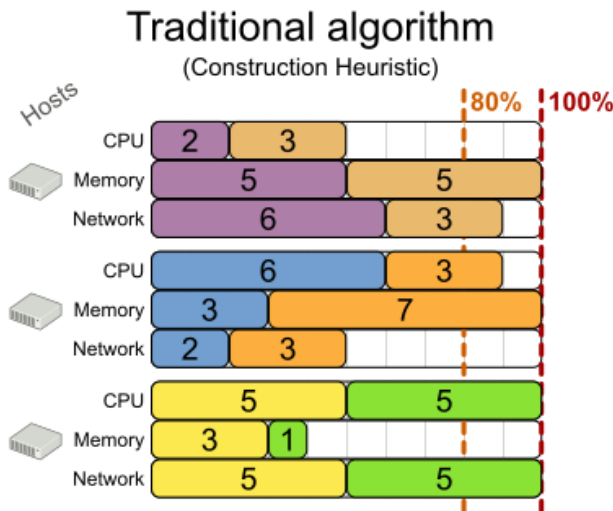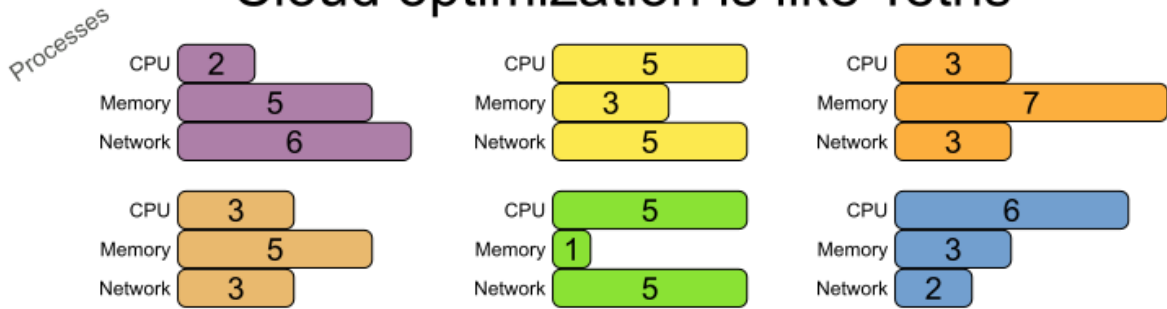
Figure 4.6. Value proposition
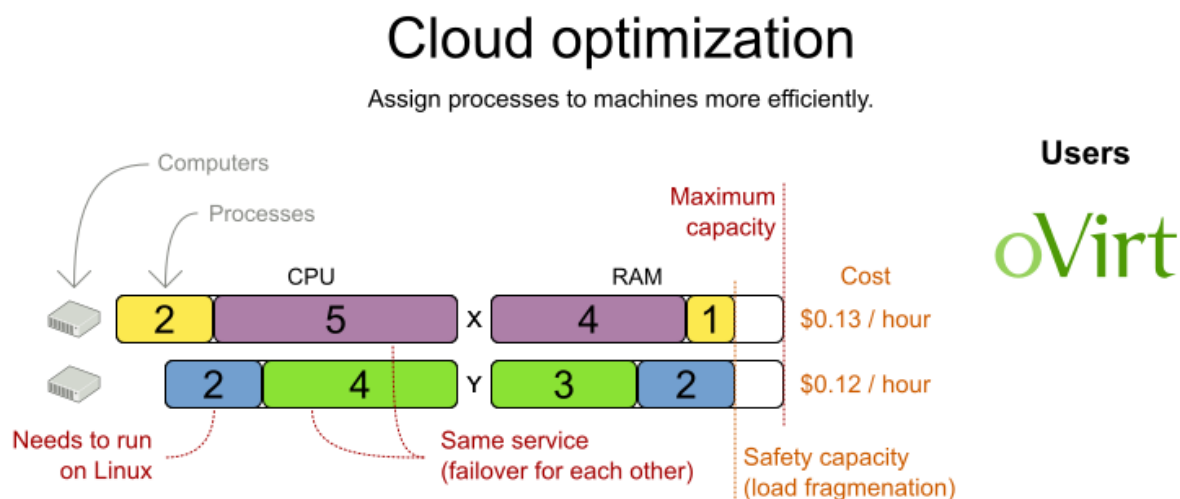
Problem size

> model_a1_1 has 2 resources, 1 neighborhoods,   4 locations,   4 machines,   79 services,   100
> processes and 1 balancePenalties with a search space of   10^60.
> model_a1_2 has 4 resources, 2 neighborhoods,   4 locations, 100 machines,   980 services, 1000
> processes and 0 balancePenalties with a search space of   10^2000.
> model_a1_3 has 3 resources, 5 neighborhoods, 25 locations, 100 machines,   216 services, 1000
> processes and 0 balancePenalties with a search space of   10^2000.
> model_a1_4 has 3 resources, 50 neighborhoods, 50 locations, 50 machines,   142 services, 1000
> processes and 1 balancePenalties with a search space of   10^1698.
> model_a1_5 has 4 resources, 2 neighborhoods,   4 locations,   12 machines,   981 services, 1000
> processes and 1 balancePenalties with a search space of   10^1079.
> model_a2_1 has 3 resources, 1 neighborhoods,   1 locations, 100 machines, 1000 services, 1000
> processes and 0 balancePenalties with a search space of   10^2000.
> model_a2_2 has 12 resources, 5 neighborhoods, 25 locations, 100 machines,   170 services, 1000
> processes and 0 balancePenalties with a search space of   10^2000.
> model_a2_3 has 12 resources, 5 neighborhoods, 25 locations, 100 machines,   129 services, 1000
> processes and 0 balancePenalties with a search space of   10^2000.
> model_a2_4 has 12 resources, 5 neighborhoods, 25 locations, 50 machines,   180 services, 1000
> processes and 1 balancePenalties with a search space of   10^1698.
> model_a2_5 has 12 resources, 5 neighborhoods, 25 locations, 50 machines,   153 services, 1000
> processes and 0 balancePenalties with a search space of   10^1698.
> model_b_1 has 12 resources, 5 neighborhoods, 10 locations, 100 machines, 2512 services, 5000
> processes and 0 balancePenalties with a search space of 10^10000.
> model_b_2 has 12 resources, 5 neighborhoods, 10 locations, 100 machines, 2462 services, 5000
> processes and 1 balancePenalties with a search space of 10^10000.

model_b_3 has 6 resources, 5 neighborhoods, 10 locations, 100 machines, 15025 services, 20000 processes and 0 balancePenalties with a search space of 10^40000.
model_b_4 has 6 resources, 5 neighborhoods, 50 locations, 500 machines, 1732 services, 20000 processes and 1 balancePenalties with a search space of 10^53979.
model_b_5 has 6 resources, 5 neighborhoods, 10 locations, 100 machines, 35082 services, 40000 processes and 0 balancePenalties with a search space of 10^80000.
model_b_6 has 6 resources, 5 neighborhoods, 50 locations, 200 machines, 14680 services, 40000 processes and 1 balancePenalties with a search space of 10^92041.
model_b_7 has 6 resources, 5 neighborhoods, 50 locations, 4000 machines, 15050 services, 40000 processes and 1 balancePenalties with a search space of 10^144082.
model_b_8 has 3 resources, 5 neighborhoods, 10 locations, 100 machines, 45030 services, 50000 processes and 0 balancePenalties with a search space of 10^100000.
model_b_9 has 3 resources, 5 neighborhoods, 100 locations, 1000 machines, 4609 services, 50000 processes and 1 balancePenalties with a search space of 10^150000.
model_b_10 has 3 resources, 5 neighborhoods, 100 locations, 5000 machines, 4896 services, 50000 processes and 1 balancePenalties with a search space of 10^184948.

Figure 4.7. Domain model
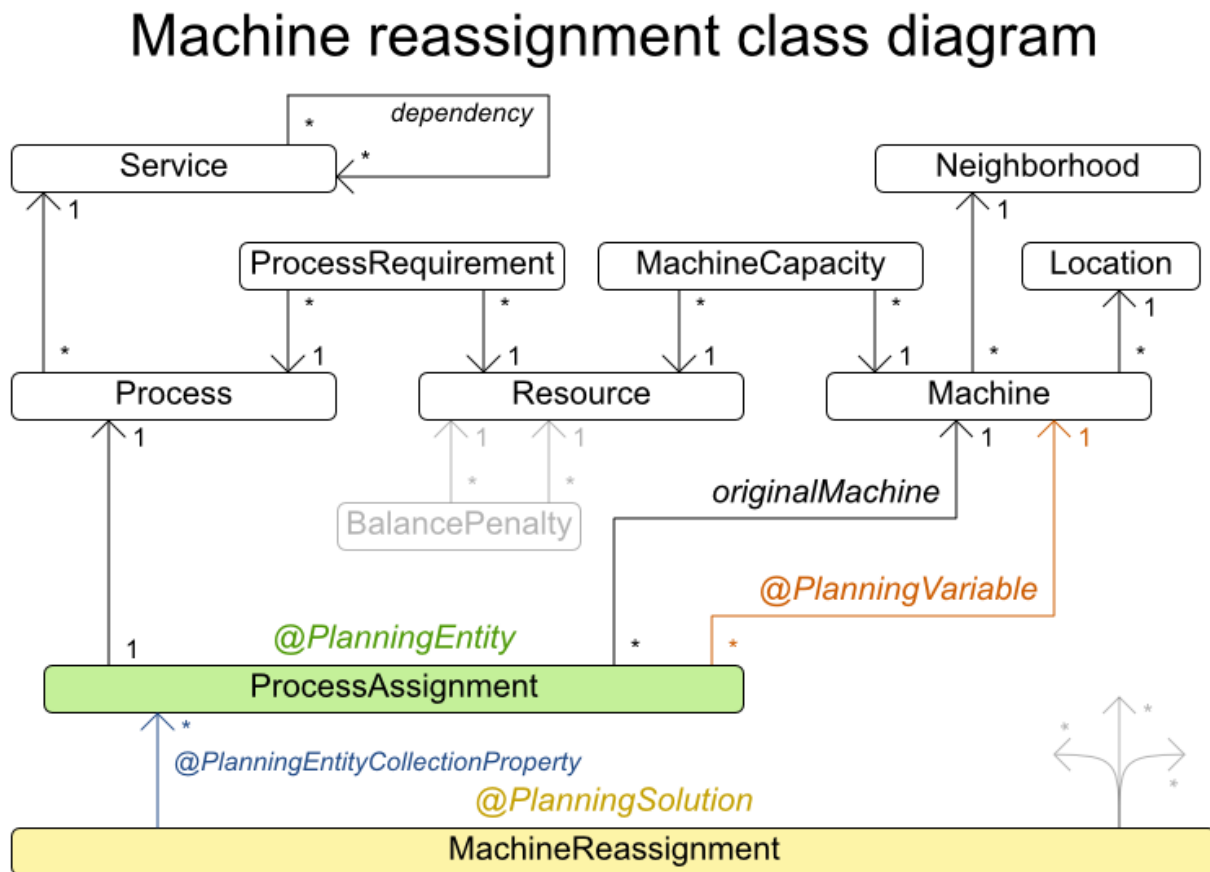
4.8. VEHICLE ROUTING

Using a fleet of vehicles, pick up the objects of each customer and bring them to the depot. Each vehicle can service multiple customers, but it has a limited capacity.
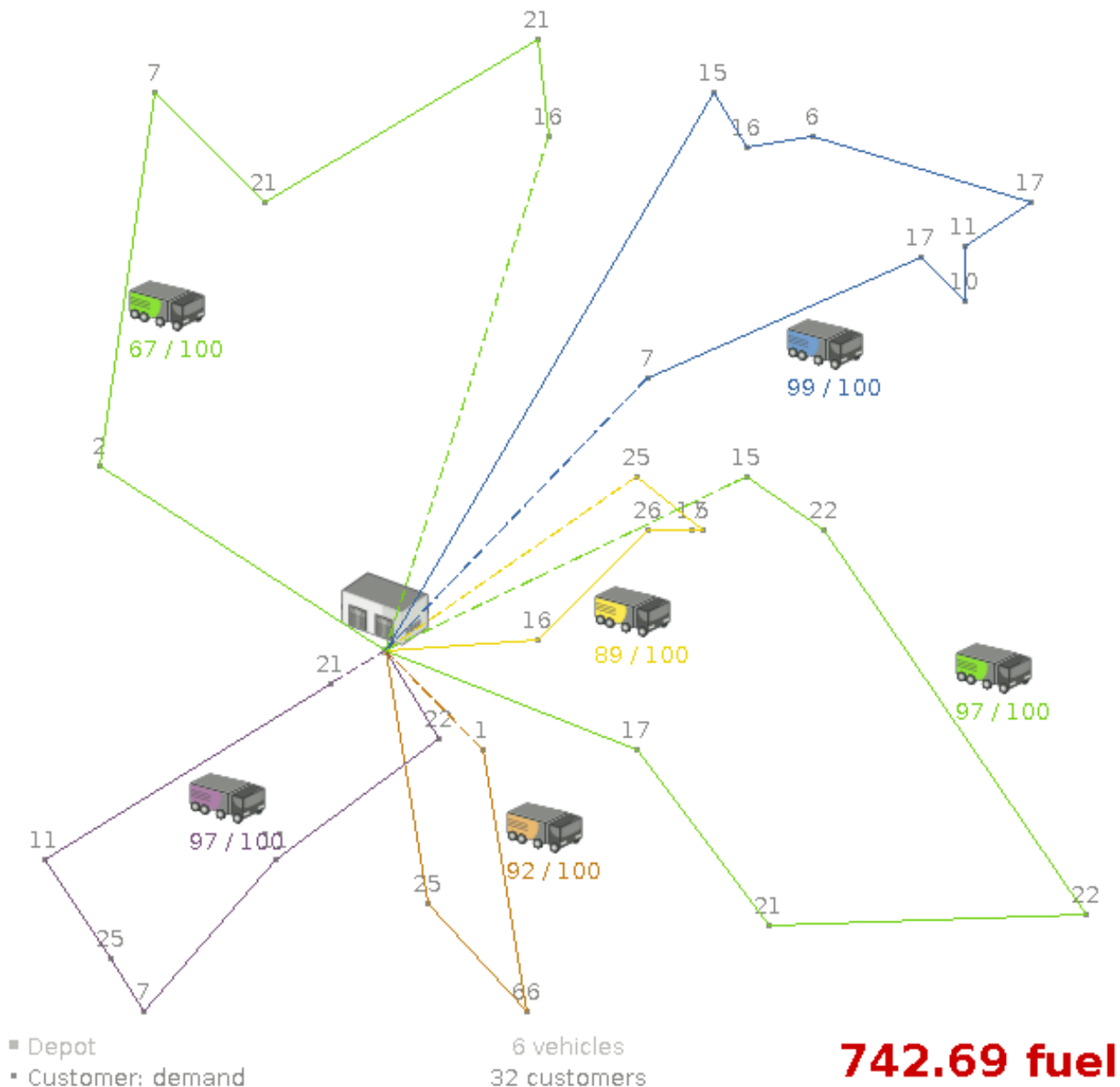
Besides the basic case (CVRP), there is also a variant with time windows (CVRPTW).

Hard constraints:

- Vehicle capacity: a vehicle cannot carry more items then its capacity.

- Time windows (only in CVRPTW):

    - Travel time: Traveling from one location to another takes time.

    - Customer service duration: a vehicle must stay at the customer for the length of the service duration.

    - Customer ready time: a vehicle may arrive before the customer's ready time, but it must wait until the ready time before servicing.

    - Customer due time: a vehicle must arrive on time, before the customer's due time.

Soft constraints:

- Total distance: minimize the total distance driven (fuel consumption) of all vehicles.

The capacitated vehicle routing problem (CVRP) and its time-windowed variant (CVRPTW) are defined by the VRP web.

Figure 4.8. Value proposition
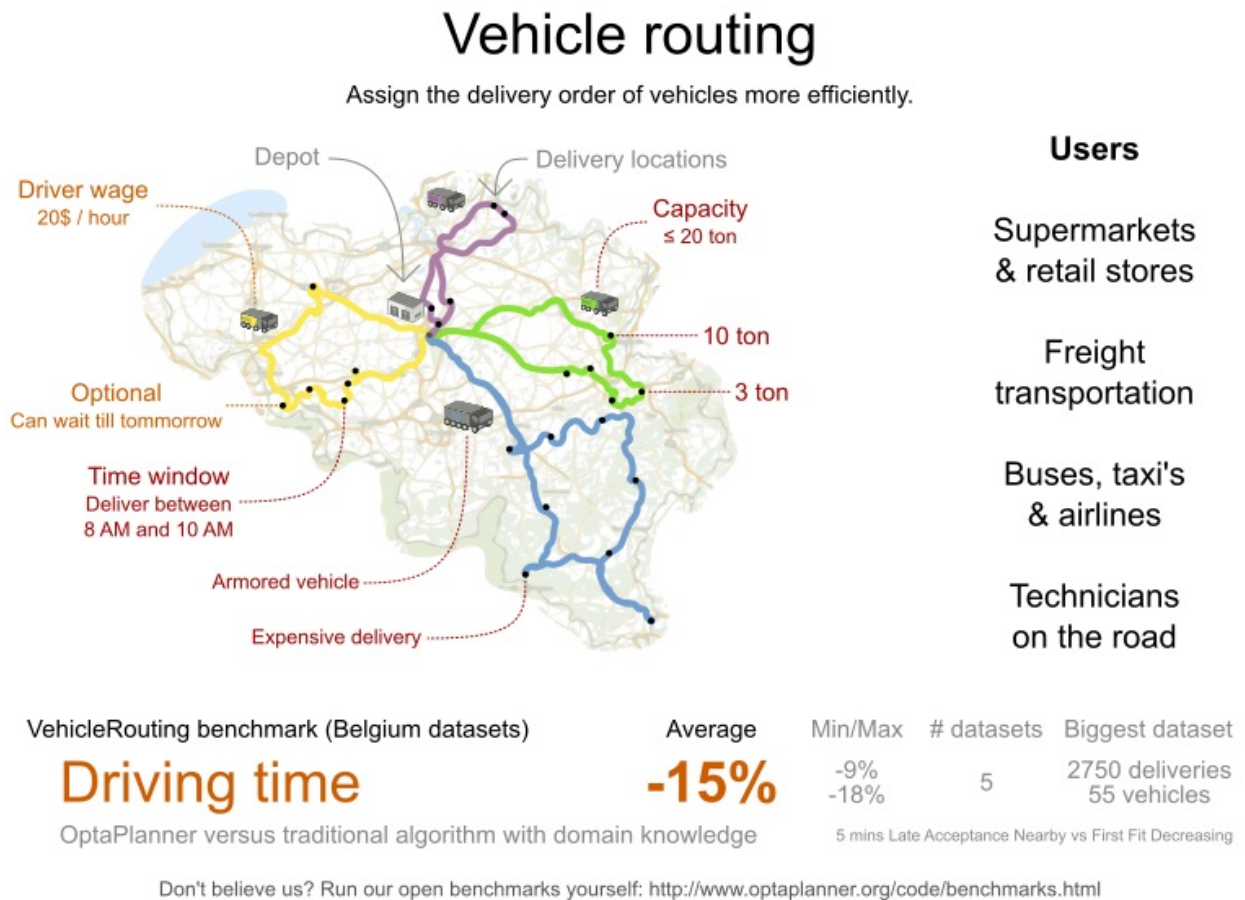
Problem size

CVRP instances (without time windows):

```
belgium-n50-k10          has  1 depots, 10 vehicles and   49 customers with a search space of
10^74.
belgium-n100-k10         has  1 depots, 10 vehicles and   99 customers with a search space of
10^170.
belgium-n500-k20         has  1 depots, 20 vehicles and  499 customers with a search space of
10^1168.
belgium-n1000-k20        has  1 depots, 20 vehicles and  999 customers with a search space of
10^2607.
belgium-n2750-k55        has  1 depots, 55 vehicles and 2749 customers with a search space of
10^8380.
belgium-road-km-n50-k10    has  1 depots, 10 vehicles and   49 customers with a search space of
10^74.
belgium-road-km-n100-k10   has  1 depots, 10 vehicles and   99 customers with a search space of
10^170.
belgium-road-km-n500-k20   has  1 depots, 20 vehicles and  499 customers with a search space of
10^1168.
belgium-road-km-n1000-k20  has  1 depots, 20 vehicles and  999 customers with a search space of
10^2607.
belgium-road-km-n2750-k55  has  1 depots, 55 vehicles and 2749 customers with a search space of
```

10^8380.

belgium-road-time-n50-k10   has 1 depots, 10 vehicles and   49 customers with a search space of 10^74.

belgium-road-time-n100-k10 has 1 depots, 10 vehicles and   99 customers with a search space of 10^170.

belgium-road-time-n500-k20 has 1 depots, 20 vehicles and 499 customers with a search space of 10^1168.

belgium-road-time-n1000-k20 has 1 depots, 20 vehicles and 999 customers with a search space of 10^2607.

belgium-road-time-n2750-k55 has 1 depots, 55 vehicles and 2749 customers with a search space of 10^8380.

belgium-d2-n50-k10   has 2 depots, 10 vehicles and   48 customers with a search space of 10^74.

belgium-d3-n100-k10   has 3 depots, 10 vehicles and   97 customers with a search space of 10^170.

belgium-d5-n500-k20   has 5 depots, 20 vehicles and 495 customers with a search space of 10^1168.

belgium-d8-n1000-k20   has 8 depots, 20 vehicles and 992 customers with a search space of 10^2607.

belgium-d10-n2750-k55   has 10 depots, 55 vehicles and 2740 customers with a search space of 10^8380.

A-n32-k5 has 1 depots, 5 vehicles and 31 customers with a search space of 10^40.
A-n33-k5 has 1 depots, 5 vehicles and 32 customers with a search space of 10^41.
A-n33-k6 has 1 depots, 6 vehicles and 32 customers with a search space of 10^42.
A-n34-k5 has 1 depots, 5 vehicles and 33 customers with a search space of 10^43.
A-n36-k5 has 1 depots, 5 vehicles and 35 customers with a search space of 10^46.
A-n37-k5 has 1 depots, 5 vehicles and 36 customers with a search space of 10^48.
A-n37-k6 has 1 depots, 6 vehicles and 36 customers with a search space of 10^49.
A-n38-k5 has 1 depots, 5 vehicles and 37 customers with a search space of 10^49.
A-n39-k5 has 1 depots, 5 vehicles and 38 customers with a search space of 10^51.
A-n39-k6 has 1 depots, 6 vehicles and 38 customers with a search space of 10^52.
A-n44-k7 has 1 depots, 7 vehicles and 43 customers with a search space of 10^61.
A-n45-k6 has 1 depots, 6 vehicles and 44 customers with a search space of 10^62.
A-n45-k7 has 1 depots, 7 vehicles and 44 customers with a search space of 10^63.
A-n46-k7 has 1 depots, 7 vehicles and 45 customers with a search space of 10^65.
A-n48-k7 has 1 depots, 7 vehicles and 47 customers with a search space of 10^68.
A-n53-k7 has 1 depots, 7 vehicles and 52 customers with a search space of 10^77.
A-n54-k7 has 1 depots, 7 vehicles and 53 customers with a search space of 10^79.
A-n55-k9 has 1 depots, 9 vehicles and 54 customers with a search space of 10^82.
A-n60-k9 has 1 depots, 9 vehicles and 59 customers with a search space of 10^91.
A-n61-k9 has 1 depots, 9 vehicles and 60 customers with a search space of 10^93.
A-n62-k8 has 1 depots, 8 vehicles and 61 customers with a search space of 10^94.
A-n63-k9 has 1 depots, 9 vehicles and 62 customers with a search space of 10^97.
A-n63-k10 has 1 depots, 10 vehicles and 62 customers with a search space of 10^98.
A-n64-k9 has 1 depots, 9 vehicles and 63 customers with a search space of 10^99.
A-n65-k9 has 1 depots, 9 vehicles and 64 customers with a search space of 10^101.
A-n69-k9 has 1 depots, 9 vehicles and 68 customers with a search space of 10^108.
A-n80-k10 has 1 depots, 10 vehicles and 79 customers with a search space of 10^130.
F-n45-k4 has 1 depots, 4 vehicles and 44 customers with a search space of 10^60.
F-n72-k4 has 1 depots, 4 vehicles and 71 customers with a search space of 10^108.
F-n135-k7 has 1 depots, 7 vehicles and 134 customers with a search space of 10^240.

CVRPTW instances (with time windows):

belgium-tw-d2-n50-k10   has 2 depots, 10 vehicles and   48 customers with a search space of

10^74.
belgium-tw-d3-n100-k10   has 3 depots, 10 vehicles and   97 customers with a search space of 10^170.
belgium-tw-d5-n500-k20   has 5 depots, 20 vehicles and 495 customers with a search space of 10^1168.
belgium-tw-d8-n1000-k20 has 8 depots, 20 vehicles and 992 customers with a search space of 10^2607.
belgium-tw-d10-n2750-k55 has 10 depots, 55 vehicles and 2740 customers with a search space of 10^8380.
belgium-tw-n50-k10   has 1 depots, 10 vehicles and   49 customers with a search space of 10^74.
belgium-tw-n100-k10   has 1 depots, 10 vehicles and   99 customers with a search space of 10^170.
belgium-tw-n500-k20   has 1 depots, 20 vehicles and 499 customers with a search space of 10^1168.
belgium-tw-n1000-k20   has 1 depots, 20 vehicles and 999 customers with a search space of 10^2607.
belgium-tw-n2750-k55   has 1 depots, 55 vehicles and 2749 customers with a search space of 10^8380.

Solomon_025_C101   has 1 depots, 25 vehicles and   25 customers with a search space of 10^40.
Solomon_025_C201   has 1 depots, 25 vehicles and   25 customers with a search space of 10^40.
Solomon_025_R101   has 1 depots, 25 vehicles and   25 customers with a search space of 10^40.
Solomon_025_R201   has 1 depots, 25 vehicles and   25 customers with a search space of 10^40.
Solomon_025_RC101   has 1 depots, 25 vehicles and   25 customers with a search space of 10^40.
Solomon_025_RC201   has 1 depots, 25 vehicles and   25 customers with a search space of 10^40.
Solomon_100_C101   has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Solomon_100_C201   has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Solomon_100_R101   has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Solomon_100_R201   has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Solomon_100_RC101   has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Solomon_100_RC201   has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Homberger_0200_C1_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.
Homberger_0200_C2_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.
Homberger_0200_R1_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.
Homberger_0200_R2_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.
Homberger_0200_RC1_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.
Homberger_0200_RC2_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.

Homberger_0400_C1_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.

Homberger_0400_C2_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.

Homberger_0400_R1_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.

Homberger_0400_R2_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.

Homberger_0400_RC1_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.

Homberger_0400_RC2_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.

Homberger_0600_C1_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.

Homberger_0600_C2_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.

Homberger_0600_R1_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.

Homberger_0600_R2_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.

Homberger_0600_RC1_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.

Homberger_0600_RC2_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.

Homberger_0800_C1_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.

Homberger_0800_C2_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.

Homberger_0800_R1_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.

Homberger_0800_R2_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.

Homberger_0800_RC1_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.

Homberger_0800_RC2_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.

Homberger_1000_C110_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.

Homberger_1000_C210_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.

Homberger_1000_R110_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.

Homberger_1000_R210_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.

Homberger_1000_RC110_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.

Homberger_1000_RC210_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.

4.8.1. Domain model for Vehicle routing

Vehicle routing class diagram

@PlanningSolution

VehicleRoutingSolution

@PlanningEntityCollectionProperty

@PlanningEntity

Standstill

Customer
readyTime
dueTime
serviceDuration
arrivalTime
departureTime

@PlanningVariable
previousStandstill

nextCustomer

0..1

vehicle

Vehicle
readyTime
dueTime

Depot

1

Location

Shadow variables

arrivalTime =
previousS.departureTime
+ travelTime

departureTime =
max(arrivalTime, readyTime)
+ serviceDuration

nextCustomer =
inverse of previousStandstill

vehicle =
(this instanceof Vehicle)
? this : previousS.vehicle

The vehicle routing with timewindows domain model makes heavily use of the shadow variable feature. This allows it to express its constraints more naturally, because properties such as **arrivalTime** and **departureTime**, are directly available on the domain model.

Road Distances Instead of Air Distances

In the real world, vehicles cannot follow a straight line from location to location: they have to use roads and highways. From a business point of view, this matters a lot:
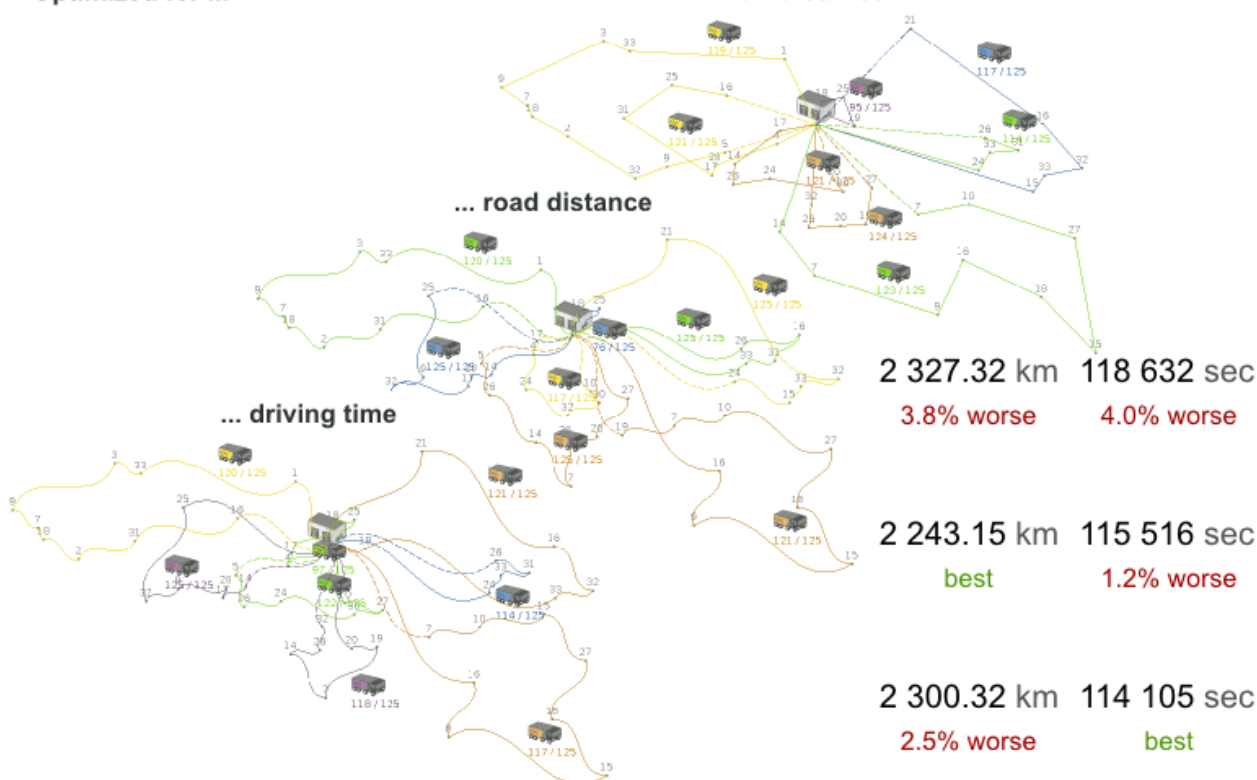
For the optimization algorithm, this does not matter much, as long as the distance between two points can be looked up (and are preferably precalculated). The road cost does not even need to be a distance, it can also be travel time, fuel cost, or a weighted function of those. There are several technologies available to precalculate road costs, such as GraphHopper (embeddable, offline Java engine), Open MapQuest (web service) and Google Maps Client API (web service).
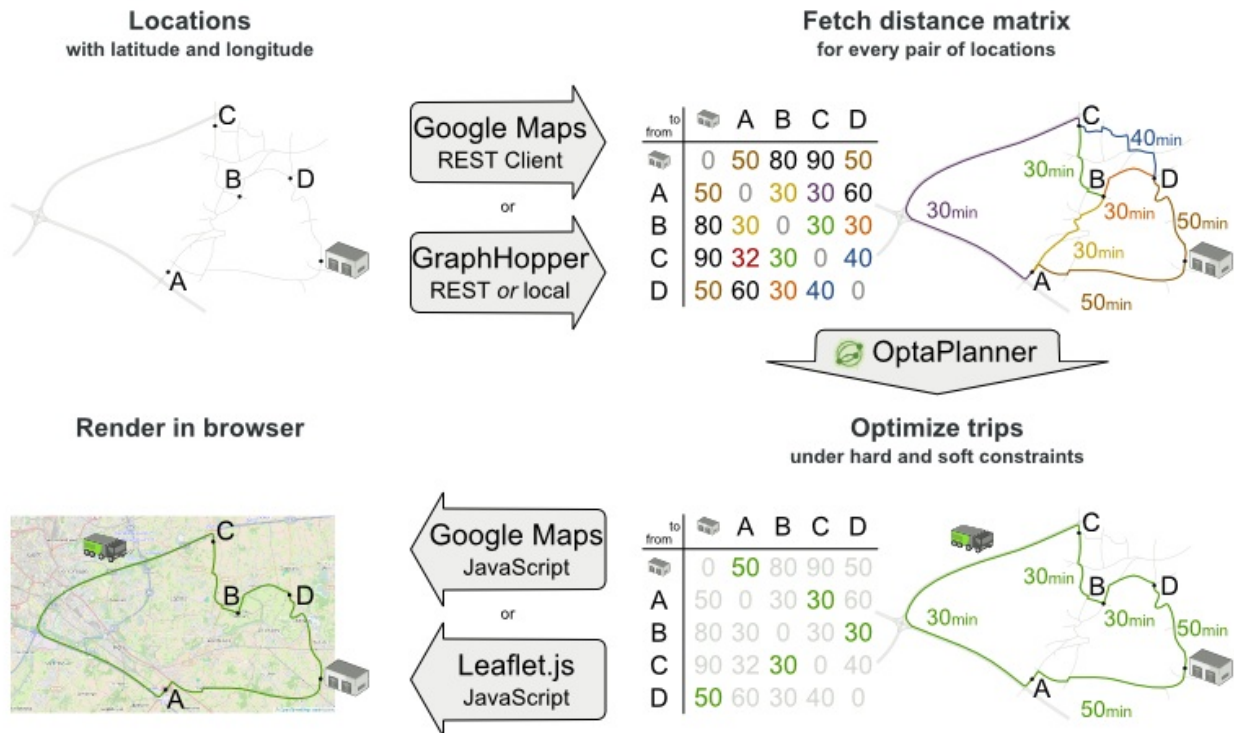
There are also several technologies to render it, such as Leaflet and Google Maps for developers.
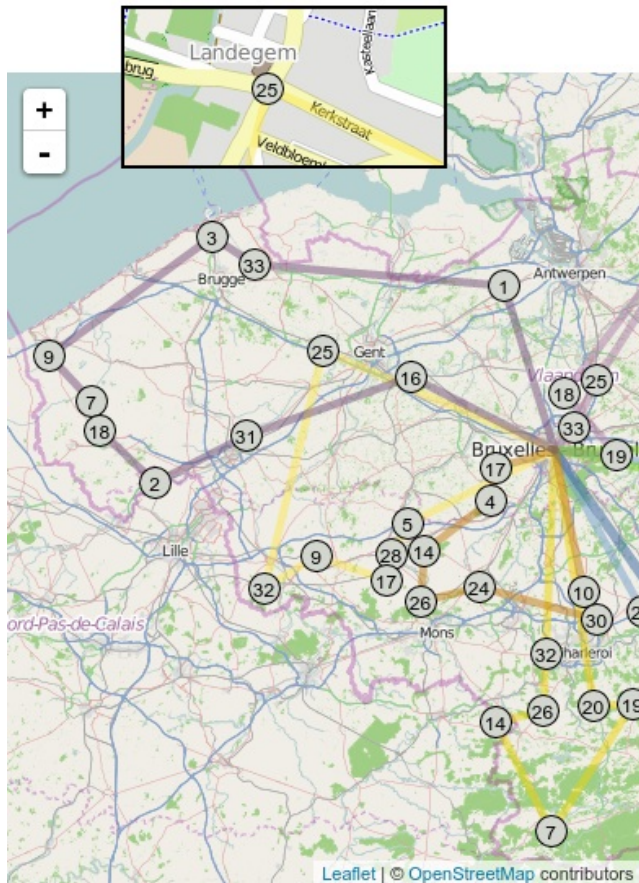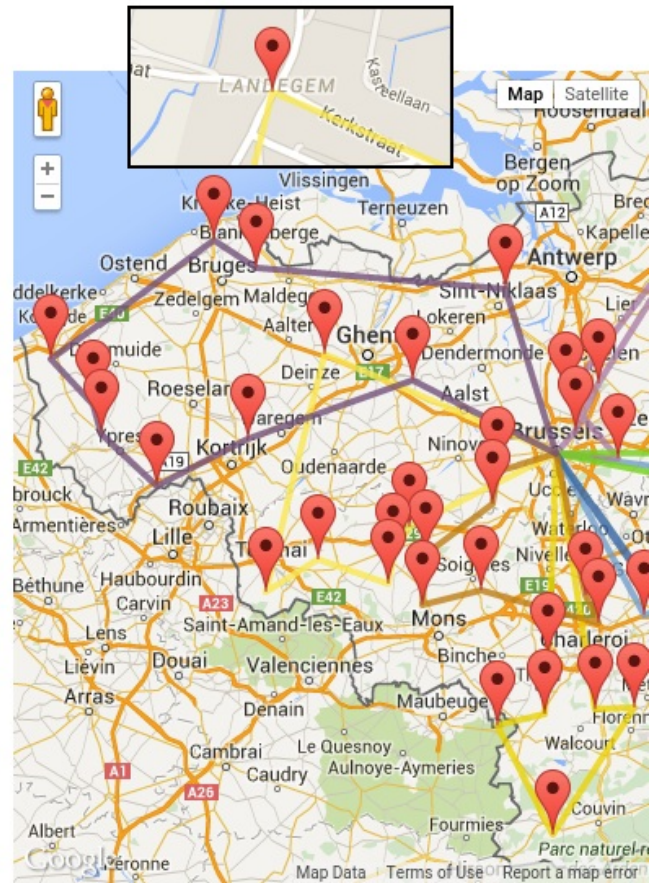
It is even possible to render the actual road routes with GraphHopper or Google Map Directions, but because of route overlaps on highways, it can become harder to see the standstill order:

Take special care that the road costs between two points use the same optimization criteria as the one used in Planner. For example, GraphHopper etc will by default return the fastest route, not the shortest route. Don't use the km (or miles) distances of the fastest GPS routes to optimize the shortest trip in Planner: this leads to a suboptimal solution as shown below:

Road distance triangle inequality

Routes and trips must optimize the same property to avoid suboptimal solutions.

Shortest GPS routes

C 60km
40min
30km
30min B 30km D
50km 30min 40km
50min 30km 50min
30min
A 60km
50min

Goal: shortest trip
using **shortest** GPS routes

C 60km
30km
B 30km D
50km 30km 40km
30km
A 60km

→ A → C → B → D →
60 + 50 + 30 + 30 + 40 = 210km
optimal

Goal: fastest trip
using **shortest** GPS routes

C
40min
30min B D
50min 30min 50min
30min
A
50min

→ A → B → C → D →
50 + 30 + 30 + 40 + 50 = 200min
suboptimal

Fastest GPS routes

C 60km
40min
30km
30min B 30km D
90km 30min 40km
30min 30km 50min
30min
A 60km
50min

In this example, only the A→C route
differs between shortest and fastest.
In the real world, almost all routes differ.

Goal: shortest trip
using **fastest** GPS routes

Violates triangle inequality!
A→C ≰ A→B + B→C
90 ≰ 30 + 30
C 60km
30km
B 30km D
90km 40km
30km
A 60km

→ A → B → C → D →
60 + 30 + 30 + 60 + 40 = 220km
suboptimal

Goal: fastest trip
using **fastest** GPS routes

C
40min
30min B D
30min 30min 50min
30min
A
50min

→ A → C → B → D →
50 + 30 + 30 + 30 + 50 = 190min
optimal

Contrary to popular belief, most users do not want the shortest route: they want the fastest route instead. They prefer highways over normal roads. They prefer normal roads over dirt roads. In the real world, the fastest and shortest route are rarely the same.

4.9. PROJECT JOB SCHEDULING

Schedule all jobs in time and execution mode to minimize project delays. Each job is part of a project. A job can be executed in different ways: each way is an execution mode that implies a different duration but also different resource usages. This is a form of flexible *job shop scheduling*.

Project job scheduling

For each job, choose an execution mode and a start time.
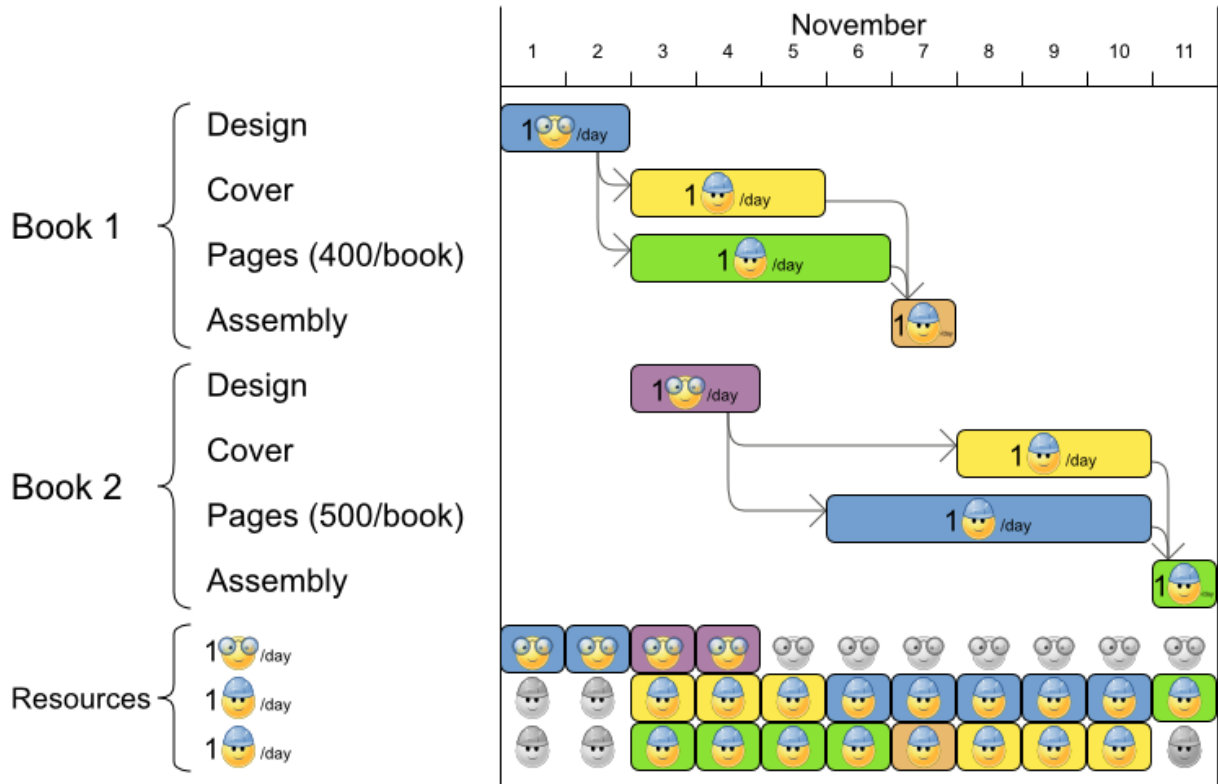
Hard constraints:

- Job precedence: a job can only start when all its predecessor jobs are finished.

- Resource capacity: do not use more resources than available.

  - Resources are local (shared between jobs of the same project) or global (shared between all jobs)

  - Resources are renewable (capacity available per day) or nonrenewable (capacity available for all days)

Medium constraints:

- Total project delay: minimize the duration (makespan) of each project.

Soft constraints:

- Total makespan: minimize the duration of the whole multi-project schedule.

The problem is defined by the MISTA 2013 challenge.

Problem size

> Schedule A-1 has 2 projects, 24 jobs, 64 execution modes, 7 resources and 150 resource requirements.
> Schedule A-2 has 2 projects, 44 jobs, 124 execution modes, 7 resources and 420 resource requirements.

Schedule A-3 has 2 projects, 64 jobs, 184 execution modes, 7 resources and 630 resource requirements.
Schedule A-4 has 5 projects, 60 jobs, 160 execution modes, 16 resources and 390 resource requirements.
Schedule A-5 has 5 projects, 110 jobs, 310 execution modes, 16 resources and 900 resource requirements.
Schedule A-6 has 5 projects, 160 jobs, 460 execution modes, 16 resources and 1440 resource requirements.
Schedule A-7 has 10 projects, 120 jobs, 320 execution modes, 22 resources and 900 resource requirements.
Schedule A-8 has 10 projects, 220 jobs, 620 execution modes, 22 resources and 1860 resource requirements.
Schedule A-9 has 10 projects, 320 jobs, 920 execution modes, 31 resources and 2880 resource requirements.
Schedule A-10 has 10 projects, 320 jobs, 920 execution modes, 31 resources and 2970 resource requirements.
Schedule B-1 has 10 projects, 120 jobs, 320 execution modes, 31 resources and 900 resource requirements.
Schedule B-2 has 10 projects, 220 jobs, 620 execution modes, 22 resources and 1740 resource requirements.
Schedule B-3 has 10 projects, 320 jobs, 920 execution modes, 31 resources and 3060 resource requirements.
Schedule B-4 has 15 projects, 180 jobs, 480 execution modes, 46 resources and 1530 resource requirements.
Schedule B-5 has 15 projects, 330 jobs, 930 execution modes, 46 resources and 2760 resource requirements.
Schedule B-6 has 15 projects, 480 jobs, 1380 execution modes, 46 resources and 4500 resource requirements.
Schedule B-7 has 20 projects, 240 jobs, 640 execution modes, 61 resources and 1710 resource requirements.
Schedule B-8 has 20 projects, 440 jobs, 1240 execution modes, 42 resources and 3180 resource requirements.
Schedule B-9 has 20 projects, 640 jobs, 1840 execution modes, 61 resources and 5940 resource requirements.
Schedule B-10 has 20 projects, 460 jobs, 1300 execution modes, 42 resources and 4260 resource requirements.

4.10. TASK ASSIGNING

Assign each task to a spot in an employee's queue. Each task has a duration which is affected by the employee's affinity level with the task's customer.

Hard constraints:

- Skill: Each task requires one or more skills. The employee must possess all these skills.

Soft level 0 constraints:

- Critical tasks: Complete critical tasks first, sooner than major and minor tasks.

Soft level 1 constraints:

- Minimize makespan: Reduce the time to complete all tasks.

  ○ Start with the longest working employee first, then the second longest working employee and so forth, to create fairness and load balancing.

Soft level 2 constraints:

- Major tasks: Complete major tasks as soon as possible, sooner than minor tasks.

Soft level 3 constraints:

- Minor tasks: Complete minor tasks as soon as possible.

Figure 4.9. Value proposition
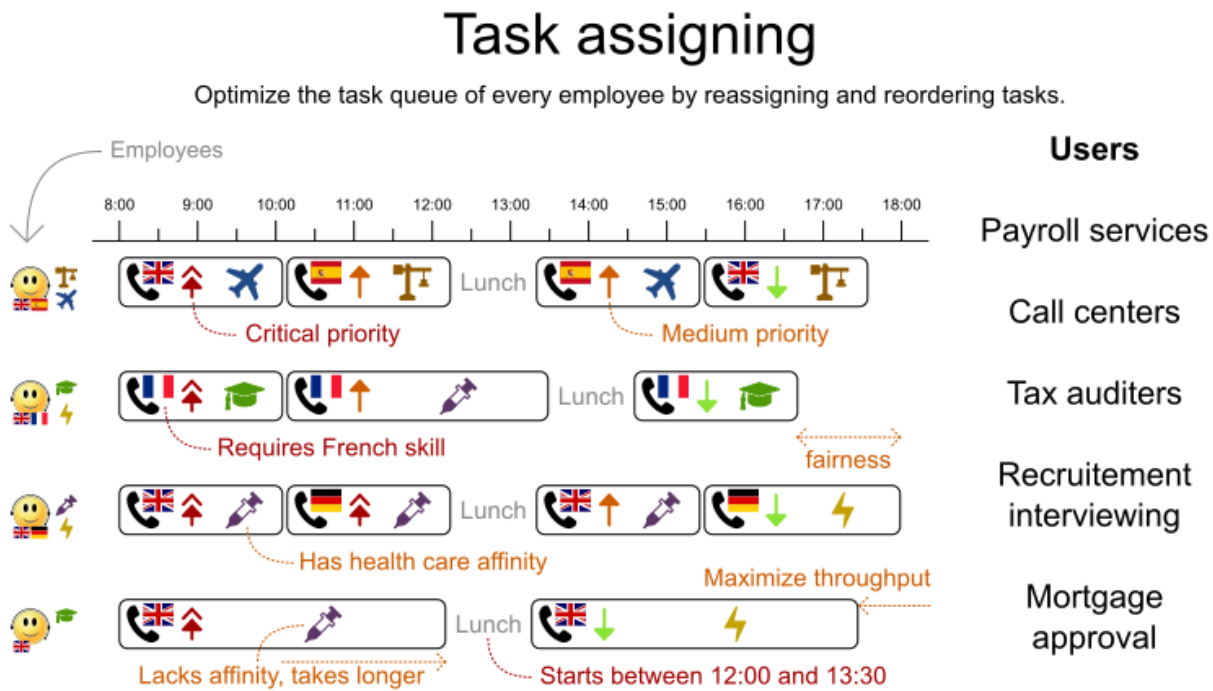
Problem size

```
24tasks-8employees   has  24 tasks, 6 skills,  8 employees,   4 task types and  4 customers with a
search space of   10^30.
50tasks-5employees   has  50 tasks, 5 skills,  5 employees,  10 task types and 10 customers with a
search space of   10^69.
100tasks-5employees  has 100 tasks, 5 skills,  5 employees,  20 task types and 15 customers with a
search space of  10^164.
500tasks-20employees has 500 tasks, 6 skills, 20 employees, 100 task types and 60 customers with
a search space of 10^1168.
```

Figure 4.10. Domain model
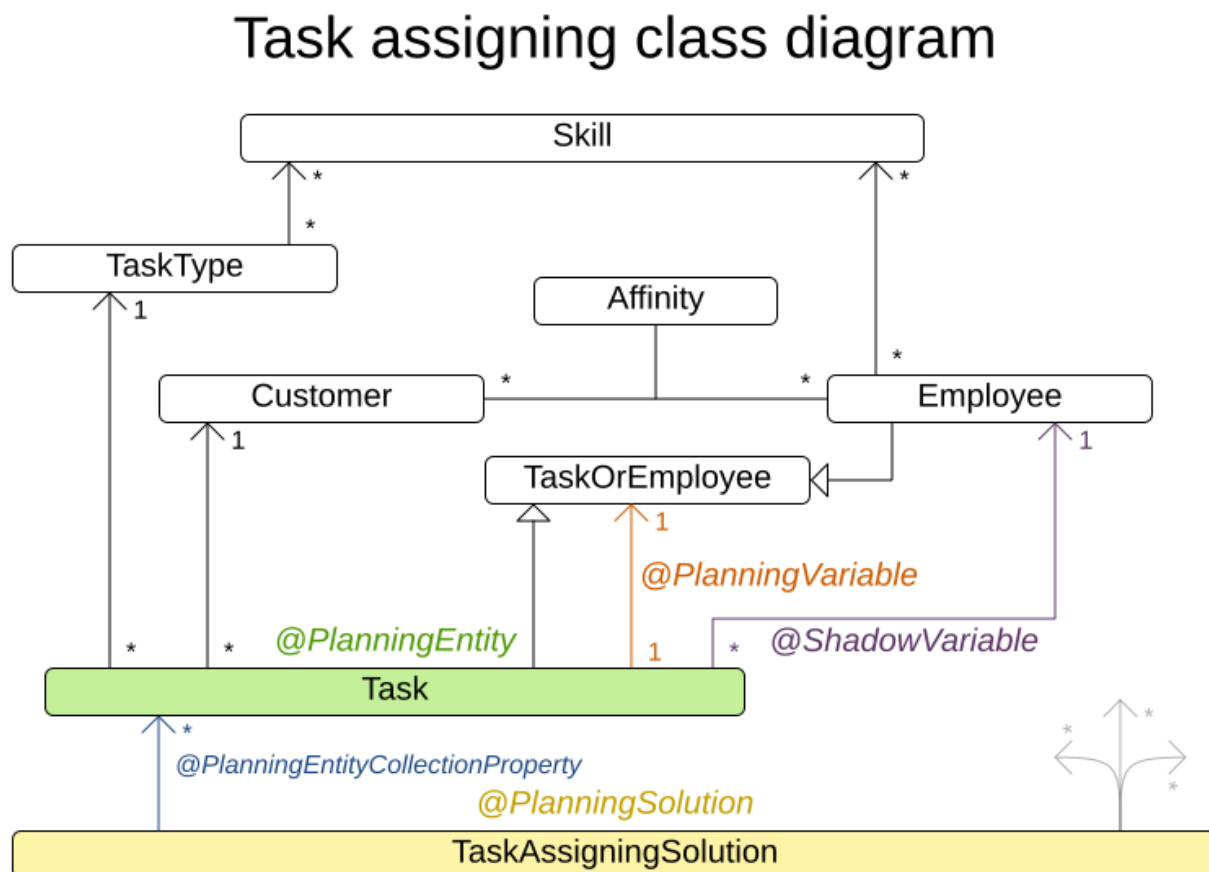
4.11. EXAM TIMETABLING (ITC 2007 TRACK 1 - EXAMINATION)

Schedule each exam into a period and into a room. Multiple exams can share the same room during the same period.
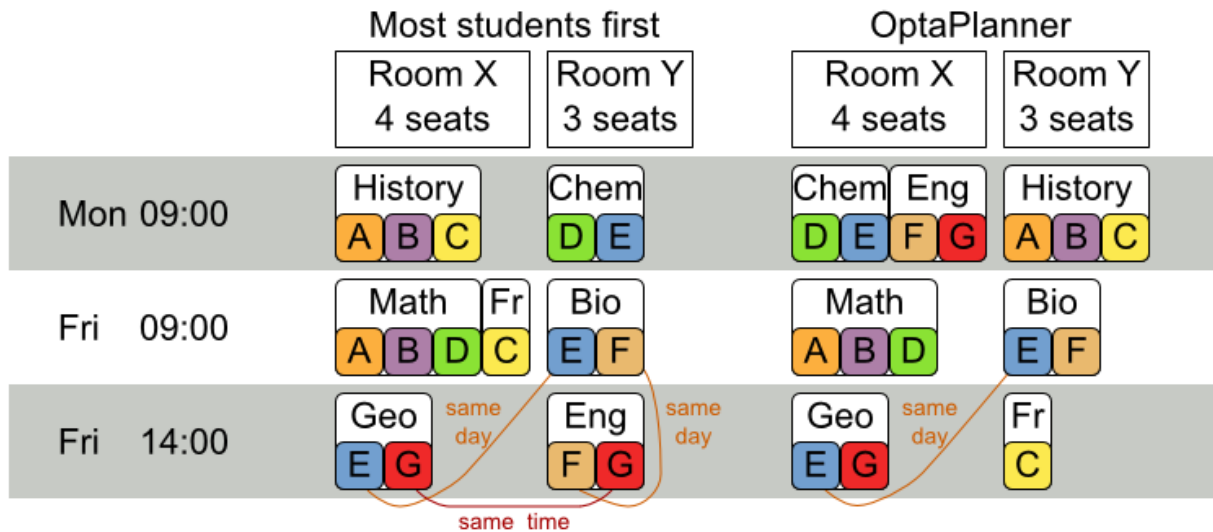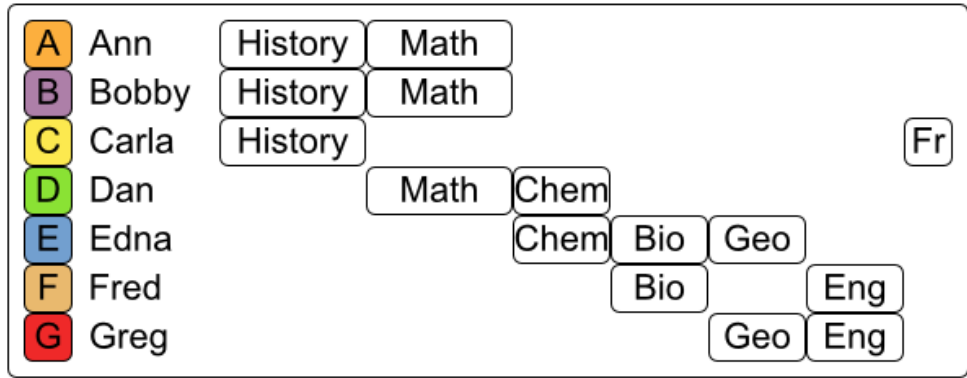
Hard constraints:

- Exam conflict: Two exams that share students must not occur in the same period.

- Room capacity: A room's seating capacity must suffice at all times.

- Period duration: A period's duration must suffice for all of its exams.

- Period related hard constraints (specified per dataset):

  - Coincidence: Two specified exams must use the same period (but possibly another room).

  - Exclusion: Two specified exams must not use the same period.

  - After: A specified exam must occur in a period after another specified exam's period.

- Room related hard constraints (specified per dataset):

  - Exclusive: One specified exam should not have to share its room with any other exam.

Soft constraints (each of which has a parametrized penalty):

- The same student should not have two exams in a row.

- The same student should not have two exams on the same day.

- Period spread: Two exams that share students should be a number of periods apart.

- Mixed durations: Two exams that share a room should not have different durations.

- Front load: Large exams should be scheduled earlier in the schedule.

- Period penalty (specified per dataset): Some periods have a penalty when used.

- Room penalty (specified per dataset): Some rooms have a penalty when used.

It uses large test data sets of real-life universities.

The problem is defined by the International Timetabling Competition 2007 track 1. Geoffrey De Smet finished 4th in that competition with a very early version of Planner. Many improvements have been made since then.

Problem Size

```
exam_comp_set1 has  7883 students,  607 exams, 54 periods,  7 rooms,  12 period constraints and
0 room constraints with a search space of 10^1564.
exam_comp_set2 has 12484 students,  870 exams, 40 periods, 49 rooms,  12 period constraints and
2 room constraints with a search space of 10^2864.
exam_comp_set3 has 16365 students,  934 exams, 36 periods, 48 rooms, 168 period constraints and
15 room constraints with a search space of 10^3023.
exam_comp_set4 has  4421 students,  273 exams, 21 periods,  1 rooms,  40 period constraints and
0 room constraints with a search space of  10^360.
exam_comp_set5 has  8719 students, 1018 exams, 42 periods,  3 rooms,  27 period constraints and
0 room constraints with a search space of 10^2138.
exam_comp_set6 has  7909 students,  242 exams, 16 periods,  8 rooms,  22 period constraints and
0 room constraints with a search space of  10^509.
exam_comp_set7 has 13795 students, 1096 exams, 80 periods, 15 rooms,  28 period constraints and
0 room constraints with a search space of 10^3374.
exam_comp_set8 has  7718 students,  598 exams, 80 periods,  8 rooms,  20 period constraints and
1 room constraints with a search space of 10^1678.
```

4.11.1. Domain model for Exam timetabling

The following diagram shows the main examination domain classes:

Figure 4.11. Examination domain class diagram
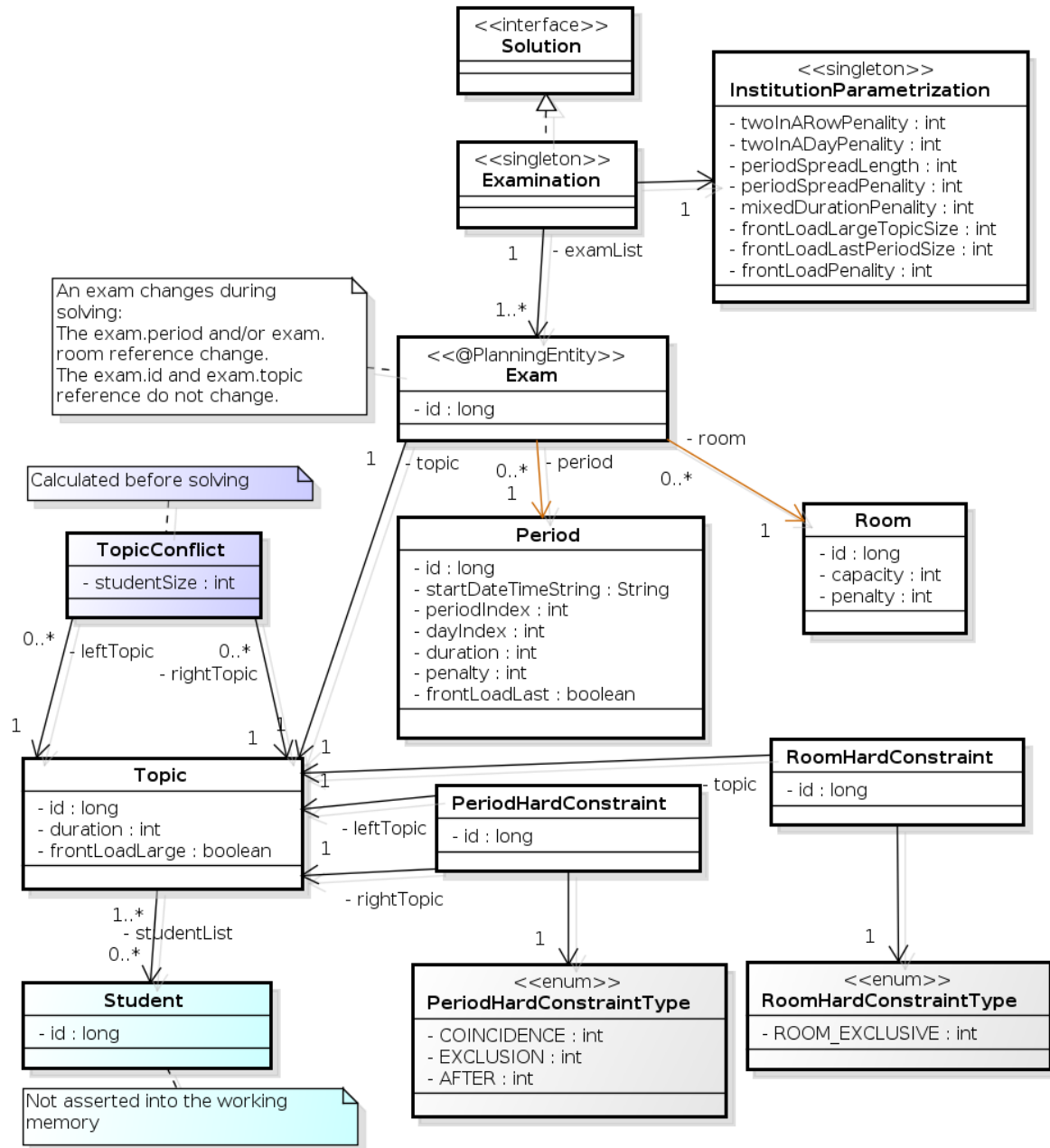
Notice that we've split up the exam concept into an **Exam** class and a **Topic** class. The **Exam** instances change during solving (this is the planning entity class), when their period or room property changes. The **Topic**, **Period** and **Room** instances never change during solving (these are problem facts, just like some other classes).

4.12. NURSE ROSTERING (INRC 2010)

For each shift, assign a nurse to work that shift.

Employee shift rostering

Populate each work shift with a nurse.
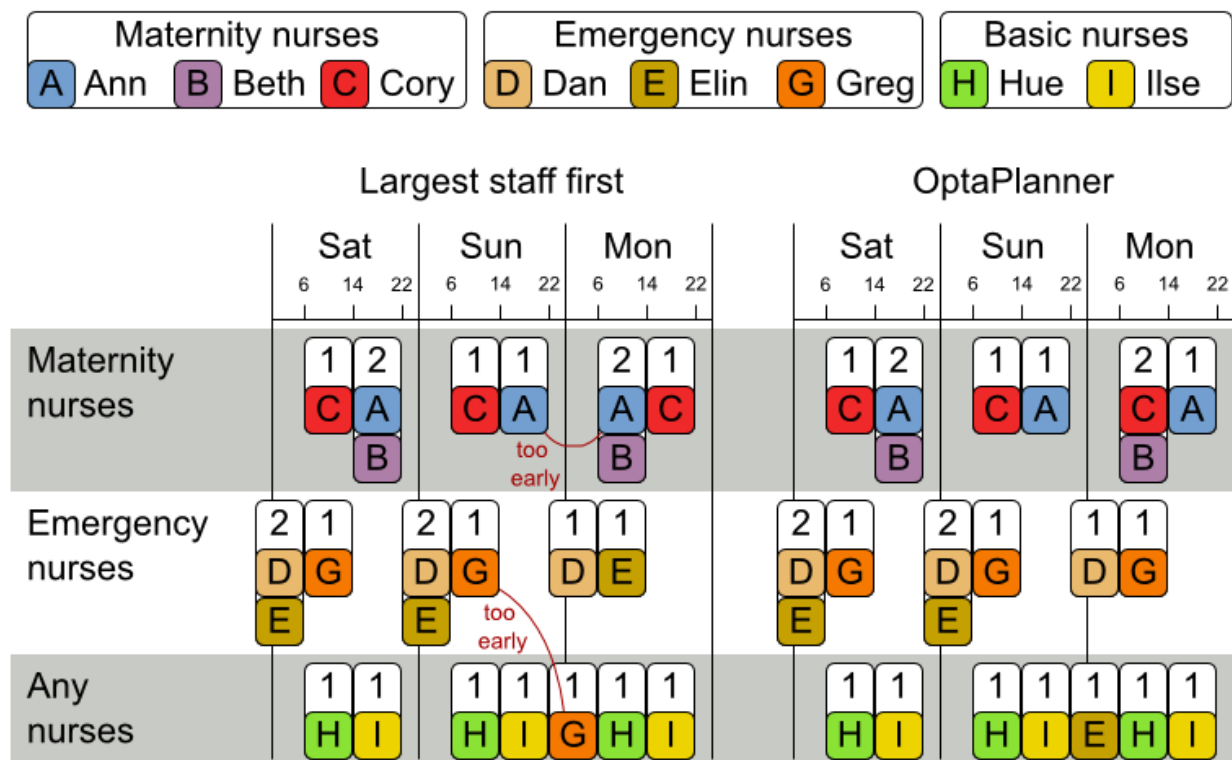
Hard constraints:

- **No unassigned shifts** (built-in): Every shift need to be assigned to an employee.

- **Shift conflict**: An employee can have only one shift per day.

Soft constraints:

- Contract obligations. The business frequently violates these, so they decided to define these as soft constraints instead of hard constraints.

    - **Minimum and maximum assignments**: Each employee needs to work more than x shifts and less than y shifts (depending on their contract).

    - **Minimum and maximum consecutive working days** Each employee needs to work between x and y days in a row (depending on their contract).

    - **Minimum and maximum consecutive free days** Each employee needs to be free between x and y days in a row (depending on their contract).

    - **Minimum and maximum consecutive working weekends** Each employee needs to work between x and y weekends in a row (depending on their contract).

    - **Complete weekends**: Each employee needs to work every day in a weekend or not at all.

    - **Identical shift types during weekend** Each weekend shift for the same weekend of the same employee must be the same shift type.

- **Unwanted patterns**: A combination of unwanted shift types in a row. For example: a late shift followed by an early shift followed by a late shift.

- Employee wishes:

  - **Day on request**: An employee wants to work on a specific day.

  - **Day off request**: An employee does not want to work on a specific day.

  - **Shift on request**: An employee wants to be assigned to a specific shift.

  - **Shift off request**: An employee does not want to be assigned to a specific shift.

- **Alternative skill**: An employee assigned to a skill should have a proficiency in every skill required by that shift.

The problem is defined by the International Nurse Rostering Competition 2010 .

Figure 4.12. Value proposition
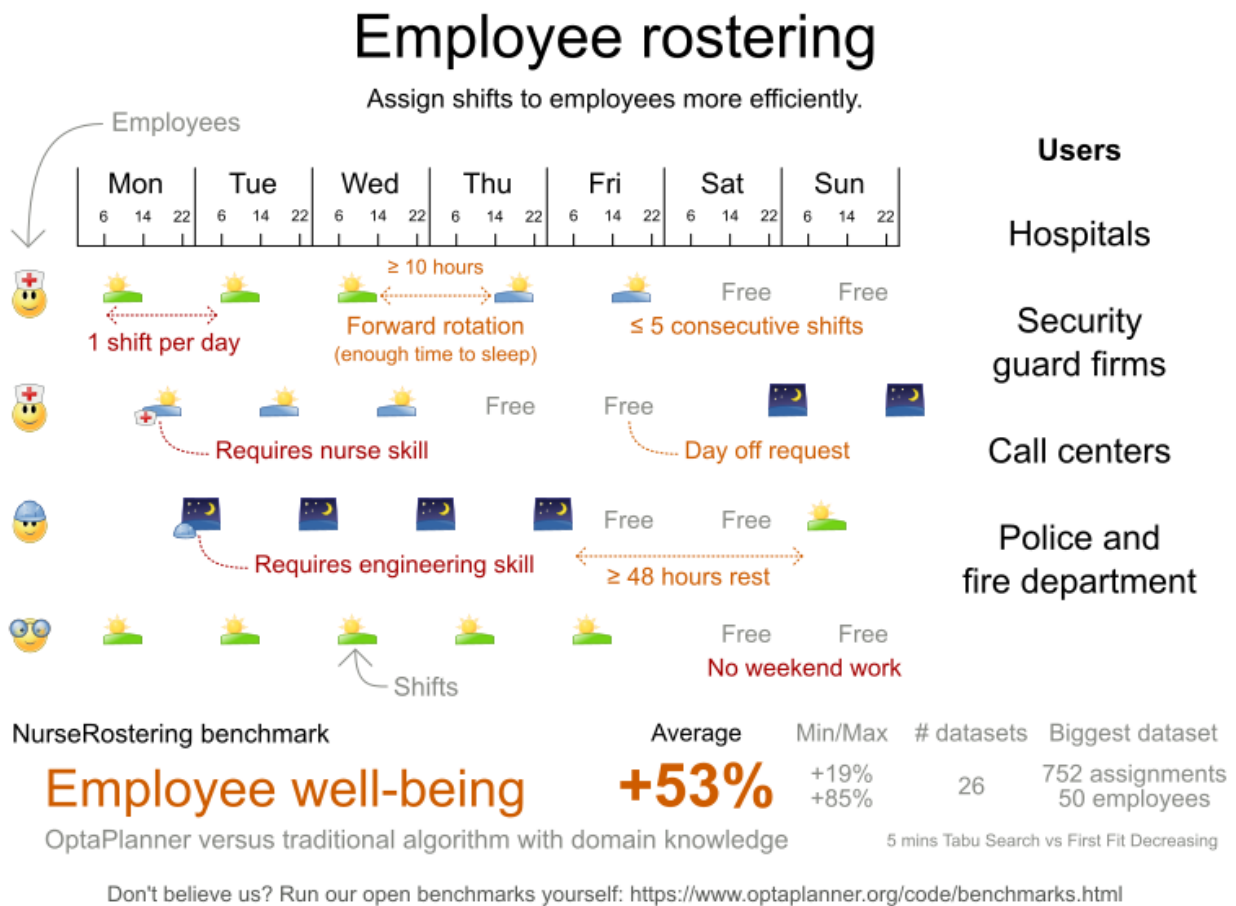
Problem size

There are three dataset types:

- sprint: must be solved in seconds.

- medium: must be solved in minutes.

- long: must be solved in hours.

toy1   has 1 skills, 3 shiftTypes, 2 patterns, 1 contracts, 6 employees, 7 shiftDates, 35 shiftAssignments and   0 requests with a search space of   10^27.
toy2   has 1 skills, 3 shiftTypes, 3 patterns, 2 contracts, 20 employees, 28 shiftDates, 180 shiftAssignments and 140 requests with a search space of 10^234.

sprint01   has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint02   has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint03   has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint04   has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint05   has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint06   has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint07   has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint08   has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint09   has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint10   has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_hint01 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_hint02 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_hint03 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_late01 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_late02 has 1 skills, 3 shiftTypes, 4 patterns, 3 contracts, 10 employees, 28 shiftDates, 144 shiftAssignments and 139 requests with a search space of 10^144.
sprint_late03 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 160 shiftAssignments and 150 requests with a search space of 10^160.
sprint_late04 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 160 shiftAssignments and 150 requests with a search space of 10^160.
sprint_late05 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_late06 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_late07 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_late08 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and   0 requests with a search space of 10^152.
sprint_late09 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and   0 requests with a search space of 10^152.
sprint_late10 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.

medium01   has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of 10^906.
medium02   has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of 10^906.

medium03   has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of 10^906.
medium04   has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of 10^906.
medium05   has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of 10^906.
medium_hint01 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of 10^632.
medium_hint02 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of 10^632.
medium_hint03 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of 10^632.
medium_late01 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 424 shiftAssignments and 390 requests with a search space of 10^626.
medium_late02 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of 10^632.
medium_late03 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of 10^632.
medium_late04 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 416 shiftAssignments and 390 requests with a search space of 10^614.
medium_late05 has 2 skills, 5 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 452 shiftAssignments and 390 requests with a search space of 10^667.

long01   has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of 10^1250.
long02   has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of 10^1250.
long03   has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of 10^1250.
long04   has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of 10^1250.
long05   has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of 10^1250.
long_hint01   has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and   0 requests with a search space of 10^1257.
long_hint02   has 2 skills, 5 shiftTypes, 7 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and   0 requests with a search space of 10^1257.
long_hint03   has 2 skills, 5 shiftTypes, 7 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and   0 requests with a search space of 10^1257.
long_late01   has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and   0 requests with a search space of 10^1277.
long_late02   has 2 skills, 5 shiftTypes, 9 patterns, 4 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and   0 requests with a search space of 10^1277.
long_late03   has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and   0 requests with a search space of 10^1277.
long_late04   has 2 skills, 5 shiftTypes, 9 patterns, 4 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and   0 requests with a search space of 10^1277.
long_late05   has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and   0 requests with a search space of 10^1257.

Figure 4.13. Domain model
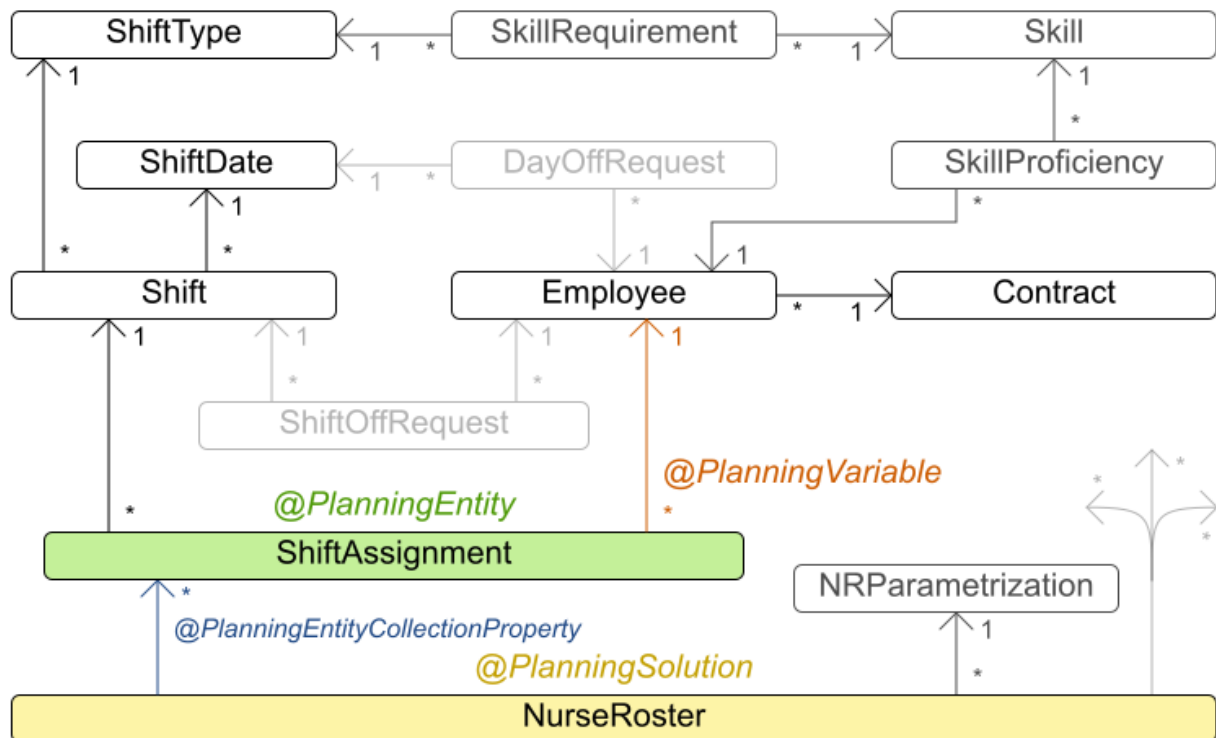
4.13. TRAVELING TOURNAMENT PROBLEM (TTP)

Schedule matches between *n* teams.
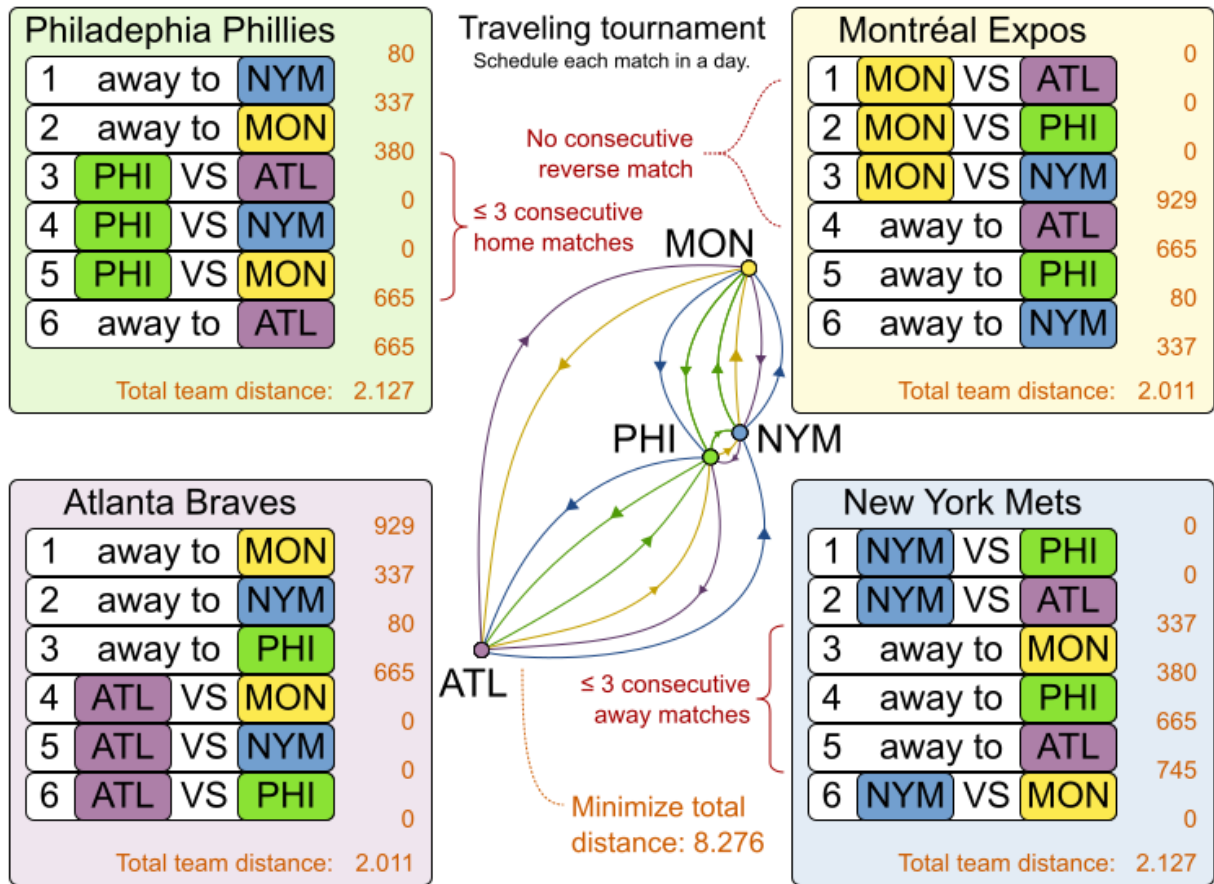
Hard constraints:

- Each team plays twice against every other team: once home and once away.

- Each team has exactly one match on each timeslot.

- No team must have more than three consecutive home or three consecutive away matches.

- No repeaters: no two consecutive matches of the same two opposing teams.

Soft constraints:

- Minimize the total distance traveled by all teams.

The problem is defined on Michael Trick's website (which contains the world records too) .

Problem size

```
1-nl04    has  6 days,  4 teams and   12 matches with a search space of    10^5.
1-nl06    has 10 days,  6 teams and   30 matches with a search space of   10^19.
1-nl08    has 14 days,  8 teams and   56 matches with a search space of   10^43.
1-nl10    has 18 days, 10 teams and   90 matches with a search space of   10^79.
1-nl12    has 22 days, 12 teams and  132 matches with a search space of  10^126.
1-nl14    has 26 days, 14 teams and  182 matches with a search space of  10^186.
1-nl16    has 30 days, 16 teams and  240 matches with a search space of  10^259.
2-bra24   has 46 days, 24 teams and  552 matches with a search space of  10^692.
3-nfl16   has 30 days, 16 teams and  240 matches with a search space of  10^259.
3-nfl18   has 34 days, 18 teams and  306 matches with a search space of  10^346.
```

```
3-nfl20     has 38 days, 20 teams and  380 matches with a search space of  10^447.
3-nfl22     has 42 days, 22 teams and  462 matches with a search space of  10^562.
3-nfl24     has 46 days, 24 teams and  552 matches with a search space of  10^692.
3-nfl26     has 50 days, 26 teams and  650 matches with a search space of  10^838.
3-nfl28     has 54 days, 28 teams and  756 matches with a search space of  10^999.
3-nfl30     has 58 days, 30 teams and  870 matches with a search space of 10^1175.
3-nfl32     has 62 days, 32 teams and  992 matches with a search space of 10^1367.
4-super04  has  6 days,  4 teams and   12 matches with a search space of    10^5.
4-super06  has 10 days,  6 teams and   30 matches with a search space of   10^19.
4-super08  has 14 days,  8 teams and   56 matches with a search space of   10^43.
4-super10  has 18 days, 10 teams and   90 matches with a search space of   10^79.
4-super12  has 22 days, 12 teams and  132 matches with a search space of  10^126.
4-super14  has 26 days, 14 teams and  182 matches with a search space of  10^186.
5-galaxy04 has  6 days,  4 teams and   12 matches with a search space of    10^5.
5-galaxy06 has 10 days,  6 teams and   30 matches with a search space of   10^19.
5-galaxy08 has 14 days,  8 teams and   56 matches with a search space of   10^43.
5-galaxy10 has 18 days, 10 teams and   90 matches with a search space of   10^79.
5-galaxy12 has 22 days, 12 teams and  132 matches with a search space of  10^126.
5-galaxy14 has 26 days, 14 teams and  182 matches with a search space of  10^186.
5-galaxy16 has 30 days, 16 teams and  240 matches with a search space of  10^259.
5-galaxy18 has 34 days, 18 teams and  306 matches with a search space of  10^346.
5-galaxy20 has 38 days, 20 teams and  380 matches with a search space of  10^447.
5-galaxy22 has 42 days, 22 teams and  462 matches with a search space of  10^562.
5-galaxy24 has 46 days, 24 teams and  552 matches with a search space of  10^692.
5-galaxy26 has 50 days, 26 teams and  650 matches with a search space of  10^838.
5-galaxy28 has 54 days, 28 teams and  756 matches with a search space of  10^999.
5-galaxy30 has 58 days, 30 teams and  870 matches with a search space of 10^1175.
5-galaxy32 has 62 days, 32 teams and  992 matches with a search space of 10^1367.
5-galaxy34 has 66 days, 34 teams and 1122 matches with a search space of 10^1576.
5-galaxy36 has 70 days, 36 teams and 1260 matches with a search space of 10^1801.
5-galaxy38 has 74 days, 38 teams and 1406 matches with a search space of 10^2042.
5-galaxy40 has 78 days, 40 teams and 1560 matches with a search space of 10^2301.
```

4.14. CHEAP TIME SCHEDULING

Schedule all tasks in time and on a machine to minimize power cost. Power prices differs in time. This is a form of *job shop scheduling*.

Hard constraints:

- Start time limits: Each task must start between its earliest start and latest start limit.

- Maximum capacity: The maximum capacity for each resource for each machine must not be exceeded.

- Startup and shutdown: Each machine must be active in the periods during which it has assigned tasks. Between tasks it is allowed to be idle to avoid startup and shutdown costs.

Medium constraints:

- Power cost: Minimize the total power cost of the whole schedule.

  - Machine power cost: Each active or idle machine consumes power, which infers a power cost (depending on the power price during that time).

- Task power cost: Each task consumes power too, which infers a power cost (depending on the power price during its time).

- Machine startup and shutdown cost: Every time a machine starts up or shuts down, an extra cost is inflicted.

Soft constraints (addendum to the original problem definition):

- Start early: Prefer starting a task sooner rather than later.

The problem is defined by the ICON challenge.

Problem size

```
sample01   has 3 resources,   2 machines, 288 periods and   25 tasks with a search space of
10^53.
sample02   has 3 resources,   2 machines, 288 periods and   50 tasks with a search space of
10^114.
sample03   has 3 resources,   2 machines, 288 periods and  100 tasks with a search space of
10^226.
sample04   has 3 resources,   5 machines, 288 periods and  100 tasks with a search space of
10^266.
sample05   has 3 resources,   2 machines, 288 periods and  250 tasks with a search space of
10^584.
sample06   has 3 resources,   5 machines, 288 periods and  250 tasks with a search space of
10^673.
sample07   has 3 resources,   2 machines, 288 periods and 1000 tasks with a search space of
10^2388.
sample08   has 3 resources,   5 machines, 288 periods and 1000 tasks with a search space of
10^2748.
sample09   has 4 resources,  20 machines, 288 periods and 2000 tasks with a search space of
10^6668.
instance00 has 1 resources,  10 machines, 288 periods and  200 tasks with a search space of
10^595.
instance01 has 1 resources,  10 machines, 288 periods and  200 tasks with a search space of
10^599.
instance02 has 1 resources,  10 machines, 288 periods and  200 tasks with a search space of
10^599.
instance03 has 1 resources,  10 machines, 288 periods and  200 tasks with a search space of
10^591.
instance04 has 1 resources,  10 machines, 288 periods and  200 tasks with a search space of
10^590.
instance05 has 2 resources,  25 machines, 288 periods and  200 tasks with a search space of
10^667.
instance06 has 2 resources,  25 machines, 288 periods and  200 tasks with a search space of
10^660.
instance07 has 2 resources,  25 machines, 288 periods and  200 tasks with a search space of
10^662.
instance08 has 2 resources,  25 machines, 288 periods and  200 tasks with a search space of
10^651.
instance09 has 2 resources,  25 machines, 288 periods and  200 tasks with a search space of
10^659.
instance10 has 2 resources,  20 machines, 288 periods and  500 tasks with a search space of
10^1657.
instance11 has 2 resources,  20 machines, 288 periods and  500 tasks with a search space of
10^1644.
```

instance12 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1637.

instance13 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1659.

instance14 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1643.

instance15 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10^1782.

instance16 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10^1778.

instance17 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10^1764.

instance18 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10^1769.

instance19 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10^1778.

instance20 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10^3689.

instance21 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10^3678.

instance22 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10^3706.

instance23 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10^3676.

instance24 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10^3681.

instance25 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10^3774.

instance26 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10^3737.

instance27 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10^3744.

instance28 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10^3731.

instance29 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10^3746.

instance30 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10^7718.

instance31 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10^7740.

instance32 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10^7686.

instance33 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10^7672.

instance34 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10^7695.

instance35 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10^7807.

instance36 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10^7814.

instance37 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10^7764.

instance38 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10^7736.

instance39 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10^7783.

> instance40 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of
> 10^15976.
> instance41 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of
> 10^15935.
> instance42 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of
> 10^15887.
> instance43 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of
> 10^15896.
> instance44 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of
> 10^15885.
> instance45 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
> 10^20173.
> instance46 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
> 10^20132.
> instance47 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
> 10^20126.
> instance48 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
> 10^20110.
> instance49 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
> 10^20078.

4.15. INVESTMENT ASSET CLASS ALLOCATION (PORTFOLIO OPTIMIZATION)

Decide the relative quantity to invest in each asset class.

Hard constraints:

- Risk maximum: the total standard deviation must not be higher than the standard deviation maximum.

    - Total standard deviation calculation takes asset class correlations into account by applying Markowitz Portfolio Theory.

- Region maximum: Each region has a quantity maximum.

- Sector maximum: Each sector has a quantity maximum.

Soft constraints:

- Maximize expected return.

Problem size

> de_smet_1 has 1 regions, 3 sectors and 11 asset classes with a search space of 10^4.
> irrinki_1 has 2 regions, 3 sectors and 6 asset classes with a search space of 10^3.

Larger datasets have not been created or tested yet, but should not pose a problem. A good source of data is this Asset Correlation website.

4.16. CONFERENCE SCHEDULING

Assign each conference talk to a timeslot and a room. Timeslots can overlap. Read/write to/from an ***.xlsx** file that can be edited with LibreOffice or Excel too.

Hard constraints:

- Talk type of timeslot: The type of a talk must match the timeslot's talk type.

- Room unavailable timeslots: A talk's room must be available during the talk's timeslot.

- Room conflict: Two talks can't use the same room during overlapping timeslots.

- Speaker unavailable timeslots: Every talk's speaker must be available during the talk's timeslot.

- Speaker conflict: Two talks can't share a speaker during overlapping timeslots.

- Generic purpose timeslot and room tags

  - Speaker required timeslot tag: If a speaker has a required timeslot tag, then all his/her talks must be assigned to a timeslot with that tag.

  - Speaker prohibited timeslot tag: If a speaker has a prohibited timeslot tag, then all his/her talks cannot be assigned to a timeslot with that tag.

  - Talk required timeslot tag: If a talk has a required timeslot tag, then it must be assigned to a timeslot with that tag.

  - Talk prohibited timeslot tag: If a talk has a prohibited timeslot tag, then it cannot be assigned to a timeslot with that tag.

  - Speaker required room tag: If a speaker has a required room tag, then all his/her talks must be assigned to a room with that tag.

  - Speaker prohibited room tag: If a speaker has a prohibited room tag, then all his/her talks cannot be assigned to a room with that tag.

  - Talk required room tag: If a talk has a required room tag, then it must be assigned to a room with that tag.

  - Talk prohibited room tag: If a talk has a prohibited room tag, then it cannot be assigned to a room with that tag.

- Talk mutually-exclusive-talks tag: Talks that share such a tag must not be scheduled in overlapping timeslots.

- Talk prerequisite talks: A talk must be scheduled after all its prerequisite talks.

Soft constraints:

- Theme track conflict: Minimize the number of talks that share a same theme tag during overlapping timeslots.

- Sector conflict: Minimize the number of talks that share a same sector tag during overlapping timeslots.

- Content audience level flow violation: For every content tag, schedule the introductory talks before the advanced talks.

- Audience level diversity: For every timeslot, maximize the number of talks with a different audience level.

- Language diversity: For every timeslot, maximize the number of talks with a different language.

- Generic purpose timeslot and room tags

  - Speaker preferred timeslot tag: If a speaker has a preferred timeslot tag, then all his/her talks should be assigned to a timeslot with that tag.

  - Speaker undesired timeslot tag: If a speaker has an undesired timeslot tag, then all his/her talks should not be assigned to a timeslot with that tag.

  - Talk preferred timeslot tag: If a talk has a preferred timeslot tag, then it should be assigned to a timeslot with that tag.

  - Talk undesired timeslot tag: If a talk has an undesired timeslot tag, then it should not be assigned to a timeslot with that tag.

  - Speaker preferred room tag: If a speaker has a preferred room tag, then all his/her talks should be assigned to a room with that tag.

  - Speaker undesired room tag: If a speaker has an undesired room tag, then all his/her talks should not be assigned to a room with that tag.

  - Talk preferred room tag: If a talk has a preferred room tag, then it should be assigned to a room with that tag.

  - Talk undesired room tag: If a talk has an undesired room tag, then it should not be assigned to a room with that tag.

- Same day talks: All talks that share a same theme tag or content tag should be scheduled in the minimum number of days (ideally in the same day).

Figure 4.14. Value proposition
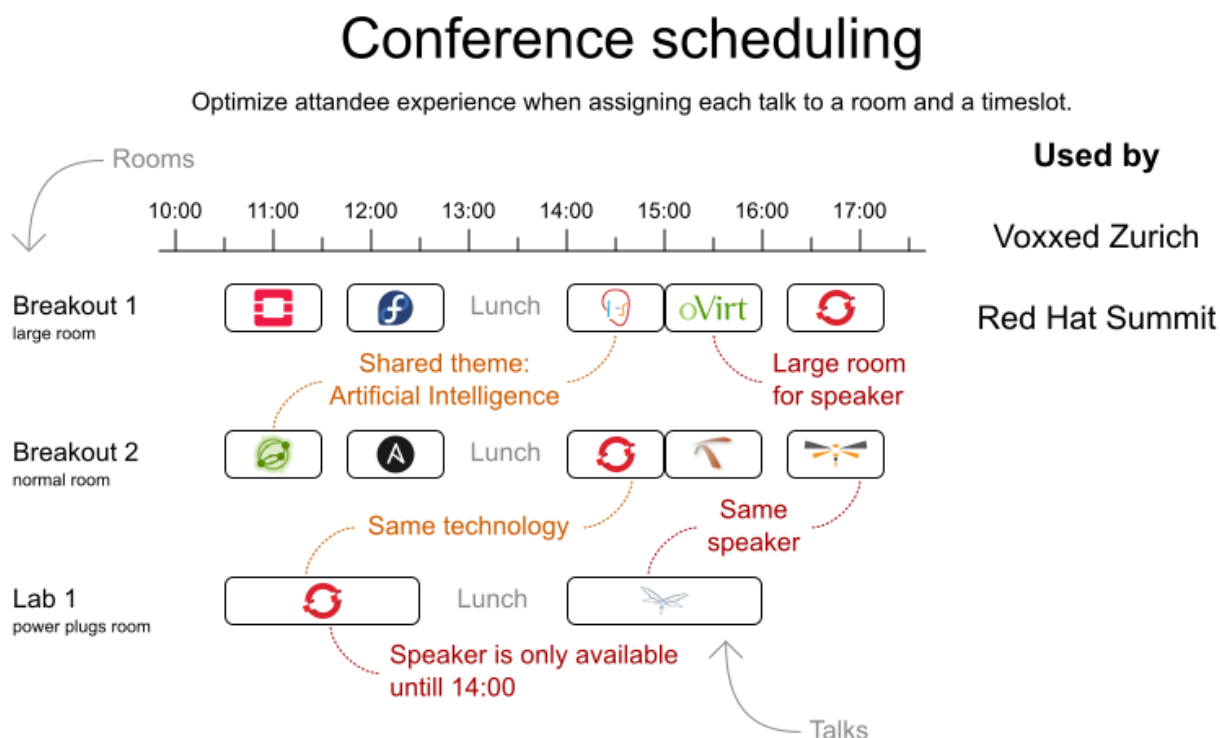
Problem size

```
18talks-6timeslots-5rooms    has  18 talks,  6 timeslots and  5 rooms with a search space of  10^26.
36talks-12timeslots-5rooms   has  36 talks, 12 timeslots and  5 rooms with a search space of  10^64.
72talks-12timeslots-10rooms  has  72 talks, 12 timeslots and 10 rooms with a search space of
10^149.
108talks-18timeslots-10rooms has 108 talks, 18 timeslots and 10 rooms with a search space of
10^243.
216talks-18timeslots-20rooms has 216 talks, 18 timeslots and 20 rooms with a search space of
10^552.
```

4.17. ROCK TOUR

Drive the rock bus from show to show, but schedule shows only on available days.

Hard constraints:

- Schedule every required show.

- Schedule as many shows as possible.

Medium constraints:

- Maximize revenue opportunity.

- Minimize driving time.

- Visit sooner than later.

Soft constraints:

- Avoid long driving times.

Problem size

> 47shows has 47 shows with a search space of 10^59.

4.18. FLIGHT CREW SCHEDULING

Assign flights to pilots and flight attendants.

Hard constraints:

- Required skill: each flight assignment has a required skill. For example, flight AB0001 requires 2 pilots and 3 flight attendants.

- Flight conflict: each employee can only attend one flight at the same time

- Transfer between two flights: between two flights, an employee must be able to transfer from the arrival airport to the departure airport. For example, Ann arrives in Brussels at 10:00 and departs in Amsterdam at 15:00.

- Employee unavailability: the employee must be available on the day of the flight. For example, Ann is on PTO on 1–Feb.

Soft constraints:

- First assignment departing from home

- Last assignment arriving at home

- Load balance flight duration total per employee

Problem size

> 175flights-7days-Europe has 2 skills, 50 airports, 150 employees, 175 flights and 875 flight
> assignments with a search space of 10^1904.
> 700flights-28days-Europe has 2 skills, 50 airports, 150 employees, 700 flights and 3500 flight
> assignments with a search space of 10^7616.
> 875flights-7days-Europe has 2 skills, 50 airports, 750 employees, 875 flights and 4375 flight
> assignments with a search space of 10^12578.
> 175flights-7days-US   has 2 skills, 48 airports, 150 employees, 175 flights and 875 flight
> assignments with a search space of 10^1904.

CHAPTER 5. DOWNLOADING RED HAT BUILD OF OPTAPLANNER EXAMPLES

You can download the Red Hat build of OptaPlanner examples as a part of the Red Hat Decision Manager add-ons package available on the Red Hat Customer Portal.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required), and select the product and version from the drop-down options:

   - Product: Decision Manager

   - Version: 7.11

2. Download **Red Hat Decision Manager 7.11 Add Ons**.

3. Extract the **rhdm-7.11.0-add-ons.zip** file. The extracted **add-ons** folder contains the **rhdm-7.11.0-planner-engine.zip** file.

4. Extract the **rhdm-7.11.0-planner-engine.zip** file.

Result

The extracted **rhdm-7.11.0-planner-engine** directory contains example source code under the following subdirectories:

- **examples/sources/src/main/java/org/optaplanner/examples**

- **examples/sources/src/main/resources/org/optaplanner/examples**

5.1. RUNNING OPTAPLANNER EXAMPLES

Red Hat build of OptaPlanner includes several examples that demonstrate a variety of planning use cases. Download and use the examples to explore different types of planning solutions.

Prerequisites

- You have downloaded and extracted the examples as described in Chapter 5, *Downloading Red Hat build of OptaPlanner examples*.

Procedure

1. To run the examples, in the **rhdm-7.11.0-planner-engine/examples** directory enter one of the following commands:
   Linux or Mac:

   ```
   $ ./runExamples.sh
   ```

   Windows:

   ```
   $ runExamples.bat
   ```

   The OptaPlanner Examples window opens.

2. Select an example to run that example.

> **NOTE**
>
> Red Hat build of OptaPlanner has no GUI dependencies. It runs just as well on a server or a mobile JVM as it does on the desktop.

5.2. RUNNING THE RED HAT BUILD OF OPTAPLANNER EXAMPLES IN AN IDE (INTELLIJ, ECLIPSE, OR NETBEANS)

If you use an integrated development environment (IDE), such as IntelliJ, Eclipse, or Netbeans, you can run your downloaded OptaPlanner examples within your development environment.

Prerequisites

- You have downloaded and extracted the OptaPlanner examples as described in Chapter 5, *Downloading Red Hat build of OptaPlanner examples* .

Procedure

1. Open the OptaPlanner examples as a new project:

    a. For IntelliJ or Netbeans, open **examples/sources/pom.xml** as the new project. The Maven integration guides you through the rest of the installation. Skip the rest of the steps in this procedure.

    b. For Eclipse, open a new project for the **/examples/binaries** directory, located under the **rhdm-7.11.0-planner-engine** directory.

2. Add all the JAR files that are in the **binaries** directory to the classpath, except for the **examples/binaries/optaplanner-examples-7.52.0.Final-redhat-00007.jar** file.

3. Add the Java source directory **src/main/java** and the Java resources directory **src/main/resources**, located under the **rhdm-7.11.0-planner-engine/examples/sources/** directory.

4. Create a run configuration:

    - Main class: **org.optaplanner.examples.app.OptaPlannerExamplesApp**

    - VM parameters (optional): **-Xmx512M -server -Dorg.optaplanner.examples.dataDir=examples/sources/data**

    - Working directory: **examples/sources**

5. Run the run configuration.

CHAPTER 6. GETTING STARTED WITH OPTAPLANNER IN BUSINESS CENTRAL: AN EMPLOYEE ROSTERING EXAMPLE

You can build and deploy the **employee-rostering** sample project in Business Central. The project demonstrates how to create each of the Business Central assets required to solve the shift rostering planning problem and use Red Hat build of OptaPlanner to find the best possible solution.

You can deploy the preconfigured **employee-rostering** project in Business Central. Alternatively, you can create the project yourself using Business Central.

> **NOTE**
>
> The **employee-rostering** sample project in Business Central does not include a data set. You must supply a data set in XML format using a REST API call.

6.1. DEPLOYING THE EMPLOYEE ROSTERING SAMPLE PROJECT IN BUSINESS CENTRAL

Business Central includes a number of sample projects that you can use to get familiar with the product and its features. The employee rostering sample project is designed and created to demonstrate the shift rostering use case for Red Hat build of OptaPlanner. Use the following procedure to deploy and run the employee rostering sample in Business Central.

Prerequisites

- Red Hat Decision Manager has been downloaded and installed. For installation options, see *Planning a Red Hat Decision Manager installation* .

- You have started Red Hat Decision Manager, as described in the installation documentation, and you are logged in to Business Central as a user with **admin** permissions.

Procedure

1. In Business Central, click **Menu → Design → Projects**.

2. In the preconfigured **MySpace** space, click **Try Samples**.

3. Select **employee-rostering** from the list of sample projects and click **Ok** in the upper-right corner to import the project.

4. After the asset list has complied, click **Build & Deploy** to deploy the employee rostering example.

The rest of this document explains each of the project assets and their configuration.

6.2. RE-CREATING THE EMPLOYEE ROSTERING SAMPLE PROJECT

The employee rostering sample project is a preconfigured project available in Business Central. You can learn about how to deploy this project in Section 6.1, "Deploying the employee rostering sample project in Business Central".

You can create the employee rostering example "from scratch". You can use the workflow in this example to create a similar project of your own in Business Central.

6.2.1. Setting up the employee rostering project

To start developing a solver in Business Central, you must set up the project.

Prerequisites

- Red Hat Decision Manager has been downloaded and installed.

- You have deployed Business Central and logged in with a user that has the **admin** role.

Procedure

1. Create a new project in Business Central by clicking **Menu → Design → Projects → Add Project**.

2. In the **Add Project** window, fill out the following fields:

   - Name: **employee-rostering**

   - **Description**(optional): Employee rostering problem optimization using OptaPlanner. Assigns employees to shifts based on their skill.

   Optionally, click **Configure Advanced Options** to populate the **Group ID**, **Artifact ID**, and **Version** information.

   - Group ID: **employeerostering**

   - Artifact ID: **employeerostering**

   - Version: **1.0.0-SNAPSHOT**

3. Click **Add** to add the project to the Business Central project repository.

6.2.2. Problem facts and planning entities

Each of the domain classes in the employee rostering planning problem is categorized as one of the following:

- An unrelated class: not used by any of the score constraints. From a planning standpoint, this data is obsolete.

- A *problem fact* class: used by the score constraints, but does not change during planning (as long as the problem stays the same), for example, **Shift** and **Employee**. All the properties of a problem fact class are problem properties.

- A *planning entity* class: used by the score constraints and changes during planning, for example, **ShiftAssignment**. The properties that change during planning are *planning variables*. The other properties are problem properties.
  Ask yourself the following questions:

- *What class changes during planning?*

- *Which class has variables that I want the* **Solver** *to change?*
  That class is a planning entity.

A planning entity class needs to be annotated with the **@PlanningEntity** annotation, or defined in Business Central using the Red Hat build of OptaPlanner dock in the domain designer.

Each planning entity class has one or more *planning variables*, and must also have one or more defining properties.

Most use cases have only one planning entity class, and only one planning variable per planning entity class.

6.2.3. Creating the data model for the employee rostering project

Use this section to create the data objects required to run the employee rostering sample project in Business Central.

Prerequisites

- You have completed the project setup described in Section 6.2.1, "Setting up the employee rostering project".

Procedure

1. With your new project, either click **Data Object** in the project perspective, or click **Add Asset →
   Data Object** to create a new data object.

2. Name the first data object **Timeslot**, and select **employeerostering.employeerostering** as the **Package**.
   Click **Ok**.

3. In the **Data Objects** perspective, click **+add field** to add fields to the **Timeslot** data object.

4. In the **id** field, type **endTime**.

5. Click the drop-down menu next to **Type** and select **LocalDateTime**.

6. Click **Create and continue** to add another field.

7. Add another field with the **id startTime** and **Type LocalDateTime**.

8. Click **Create**.

9. Click **Save** in the upper-right corner to save the **Timeslot** data object.

10. Click the **x** in the upper-right corner to close the **Data Objects** perspective and return to the **Assets** menu.

11. Using the previous steps, create the following data objects and their attributes:

    Table 6.1. Skill

id	Type
name	**String**

    Table 6.2. Employee

id	Type
name	**String**
skills	**employeerostering.employeerostering.Skill[List]**

Table 6.3. Shift

id	Type
requiredSkill	**employeerostering.employeerostering.Skill**
timeslot	**employeerostering.employeerostering.Timeslot**

Table 6.4. DayOffRequest

id	Type
date	**LocalDate**
employee	**employeerostering.employeerostering.Employee**

Table 6.5. ShiftAssignment

id	Type
employee	**employeerostering.employeerostering.Employee**
shift	**employeerostering.employeerostering.Shift**

For more examples of creating data objects, see *Getting started with decision services* .

6.2.3.1. Creating the employee roster planning entity

In order to solve the employee rostering planning problem, you must create a planning entity and a solver. The planning entity is defined in the domain designer using the attributes available in the Red Hat build of OptaPlanner dock.

Use the following procedure to define the **ShiftAssignment** data object as the planning entity for the employee rostering example.

Prerequisites

- You have created the relevant data objects and planning entity required to run the employee rostering example by completing the procedures in Section 6.2.3, "Creating the data model for the employee rostering project".

Procedure

1. From the project **Assets** menu, open the **ShiftAssignment** data object.

2. In the **Data Objects** perspective, open the OptaPlanner dock by clicking the   on the right.

3. Select **Planning Entity**.

4. Select **employee** from the list of fields under the **ShiftAssignment** data object.

5. In the OptaPlanner dock, select **Planning Variable**.
   In the **Value Range Id** input field, type **employeeRange**. This adds the **@ValueRangeProvider** annotation to the planning entity, which you can view by clicking the **Source** tab in the designer.

   The value range of a planning variable is defined with the **@ValueRangeProvider** annotation. A **@ValueRangeProvider** annotation always has a property **id**, which is referenced by the **@PlanningVariable** property **valueRangeProviderRefs**.

6. Close the dock and click **Save** to save the data object.

6.2.3.2. Creating the employee roster planning solution

The employee roster problem relies on a defined planning solution. The planning solution is defined in the domain designer using the attributes available in the Red Hat build of OptaPlanner dock.

Prerequisites

- You have created the relevant data objects and planning entity required to run the employee rostering example by completing the procedures in Section 6.2.3, "Creating the data model for the employee rostering project" and Section 6.2.3.1, "Creating the employee roster planning entity".

Procedure

1. Create a new data object with the identifier **EmployeeRoster**.

2. Create the following fields:

   Table 6.6. EmployeeRoster

id	Type
dayOffRequestList	**employeerostering.employeerostering.DayOffRequest[List]**
shiftAssignmentList	**employeerostering.employeerostering.ShiftAssignment[List]**

id	Type
shiftList	employeerostering.employeerostering.Shift[List]
skillList	employeerostering.employeerostering.Skill[List]
timeslotList	employeerostering.employeerostering.Timeslot[List]

3. In the **Data Objects** perspective, open the OptaPlanner dock by clicking the   on the right.

4. Select **Planning Solution**.

5. Leave the default **Hard soft score** as the **Solution Score Type**. This automatically generates a **score** field in the **EmployeeRoster** data object with the solution score as the type.

6. Add a new field with the following attributes:

id	Type
employeeList	employeerostering.employeerostering.Employee[List]

7. With the **employeeList** field selected, open the OptaPlanner dock and select the **Planning Value Range Provider** box.
   In the **id** field, type **employeeRange**. Close the dock.

8. Click **Save** in the upper-right corner to save the asset.

6.2.4. Employee rostering constraints

Employee rostering is a planning problem. All planning problems include constraints that must be satisfied in order to find an optimal solution.

The employee rostering sample project in Business Central includes the following hard and soft constraints:

Hard constraint

- Employees are only assigned one shift per day.

- All shifts that require a particular employee skill are assigned an employee with that particular skill.

Soft constraints

- All employees are assigned a shift.

- If an employee requests a day off, their shift is reassigned to another employee.

Hard and soft constraints are defined in Business Central using either the free-form DRL designer, or using guided rules.

6.2.4.1. DRL (Drools Rule Language) rules

DRL (Drools Rule Language) rules are business rules that you define directly in **.drl** text files. These DRL files are the source in which all other rule assets in Business Central are ultimately rendered. You can create and manage DRL files within the Business Central interface, or create them externally as part of a Maven or Java project using Red Hat CodeReady Studio or another integrated development environment (IDE). A DRL file can contain one or more rules that define at a minimum the rule conditions (**when**) and actions (**then**). The DRL designer in Business Central provides syntax highlighting for Java, DRL, and XML.

DRL files consist of the following components:

Components in a DRL file

```
package

import

function  // Optional

query  // Optional

declare   // Optional

global   // Optional

rule "rule name"
   // Attributes
   when
      // Conditions
   then
      // Actions
end

rule "rule2 name"

...
```

The following example DRL rule determines the age limit in a loan application decision service:

Example rule for loan application age limit

```
rule "Underage"
  salience 15
  agenda-group "applicationGroup"
  when
    $application : LoanApplication()
    Applicant( age < 21 )
  then
```

```
    $application.setApproved( false );
    $application.setExplanation( "Underage" );
end
```

A DRL file can contain single or multiple rules, queries, and functions, and can define resource declarations such as imports, globals, and attributes that are assigned and used by your rules and queries. The DRL package must be listed at the top of a DRL file and the rules are typically listed last. All other DRL components can follow any order.

Each rule must have a unique name within the rule package. If you use the same rule name more than once in any DRL file in the package, the rules fail to compile. Always enclose rule names with double quotation marks (**rule "rule name"**) to prevent possible compilation errors, especially if you use spaces in rule names.

All data objects related to a DRL rule must be in the same project package as the DRL file in Business Central. Assets in the same package are imported by default. Existing assets in other packages can be imported with the DRL rule.

6.2.4.2. Defining constraints for employee rostering using the DRL designer

You can create constraint definitions for the employee rostering example using the free-form DRL designer in Business Central.

Use this procedure to create a *hard constraint* where no employee is assigned a shift that begins less than 10 hours after their previous shift ended.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.

2. Click **Add Asset → DRL file**.

3. In the **DRL file** name field, type **ComplexScoreRules**.

4. Select the **employeerostering.employeerostering** package.

5. Click **+Ok** to create the DRL file.

6. In the **Model** tab of the DRL designer, define the **Employee10HourShiftSpace** rule as a DRL file:

   ```
   package employeerostering.employeerostering;

   rule "Employee10HourShiftSpace"
       when
           $shiftAssignment : ShiftAssignment( $employee : employee != null, $shiftEndDateTime :
   shift.timeslot.endTime)
           ShiftAssignment( this != $shiftAssignment, $employee == employee, $shiftEndDateTime
   <= shift.timeslot.endTime,
                   $shiftEndDateTime.until(shift.timeslot.startTime,
   java.time.temporal.ChronoUnit.HOURS) <10)
       then
           scoreHolder.addHardConstraintMatch(kcontext, -1);
   end
   ```

7. Click **Save** to save the DRL file.

For more information about creating DRL files, see *Designing a decision service using DRL rules* .

6.2.5. Creating rules for employee rostering using guided rules

You can create rules that define hard and soft constraints for employee rostering using the guided rules designer in Business Central.

6.2.5.1. Guided rules

Guided rules are business rules that you create in a UI-based guided rules designer in Business Central that leads you through the rule-creation process. The guided rules designer provides fields and options for acceptable input based on the data objects for the rule being defined. The guided rules that you define are compiled into Drools Rule Language (DRL) rules as with all other rule assets.

All data objects related to a guided rule must be in the same project package as the guided rule. Assets in the same package are imported by default. After you create the necessary data objects and the guided rule, you can use the **Data Objects** tab of the guided rules designer to verify that all required data objects are listed or to import other existing data objects by adding a **New item**.

6.2.5.2. Creating a guided rule to balance employee shift numbers

The **BalanceEmployeesShiftNumber** guided rule creates a soft constraint that ensures shifts are assigned to employees in a way that is balanced as evenly as possible. It does this by creating a score penalty that increases when shift distribution is less even. The score formula, implemented by the rule, incentivizes the Solver to distribute shifts in a more balanced way.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.

2. Click **Add Asset → Guided Rule**.

3. Enter **BalanceEmployeesShiftNumber** as the **Guided Rule** name and select the **employeerostering.employeerostering** Package.

4. Click **Ok** to create the rule asset.

5. Add a **WHEN** condition by clicking the ✛ in the **WHEN** field.

6. Select **Employee** in the **Add a condition to the rule**window. Click **+Ok**.

7. Click the **Employee** condition to modify the constraints and add the variable name **$employee**.

8. Add the WHEN condition **From Accumulate**.

   a. Above the **From Accumulate** condition, click **click to add pattern** and select **Number** as the fact type from the drop-down list.

   b. Add the variable name **$shiftCount** to the **Number** condition.

   c. Below the **From Accumulate** condition, click **click to add pattern** and select the **ShiftAssignment** fact type from the drop-down list.

   d. Add the variable name **$shiftAssignment** to the **ShiftAssignment** fact type.

   e. Click the **ShiftAssignment** condition again and from the **Add a restriction on a field** drop-down list, select **employee**.

   f. Select **equal to** from the drop-down list next to the **employee** constraint.

   g. Click the ✎ icon next to the drop-down button to add a variable, and click **Bound variable** in the **Field value** window.

   h. Select **$employee** from the drop-down list.

   i. In the **Function** box type **count($shiftAssignment)**.

9. Add the THEN condition by clicking the ✚ in the **THEN** field.

10. Select **Modify Soft Score** in the **Add a new action** window. Click **+Ok**.

   a. Type the following expression into the box: **-($shiftCount.intValue()*$shiftCount.intValue())**

11. Click **Validate** in the upper-right corner to check all rule conditions are valid. If the rule validation fails, address any problems described in the error message, review all components in the rule, and try again to validate the rule until the rule passes.

12. Click **Save** to save the rule.

For more information about creating guided rules, see *Designing a decision service using guided rules* .

6.2.5.3. Creating a guided rule for no more than one shift per day

The **OneEmployeeShiftPerDay** guided rule creates a hard constraint that employees are not assigned more than one shift per day. In the employee rostering example, this constraint is created using the guided rule designer.

OneEmployeeShiftPerDay.rdrl - Guided Rules ∨   Save | Delete | Rename | Copy | Validate | Latest Version ∨

Editor   Overview   Source   Data Objects

EXTENDS   - None -   ∨

WHEN

1. $shiftAssignment : ShiftAssignment(employee != null)
         ShiftAssignment(this != $shiftAssignment , employee == $shiftAssignment.employee , shift.timeslot.startTime.toLocalDate() ==
      $shiftAssignment.shift.timeslot.startTime.toLocalDate())

THEN

1.   scoreHolder.addHardConstraintMatch(kcontext, -1);

      (show
      options...)

Messages   Clear | ⟳ | ⤢ | ✕

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.

2. Click **Add Asset → Guided Rule**.

3. Enter **OneEmployeeShiftPerDay** as the **Guided Rule** name and select the **employeerostering.employeerostering** Package.

4. Click **Ok** to create the rule asset.

5. Add a **WHEN** condition by clicking the ✚ in the **WHEN** field.

6. Select **Free form DRL** from the **Add a condition to the rule** window.

7. In the free form DRL box, type the following condition:

   > $shiftAssignment : ShiftAssignment(employee != null)
   >   ShiftAssignment(this != $shiftAssignment , employee == $shiftAssignment.employee ,
   > shift.timeslot.startTime.toLocalDate() ==
   > $shiftAssignment.shift.timeslot.startTime.toLocalDate())

   This condition states that a shift cannot be assigned to an employee that already has another shift assignment on the same day.

8. Add the **THEN** condition by clicking the ✚ in the **THEN** field.

9. Select **Add free form DRL** from the **Add a new action** window.

10. In the free form DRL box, type the following condition:

    > scoreHolder.addHardConstraintMatch(kcontext, -1);

11. Click **Validate** in the upper-right corner to check all rule conditions are valid. If the rule validation fails, address any problems described in the error message, review all components in the rule, and try again to validate the rule until the rule passes.

12. Click **Save** to save the rule.

For more information about creating guided rules, see *Designing a decision service using guided rules* .

6.2.5.4. Creating a guided rule to match skills to shift requirements

The **ShiftReqiredSkillsAreMet** guided rule creates a hard constraint that ensures all shifts are assigned an employee with the correct set of skills. In the employee rostering example, this constraint is created using the guided rule designer.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.

2. Click **Add Asset → Guided Rule**.

3. Enter **ShiftReqiredSkillsAreMet** as the **Guided Rule** name and select the **employeerostering.employeerostering** Package.

4. Click **Ok** to create the rule asset.

5. Add a **WHEN** condition by clicking the ✚ in the **WHEN** field.

6. Select **ShiftAssignment** in the **Add a condition to the rule** window. Click **+Ok**.

7. Click the **ShiftAssignment** condition, and select **employee** from the **Add a restriction on a field** drop-down list.

8. In the designer, click the drop-down list next to **employee** and select **is not null**.

9. Click the **ShiftAssignment** condition, and click **Expression editor**.

   a. In the designer, click **[not bound]** to open the **Expression editor**, and bind the expression to the variable **$requiredSkill**. Click **Set**.

   b. In the designer, next to **$requiredSkill**, select **shift** from the first drop-down list, then **requiredSkill** from the next drop-down list.

10. Click the **ShiftAssignment** condition, and click **Expression editor**.

   a. In the designer, next to **[not bound]**, select **employee** from the first drop-down list, then **skills** from the next drop-down list.

   b. Leave the next drop-down list as **Choose**.

   c. In the next drop-down box, change **please choose** to **excludes**.

   d. Click the ✎ icon next to **excludes**, and in the **Field value** window, click the **New formula** button.

    e. Type **$requiredSkill** into the formula box.

11. Add the **THEN** condition by clicking the ➕ in the **THEN** field.

12. Select **Modify Hard Score** in the **Add a new action** window. Click **+Ok**.

13. Type **-1** into the score actions box.

14. Click **Validate** in the upper-right corner to check all rule conditions are valid. If the rule validation fails, address any problems described in the error message, review all components in the rule, and try again to validate the rule until the rule passes.

15. Click **Save** to save the rule.

For more information about creating guided rules, see *Designing a decision service using guided rules* .

6.2.5.5. Creating a guided rule to manage day off requests

The **DayOffRequest** guided rule creates a soft constraint. This constraint allows a shift to be reassigned to another employee in the event the employee who was originally assigned the shift is no longer able to work that day. In the employee rostering example, this constraint is created using the guided rule designer.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.

2. Click **Add Asset → Guided Rule**.

3. Enter **DayOffRequest** as the **Guided Rule** name and select the **employeerostering.employeerostering** Package.

4. Click **Ok** to create the rule asset.

5. Add a **WHEN** condition by clicking the ➕ in the **WHEN** field.

6. Select **Free form DRL** from the **Add a condition to the rule** window.

7. In the free form DRL box, type the following condition:

```
$dayOffRequest : DayOffRequest( )
  ShiftAssignment( employee == $dayOffRequest.employee ,
shift.timeslot.startTime.toLocalDate() == $dayOffRequest.date )
```

This condition states if a shift is assigned to an employee who has made a day off request, the employee can be unassigned the shift on that day.

8. Add the **THEN** condition by clicking the ✚ in the **THEN** field.

9. Select **Add free form DRL** from the **Add a new action** window.

10. In the free form DRL box, type the following condition:

    > scoreHolder.addSoftConstraintMatch(kcontext, -100);

11. Click **Validate** in the upper-right corner to check all rule conditions are valid. If the rule validation fails, address any problems described in the error message, review all components in the rule, and try again to validate the rule until the rule passes.

12. Click **Save** to save the rule.

For more information about creating guided rules, see *Designing a decision service using guided rules* .

6.2.6. Creating a solver configuration for employee rostering

You can create and edit Solver configurations in Business Central. The Solver configuration designer creates a solver configuration that can be run after the project is deployed.

Prerequisites

- Red Hat Decision Manager has been downloaded and installed.

- You have created and configured all of the relevant assets for the employee rostering example.

Procedure

1. In Business Central, click **Menu → Projects**, and click your project to open it.

2. In the **Assets** perspective, click **Add Asset → Solver configuration**

3. In the **Create new Solver configuration** window, type the name **EmployeeRosteringSolverConfig** for your Solver and click **Ok**.
   This opens the **Solver configuration** designer.

4. In the **Score Director Factory** configuration section, define a KIE base that contains scoring rule definitions. The employee rostering sample project uses **defaultKieBase**.

   a. Select one of the KIE sessions defined within the KIE base. The employee rostering sample project uses **defaultKieSession**.

5. Click **Validate** in the upper-right corner to check the **Score Director Factory** configuration is correct. If validation fails, address any problems described in the error message, and try again to validate until the configuration passes.

6. Click **Save** to save the Solver configuration.

6.2.7. Configuring Solver termination for the employee rostering project

You can configure the Solver to terminate after a specified amount of time. By default, the planning engine is given an unlimited time period to solve a problem instance.

The employee rostering sample project is set up to run for 30 seconds.

Prerequisites

- You have created all relevant assets for the employee rostering project and created the **EmployeeRosteringSolverConfig** solver configuration in Business Central as described in Section 6.2.6, "Creating a solver configuration for employee rostering".

Procedure

1. Open the **EmployeeRosteringSolverConfig** from the **Assets** perspective. This will open the **Solver configuration** designer.

2. In the **Termination** section, click **Add** to create new termination element within the selected logical group.

3. Select the **Time spent** termination type from the drop-down list. This is added as an input field in the termination configuration.

4. Use the arrows next to the time elements to adjust the amount of time spent to 30 seconds.

5. Click **Validate** in the upper-right corner to check the **Score Director Factory** configuration is correct. If validation fails, address any problems described in the error message, and try again to validate until the configuration passes.

6. Click **Save** to save the Solver configuration.

6.3. ACCESSING THE SOLVER USING THE REST API

After deploying or re-creating the sample solver, you can access it using the REST API.

You must register a solver instance using the REST API. Then you can supply data sets and retrieve optimized solutions.

Prerequisites

- The employee rostering project is set up and deployed according to the previous sections in this document. You can either deploy the sample project, as described in Section 6.1, "Deploying the employee rostering sample project in Business Central", or re-create the project, as described in Section 6.2, "Re-creating the employee rostering sample project".

6.3.1. Registering the Solver using the REST API

You must register the solver instance using the REST API before you can use the solver.

Each solver instance is capable of optimizing one planning problem at a time.

Procedure

1. Create a HTTP request using the following header:

authorization: admin:admin
X-KIE-ContentType: xstream
content-type: application/xml

2. Register the Solver using the following request:

PUT

http://localhost:8080/kie-server/services/rest/server/containers/employeerostering_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver

Request body

```
<solver-instance>
  <solver-config-file>employeerostering/employeerostering/EmployeeRosteringSolverConfig.solver.xml</solver-config-file>
</solver-instance>
```

6.3.2. Calling the Solver using the REST API

After registering the solver instance, you can use the REST API to submit a data set to the solver and to retrieve an optimized solution.

Procedure

1. Create a HTTP request using the following header:

   authorization: admin:admin
   X-KIE-ContentType: xstream
   content-type: application/xml

2. Submit a request to the Solver with a data set, as in the following example:

   POST

   http://localhost:8080/kie-server/services/rest/server/containers/employeerostering_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver/state/solving

   Request body

   ```
   <employeerostering.employeerostering.EmployeeRoster>
     <employeeList>
       <employeerostering.employeerostering.Employee>
         <name>John</name>
         <skills>
           <employeerostering.employeerostering.Skill>
             <name>reading</name>
           </employeerostering.employeerostering.Skill>
         </skills>
       </employeerostering.employeerostering.Employee>
       <employeerostering.employeerostering.Employee>
         <name>Mary</name>
         <skills>
   ```

```xml
        <employeerostering.employeerostering.Skill>
          <name>writing</name>
        </employeerostering.employeerostering.Skill>
      </skills>
    </employeerostering.employeerostering.Employee>
    <employeerostering.employeerostering.Employee>
      <name>Petr</name>
      <skills>
        <employeerostering.employeerostering.Skill>
          <name>speaking</name>
        </employeerostering.employeerostering.Skill>
      </skills>
    </employeerostering.employeerostering.Employee>
  </employeeList>
  <shiftList>
    <employeerostering.employeerostering.Shift>
      <timeslot>
        <startTime>2017-01-01T00:00:00</startTime>
        <endTime>2017-01-01T01:00:00</endTime>
      </timeslot>
      <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee/skills/emplo
yeerostering.employeerostering.Skill"/>
    </employeerostering.employeerostering.Shift>
    <employeerostering.employeerostering.Shift>
      <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
      <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emp
loyeerostering.employeerostering.Skill"/>
    </employeerostering.employeerostering.Shift>
    <employeerostering.employeerostering.Shift>
      <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
      <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emp
loyeerostering.employeerostering.Skill"/>
    </employeerostering.employeerostering.Shift>
  </shiftList>
  <skillList>
    <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee/skills/employe
erostering.employeerostering.Skill"/>
    <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emplo
yeerostering.employeerostering.Skill"/>
    <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emplo
yeerostering.employeerostering.Skill"/>
  </skillList>
  <timeslotList>
    <employeerostering.employeerostering.Timeslot
reference="../../shiftList/employeerostering.employeerostering.Shift/timeslot"/>
  </timeslotList>
  <dayOffRequestList/>
  <shiftAssignmentList>
    <employeerostering.employeerostering.ShiftAssignment>
      <shift reference="../../../shiftList/employeerostering.employeerostering.Shift"/>
```

```
    </employeerostering.employeerostering.ShiftAssignment>
    <employeerostering.employeerostering.ShiftAssignment>
      <shift reference="../../../shiftList/employeerostering.employeerostering.Shift[3]"/>
    </employeerostering.employeerostering.ShiftAssignment>
    <employeerostering.employeerostering.ShiftAssignment>
      <shift reference="../../../shiftList/employeerostering.employeerostering.Shift[2]"/>
    </employeerostering.employeerostering.ShiftAssignment>
  </shiftAssignmentList>
</employeerostering.employeerostering.EmployeeRoster>
```

3. Request the best solution to the planning problem:

   GET

   http://localhost:8080/kie-server/services/rest/server/containers/employeerostering_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver/bestsolution

   Example response

```
<solver-instance>
  <container-id>employee-rostering</container-id>
  <solver-id>solver1</solver-id>
  <solver-config-file>employeerostering/employeerostering/EmployeeRosteringSolverConfig.solver.xml</solver-config-file>
  <status>NOT_SOLVING</status>
  <score scoreClass="org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore">0hard/0soft</score>
  <best-solution class="employeerostering.employeerostering.EmployeeRoster">
    <employeeList>
      <employeerostering.employeerostering.Employee>
        <name>John</name>
        <skills>
          <employeerostering.employeerostering.Skill>
            <name>reading</name>
          </employeerostering.employeerostering.Skill>
        </skills>
      </employeerostering.employeerostering.Employee>
      <employeerostering.employeerostering.Employee>
        <name>Mary</name>
        <skills>
          <employeerostering.employeerostering.Skill>
            <name>writing</name>
          </employeerostering.employeerostering.Skill>
        </skills>
      </employeerostering.employeerostering.Employee>
      <employeerostering.employeerostering.Employee>
        <name>Petr</name>
        <skills>
          <employeerostering.employeerostering.Skill>
            <name>speaking</name>
          </employeerostering.employeerostering.Skill>
        </skills>
      </employeerostering.employeerostering.Employee>
```

```xml
    </employeeList>
    <shiftList>
      <employeerostering.employeerostering.Shift>
        <timeslot>
          <startTime>2017-01-01T00:00:00</startTime>
          <endTime>2017-01-01T01:00:00</endTime>
        </timeslot>
        <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee/skills/emplo
yeerostering.employeerostering.Skill"/>
      </employeerostering.employeerostering.Shift>
      <employeerostering.employeerostering.Shift>
        <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
        <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emp
loyeerostering.employeerostering.Skill"/>
      </employeerostering.employeerostering.Shift>
      <employeerostering.employeerostering.Shift>
        <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
        <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emp
loyeerostering.employeerostering.Skill"/>
      </employeerostering.employeerostering.Shift>
    </shiftList>
    <skillList>
      <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee/skills/employe
erostering.employeerostering.Skill"/>
      <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emplo
yeerostering.employeerostering.Skill"/>
      <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emplo
yeerostering.employeerostering.Skill"/>
    </skillList>
    <timeslotList>
      <employeerostering.employeerostering.Timeslot
reference="../../shiftList/employeerostering.employeerostering.Shift/timeslot"/>
    </timeslotList>
    <dayOffRequestList/>
    <shiftAssignmentList/>
    <score>0hard/0soft</score>
  </best-solution>
</solver-instance>
```

CHAPTER 7. GETTING STARTED WITH OPTAPLANNER AND QUARKUS

You can use the **https://code.quarkus.redhat.com** website to generate a Red Hat build of OptaPlanner Quarkus Maven project and automatically add and configure the extensions that you want to use in your application. You can then download the Quarkus Maven repository or use the online Maven repository with your project.

7.1. APACHE MAVEN AND RED HAT BUILD OF QUARKUS

Apache Maven is a distributed build automation tool used in Java application development to create, manage, and build software projects. Maven uses standard configuration files called Project Object Model (POM) files to define projects and manage the build process. POM files describe the module and component dependencies, build order, and targets for the resulting project packaging and output using an XML file. This ensures that the project is built in a correct and uniform manner.

Maven repositories

A Maven repository stores Java libraries, plug-ins, and other build artifacts. The default public repository is the Maven 2 Central Repository, but repositories can be private and internal within a company to share common artifacts among development teams. Repositories are also available from third parties.

You can use the online Maven repository with your Quarkus projects or you can download the Red Hat build of Quarkus Maven repository.

Maven plug-ins

Maven plug-ins are defined parts of a POM file that achieve one or more goals. Quarkus applications use the following Maven plug-ins:

- Quarkus Maven plug-in (**quarkus-maven-plugin**): Enables Maven to create Quarkus projects, supports the generation of uber-JAR files, and provides a development mode.

- Maven Surefire plug-in (**maven-surefire-plugin**): Used during the test phase of the build lifecycle to execute unit tests on your application. The plug-in generates text and XML files that contain the test reports.

7.2. CONFIGURING THE MAVEN SETTINGS.XML FILE FOR THE ONLINE REPOSITORY

You can use the online Maven repository with your Maven project by configuring your user **settings.xml** file. This is the recommended approach. Maven settings used with a repository manager or repository on a shared server provide better control and manageability of projects.

> **NOTE**
>
> When you configure the repository by modifying the Maven **settings.xml** file, the changes apply to all of your Maven projects.

Procedure

1. Open the Maven **~/.m2/settings.xml** file in a text editor or integrated development environment (IDE).
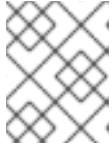
> **NOTE**
>
> If there is not a **settings.xml** file in the **~/.m2/** directory, copy the **settings.xml** file from the **$MAVEN_HOME/.m2/conf/** directory into the **~/.m2/** directory.

2. Add the following lines to the **<profiles>** element of the **settings.xml** file:

```xml
<!-- Configure the Maven repository -->
<profile>
  <id>red-hat-enterprise-maven-repository</id>
  <repositories>
    <repository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

3. Add the following lines to the **<activeProfiles>** element of the **settings.xml** file and save the file.

```xml
<activeProfile>red-hat-enterprise-maven-repository</activeProfile>
```

7.3. DOWNLOADING AND CONFIGURING THE QUARKUS MAVEN REPOSITORY

If you do not want to use the online Maven repository, you can download and configure the Quarkus Maven repository to create a Quarkus application with Maven. The Quarkus Maven repository contains many of the requirements that Java developers typically use to build their applications. This procedure describes how to edit the **settings.xml** file to configure the Quarkus Maven repository.

> **NOTE**
>
> When you configure the repository by modifying the Maven **settings.xml** file, the changes apply to all of your Maven projects.

Procedure

1. Download the Red Hat build of Quarkus Maven repository ZIP file from the Software Downloads page of the Red Hat Customer Portal (login required).

2. Expand the downloaded archive.

3. Change directory to the **~/.m2/** directory and open the Maven **settings.xml** file in a text editor or integrated development environment (IDE).

4. Add the following lines to the **<profiles>** element of the **settings.xml** file, where **QUARKUS_MAVEN_REPOSITORY** is the path of the Quarkus Maven repository that you downloaded. The format of **QUARKUS_MAVEN_REPOSITORY** must be **file://$PATH**, for example **file:///home/userX/rh-quarkus-1.11.6.GA-maven-repository/maven-repository**.

```xml
<!-- Configure the Quarkus Maven repository -->
<profile>
  <id>red-hat-enterprise-maven-repository</id>
  <repositories>
    <repository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>QUARKUS_MAVEN_REPOSITORY</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>QUARKUS_MAVEN_REPOSITORY</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

5. Add the following lines to the **<activeProfiles>** element of the **settings.xml** file and save the file.

```xml
<activeProfile>red-hat-enterprise-maven-repository</activeProfile>
```

IMPORTANT

If your Maven repository contains outdated artifacts, you might encounter one of the following Maven error messages when you build or deploy your project, where *ARTIFACT_NAME* is the name of a missing artifact and *PROJECT_NAME* is the name of the project you are trying to build:

- **Missing artifact *PROJECT_NAME***

- **[ERROR] Failed to execute goal on project *ARTIFACT_NAME*; Could not resolve dependencies for *PROJECT_NAME***

To resolve the issue, delete the cached version of your local repository located in the **~/.m2/repository** directory to force a download of the latest Maven artifacts.

7.4. CREATING A RED HAT BUILD OF QUARKUS MAVEN PROJECT USING CODE.QUARKUS.REDHAT.COM

You can use the **code.quarkus.redhat.com** website to generate a Red Hat build of OptaPlanner Quarkus Maven project and automatically add and configure the extensions that you want to use in your application. In addition, **code.quarkus.redhat.com** automatically manages the configuration parameters required to compile your project into a native executable.

This section walks you through the process of generating an OptaPlanner Maven project and includes the following topics:

- Specifying basic details about your application.

- Choosing the extensions that you want to include in your project.

- Generating a downloadable archive with your project files.

- Using the custom commands for compiling and starting your application.

Prerequisites

- You have a web browser.

Procedure

1. Open **https://code.quarkus.redhat.com** in your web browser:

2. Specify details about your project:

3. Enter a group name for your project. The format of the name follows the Java package naming convention, for example, **com.example**.

4. Enter a name that you want to use for Maven artifacts generated from your project, for example **code-with-quarkus**.

5. Select **Build Tool** > **Maven** to specify that you want to create a Maven project. The build tool that you choose determines the items:

   - The directory structure of your generated project

   - The format of configuration files used in your generated project

- The custom build script and command for compiling and starting your application that **code.quarkus.redhat.com** displays for you after you generate your project

> **NOTE**
>
> Red Hat provides support for using **code.quarkus.redhat.com** to create OptaPlanner Maven projects only. Generating Gradle projects is not supported by Red Hat.

6. Enter a version to be used in artifacts generated from your project. The default value of this field is **1.0.0-SNAPSHOT**. Using semantic versioning is recommended, but you can use a different type of versioning if you prefer.

7. Enter the package name of artifacts that the build tool generates when you package your project.
   According to the Java package naming conventions the package name should match the group name that you use for your project, but you can specify a different name.

> **NOTE**
>
> The **code.quarkus.redhat.com** website automatically uses the latest release of OptaPlanner. You can manually change the BOM version in the **pom.xml** file after you generate your project.

8. Select the following extensions to include as dependencies:

   - RESTEasy JAX-RS (quarkus-resteasy)

   - RESTEasy Jackson (quarkus-resteasy-jackson)

   - OptaPlanner AI constraint solver(optaplanner-quarkus)

   - OptaPlanner Jackson (optaplanner-quarkus-jackson)
     Red Hat provides different levels of support for individual extensions on the list, which are indicated by labels next to the name of each extension:

     - *SUPPORTED* extensions are fully supported by Red Hat for use in enterprise applications in production environments.

     - *TECH-PREVIEW* extensions are subject to limited support by Red Hat in production environments under the Technology Preview Features Support Scope .

     - *DEV-SUPPORT* extensions are not supported by Red Hat for use in production environments, but the core functionalities that they provide are supported by Red Hat developers for use in developing new applications.

     - *DEPRECATED* extension are planned to be replaced with a newer technology or implementation that provides the same functionality.
       Unlabeled extensions are not supported by Red Hat for use in production environments.

9. Select *Generate your application* to confirm your choices and display the overlay screen with the download link for the archive that contains your generated project. The overlay screen also shows the custom command that you can use to compile and start your application.

10. Select **Download the ZIP** to save the archive with the generated project files to your system.

11. Extract the contents of the archive.

12. Navigate to the directory that contains your extracted project files:

    ```
    cd <directory_name>
    ```

13. Compile and start your application in development mode:

    ```
    ./mvnw compile quarkus:dev
    ```

PART III. RED HAT BUILD OF OPTAPLANNER QUICK START GUIDES

Red Hat build of OptaPlanner provides the following quick start guides to demonstrate how OptaPlanner can integrate with different techologies:

- Red Hat build of OptaPlanner on Red Hat build of Quarkus: a school timetable quick start guide

- Red Hat build of OptaPlanner on Red Hat build of Quarkus: a vaccination appointment scheduler quick start guide

- Red Hat build of OptaPlanner on Spring Boot: a school timetable quick start guide

- Red Hat build of OptaPlanner with Java solvers: a cloud balancing quick start guide

CHAPTER 8. RED HAT BUILD OF OPTAPLANNER ON RED HAT BUILD OF QUARKUS: A SCHOOL TIMETABLE QUICK START GUIDE

This guide walks you through the process of creating a Red Hat build of Quarkus application with Red Hat build of OptaPlanner's constraint solving artificial intelligence (AI). You will build a REST application that optimizes a school timetable for students and teachers

Your service will assign **Lesson** instances to **Timeslot** and **Room** instances automatically by using AI to adhere to the following hard and soft *scheduling constraints*:

- A room can have at most one lesson at the same time.

- A teacher can teach at most one lesson at the same time.

- A student can attend at most one lesson at the same time.

- A teacher prefers to teach in a single room.

- A teacher prefers to teach sequential lessons and dislikes gaps between lessons.

Mathematically speaking, school timetabling is an *NP-hard* problem. That means it is difficult to scale. Simply iterating through all possible combinations with brute force would take millions of years for a non-trivial dataset, even on a supercomputer. Fortunately, AI constraint solvers such as Red Hat build of OptaPlanner have advanced algorithms that deliver a near-optimal solution in a reasonable amount of time. What is considered to be a reasonable amount of time is subjective and depends on the goals of your problem.

Prerequisites

.

- OpenJDK 11 or later is installed. Red Hat build of Open JDK is available from the Software Downloads page in the Red Hat Customer Portal (login required).

- Apache Maven 3.6 or higher is installed. Maven is available from the Apache Maven Project website.

- An IDE, such as IntelliJ IDEA, VSCode, Eclipse, or NetBeans is available.

8.1. CREATING THE SCHOOL TIMETABLE PROJECT

The school timetable project lets you get up and running with a Red Hat build of OptaPlanner and Quarkus application using Apache Maven and the Quarkus Maven plug-in.
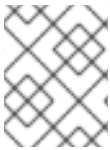
> **NOTE**
>
> If you prefer, you can create a Quakus OptaPlanner project as described in Chapter 7, Getting Started with OptaPlanner and Quarkus .

Procedure

1. In a command terminal, enter the following command to verify that Maven is using JDK 11 and that the Maven version is 3.6 or higher:

   ```
   mvn --version
   ```

2. If the preceding command does not return JDK 11, add the path to JDK 11 to the PATH environment variable and enter the preceding command again.

3. To generate the project, enter one of the following commands:
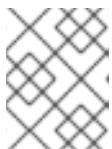
   > **NOTE**
   >
   > Apple macOS and Microsoft Windows are not supported production environments.

   - If you are using Linux or Apple macOS, enter the following command:

     ```
     mvn io.quarkus:quarkus-maven-plugin:1.11.6.Final-redhat-00001:create \
         -DprojectGroupId=com.example \
         -DprojectArtifactId=optaplanner-quickstart  \
         -Dextensions="resteasy,resteasy-jackson,optaplanner-quarkus,optaplanner-quarkus-jackson" \
         -DplatformGroupId=com.redhat.quarkus \
         -DplatformVersion=1.11.6.Final-redhat-00001 \
         -DnoExamples
     ```

     This command create the following elements in the **./optaplanner-quickstart** directory:

     - The Maven structure

     - Example **Dockerfile** file in **src/main/docker**

     - The application configuration file

Table 8.1. Properties used in the mvn **io.quarkus:quarkus-maven-plugin:1.11.6.Final-redhat-00001:create** command

Property	Description
projectGroupId	The group ID of the project.
projectArtifactId	The artifact ID of the project.
extensions	A comma-separated list of Quarkus extensions to use with this project. For a full list of Quarkus extensions, enter **mvn quarkus:list-extensions** on the command line.
platformGroupId	The Group ID of the target platform.
platformVersion	The version of the platform that you want the project to use.
noExamples	Creates a project with the project structure but without tests or classes.

The values of the **projectGroupID** and the **projectArtifactID** properties are used to generate the project version. The default project version is **1.0.0-SNAPSHOT**. The values of the **platformGroupId** and **platformVersion** properties are used by **quarkus-universe-bom** to manage project dependencies.

4. After the directory structure is created, open the **optaplanner-quickstart/pom.xml** file in a text editor and examine the contents of the file to ensure it contains the following elements:

```xml
<properties>
...
<quarkus-plugin.version>1.11.6.Final-redhat-00001</quarkus-plugin.version>
<quarkus.platform.artifact-id>quarkus-universe-bom</quarkus.platform.artifact-id>
<quarkus.platform.group-id>com.redhat.quarkus</quarkus.platform.group-id>
<quarkus.platform.version>1.11.6.Final-redhat-00001.Final-redhat-00001</quarkus.platform.version>
</properties>
<dependencyManagement>
<dependencies>
  <dependency>
    <groupId>${quarkus.platform.group-id}</groupId>
    <artifactId>${quarkus.platform.artifact-id}</artifactId>
    <version>${quarkus.platform.version}</version>
    <type>pom</type>
    <scope>import</scope>
  </dependency>
</dependencies>
</dependencyManagement>
<dependencies>
<dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-quarkus</artifactId>
</dependency>
<dependency>
```

```
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-quarkus-jackson</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy-jackson</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-arc</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-junit5</artifactId>
    <scope>test</scope>
  </dependency>
```

The Quarkus BOM is imported into the **pom.xml** file. Therefore, you do not need to list the versions of individual Quarkus dependencies in the **pom.xml** file.

5. Review the **quarkus-resteasy** dependency in the **pom.xml** file. This dependency enables you to develop REST applications:

```
    <dependency>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-resteasy</artifactId>
    </dependency>
```

8.2. MODEL THE DOMAIN OBJECTS

The goal of the Red Hat build of OptaPlanner timetable project is to assign each lesson to a time slot and a room. To do this, add three classes, **Timeslot**, **Lesson**, and **Room**, as shown in the following diagram:

Time table class diagram

Timeslot

The **Timeslot** class represents a time interval when lessons are taught, for example, **Monday 10:30 - 11:30** or **Tuesday 13:30 - 14:30**. In this example, all time slots have the same duration and there are no time slots during lunch or other breaks.

A time slot has no date because a high school schedule just repeats every week. There is no need for continuous planning. A timeslot is called a *problem fact* because no **Timeslot** instances change during solving. Such classes do not require any OptaPlanner-specific annotations.

Room

The **Room** class represents a location where lessons are taught, for example, **Room A** or **Room B**. In this example, all rooms are without capacity limits and they can accommodate all lessons.

Room instances do not change during solving so **Room** is also a *problem fact*.

Lesson

During a lesson, represented by the **Lesson** class, a teacher teaches a subject to a group of students, for example, **Math by A.Turing for 9th grade** or **Chemistry by M.Curie for 10th grade**. If a subject is taught multiple times each week by the same teacher to the same student group, there are multiple **Lesson** instances that are only distinguishable by **id**. For example, the 9th grade has six math lessons a week.

During solving, OptaPlanner changes the **timeslot** and **room** fields of the **Lesson** class to assign each lesson to a time slot and a room. Because OptaPlanner changes these fields, **Lesson** is a *planning entity*:

Time table class diagram

The timeslot and room fields are
normally null before solving
and non-null after solving

Most of the fields in the previous diagram contain input data, except for the orange fields. A lesson's **timeslot** and **room** fields are unassigned (**null**) in the input data and assigned (not **null**) in the output data. OptaPlanner changes these fields during solving. Such fields are called planning variables. In order for OptaPlanner to recognize them, both the **timeslot** and **room** fields require an **@PlanningVariable** annotation. Their containing class, **Lesson**, requires an **@PlanningEntity** annotation.

Procedure

1. Create the **src/main/java/com/example/domain/Timeslot.java** class:

```java
package com.example.domain;

import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;

    private Timeslot() {
    }

    public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
        this.dayOfWeek = dayOfWeek;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    @Override
    public String toString() {
        return dayOfWeek + " " + startTime.toString();
    }
}
```

```
// **********************************
// Getters and setters
// **********************************

public DayOfWeek getDayOfWeek() {
    return dayOfWeek;
}

public LocalTime getStartTime() {
    return startTime;
}

public LocalTime getEndTime() {
    return endTime;
}

}
```

Notice the **toString()** method keeps the output short so it is easier to read OptaPlanner's **DEBUG** or **TRACE** log, as shown later.

2. Create the **src/main/java/com/example/domain/Room.java** class:

```
package com.example.domain;

public class Room {

    private String name;

    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    // **********************************
    // Getters and setters
    // **********************************

    public String getName() {
        return name;
    }

}
```

3. Create the **src/main/java/com/example/domain/Lesson.java** class:

```
package com.example.domain;
```

```java
import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
    }

    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // ******************************
    // Getters and setters
    // ******************************

    public Long getId() {
        return id;
    }

    public String getSubject() {
        return subject;
    }

    public String getTeacher() {
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {
        return timeslot;
    }
```

```
        public void setTimeslot(Timeslot timeslot) {
            this.timeslot = timeslot;
        }

        public Room getRoom() {
            return room;
        }

        public void setRoom(Room room) {
            this.room = room;
        }

    }
```

The **Lesson** class has an **@PlanningEntity** annotation, so OptaPlanner knows that this class changes during solving because it contains one or more planning variables.

The **timeslot** field has an **@PlanningVariable** annotation, so OptaPlanner knows that it can change its value. In order to find potential **Timeslot** instances to assign to this field, OptaPlanner uses the **valueRangeProviderRefs** property to connect to a value range provider that provides a **List<Timeslot>** to pick from. See Section 8.4, "Gather the domain objects in a planning solution" for information about value range providers.

The **room** field also has an **@PlanningVariable** annotation for the same reasons.

8.3. DEFINE THE CONSTRAINTS AND CALCULATE THE SCORE

When solving a problem, a *score* represents the quality of a specific solution. The higher the score the better. Red Hat build of OptaPlanner looks for the best solution, which is the solution with the highest score found in the available time. It might be the *optimal* solution.

Because the timetable example use case has hard and soft constraints, use the **HardSoftScore** class to represent the score:

- Hard constraints must not be broken. For example: *A room can have at most one lesson at the same time.*

- Soft constraints should not be broken. For example: *A teacher prefers to teach in a single room.*

Hard constraints are weighted against other hard constraints. Soft constraints are weighted against other soft constraints. Hard constraints always outweigh soft constraints, regardless of their respective weights.

To calculate the score, you could implement an **EasyScoreCalculator** class:

```
public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

    @Override
    public HardSoftScore calculateScore(TimeTable timeTable) {
        List<Lesson> lessonList = timeTable.getLessonList();
        int hardScore = 0;
        for (Lesson a : lessonList) {
            for (Lesson b : lessonList) {
                if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
                        && a.getId() < b.getId()) {
```

```
                // A room can accommodate at most one lesson at the same time.
                if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                    hardScore--;
                }
                // A teacher can teach at most one lesson at the same time.
                if (a.getTeacher().equals(b.getTeacher())) {
                    hardScore--;
                }
                // A student can attend at most one lesson at the same time.
                if (a.getStudentGroup().equals(b.getStudentGroup())) {
                    hardScore--;
                }
            }
        }
    }
    int softScore = 0;
    // Soft constraints are only implemented in the "complete" implementation
    return HardSoftScore.of(hardScore, softScore);
    }

}
```

Unfortunately, this solution does not scale well because it is non-incremental: every time a lesson is assigned to a different time slot or room, all lessons are re-evaluated to calculate the new score.

A better solution is to create a **src/main/java/com/example/solver/TimeTableConstraintProvider.java** class to perform incremental score calculation. This class uses OptaPlanner's ConstraintStream API which is inspired by Java 8 Streams and SQL. The **ConstraintProvider** scales an order of magnitude better than the **EasyScoreCalculator**: $O(n)$ instead of $O(n^2)$.

Procedure

Create the following **src/main/java/com/example/solver/TimeTableConstraintProvider.java** class:

```java
package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
                // Hard constraints
                roomConflict(constraintFactory),
                teacherConflict(constraintFactory),
                studentGroupConflict(constraintFactory),
                // Soft constraints are only implemented in the "complete" implementation
        };
    }
```

```java
    private Constraint roomConflict(ConstraintFactory constraintFactory) {
        // A room can accommodate at most one lesson at the same time.

        // Select a lesson ...
        return constraintFactory.from(Lesson.class)
                // ... and pair it with another lesson ...
                .join(Lesson.class,
                        // ... in the same timeslot ...
                        Joiners.equal(Lesson::getTimeslot),
                        // ... in the same room ...
                        Joiners.equal(Lesson::getRoom),
                        // ... and the pair is unique (different id, no reverse pairs)
                        Joiners.lessThan(Lesson::getId))
                // then penalize each pair with a hard weight.
                .penalize("Room conflict", HardSoftScore.ONE_HARD);
    }

    private Constraint teacherConflict(ConstraintFactory constraintFactory) {
        // A teacher can teach at most one lesson at the same time.
        return constraintFactory.from(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getTeacher),
                        Joiners.lessThan(Lesson::getId))
                .penalize("Teacher conflict", HardSoftScore.ONE_HARD);
    }

    private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
        // A student can attend at most one lesson at the same time.
        return constraintFactory.from(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getStudentGroup),
                        Joiners.lessThan(Lesson::getId))
                .penalize("Student group conflict", HardSoftScore.ONE_HARD);
    }

}
```

8.4. GATHER THE DOMAIN OBJECTS IN A PLANNING SOLUTION

A **TimeTable** instance wraps all **Timeslot**, **Room**, and **Lesson** instances of a single dataset. Furthermore, because it contains all lessons, each with a specific planning variable state, it is a *planning solution* and it has a score:

- If lessons are still unassigned, then it is an *uninitialized* solution, for example, a solution with the score **-4init/0hard/0soft**.

- If it breaks hard constraints, then it is an *infeasible* solution, for example, a solution with the score **-2hard/-3soft**.

- If it adheres to all hard constraints, then it is a *feasible* solution, for example, a solution with the score **0hard/-7soft**.

The **TimeTable** class has an **@PlanningSolution** annotation, so Red Hat build of OptaPlanner knows that this class contains all of the input and output data.

Specifically, this class is the input of the problem:

- A **timeslotList** field with all time slots

  - This is a list of problem facts, because they do not change during solving.

- A **roomList** field with all rooms

  - This is a list of problem facts, because they do not change during solving.

- A **lessonList** field with all lessons

  - This is a list of planning entities because they change during solving.

  - Of each **Lesson**:

    - The values of the **timeslot** and **room** fields are typically still **null**, so unassigned. They are planning variables.

    - The other fields, such as **subject**, **teacher** and **studentGroup**, are filled in. These fields are problem properties.

However, this class is also the output of the solution:

- A **lessonList** field for which each **Lesson** instance has non-null **timeslot** and **room** fields after solving

- A **score** field that represents the quality of the output solution, for example, **0hard/-5soft**

Procedure

Create the **src/main/java/com/example/domain/TimeTable.java** class:

```java
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;

@PlanningSolution
public class TimeTable {

    @ValueRangeProvider(id = "timeslotRange")
    @ProblemFactCollectionProperty
    private List<Timeslot> timeslotList;

    @ValueRangeProvider(id = "roomRange")
    @ProblemFactCollectionProperty
    private List<Room> roomList;

    @PlanningEntityCollectionProperty
    private List<Lesson> lessonList;
```

```
        @PlanningScore
        private HardSoftScore score;

        private TimeTable() {
        }

        public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
                List<Lesson> lessonList) {
            this.timeslotList = timeslotList;
            this.roomList = roomList;
            this.lessonList = lessonList;
        }

        // ******************************
        // Getters and setters
        // ******************************

        public List<Timeslot> getTimeslotList() {
            return timeslotList;
        }

        public List<Room> getRoomList() {
            return roomList;
        }

        public List<Lesson> getLessonList() {
            return lessonList;
        }

        public HardSoftScore getScore() {
            return score;
        }
    }
```

The value range providers

The **timeslotList** field is a value range provider. It holds the **Timeslot** instances which OptaPlanner can pick from to assign to the **timeslot** field of **Lesson** instances. The **timeslotList** field has an **@ValueRangeProvider** annotation to connect those two, by matching the **id** with the **valueRangeProviderRefs** of the **@PlanningVariable** in the **Lesson**.

Following the same logic, the **roomList** field also has an **@ValueRangeProvider** annotation.

The problem fact and planning entity properties

Furthermore, OptaPlanner needs to know which **Lesson** instances it can change as well as how to retrieve the **Timeslot** and **Room** instances used for score calculation by your **TimeTableConstraintProvider**.

The **timeslotList** and **roomList** fields have an **@ProblemFactCollectionProperty** annotation, so your **TimeTableConstraintProvider** can select from those instances.

The **lessonList** has an **@PlanningEntityCollectionProperty** annotation, so OptaPlanner can change them during solving and your **TimeTableConstraintProvider** can select from those too.

8.5. CREATE THE SOLVER SERVICE

Solving planning problems on REST threads causes HTTP timeout issues. Therefore, the Quarkus extension injects a SolverManager, which runs solvers in a separate thread pool and can solve multiple data sets in parallel.

Procedure

Create the **src/main/java/org/acme/optaplanner/rest/TimeTableResource.java** class:

```java
package org.acme.optaplanner.rest;

import java.util.UUID;
import java.util.concurrent.ExecutionException;
import javax.inject.Inject;
import javax.ws.rs.POST;
import javax.ws.rs.Path;

import org.acme.optaplanner.domain.TimeTable;
import org.optaplanner.core.api.solver.SolverJob;
import org.optaplanner.core.api.solver.SolverManager;

@Path("/timeTable")
public class TimeTableResource {

    @Inject
    SolverManager<TimeTable, UUID> solverManager;

    @POST
    @Path("/solve")
    public TimeTable solve(TimeTable problem) {
        UUID problemId = UUID.randomUUID();
        // Submit the problem to start solving
        SolverJob<TimeTable, UUID> solverJob = solverManager.solve(problemId, problem);
        TimeTable solution;
        try {
            // Wait until the solving ends
            solution = solverJob.getFinalBestSolution();
        } catch (InterruptedException | ExecutionException e) {
            throw new IllegalStateException("Solving failed.", e);
        }
        return solution;
    }

}
```

This initial implementation waits for the solver to finish, which can still cause an HTTP timeout. The complete implementation avoids HTTP timeouts much more elegantly.

8.6. SET THE SOLVER TERMINATION TIME

If your planning application does not have a termination setting or a termination event, it theoretically runs forever and in reality eventually causes an HTTP timeout error. To prevent this from occurring, use the **optaplanner.solver.termination.spent-limit** parameter to specify the length of time after which

the application terminates. In most applications, set the time to at least five minutes (**5m**). However, in the Timetable example, limit the solving time to five second, which is short enough to avoid the HTTP timeout.

Procedure

Create the **src/main/resources/application.properties** file with the following content:

```
quarkus.optaplanner.solver.termination.spent-limit=5s
```

8.7. RUNNING THE SCHOOL TIMETABLE APPLICATION

After you have created the school timetable project, run it in development mode. In development mode, you can update the application sources and configurations while your application is running. Your changes will appear in the running application.

Prerequisites

* You have created the school timetable project.

Procedure

1. To compile the application in development mode, enter the following command from the project directory:

   ```
   ./mvnw compile quarkus:dev
   ```

2. Test the REST service. You can use any REST client. The following example uses the Linux command **curl** to send a POST request:

   ```
   $ curl -i -X POST http://localhost:8080/timeTable/solve -H "Content-Type:application/json" -d
   '{"timeslotList":[{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},
   {"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"}],"roomList":
   [{"name":"Room A"},{"name":"Room B"}],"lessonList":[{"id":1,"subject":"Math","teacher":"A.
   Turing","studentGroup":"9th grade"},{"id":2,"subject":"Chemistry","teacher":"M.
   Curie","studentGroup":"9th grade"},{"id":3,"subject":"French","teacher":"M.
   Curie","studentGroup":"10th grade"},{"id":4,"subject":"History","teacher":"I.
   Jones","studentGroup":"10th grade"}]}'
   ```

   After the time period specified in **termination spent time** defined in your **application.properties** file, the service returns output similar to the following example:

   ```
   HTTP/1.1 200
   Content-Type: application/json
   ...

   {"timeslotList":...,"roomList":...,"lessonList":[{"id":1,"subject":"Math","teacher":"A.
   Turing","studentGroup":"9th grade","timeslot":
   {"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":
   {"name":"Room A"}},{"id":2,"subject":"Chemistry","teacher":"M. Curie","studentGroup":"9th
   grade","timeslot":
   {"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":
   {"name":"Room A"}},{"id":3,"subject":"French","teacher":"M. Curie","studentGroup":"10th
   grade","timeslot":
   ```

{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":
{"name":"Room B"}},{"id":4,"subject":"History","teacher":"I. Jones","studentGroup":"10th
grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":
{"name":"Room B"}}],"score":"0hard/0soft"}

Notice that your application assigned all four lessons to one of the two time slots and one of the two rooms. Also notice that it conforms to all hard constraints. For example, M. Curie's two lessons are in different time slots.

3. To review what OptaPlanner did during the solving time, review the info log on the server side. The following is sample info log output:

```
... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score
calculation speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score
calculation speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed
(28524/sec), phase total (2), environment mode (REPRODUCIBLE).
```

8.7.1. Test the application

A good application includes test coverage. This example tests the Red Hat build of OptaPlanner school timetable project on Red Hat build of Quarkus. It uses a JUnit test to generate a test dataset and send it to the **TimeTableController** to solve.

Procedure

1. Create the **src/test/java/com/example/rest/TimeTableResourceTest.java** class with the following content:

```java
package com.exmaple.optaplanner.rest;

import java.time.DayOfWeek;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import com.exmaple.optaplanner.domain.Room;
import com.exmaple.optaplanner.domain.Timeslot;
import com.exmaple.optaplanner.domain.Lesson;
import com.exmaple.optaplanner.domain.TimeTable;
import com.exmaple.optaplanner.rest.TimeTableResource;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;
```

```java
@QuarkusTest
public class TimeTableResourceTest {

    @Inject
    TimeTableResource timeTableResource;

    @Test
    @Timeout(600_000)
    public void solve() {
        TimeTable problem = generateProblem();
        TimeTable solution = timeTableResource.solve(problem);
        assertFalse(solution.getLessonList().isEmpty());
        for (Lesson lesson : solution.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(solution.getScore().isFeasible());
    }

    private TimeTable generateProblem() {
        List<Timeslot> timeslotList = new ArrayList<>();
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30),
LocalTime.of(9, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>();
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
        lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
        lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
        lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
        return new TimeTable(timeslotList, roomList, lessonList);
    }

}
```

This test verifies that after solving, all lessons are assigned to a time slot and a room. It also verifies that it found a feasible solution (no hard constraints broken).

2. Add test properties to the **src/main/resources/application.properties** file:

```
# The solver runs only for 5 seconds to avoid a HTTP timeout in this simple implementation.
# It's recommended to run for at least 5 minutes ("5m") otherwise.
quarkus.optaplanner.solver.termination.spent-limit=5s

# Effectively disable this termination in favor of the best-score-limit
%test.quarkus.optaplanner.solver.termination.spent-limit=1h
%test.quarkus.optaplanner.solver.termination.best-score-limit=0hard/*soft
```

Normally, the solver finds a feasible solution in less than 200 milliseconds. Notice how the **application.properties** file overwrites the solver termination during tests to terminate as soon as a feasible solution **(0hard/*soft)** is found. This avoids hard coding a solver time, because the unit test might run on arbitrary hardware. This approach ensures that the test runs long enough to find a feasible solution, even on slow systems. But it does not run a millisecond longer than it strictly must, even on fast systems.

8.7.2. Logging

After you complete the Red Hat build of OptaPlanner school timetable project, you can use logging information to help you fine-tune the constraints in the **ConstraintProvider**. Review the score calculation speed in the **info** log file to assess the impact of changes to your constraints. Run the application in debug mode to show every step that your application takes or use trace logging to log every step and every move.

Procedure

1. Run the school timetable application for a fixed amount of time, for example, five minutes.

2. Review the score calculation speed in the **log** file as shown in the following example:

   ```
   ... Solving ended: ..., score calculation speed (29455/sec), ...
   ```

3. Change a constraint, run the planning application again for the same amount of time, and review the score calculation speed recorded in the **log** file.

4. Run the application in debug mode to log every step that the application makes:

   - To run debug mode from the command line, use the **-D** system property.

   - To permanently enable debug mode, add the following line to the **application.properties** file:

     ```
     quarkus.log.category."org.optaplanner".level=debug
     ```

     The following example shows output in the **log** file in debug mode:

     ```
     ... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode
     (REPRODUCIBLE), random (JDK with seed 0).
     ...     CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15),
     picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
     ...     CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15),
     picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
     ...
     ```

5. Use **trace** logging to show every step and every move for each step.

CHAPTER 9. RED HAT BUILD OF OPTAPLANNER ON RED HAT BUILD OF QUARKUS: A VACCINATION APPOINTMENT SCHEDULER QUICK START GUIDE

You can use the OptaPlanner vaccination appointment scheduler quick start to develop a vaccination schedule that is both efficient and fair. The vaccination appointment scheduler uses artificial intelligence (AI) to prioritize people and allocate time slots based on multiple constraints and priorities.

Prerequisites

- OpenJDK 11 or later is installed. Red Hat build of Open JDK is available from the Software Downloads page in the Red Hat Customer Portal (login required).

- Apache Maven 3.6 or higher is installed. Maven is available from the Apache Maven Project website.

- An IDE, such as IntelliJ IDEA, VSCode, Eclipse, or NetBeans is available.

- You have created a Quakus OptaPlanner project as described in Chapter 7, *Getting Started with OptaPlanner and Quarkus*.

9.1. HOW THE OPTAPLANNER VACCINATION APPOINTMENT SCHEDULER WORKS

There are two main approaches to scheduling appointments. The system can either let a person choose an appointment slot (user-selects) or the system assigns a slot and tells the person when and where to attend (system-automatically-assigns). The OptaPlanner vaccination appointment scheduler uses the system-automatically-assigns approach. With the OptaPlanner vaccination appointment scheduler, you can create an application where people provide their information to the system and the system assigns an appointment.

Characteristics of this approach:

- Appointment slots are allocated based on priority.

- The system allocates the best appointment time and location based on preconfigured planning constraints.

- The system is not overwhelmed by a large number of users competing for a limited number of appointments.

This approach solves the problem of vaccinating as many people as possible by using planning constraints to create a score for each person. The person's score determines when they get an appointment. The higher the person's score, the better chance they have of receiving an earlier appointment.

9.1.1. OptaPlanner vaccination appointment scheduler constraints

OptaPlanner vaccination appointment scheduler constraints are either hard, medium, or soft:

- Hard constraints cannot be broken. If any hard constraint is broken, the plan is unfeasible and cannot be executed:

  - Capacity: Do not over-book vaccine capacity at any time at any location.

- Vaccine max age: If a vaccine has a maximum age, do not administer it to people who at the time of the first dose vaccination are older than the vaccine maximum age. Ensure people are given a vaccine type appropriate for their age. For example, do not assign a 75 year old person an appointment for a vaccine that has a maximum age restriction of 65 years.

- Required vaccine type: Use the required vaccine type. For example, the second dose of a vaccine must be the same vaccine type as the first dose.

- Ready date: Administer the vaccine on or after the specified date. For example, if a person receives a second dose, do not administer it before the recommended earliest possible vaccination date for the specific vaccine type, for example 26 days after the first dose.

- Due date: Administer the vaccine on or before the specified date. For example, if a person receives a second dose, administer it before the recommended vaccination final due date for the specific vaccine, for example three months after the first dose.

- Restrict maximum travel distance: Assign each person to one of a group of vaccination centers nearest to them. This is typically one of three centers. This restriction is calculated by travel time, not distance, so a person that lives in an urban area usually has a lower maximum distance to travel than a person living in a rural area.

- Medium constraints decide who does not get an appointment when there is not enough capacity to assign appointments to everyone. This is called overconstrained planning:

  - Schedule second dose vaccinations: Do not leave any second dose vaccination appointments unassigned unless the ideal date falls outside of the planning window.

  - Schedule people based on their priority rating: Each person has a priority rating. This is typically their age but it can be much higher if they are, for example, a health care worker. Leave only people with the lowest priority ratings unassigned. They will be considered in the next run. This constraint is softer than the previous constraint because the second dose is always prioritized over priority rating.

- Soft constraints should not be broken:

  - Preferred vaccination center: If a person has a preferred vaccination center, give them an appointment at that center.

  - Distance: Minimize the distance that a person must travel to their assigned vaccination center.

  - Ideal date: Administer the vaccine on or as close to the specified date as possible. For example, if a person receives a second dose, administer it on the ideal date for the specific vaccine, for example 28 days after the first dose. This constraint is softer than the distance constraint to avoid sending people halfway across the country just to be one day closer to their ideal date.

  - Priority rating: Schedule people with a higher priority rating earlier in the planning window. This constraint is softer than the distance constraint to avoid sending people halfway across the country. This constraint is also softer than the ideal date constraint because the second dose is prioritized over priority rating.

Hard constraints are weighted against other hard constraints. Soft constraints are weighted against other soft constraints. However, hard constraints always take precedence over medium and soft constraints. If a hard constraint is broken, then the plan is not feasible. But if no hard constraints are broken then soft and medium constraints are considered in order to determine priority. Because there are often more people than available appointment slots, you must prioritize. Second dose appointments

are always assigned first to avoid creating a backlog that would overwhelm your system later. After that, people are assigned based on their priority rating. Everyone starts with a priority rating that is their age. Doing this prioritizes older people over younger people. After that, people that are in specific priority groups receive, for example, a few hundred extra points. This varies based on the priority of their group. For example, nurses might receive an extra 1000 points. This way, older nurses are prioritized over younger nurses and young nurses are prioritized over people who are not nurses. The following table illustrates this concept:

Table 9.1. Priority rating table

Age	Job	Priority rating
60	nurse	1060
33	nurse	1033
71	retired	71
52	office worker	52

9.1.2. The OptaPlanner solver

At the core of OptaPlanner is the solver, the engine that takes the problem data set and overlays the planning constraints and configurations. The problem data set includes all of the information about the people, the vaccines, and the vaccination centers. The solver works through the various combinations of data and eventually determines an optimized appointment schedule with people assigned to vaccination appointments at a specific center. The following illustration shows a schedule that the solver created:
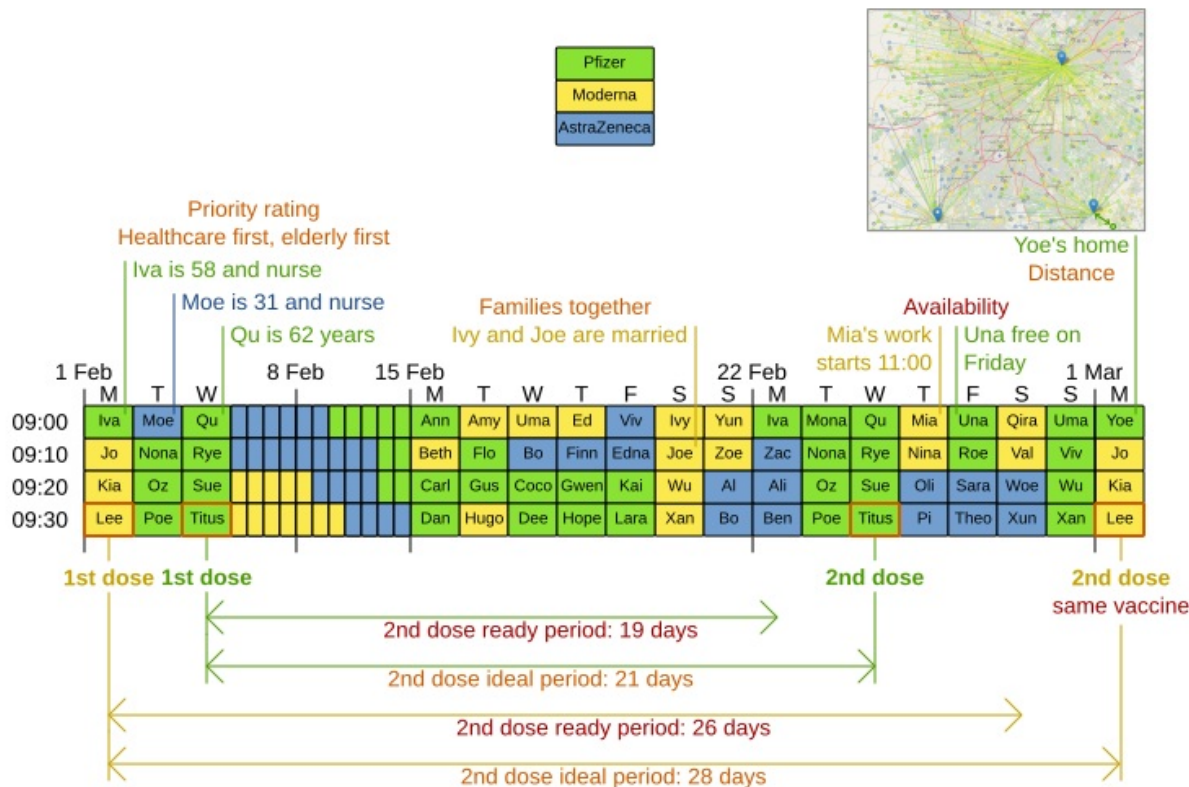
9.1.3. Continuous planning

Continuous planning is the technique of managing one or more upcoming planning periods at the same time and repeating that process monthly, weekly, daily, hourly, or even more frequently. The planning window advances incrementally by a specified interval. The following illustration shows a two week planning window that is updated daily:

The two week planning window is divided in half. The first week is in the published state and the second week is in the draft state. People are assigned to appointments in both the published and draft parts of the planning window. However, only people in the published part of the planning window are notified of their appointments. The other appointments can still change easily in the next run. Doing this gives OptaPlanner the flexibility to change the appointments in the draft part when you run the solver again, if necessary. For example, if a person who needs a second dose has a ready date of Monday and an ideal date of Wednesday, OptaPlanner does not have to give them an appointment for Monday if you can prove OptaPlanner can demonstrate that it can give them a draft appointment later in the week.

You can determine the size of the planning window but just be aware of the size of the problem space. The problem space is all of the various elements that go into creating the schedule. The more days you plan ahead, the larger the problem space.

9.1.4. Pinned planning entities

If you are continuously planning on a daily basis, there will be appointments within the two week period that are already allocated to people. To ensure that appointments are not double-booked, OptaPlanner marks existing appointments as allocated by pinning them. Pinning is used to anchor one or more specific assignments and force OptaPlanner to schedule around those fixed assignments. A pinned planning entity, such as an appointment, does not change during solving.

Whether an entity is pinned or not is determined by the appointment state. An appointment can have five states : **Open**, **Invited**, **Accepted**, **Rejected**, or **Rescheduled**.

> **NOTE**
>
> You do not actually see these states directly in the quick start demo code because the OptaPlanner engine is only interested in whether the appointment is pinned or not.

You need to be able to plan around appointments that have already been scheduled. An appointment with the **Invited** or **Accepted** state is pinned. Appointments with the **Open**, **Reschedule**, and **Rejected** state are not pinned and are available for scheduling.

In this example, when the solver runs it searches across the entire two week planning window in both the published and draft ranges. The solver considers any unpinned entities, appointments with the **Open**, **Reschedule**, or **Rejected** states, in addition to the unscheduled input data, to find the optimal solution. If the solver is run daily, you will see a new day added to the schedule before you run the solver.

Notice that the appointments on the new day have been assigned and Amy and Edna who were previously scheduled in the draft part of the planning window are now scheduled in the published part of the window. This was possible because Gus and Hugo requested a reschedule. This will not cause any confusion because Amy and Edna were never notified about their draft dates. Now, because they have appointments in the published section of the planning window, they will be notified and asked to accept or reject their appointments, and their appointments are now pinned.

9.2. DOWNLOADING AND RUNNING THE OPTAPLANNER VACCINATION APPOINTMENT SCHEDULER

Download the OptaPlanner vaccination appointment scheduler quick start archive, start it in Quarkus development mode, and view the application in a browser. Quarkus development mode enables you to make changes and update your application while it is running.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required), and select the product and version from the drop-down options:

   - Product: Decision Manager

   - Version: 7.11

2. Download **Red Hat Decision Manager 7.11.0 Kogito and OptaPlanner 8 Decision Services Quickstarts** (**rhdm-7.11.0-decision-services-quickstarts.zip**).

3. Extract the **rhdm-7.11.0-decision-services-quickstarts.zip** file.

4. Navigate to the **optaplanner-quickstarts-8.5.0.Final-redhat-00004** directory.

5. Navigate to the **optaplanner-quickstarts-8.5.0.Final-redhat-00004/quarkus-vaccination-scheduling** directory.

6. Enter the following command to start the OptaPlanner vaccination appointment scheduler in development mode:

   ```
   $ mvn quarkus:dev
   ```

7. To view the OptaPlanner vaccination appointment scheduler, enter the following URL in a web browser.

> http://localhost:8080/

8. To run the OptaPlanner vaccination appointment scheduler, click **Solve**.

9. Make changes to the source code then press the F5 key to refresh your browser. Notice that the changes that you made are now available.

9.3. PACKAGE AND RUN THE OPTAPLANNER VACCINATION APPOINTMENT SCHEDULER

When you have completed development work on the OptaPlanner vaccination appointment scheduler in **quarkus:dev** mode, run the application as a conventional jar file.

Prerequisites

- You have downloaded the OptaPlanner vaccination appointment scheduler quick start. For more information, see Section 9.2, "Downloading and running the OptaPlanner vaccination appointment scheduler".

Procedure

1. Navigate to the **quarkus-vaccination-scheduling** directory.

2. To compile the OptaPlanner vaccination appointment scheduler, enter the following command:

   > $ mvn package

3. To run the compiled OptaPlanner vaccination appointment scheduler, enter the following command:

   > $ java -jar ./target/*-runner.jar
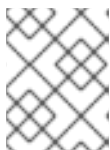
   > **NOTE**
   >
   > To run the application on port 8081, add **-Dquarkus.http.port=8081** to the preceding command.

4. To start the OptaPlanner vaccination appointment scheduler, enter the following URL in a web browser.

   > http://localhost:8080/

9.4. RUN THE OPTAPLANNER VACCINATION APPOINTMENT SCHEDULER AS A NATIVE EXECUTABLE

To take advantage of the small memory footprint and access speeds that Quarkus offers, compile the OptaPlanner vaccination appointment scheduler in Quarkus native mode.

Procedure

1. Install GraalVM and the **native-image** tool. For information, see [Configuring GraalVM](#) on the Quarkus website.

2. Navigate to the **quarkus-vaccination-scheduling** directory.

3. To compile the OptaPlanner vaccination appointment scheduler natively, enter the following command:

   ```
   $ mvn package -Dnative -DskipTests
   ```

4. To run the native executable, enter the following command:

   ```
   $ ./target/*-runner
   ```

5. To start the OptaPlanner vaccination appointment scheduler, enter the following URL in a web browser.

   ```
   http://localhost:8080/
   ```

9.5. ADDITIONAL RESOURCES

- [Vaccination appointment scheduling video](#)

CHAPTER 10. RED HAT BUILD OF OPTAPLANNER ON SPRING BOOT: A SCHOOL TIMETABLE QUICK START GUIDE

This guide walks you through the process of creating a Spring Boot application with OptaPlanner's constraint solving artificial intelligence (AI). You will build a REST application that optimizes a school timetable for students and teachers.

Your service will assign **Lesson** instances to **Timeslot** and **Room** instances automatically by using AI to adhere to the following hard and soft *scheduling constraints*:

- A room can have at most one lesson at the same time.

- A teacher can teach at most one lesson at the same time.

- A student can attend at most one lesson at the same time.

- A teacher prefers to teach in a single room.

- A teacher prefers to teach sequential lessons and dislikes gaps between lessons.

Mathematically speaking, school timetabling is an *NP-hard* problem. That means it is difficult to scale. Simply iterating through all possible combinations with brute force would take millions of years for a non-trivial dataset, even on a supercomputer. Fortunately, AI constraint solvers such as OptaPlanner have advanced algorithms that deliver a near-optimal solution in a reasonable amount of time. What is considered to be a reasonable amount of time is subjective and depends on the goals of your problem.

Prerequisites

- OpenJDK 11 or later is installed. Red Hat build of Open JDK is available from the Software Downloads page in the Red Hat Customer Portal (login required).

- Apache Maven 3.6 or higher is installed. Maven is available from the Apache Maven Project website.

- An IDE, such as IntelliJ IDEA, VSCode, Eclipse, or NetBeans is available.

10.1. DOWNLOADING AND BUILDING THE SPRING BOOT SCHOOL TIMETABLE QUICK START

If you want to see a completed example of the school timetable project for Red Hat build of OptaPlanner with Spring Boot product, download the starter application from the Red Hat Customer Portal.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required), and select the product and version from the drop-down options:

   - Product: Decision Manager

   - Version: 7.11

2. Download **Red Hat Decision Manager 7.11.0 Kogito and OptaPlanner 8 Decision Services Quickstarts** (**rhdm-7.11.0-decision-services-quickstarts.zip**).

3. Extract the **rhdm-7.11.0-decision-services-quickstarts.zip** file.

4. Download the **Red Hat Decision Manager 7.11.0 Kogito and OptaPlanner 8 Decision Services Maven Repositroy** (**rhdm-7.11.0-kogito-maven-repository.zip**).

5. Extract the **rhdm-7.11.0-kogito-maven-repository.zip** file.

6. Copy the contents of the **rhdm-7.11.0-kogito-maven-repository**/**maven-repository** subdirectory into the **~/.m2/repository** directory.

7. Navigate to the **optaplanner-quickstarts-8.5.0.Final-redhat-00004/spring-boot-school-timetabling** directory.

8. Enter the following command to build the Spring Boot school timetabling project:

   ```
   mvn clean install -DskipTests
   ```

9. To build the Spring Boot school timetabling project, enter the following command:

   ```
   mvn spring-boot:run -DskipTests
   ```

10. To view the project, enter the following URL in a web browser:

    ```
    http://localhost:8080/
    ```

10.2. MODEL THE DOMAIN OBJECTS

The goal of the Red Hat build of OptaPlanner timetable project is to assign each lesson to a time slot and a room. To do this, add three classes, **Timeslot**, **Lesson**, and **Room**, as shown in the following diagram:

Time table class diagram

Timeslot

The **Timeslot** class represents a time interval when lessons are taught, for example, **Monday 10:30 - 11:30** or **Tuesday 13:30 - 14:30**. In this example, all time slots have the same duration and there are no time slots during lunch or other breaks.

A time slot has no date because a high school schedule just repeats every week. There is no need for continuous planning. A timeslot is called a *problem fact* because no **Timeslot** instances change during solving. Such classes do not require any OptaPlanner-specific annotations.

Room

The **Room** class represents a location where lessons are taught, for example, **Room A** or **Room B**. In this example, all rooms are without capacity limits and they can accommodate all lessons.

Room instances do not change during solving so **Room** is also a *problem fact*.

Lesson

During a lesson, represented by the **Lesson** class, a teacher teaches a subject to a group of students, for example, **Math by A.Turing for 9th grade** or **Chemistry by M.Curie for 10th grade**. If a subject is taught multiple times each week by the same teacher to the same student group, there are multiple **Lesson** instances that are only distinguishable by **id**. For example, the 9th grade has six math lessons a week.

During solving, OptaPlanner changes the **timeslot** and **room** fields of the **Lesson** class to assign each lesson to a time slot and a room. Because OptaPlanner changes these fields, **Lesson** is a *planning entity*:

Time table class diagram

The timeslot and room fields are normally null before solving and non-null after solving

Most of the fields in the previous diagram contain input data, except for the orange fields. A lesson's **timeslot** and **room** fields are unassigned (**null**) in the input data and assigned (not **null**) in the output data. OptaPlanner changes these fields during solving. Such fields are called planning variables. In order for OptaPlanner to recognize them, both the **timeslot** and **room** fields require an **@PlanningVariable** annotation. Their containing class, **Lesson**, requires an **@PlanningEntity** annotation.

Procedure

1. Create the **src/main/java/com/example/domain/Timeslot.java** class:

```java
package com.example.domain;

import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;

    private Timeslot() {
    }

    public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
        this.dayOfWeek = dayOfWeek;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    @Override
    public String toString() {
        return dayOfWeek + " " + startTime.toString();
    }
```

```
    // ********************************
    // Getters and setters
    // ********************************

    public DayOfWeek getDayOfWeek() {
        return dayOfWeek;
    }

    public LocalTime getStartTime() {
        return startTime;
    }

    public LocalTime getEndTime() {
        return endTime;
    }

}
```

Notice the **toString()** method keeps the output short so it is easier to read OptaPlanner's
DEBUG or **TRACE** log, as shown later.

2. Create the **src/main/java/com/example/domain/Room.java** class:

```
package com.example.domain;

public class Room {

    private String name;

    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public String getName() {
        return name;
    }

}
```

3. Create the **src/main/java/com/example/domain/Lesson.java** class:

```
package com.example.domain;
```

```java
import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
    }

    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // ******************************
    // Getters and setters
    // ******************************

    public Long getId() {
        return id;
    }

    public String getSubject() {
        return subject;
    }

    public String getTeacher() {
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {
        return timeslot;
    }
```

```
        public void setTimeslot(Timeslot timeslot) {
            this.timeslot = timeslot;
        }

        public Room getRoom() {
            return room;
        }

        public void setRoom(Room room) {
            this.room = room;
        }

    }
```

The **Lesson** class has an **@PlanningEntity** annotation, so OptaPlanner knows that this class changes during solving because it contains one or more planning variables.

The **timeslot** field has an **@PlanningVariable** annotation, so OptaPlanner knows that it can change its value. In order to find potential **Timeslot** instances to assign to this field, OptaPlanner uses the **valueRangeProviderRefs** property to connect to a value range provider that provides a **List<Timeslot>** to pick from. See Section 10.4, "Gather the domain objects in a planning solution" for information about value range providers.

The **room** field also has an **@PlanningVariable** annotation for the same reasons.

10.3. DEFINE THE CONSTRAINTS AND CALCULATE THE SCORE

When solving a problem, a *score* represents the quality of a specific solution. The higher the score the better. Red Hat build of OptaPlanner looks for the best solution, which is the solution with the highest score found in the available time. It might be the *optimal* solution.

Because the timetable example use case has hard and soft constraints, use the **HardSoftScore** class to represent the score:

- Hard constraints must not be broken. For example: *A room can have at most one lesson at the same time.*

- Soft constraints should not be broken. For example: *A teacher prefers to teach in a single room.*

Hard constraints are weighted against other hard constraints. Soft constraints are weighted against other soft constraints. Hard constraints always outweigh soft constraints, regardless of their respective weights.

To calculate the score, you could implement an **EasyScoreCalculator** class:

```
public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

    @Override
    public HardSoftScore calculateScore(TimeTable timeTable) {
        List<Lesson> lessonList = timeTable.getLessonList();
        int hardScore = 0;
        for (Lesson a : lessonList) {
            for (Lesson b : lessonList) {
                if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
                        && a.getId() < b.getId()) {
```

```
                // A room can accommodate at most one lesson at the same time.
                if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                    hardScore--;
                }
                // A teacher can teach at most one lesson at the same time.
                if (a.getTeacher().equals(b.getTeacher())) {
                    hardScore--;
                }
                // A student can attend at most one lesson at the same time.
                if (a.getStudentGroup().equals(b.getStudentGroup())) {
                    hardScore--;
                }
            }
        }
    }
    int softScore = 0;
    // Soft constraints are only implemented in the "complete" implementation
    return HardSoftScore.of(hardScore, softScore);
  }

}
```

Unfortunately, this solution does not scale well because it is non-incremental: every time a lesson is assigned to a different time slot or room, all lessons are re-evaluated to calculate the new score.

A better solution is to create a **src/main/java/com/example/solver/TimeTableConstraintProvider.java** class to perform incremental score calculation. This class uses OptaPlanner's ConstraintStream API which is inspired by Java 8 Streams and SQL. The **ConstraintProvider** scales an order of magnitude better than the **EasyScoreCalculator**: $O(n)$ instead of $O(n^2)$.

Procedure

Create the following **src/main/java/com/example/solver/TimeTableConstraintProvider.java** class:

```java
package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
                // Hard constraints
                roomConflict(constraintFactory),
                teacherConflict(constraintFactory),
                studentGroupConflict(constraintFactory),
                // Soft constraints are only implemented in the "complete" implementation
        };
    }
```

```java
    private Constraint roomConflict(ConstraintFactory constraintFactory) {
        // A room can accommodate at most one lesson at the same time.

        // Select a lesson ...
        return constraintFactory.from(Lesson.class)
                // ... and pair it with another lesson ...
                .join(Lesson.class,
                        // ... in the same timeslot ...
                        Joiners.equal(Lesson::getTimeslot),
                        // ... in the same room ...
                        Joiners.equal(Lesson::getRoom),
                        // ... and the pair is unique (different id, no reverse pairs)
                        Joiners.lessThan(Lesson::getId))
                // then penalize each pair with a hard weight.
                .penalize("Room conflict", HardSoftScore.ONE_HARD);
    }

    private Constraint teacherConflict(ConstraintFactory constraintFactory) {
        // A teacher can teach at most one lesson at the same time.
        return constraintFactory.from(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getTeacher),
                        Joiners.lessThan(Lesson::getId))
                .penalize("Teacher conflict", HardSoftScore.ONE_HARD);
    }

    private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
        // A student can attend at most one lesson at the same time.
        return constraintFactory.from(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getStudentGroup),
                        Joiners.lessThan(Lesson::getId))
                .penalize("Student group conflict", HardSoftScore.ONE_HARD);
    }

}
```

10.4. GATHER THE DOMAIN OBJECTS IN A PLANNING SOLUTION

A **TimeTable** instance wraps all **Timeslot**, **Room**, and **Lesson** instances of a single dataset. Furthermore, because it contains all lessons, each with a specific planning variable state, it is a *planning solution* and it has a score:

- If lessons are still unassigned, then it is an *uninitialized* solution, for example, a solution with the score **-4init/0hard/0soft**.

- If it breaks hard constraints, then it is an *infeasible* solution, for example, a solution with the score **-2hard/-3soft**.

- If it adheres to all hard constraints, then it is a *feasible* solution, for example, a solution with the score **0hard/-7soft**.

The **TimeTable** class has an **@PlanningSolution** annotation, so Red Hat build of OptaPlanner knows that this class contains all of the input and output data.

Specifically, this class is the input of the problem:

- A **timeslotList** field with all time slots

  - This is a list of problem facts, because they do not change during solving.

- A **roomList** field with all rooms

  - This is a list of problem facts, because they do not change during solving.

- A **lessonList** field with all lessons

  - This is a list of planning entities because they change during solving.

  - Of each **Lesson**:

    - The values of the **timeslot** and **room** fields are typically still **null**, so unassigned. They are planning variables.

    - The other fields, such as **subject**, **teacher** and **studentGroup**, are filled in. These fields are problem properties.

However, this class is also the output of the solution:

- A **lessonList** field for which each **Lesson** instance has non-null **timeslot** and **room** fields after solving

- A **score** field that represents the quality of the output solution, for example, **0hard/-5soft**

Procedure

Create the **src/main/java/com/example/domain/TimeTable.java** class:

```java
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;

@PlanningSolution
public class TimeTable {

    @ValueRangeProvider(id = "timeslotRange")
    @ProblemFactCollectionProperty
    private List<Timeslot> timeslotList;

    @ValueRangeProvider(id = "roomRange")
    @ProblemFactCollectionProperty
    private List<Room> roomList;

    @PlanningEntityCollectionProperty
    private List<Lesson> lessonList;
```

```java
    @PlanningScore
    private HardSoftScore score;

    private TimeTable() {
    }

    public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
            List<Lesson> lessonList) {
        this.timeslotList = timeslotList;
        this.roomList = roomList;
        this.lessonList = lessonList;
    }

    // ******************************
    // Getters and setters
    // ******************************

    public List<Timeslot> getTimeslotList() {
        return timeslotList;
    }

    public List<Room> getRoomList() {
        return roomList;
    }

    public List<Lesson> getLessonList() {
        return lessonList;
    }

    public HardSoftScore getScore() {
        return score;
    }

}
```

The value range providers

The **timeslotList** field is a value range provider. It holds the **Timeslot** instances which OptaPlanner can pick from to assign to the **timeslot** field of **Lesson** instances. The **timeslotList** field has an **@ValueRangeProvider** annotation to connect those two, by matching the **id** with the **valueRangeProviderRefs** of the **@PlanningVariable** in the **Lesson**.

Following the same logic, the **roomList** field also has an **@ValueRangeProvider** annotation.

The problem fact and planning entity properties

Furthermore, OptaPlanner needs to know which **Lesson** instances it can change as well as how to retrieve the **Timeslot** and **Room** instances used for score calculation by your **TimeTableConstraintProvider**.

The **timeslotList** and **roomList** fields have an **@ProblemFactCollectionProperty** annotation, so your **TimeTableConstraintProvider** can select from those instances.

The **lessonList** has an **@PlanningEntityCollectionProperty** annotation, so OptaPlanner can change them during solving and your **TimeTableConstraintProvider** can select from those too.

10.5. CREATE THE TIMETABLE SERVICE

Now you are ready to put everything together and create a REST service. But solving planning problems on REST threads causes HTTP timeout issues. Therefore, the Spring Boot starter injects a **SolverManager**, which runs solvers in a separate thread pool and can solve multiple datasets in parallel.

Procedure

Create the **src/main/java/com/example/solver/TimeTableController.java** class:

```java
package com.example.solver;

import java.util.UUID;
import java.util.concurrent.ExecutionException;

import com.example.domain.TimeTable;
import org.optaplanner.core.api.solver.SolverJob;
import org.optaplanner.core.api.solver.SolverManager;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/timeTable")
public class TimeTableController {

    @Autowired
    private SolverManager<TimeTable, UUID> solverManager;

    @PostMapping("/solve")
    public TimeTable solve(@RequestBody TimeTable problem) {
        UUID problemId = UUID.randomUUID();
        // Submit the problem to start solving
        SolverJob<TimeTable, UUID> solverJob = solverManager.solve(problemId, problem);
        TimeTable solution;
        try {
            // Wait until the solving ends
            solution = solverJob.getFinalBestSolution();
        } catch (InterruptedException | ExecutionException e) {
            throw new IllegalStateException("Solving failed.", e);
        }
        return solution;
    }

}
```

In this example, the initial implementation waits for the solver to finish, which can still cause an HTTP timeout. The *complete* implementation avoids HTTP timeouts much more elegantly.

10.6. SET THE SOLVER TERMINATION TIME

If your planning application does not have a termination setting or a termination event, it theoretically runs forever and in reality eventually causes an HTTP timeout error. To prevent this from occurring, use

the **optaplanner.solver.termination.spent-limit** parameter to specify the length of time after which the application terminates. In most applications, set the time to at least five minutes (**5m**). However, in the Timetable example, limit the solving time to five second, which is short enough to avoid the HTTP timeout.

Procedure

Create the **src/main/resources/application.properties** file with the following content:

```
quarkus.optaplanner.solver.termination.spent-limit=5s
```

10.7. MAKE THE APPLICATION EXECUTABLE

After you complete the Red Hat build of OptaPlanner Spring Boot timetable project, package everything into a single executable JAR file driven by a standard Java **main()** method.

Prerequisites

- You have a completed OptaPlanner Spring Boot timetable project.

Procedure

1. Create the **TimeTableSpringBootApp.java** class with the following content:

   ```java
   package com.example;

   import org.springframework.boot.SpringApplication;
   import org.springframework.boot.autoconfigure.SpringBootApplication;

   @SpringBootApplication
   public class TimeTableSpringBootApp {

       public static void main(String[] args) {
           SpringApplication.run(TimeTableSpringBootApp.class, args);
       }

   }
   ```

2. Replace the **src/main/java/com/example/DemoApplication.java** class created by Spring Initializr with the **TimeTableSpringBootApp.java** class.

3. Run the **TimeTableSpringBootApp.java** class as the main class of a regular Java application.

10.7.1. Try the timetable application

After you start the Red Hat build of OptaPlanner Spring Boot timetable application, you can test the REST service with any REST client that you want. This example uses the Linux **curl** command to send a POST request.

Prerequisites

- The OptaPlanner Spring Boot timetable application is running.

Procedure

Enter the following command:

```
$ curl -i -X POST http://localhost:8080/timeTable/solve -H "Content-Type:application/json" -d
'{"timeslotList":[{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"}],"roomList":[{"name":"Room
A"},{"name":"Room B"}],"lessonList":[{"id":1,"subject":"Math","teacher":"A. Turing","studentGroup":"9th
grade"},{"id":2,"subject":"Chemistry","teacher":"M. Curie","studentGroup":"9th grade"},
{"id":3,"subject":"French","teacher":"M. Curie","studentGroup":"10th grade"},
{"id":4,"subject":"History","teacher":"I. Jones","studentGroup":"10th grade"}]}'
```

After about five seconds, the termination spent time defined in **application.properties**, the service
returns an output similar to the following example:

```
HTTP/1.1 200
Content-Type: application/json
...

{"timeslotList":...,"roomList":...,"lessonList":[{"id":1,"subject":"Math","teacher":"A.
Turing","studentGroup":"9th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":{"name":"Room A"}},
{"id":2,"subject":"Chemistry","teacher":"M. Curie","studentGroup":"9th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":{"name":"Room A"}},
{"id":3,"subject":"French","teacher":"M. Curie","studentGroup":"10th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":{"name":"Room B"}},
{"id":4,"subject":"History","teacher":"I. Jones","studentGroup":"10th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":{"name":"Room
B"}}],"score":"0hard/0soft"}
```

Notice that the application assigned all four lessons to one of the two time slots and one of the two
rooms. Also notice that it conforms to all hard constraints. For example, M. Curie's two lessons are in
different time slots.

On the server side, the **info** log shows what OptaPlanner did in those five seconds:

```
... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score calculation
speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score calculation
speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed (28524/sec),
phase total (2), environment mode (REPRODUCIBLE).
```

10.7.2. Test the application

A good application includes test coverage. This example tests the Timetable Red Hat build of
OptaPlanner Spring Boot application. It uses a JUnit test to generate a test dataset and send it to the
TimeTableController to solve.

Procedure

Create the **src/test/java/com/example/solver/TimeTableControllerTest.java** class with the following
content:

```java
package com.example.solver;

import java.time.DayOfWeek;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

import com.example.domain.Lesson;
import com.example.domain.Room;
import com.example.domain.TimeTable;
import com.example.domain.Timeslot;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@SpringBootTest(properties = {
        "optaplanner.solver.termination.spent-limit=1h", // Effectively disable this termination in favor of
the best-score-limit
        "optaplanner.solver.termination.best-score-limit=0hard/*soft"})
public class TimeTableControllerTest {

    @Autowired
    private TimeTableController timeTableController;

    @Test
    @Timeout(600_000)
    public void solve() {
        TimeTable problem = generateProblem();
        TimeTable solution = timeTableController.solve(problem);
        assertFalse(solution.getLessonList().isEmpty());
        for (Lesson lesson : solution.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(solution.getScore().isFeasible());
    }

    private TimeTable generateProblem() {
        List<Timeslot> timeslotList = new ArrayList<>();
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30), LocalTime.of(9,
30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30), LocalTime.of(10,
30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30), LocalTime.of(11,
30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30), LocalTime.of(14,
30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30), LocalTime.of(15,
30)));

        List<Room> roomList = new ArrayList<>();
```

```
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
        lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
        lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
        lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
        return new TimeTable(timeslotList, roomList, lessonList);
    }

}
```

This test verifies that after solving, all lessons are assigned to a time slot and a room. It also verifies that it found a feasible solution (no hard constraints broken).

Normally, the solver finds a feasible solution in less than 200 milliseconds. Notice how the **@SpringBootTest** annotation's **properties** overwrites the solver termination to terminate as soon as a feasible solution (**0hard/*soft**) is found. This avoids hard coding a solver time, because the unit test might run on arbitrary hardware. This approach ensures that the test runs long enough to find a feasible solution, even on slow systems. However, it does not run a millisecond longer than it strictly must, even on fast systems.

10.7.3. Logging

After you complete the Red Hat build of OptaPlanner Spring Boot timetable application, you can use logging information to help you fine-tune the constraints in the **ConstraintProvider**. Review the score calculation speed in the **info** log file to assess the impact of changes to your constraints. Run the application in debug mode to show every step that your application takes or use trace logging to log every step and every move.

Procedure

1. Run the timetable application for a fixed amount of time, for example, five minutes.

2. Review the score calculation speed in the **log** file as shown in the following example:

   > ... Solving ended: ..., score calculation speed (29455/sec), ...

3. Change a constraint, run the planning application again for the same amount of time, and review the score calculation speed recorded in the **log** file.

4. Run the application in debug mode to log every step:

   - To run debug mode from the command line, use the **-D** system property.

   - To change logging in the **application.properties** file, add the following line to that file:

> logging.level.org.optaplanner=debug

The following example shows output in the **log** file in debug mode:

> ... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode
> (REPRODUCIBLE), random (JDK with seed 0).
> ...   CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15),
> picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
> ...   CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15),
> picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
> ...

5. Use **trace** logging to show every step and every move for each step.

10.8. ADD DATABASE AND UI INTEGRATION

After you create the Red Hat build of OptaPlanner application example with Spring Boot, add database and UI integration.

Prerequisite

- You have created the OptaPlanner Spring Boot timetable example.

Procedure

1. Create Java Persistence API (JPA) repositories for **Timeslot**, **Room**, and **Lesson**. For information about creating JPA repositories, see *Accessing Data with JPA* on the Spring website.

2. Expose the JPA repositories through REST. For information about exposing the repositories, see *Accessing JPA Data with REST* on the Spring website.

3. Build a **TimeTableRepository** facade to read and write a **TimeTable** in a single transaction.

4. Adjust the **TimeTableController** as shown in the following example:

   ```java
   package com.example.solver;

   import com.example.domain.TimeTable;
   import com.example.persistence.TimeTableRepository;
   import org.optaplanner.core.api.score.ScoreManager;
   import org.optaplanner.core.api.solver.SolverManager;
   import org.optaplanner.core.api.solver.SolverStatus;
   import org.springframework.beans.factory.annotation.Autowired;
   import org.springframework.web.bind.annotation.GetMapping;
   import org.springframework.web.bind.annotation.PostMapping;
   import org.springframework.web.bind.annotation.RequestMapping;
   import org.springframework.web.bind.annotation.RestController;

   @RestController
   @RequestMapping("/timeTable")
   public class TimeTableController {

       @Autowired
   ```

```java
    private TimeTableRepository timeTableRepository;
    @Autowired
    private SolverManager<TimeTable, Long> solverManager;
    @Autowired
    private ScoreManager<TimeTable> scoreManager;

    // To try, GET http://localhost:8080/timeTable
    @GetMapping()
    public TimeTable getTimeTable() {
        // Get the solver status before loading the solution
        // to avoid the race condition that the solver terminates between them
        SolverStatus solverStatus = getSolverStatus();
        TimeTable solution =
timeTableRepository.findById(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
        scoreManager.updateScore(solution); // Sets the score
        solution.setSolverStatus(solverStatus);
        return solution;
    }

    @PostMapping("/solve")
    public void solve() {
        solverManager.solveAndListen(TimeTableRepository.SINGLETON_TIME_TABLE_ID,
            timeTableRepository::findById,
            timeTableRepository::save);
    }

    public SolverStatus getSolverStatus() {
        return
solverManager.getSolverStatus(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
    }

    @PostMapping("/stopSolving")
    public void stopSolving() {
        solverManager.terminateEarly(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
    }

}
```

For simplicity, this code handles only one **TimeTable** instance, but it is straightforward to enable multi-tenancy and handle multiple **TimeTable** instances of different high schools in parallel.

The **getTimeTable()** method returns the latest timetable from the database. It uses the **ScoreManager** (which is automatically injected) to calculate the score of that timetable so the UI can show the score.

The **solve()** method starts a job to solve the current timetable and store the time slot and room assignments in the database. It uses the **SolverManager.solveAndListen()** method to listen to intermediate best solutions and update the database accordingly. This enables the UI to show progress while the backend is still solving.

5. Now that the **solve()** method returns immediately, adjust the **TimeTableControllerTest** as shown in the following example:

```java
package com.example.solver;

import com.example.domain.Lesson;
```

```java
import com.example.domain.TimeTable;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.optaplanner.core.api.solver.SolverStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@SpringBootTest(properties = {
        "optaplanner.solver.termination.spent-limit=1h", // Effectively disable this termination in favor of the best-score-limit
        "optaplanner.solver.termination.best-score-limit=0hard/*soft"})
public class TimeTableControllerTest {

    @Autowired
    private TimeTableController timeTableController;

    @Test
    @Timeout(600_000)
    public void solveDemoDataUntilFeasible() throws InterruptedException {
        timeTableController.solve();
        TimeTable timeTable = timeTableController.getTimeTable();
        while (timeTable.getSolverStatus() != SolverStatus.NOT_SOLVING) {
            // Quick polling (not a Test Thread Sleep anti-pattern)
            // Test is still fast on fast systems and doesn't randomly fail on slow systems.
            Thread.sleep(20L);
            timeTable = timeTableController.getTimeTable();
        }
        assertFalse(timeTable.getLessonList().isEmpty());
        for (Lesson lesson : timeTable.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(timeTable.getScore().isFeasible());
    }

}
```

6. Poll for the latest solution until the solver finishes solving.

7. To visualize the time table, build an attractive web UI on top of these REST methods.

CHAPTER 11. RED HAT BUILD OF OPTAPLANNER WITH JAVA SOLVERS: A CLOUD BALANCING QUICK START GUIDE

This guide walks you through the process of creating Java solvers with OptaPlanner's constraint solving artificial intelligence (AI).

11.1. GETTING STARTED WITH JAVA SOLVERS: A CLOUD BALANCING EXAMPLE

An example demonstrates development of a basic Red Hat build of OptaPlanner solver using Java code.

Suppose your company owns a number of cloud computers and needs to run a number of processes on those computers. You must assign each process to a computer.

The following hard constraints must be fulfilled:

- Every computer must be able to handle the minimum hardware requirements of the sum of its processes:

  - **CPU capacity**: The CPU power of a computer must be at least the sum of the CPU power required by the processes assigned to that computer.

  - **Memory capacity**: The RAM memory of a computer must be at least the sum of the RAM memory required by the processes assigned to that computer.

  - **Network capacity**: The network bandwidth of a computer must be at least the sum of the network bandwidth required by the processes assigned to that computer.
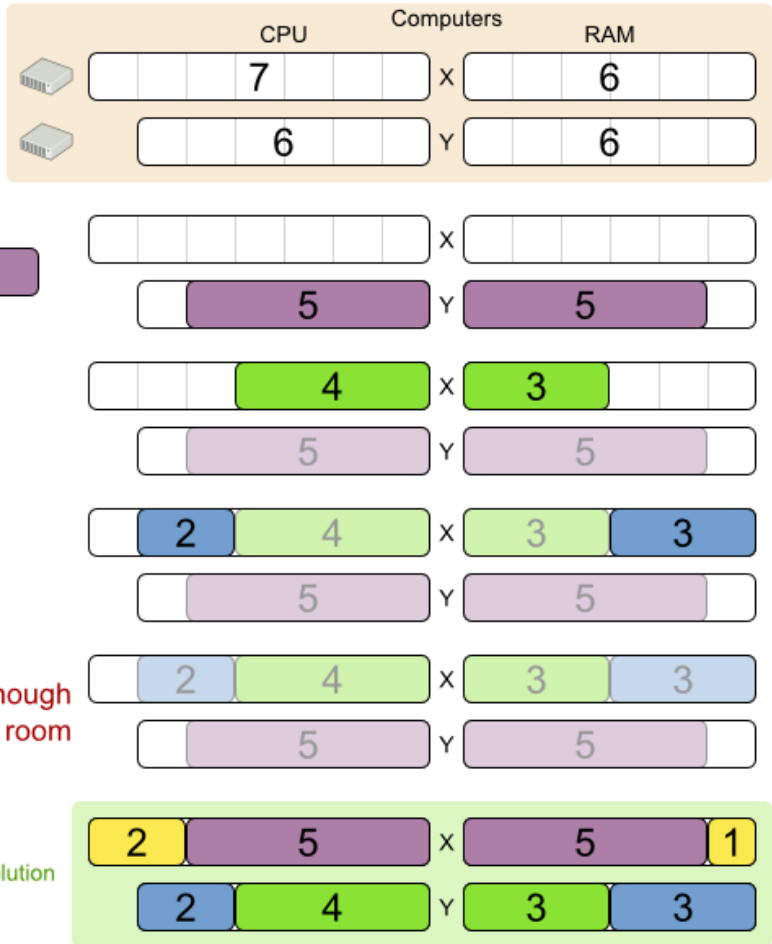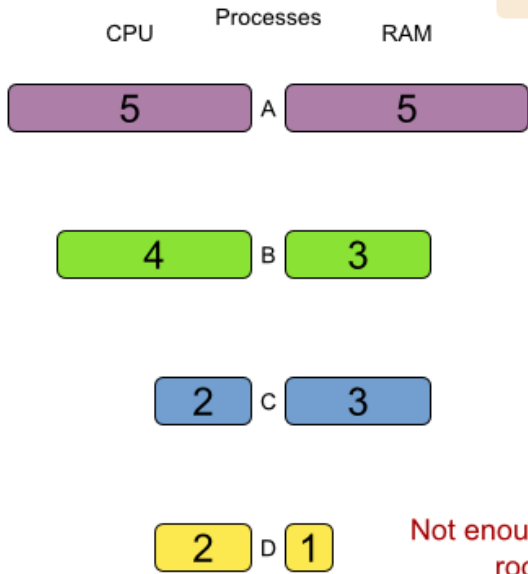
The following soft constraints should be optimized:

- Each computer that has one or more processes assigned incurs a maintenance cost which is fixed for each computer.

  - **Cost**: Minimize the total maintenance cost.

This problem is a form of *bin packing*. In the following simplified example, we assign four processes to two computers with two constraints (CPU and RAM) with a simple algorithm:
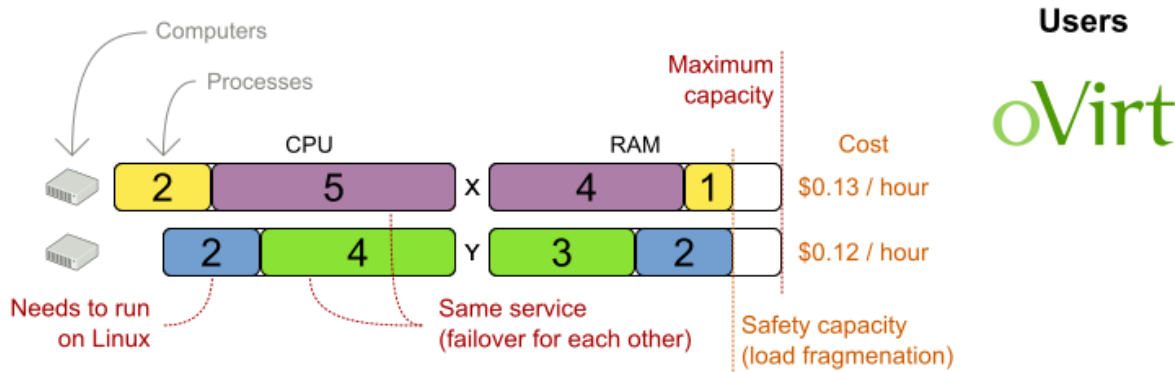
The simple algorithm used here is the *First Fit Decreasing* algorithm, which assigns the bigger processes first and assigns the smaller processes to the remaining space. As you can see, it is not optimal, because it does not leave enough room to assign the yellow process **D**.

OptaPlanner finds a more optimal solution by using additional, smarter algorithms. It also scales: both in data (more processes, more computers) and constraints (more hardware requirements, other constraints).

The following summary applies to this example, as well as to an advanced implementation with more constraints that is described in Section 4.7, "Machine reassignment (Google ROADEF 2012)":

Table 11.1. Cloud balancing problem size

Problem size	Computers	Processes	Search space
2computers-6processes	2	6	64
3computers-9processes	3	9	10^4
4computers-012processes	4	12	10^7
100computers-300processes	100	300	10^600
200computers-600processes	200	600	10^1380
400computers-1200processes	400	1200	10^3122
800computers-2400processes	800	2400	10^6967

11.1.1. Domain Model Design

Using a *domain model* helps determine which classes are planning entities and which of their properties are planning variables. It also helps to simplify constraints, improve performance, and increase flexibility for future needs.

11.1.1.1. Designing a domain model

To create a domain model, define all the objects that represent the input data for the problem. In this example, the objects are processes and computers.

A separate object in the domain model must represent a full data set of the problem, which contains the input data as well as a solution. In this example, this object holds a list of computers and a list of processes. Each process is assigned to a computer; the distribution of processes between computers is the solution.

Procedure

1. Draw a class diagram of your domain model.

2. Normalize it to remove duplicate data.

3. Write down some *sample instances* for each class. Sample instances are entity properties that are relevant for planning purposes.

   - **Computer**: Represents a computer with certain hardware and maintenance costs.
     In this example, the sample instances for the **Computer** class are **cpuPower**, **memory**, **networkBandwidth**, **cost**.

   - **Process**: Represents a process with a demand. Needs to be assigned to a **Computer** by Planner.
     Sample instances for **Process** are **requiredCpuPower**, **requiredMemory**, and **requiredNetworkBandwidth**.

   - **CloudBalance**: Represents the distribution of processes between computers. Contains every **Computer** and **Process** for a certain data set.
     For an object representing the full data set and solution, a sample instance holding the *score* must be present. OptaPlanner can calculate and compare the scores for different solutions; the solution with the highest score is the optimal solution. Therefore, the sample instance for **CloudBalance** is **score**.

4. Determine which relationships (or fields) change during planning:

   - *Planning entity*: The class (or classes) that OptaPlanner can change during solving. In this example, it is the class **Process**, because we can move processes to different computers.

     - A class representing input data that OptaPlanner can not change is known as a *problem fact*.

   - *Planning variable*: The property (or properties) of a planning entity class that changes during solving. In this example, it is the property **computer** on the class **Process**.

   - *Planning solution*: The class that represents a solution to the problem. This class must represent the full data set and contain all planning entities. In this example that is the class **CloudBalance**.

In the UML class diagram below, the OptaPlanner concepts are already annotated:

Cloud balance class diagram
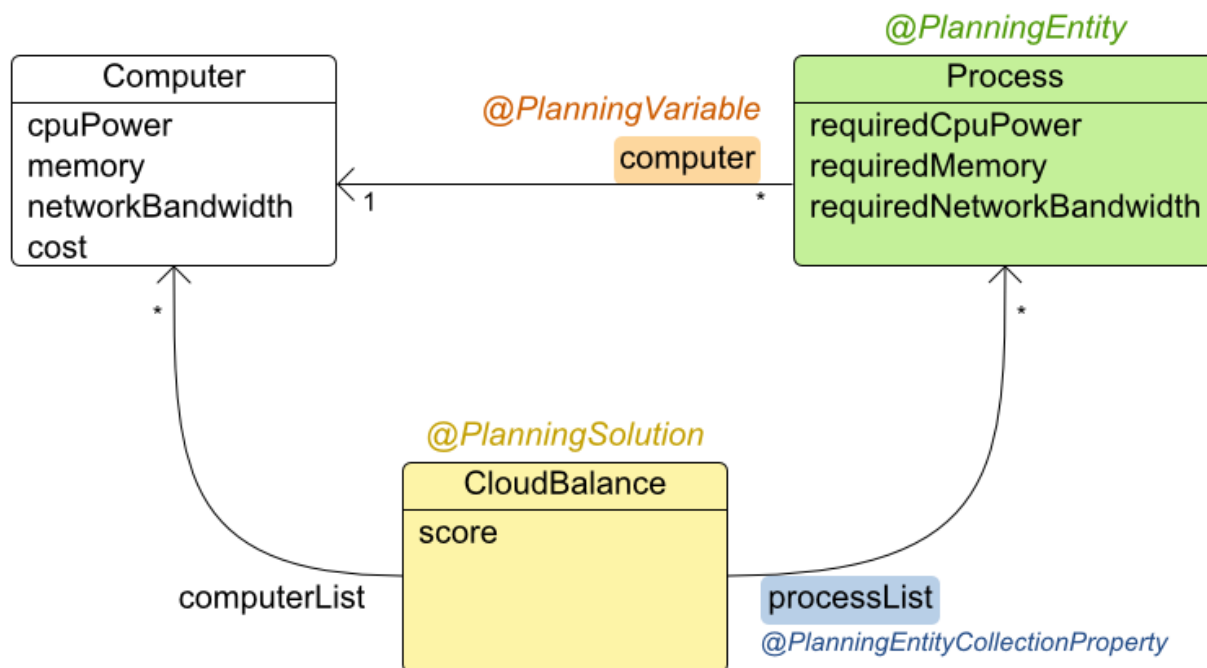
You can find the class definitions for this example in the **examples/sources/src/main/java/org/optaplanner/examples/cloudbalancing/domain** directory.

11.1.1.2. The **Computer** Class

The **Computer** class is a Java object that stores data, sometimes known as a POJO (Plain Old Java Object). Usually, you will have more of this kind of classes with input data.

> **Example 11.1. CloudComputer.java**
>
> ```java
> public class CloudComputer ... {
>
>     private int cpuPower;
>     private int memory;
>     private int networkBandwidth;
>     private int cost;
>
>     ... // getters
> }
> ```

11.1.1.3. The **Process** Class

The **Process** class is the class that is modified during solving.

We need to tell OptaPlanner that it can change the property **computer**. To do this, annotate the class with **@PlanningEntity** and annotate the **getComputer()** getter with **@PlanningVariable**.

Of course, the property **computer** needs a setter too, so OptaPlanner can change it during solving.

Example 11.2. CloudProcess.java

```java
@PlanningEntity(...)
public class CloudProcess ... {

    private int requiredCpuPower;
    private int requiredMemory;
    private int requiredNetworkBandwidth;

    private CloudComputer computer;

    ... // getters

    @PlanningVariable(valueRangeProviderRefs = {"computerRange"})
    public CloudComputer getComputer() {
        return computer;
    }

    public void setComputer(CloudComputer computer) {
        computer = computer;
    }

    // ************************************************************************
    // Complex methods
    // ************************************************************************

    ...

}
```

OptaPlanner needs to know which values it can choose from to assign to the property **computer**. Those values are retrieved from the method **CloudBalance.getComputerList()** on the planning solution, which returns a list of all computers in the current data set.

The **@PlanningVariable**'s **valueRangeProviderRefs** parameter on **CloudProcess.getComputer()** needs to match with the **@ValueRangeProvider**'s **id** on **CloudBalance.getComputerList()**.

> **NOTE**
>
> You can also use annotations on fields instead of getters.

11.1.1.4. The **CloudBalance** Class

The **CloudBalance** class has a *@PlanningSolution* annotation.

This class holds a list of all computers and processes. It represents both the planning problem and (if it is initialized) the planning solution.

The **CloudBalance** class has the following key attributes:

- It holds a collection of processes that OptaPlanner can change. We annotate the getter **getProcessList()** with **@PlanningEntityCollectionProperty**, so that OptaPlanner can retrieve the processes that it can change. To save a solution, OptaPlanner initializes a new instance of the class with the list of changed processes.

  1. It also has a **@PlanningScore** annotated property **score**, which is the **Score** of that solution in its current state. OptaPlanner automatically updates it when it calculates a **Score** for a solution instance; therefore, this property needs a setter.

  2. Especially for score calculation with Drools, the property **computerList** needs to be annotated with a **@ProblemFactCollectionProperty** so that OptaPlanner can retrieve a list of computers (problem facts) and make it available to the decision engine.

Example 11.3. CloudBalance.java

```java
@PlanningSolution
public class CloudBalance ... {

    private List<CloudComputer> computerList;

    private List<CloudProcess> processList;

    private HardSoftScore score;

    @ValueRangeProvider(id = "computerRange")
    @ProblemFactCollectionProperty
    public List<CloudComputer> getComputerList() {
        return computerList;
    }

    @PlanningEntityCollectionProperty
    public List<CloudProcess> getProcessList() {
        return processList;
    }

    @PlanningScore
    public HardSoftScore getScore() {
        return score;
    }

    public void setScore(HardSoftScore score) {
        this.score = score;
    }

    ...
}
```

11.1.2. Running the cloud balancing Hello World application

You can run the sample Hello World application to demonstrate the solver.

Procedure

1. Download and configure the examples in your preferred IDE. For instructions on downloading and configuring examples in an IDE, see Section 5.2, "Running the Red Hat build of OptaPlanner examples in an IDE (IntelliJ, Eclipse, or Netbeans)".

2. Create a run configuration with the following main class:
   org.optaplanner.examples.cloudbalancing.app.CloudBalancingHelloWorld
   By default, the Cloud Balancing Hello World is configured to run for 120 seconds.

Result

The application executes the following code:

Example 11.4. CloudBalancingHelloWorld.java

```java
public class CloudBalancingHelloWorld {

    public static void main(String[] args) {
        // Build the Solver
        SolverFactory<CloudBalance> solverFactory =
SolverFactory.createFromXmlResource("org/optaplanner/examples/cloudbalancing/solver/cloudBalancingSolverConfig.xml");
        Solver<CloudBalance> solver = solverFactory.buildSolver();

        // Load a problem with 400 computers and 1200 processes
        CloudBalance unsolvedCloudBalance = new
CloudBalancingGenerator().createCloudBalance(400, 1200);

        // Solve the problem
        CloudBalance solvedCloudBalance = solver.solve(unsolvedCloudBalance);

        // Display the result
        System.out.println("\nSolved cloudBalance with 400 computers and 1200 processes:\n" +
toDisplayString(solvedCloudBalance));
    }

    ...
}
```

The code example does the following:

1. Builds the **Solver** based on a solver configuration (in this case an XML file, **cloudBalancingSolverConfig.xml**, from the classpath).
   Building the **Solver** is the most complicated part of this procedure. For more details, see Section 11.1.3, "Solver Configuration".

   ```java
   SolverFactory<CloudBalance> solverFactory = SolverFactory.createFromXmlResource(

   "org/optaplanner/examples/cloudbalancing/solver/cloudBalancingSolverConfig.xml");
       Solver solver<CloudBalance> = solverFactory.buildSolver();
   ```

2. Loads the problem.
   CloudBalancingGenerator generates a random problem: you will replace this with a class that loads a real problem, for example from a database.

```
        CloudBalance unsolvedCloudBalance = new
    CloudBalancingGenerator().createCloudBalance(400, 1200);
```

3. Solve the problem.

```
        CloudBalance solvedCloudBalance = solver.solve(unsolvedCloudBalance);
```

4. Displays the result.

```
        System.out.println("\nSolved cloudBalance with 400 computers and 1200 processes:\n"
            + toDisplayString(solvedCloudBalance));
```

11.1.3. Solver Configuration

The solver configuration file determines how the solving process works; it is considered a part of the code. The file is named **examples/sources/src/main/resources/org/optaplanner/examples/cloudbalancing/solver/cloudBalancingSolverConfig.xml**.

Example 11.5. cloudBalancingSolverConfig.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<solver>
  <!-- Domain model configuration -->
  <scanAnnotatedClasses/>

  <!-- Score configuration -->
  <scoreDirectorFactory>

<easyScoreCalculatorClass>org.optaplanner.examples.cloudbalancing.optional.score.CloudBalancingEasyScoreCalculator</easyScoreCalculatorClass>
    <!--
<scoreDrl>org/optaplanner/examples/cloudbalancing/solver/cloudBalancingScoreRules.drl</scoreDrl>
>-->
  </scoreDirectorFactory>

  <!-- Optimization algorithms configuration -->
  <termination>
    <secondsSpentLimit>30</secondsSpentLimit>
  </termination>
</solver>
```

This solver configuration consists of three parts:

1. **Domain model configuration**: *What can OptaPlanner change?*
   We need to make OptaPlanner aware of our domain classes. In this configuration, it will automatically scan all classes in your classpath (for a **@PlanningEntity** or **@PlanningSolution** annotation):

```xml
    <scanAnnotatedClasses/>
```

2. **Score configuration**: *How should OptaPlanner optimize the planning variables? What is our goal?*
   Since we have hard and soft constraints, we use a **HardSoftScore**. But we need to tell OptaPlanner how to calculate the score, depending on our business requirements. Further down, we will look into two alternatives to calculate the score: using a basic Java implementation and using Drools DRL.

   ```
   <scoreDirectorFactory>

   <easyScoreCalculatorClass>org.optaplanner.examples.cloudbalancing.optional.score.CloudBal
   ancingEasyScoreCalculator</easyScoreCalculatorClass>
       <!--
   <scoreDrl>org/optaplanner/examples/cloudbalancing/solver/cloudBalancingScoreRules.drl</scor
   eDrl>-->
     </scoreDirectorFactory>
   ```

3. **Optimization algorithms configuration**: *How should OptaPlanner optimize it?* In this case, we use the default optimization algorithms (because no explicit optimization algorithms are configured) for 30 seconds:

   ```
   <termination>
     <secondsSpentLimit>30</secondsSpentLimit>
   </termination>
   ```

   OptaPlanner should get a good result in seconds (and even in less than 15 milliseconds if the real-time planning feature is used), but the more time it has, the better the result will be. Advanced use cases might use different termination criteria than a hard time limit.

   The default algorithms will already easily surpass human planners and most in-house implementations. You can use the advanced Benchmarker feature to power tweak to get even better results.

11.1.4. Score Configuration

OptaPlanner will search for the **Solution** with the highest **Score**. This example uses a **HardSoftScore**, which means OptaPlanner will look for the solution with no hard constraints broken (fulfill hardware requirements) and as little as possible soft constraints broken (minimize maintenance cost).
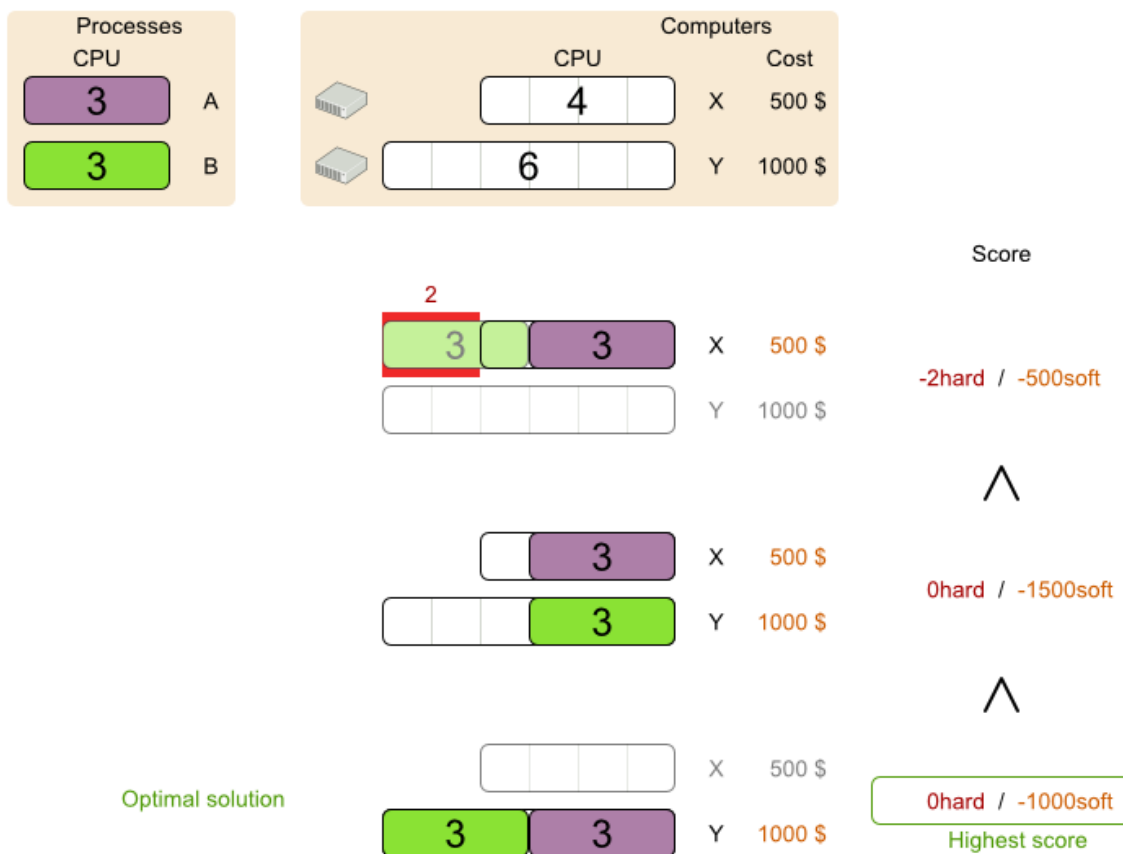
You can define constraints using plain Java, Drools, or the OptaPlanner ConstraintStream API. For information about the ConstraintStream API, see Section 10.3, "Define the constraints and calculate the score".

11.1.4.1. Configuring score calculation using Java

One way to define a score function is to implement the interface **EasyScoreCalculator** in plain Java.

Procedure

1. In the **cloudBalancingSolverConfig.xml** file, add or uncomment the setting:

    ```
    <scoreDirectorFactory>

    <easyScoreCalculatorClass>org.optaplanner.examples.cloudbalancing.optional.score.CloudBal
    ancingEasyScoreCalculator</easyScoreCalculatorClass>
    </scoreDirectorFactory>
    ```

2. Implement the **calculateScore(Solution)** method to return a **HardSoftScore** instance.

    Example 11.6. CloudBalancingEasyScoreCalculator.java

    ```java
    public class CloudBalancingEasyScoreCalculator implements
    EasyScoreCalculator<CloudBalance> {

        /**
         * A very simple implementation. The double loop can easily be removed by using
    ```

```java
Maps as shown in
   * {@link
CloudBalancingMapBasedEasyScoreCalculator#calculateScore(CloudBalance)}.
   */
 public HardSoftScore calculateScore(CloudBalance cloudBalance) {
    int hardScore = 0;
    int softScore = 0;
    for (CloudComputer computer : cloudBalance.getComputerList()) {
        int cpuPowerUsage = 0;
        int memoryUsage = 0;
        int networkBandwidthUsage = 0;
        boolean used = false;

        // Calculate usage
        for (CloudProcess process : cloudBalance.getProcessList()) {
            if (computer.equals(process.getComputer())) {
                cpuPowerUsage += process.getRequiredCpuPower();
                memoryUsage += process.getRequiredMemory();
                networkBandwidthUsage += process.getRequiredNetworkBandwidth();
                used = true;
            }
        }

        // Hard constraints
        int cpuPowerAvailable = computer.getCpuPower() - cpuPowerUsage;
        if (cpuPowerAvailable < 0) {
            hardScore += cpuPowerAvailable;
        }
        int memoryAvailable = computer.getMemory() - memoryUsage;
        if (memoryAvailable < 0) {
            hardScore += memoryAvailable;
        }
        int networkBandwidthAvailable = computer.getNetworkBandwidth() -
networkBandwidthUsage;
        if (networkBandwidthAvailable < 0) {
            hardScore += networkBandwidthAvailable;
        }

        // Soft constraints
        if (used) {
            softScore -= computer.getCost();
        }
    }
    return HardSoftScore.valueOf(hardScore, softScore);
 }

}
```

Even if we optimize the code above to use **Map**s to iterate through the **processList** only once, *it is still slow* because it does not do incremental score calculation.

To fix that, either use incremental Java score calculation or Drools score calculation. Incremental Java score calculation is not covered in this guide.

11.1.4.2. Configuring score calculation using Drools

You can use Drools rule language (DRL) to define constraints. Drools score calculation uses incremental calculation, where every score constraint is written as one or more score rules.

Using the decision engine for score calculation enables you to integrate with other Drools technologies, such as decision tables (XLS or web based), Business Central, and other supported features.

Procedure

1. Add a **scoreDrl** resource in the classpath to use the decision engine as a score function. In the **cloudBalancingSolverConfig.xml** file, add or uncomment the setting:

   ```
   <scoreDirectorFactory>

   <scoreDrl>org/optaplanner/examples/cloudbalancing/solver/cloudBalancingScoreRules.drl</scoreDrl>
     </scoreDirectorFactory>
   ```

2. Create the hard constraints. These constraints ensure that all computers have enough CPU, RAM and network bandwidth to support all their processes:

   Example 11.7. cloudBalancingScoreRules.drl – Hard Constraints

   ```
   ...

   import org.optaplanner.examples.cloudbalancing.domain.CloudBalance;
   import org.optaplanner.examples.cloudbalancing.domain.CloudComputer;
   import org.optaplanner.examples.cloudbalancing.domain.CloudProcess;

   global HardSoftScoreHolder scoreHolder;

   //
   ################################################################################
   ####
   // Hard constraints
   //
   ################################################################################
   ####

   rule "requiredCpuPowerTotal"
     when
       $computer : CloudComputer($cpuPower : cpuPower)
       accumulate(
         CloudProcess(
           computer == $computer,
           $requiredCpuPower : requiredCpuPower);
         $requiredCpuPowerTotal : sum($requiredCpuPower);
         $requiredCpuPowerTotal > $cpuPower
       )
     then
       scoreHolder.addHardConstraintMatch(kcontext, $cpuPower -
   $requiredCpuPowerTotal);
     end

   rule "requiredMemoryTotal"
   ```

```
    ...
end

rule "requiredNetworkBandwidthTotal"
    ...
end
```

3. Create a soft constraint. This constraint minimizes the maintenance cost. It is applied only if hard constraints are met:

Example 11.8. cloudBalancingScoreRules.drl – Soft Constraints

```
//
#############################################################################
####
// Soft constraints
//
#############################################################################
####

rule "computerCost"
    when
        $computer : CloudComputer($cost : cost)
        exists CloudProcess(computer == $computer)
    then
        scoreHolder.addSoftConstraintMatch(kcontext, - $cost);
end
```

11.1.5. Further development of the solver

Now that this example works, you can try developing it further. For example, you can enrich the domain model and add extra constraints such as these:

- Each **Process** belongs to a **Service**. A computer might crash, so processes running the same service should (or must) be assigned to different computers.

- Each **Computer** is located in a **Building**. A building might burn down, so processes of the same services should (or must) be assigned to computers in different buildings.

PART IV. RED HAT BUILD OF OPTAPLANNER STARTER APPLICATIONS

Red Hat build of OptaPlanner provides the following starter applications that you can deploy out-of-the-box on Red Hat OpenShift Container Platform:

- Employee Rostering starter application

- Vechile Route Planning starter application

OptaPlanner starter applications are more developed than examples and quick start guides. They focus on specific use cases and use the best technologies available to build a planning solution.

CHAPTER 12. USING RED HAT BUILD OF OPTAPLANNER IN AN IDE: AN EMPLOYEE ROSTERING EXAMPLE

As a business rules developer, you can use an IDE to build, run, and modify the **optaweb-employee-rostering** starter application that uses the Red Hat build of OptaPlanner functionality.

Prerequisites

- You use an integrated development environment, such as Red Hat CodeReady Studio or IntelliJ IDEA.

- You have an understanding of the Java language.

- You have an understanding of React and TypeScript. This requirement is necessary to develop the OptaWeb UI.

12.1. OVERVIEW OF THE EMPLOYEE ROSTERING STARTER APPLICATION

The employee rostering starter application assigns employees to shifts on various positions in an organization. For example, you can use the application to distribute shifts in a hospital between nurses, guard duty shifts across a number of locations, or shifts on an assembly line between workers.

Optimal employee rostering must take a number of variables into account. For example, different skills can be required for shifts in different positions. Also, some employees might be unavailable for some time slots or might prefer a particular time slot. Moreover, an employee can have a contract that limits the number of hours that the employee can work in a single time period.

The Red Hat build of OptaPlanner rules for this starter application use both hard and soft constraints. During an optimization, the planning engine may not violate hard constraints, for example, if an employee is unavailable (out sick), or that an employee cannot work two spots in a single shift. The planning engine tries to adhere to soft constraints, such as an employee's preference to not work a specific shift, but can violate them if the optimal solution requires it.

12.2. BUILDING AND RUNNING THE EMPLOYEE ROSTERING STARTER APPLICATION

You can build the employee rostering starter application from the source code and run it as a JAR file.

Alternatively, you can use your IDE, for example, Eclipse (including Red Hat CodeReady Studio), to build and run the application.

12.2.1. Preparing deployment files

You must download and prepare the deployment files before building and deploying the application.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required), and select the product and version from the drop-down options:

   - Product: Decision Manager

- Version: 7.11

2. Download **Red Hat Decision Manager 7.11.0 Kogito and OptaPlanner 8 Decision Services Quickstarts** (**rhdm-7.11.0-decision-services-quickstarts.zip**).

3. Extract the **rhdm-7.11.0-decision-services-quickstarts.zip** file.

4. Download the **Red Hat Decision Manager 7.11 Maven Repository Kogito and OptaPlanner 8 Maven Repository** (**rhdm-7.11.0-kogito-maven-repository.zip**) file.

5. Extract the **rhdm-7.11.0-kogito-maven-repository.zip** file.

6. Copy the contents of the **rhdm-7.11.0-kogito-maven-repository/maven-repository** subdirectory into the **~/.m2/repository** directory.

7. Navigate to the **optaweb-8.5.0.Final-redhat-00004/optaweb-employee-rostering** directory. This folder is the base folder in subsequent parts of this document.

> **NOTE**
>
> File and folder names might have higher version numbers than specifically noted in this document.

12.2.2. Running the Employee Rostering starter application JAR file

You can run the Employee Rostering starter application from a JAR file included in the **Red Hat Decision Manager 7.11.0 Kogito and OptaPlanner 8 Decision Services Quickstarts** download.

Prerequisites

- You have downloaded and extracted the **rhdm-7.11.0-decision-services-quickstarts.zip** file as described in Section 12.2.1, "Preparing deployment files".

- A Java Development Kit is installed.

- Maven is installed.

- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.

Procedure

1. In a command terminal, change to the **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-employee-rostering** directory.

2. Enter the following command:

   ```
   mvn clean install -DskipTests
   ```

3. Wait for the build process to complete.

4. Navigate to the **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-employee-rostering/optaweb-employee-rostering-standalone/target** directory.

5. Enter the following command to run the Employee Rostering JAR file:

```
java -jar quarkus-app/quarkus-run.jar
```

> **NOTE**
>
> The value of the **quarkus.datasource.db-kind** parameter is set to **H2** by default at build time. To use a different database, you must rebuild the standalone module and specify the database type on the command line. For example, to use a PostgreSQL database, enter the following command:
>
> **mvn clean install -DskipTests -Dquarkus.profile=postgres**

6. To access the application, enter **http://localhost:8080/** in a web browser.

12.2.3. Building and running the Employee Rostering starter application using Maven

You can use the command line to build and run the employee rostering starter application.

If you use this procedure, the data is stored in memory and is lost when the server is stopped. To build and run the application with a database server for persistent storage, see Section 12.2.4, "Building and running the employee rostering starter application with persistent data storage from the command line".

Prerequisites

- You have prepared the deployment files as described in Section 12.2.1, "Preparing deployment files".

- A Java Development Kit is installed.

- Maven is installed.

- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.
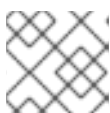
Procedure

1. Navigate to the **optaweb-employee-rostering-backend** directory.

2. Enter the following command:

```
mvn quarkus:dev
```

3. Navigate to the **optaweb-employee-rostering-frontend** directory.

4. Enter the following command:

```
npm start
```

> **NOTE**
>
> If you use **npm** to start the server, **npm** monitors code changes.

5. To access the application, enter **http://localhost:3000/** in a web browser.

12.2.4. Building and running the employee rostering starter application with persistent data storage from the command line

If you use the command line to build the employee rostering starter application and run it, you can provide a database server for persistent data storage.

Prerequisites

- You have prepared the deployment files as described in Section 12.2.1, "Preparing deployment files".

- A Java Development Kit is installed.

- Maven is installed.

- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.

- You have a deployed MySQL or PostrgeSQL database server.

Procedure

1. In a command terminal, navigate to the **optaweb-employee-rostering-standalone/target** directory.

2. Enter the following command to run the Employee Rostering JAR file:

```
java \
-Dquarkus.datasource.username=<DATABASE_USER> \
-Dquarkus.datasource.password=<DATABASE_PASSWORD> \
-Dquarkus.datasource.jdbc.url=<DATABASE_URL> \
-jar quarkus-app/quarkus-run.jar
```

In this example, replace the following placeholders:

- **<DATABASE_URL>**: URL to connect to the database

- **<DATABASE_USER>**: The user to connect to the database

- **<DATABASE_PASSWORD>**: The password for **<DATABASE_USER>**

> **NOTE**
>
> The value of the **quarkus.datasource.db-kind** parameter is set to **H2** by default at build time. To use a different database, you must rebuild the standalone module and specify the database type on the command line. For example, to use a PostgreSQL database, enter the following command:
>
> **mvn clean install -DskipTests -Dquarkus.profile=postgres**

12.2.5. Building and running the employee rostering starter application using IntelliJ IDEA

You can use IntelliJ IDEA to build and run the employee rostering starter application.

Prerequisites

- You have downloaded the Employee Rostering source code, available from the Employee Rostering GitHub page.

- IntelliJ IDEA, Maven, and Node.js are installed.

- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.

Procedure

1. Start IntelliJ IDEA.

2. From the IntelliJ IDEA main menu, select **File → Open**.

3. Select the root directory of the application source and click **OK**.

4. From the main menu, select **Run → Edit Configurations**.

5. In the window that appears, expand **Templates** and select **Maven**. The Maven sidebar appears.

6. In the Maven sidebar, select **optaweb-employee-rostering-backend** from the **Working Directory** menu.

7. In **Command Line**, enter **mvn quarkus:dev**.

8. To start the back end, click **OK** .

9. In a command terminal, navigate to the **optaweb-employee-rostering-frontend** directory.

10. Enter the following command to start the front end:

    ```
    npm start
    ```

11. To access the application, enter **http://localhost:3000/** in a web browser.

12.3. OVERVIEW OF THE SOURCE CODE OF THE EMPLOYEE ROSTERING STARTER APPLICATION

The employee rostering starter application consists of the following principal components:

- A **backend** that implements the rostering logic using Red Hat build of OptaPlanner and provides a REST API

- A **frontend** module that implements a user interface using React and interacts with the **backend** module through the REST API

You can build and use these components independently. In particular, you can implement a different user interface and use the REST API to call the server.

In addition to the two main components, the employee rostering template contains a generator of random source data (useful for demonstration and testing purposes) and a benchmarking application.

Modules and key classes

The Java source code of the employee rostering template contains several Maven modules. Each of these modules includes a separate Maven project file (**pom.xml**), but they are intended for building in a common project.

The modules contain a number of files, including Java classes. This document lists all the modules, as well as the classes and other files that contain the key information for the employee rostering calculations.

- **optaweb-employee-rostering-benchmark** module: Contains an additional application that generates random data and benchmarks the solution.

- **optaweb-employee-rostering-distribution** module: Contains README files.

- **optaweb-employee-rostering-docs** module: Contains documentation files.

- **optaweb-employee-rostering-frontend** module: Contains the client application with the user interface, developed in React.

- **optaweb-employee-rostering-backend** module: Contains the server application that uses OptaPlanner to perform the rostering calculation.

  - **src/main/java/org.optaweb.employeerostering.service.roster/rosterGenerator.java**: Generates random input data for demonstration and testing purposes. If you change the required input data, change the generator accordingly.

  - **src/main/java/org.optaweb.employeerostering.domain.employee/EmployeeAvailability .java**: Defines availability information for an employee. For every time slot, an employee can be unavailable, available, or the time slot can be designated a preferred time slot for the employee.

  - **src/main/java/org.optaweb.employeerostering.domain.employee/Employee.java**: Defines an employee. An employee has a name, a list of skills, and works under a contract. Skills are represented by skill objects.

  - **src/main/java/org.optaweb.employeerostering.domain.roster/Roster.java**: Defines the calculated rostering information.

  - **src/main/java/org.optaweb.employeerostering.domain.shift/Shift.java**: Defines a shift to which an employee can be assigned. A shift is defined by a time slot and a spot. For example, in a diner there could be a shift in the **Kitchen** spot for the February 20 8AM–4PM time slot. Multiple shifts can be defined for a specific spot and time slot. In this case, multiple employees are required for this spot and time slot.

  - **src/main/java/org.optaweb.employeerostering.domain.skill/Skill.java**: Defines a skill that an employee can have.

  - **src/main/java/org.optaweb.employeerostering.domain.spot/Spot.java**: Defines a spot where employees can be placed. For example, a **Kitchen** can be a spot.

  - **src/main/java/org.optaweb.employeerostering.domain.contract/Contract.java**: Defines a contract that sets limits on work time for an employee in various time periods.

  - **src/main/java/org.optaweb.employeerostering.domain.tenant/Tenant.java**: Defines a tenant. Each tenant represents an independent set of data. Changes in the data for one tenant do not affect any other tenants.

- ***View.java**: Classes related to domain objects that define value sets that are calculated from other information; the client application can read these values through the REST API, but not write them.

- ***Service.java**: Interfaces located in the service package that define the REST API. Both the server and the client application separately define implementations of these interfaces.

- **optaweb-employee-rostering-standalone** module: Contains the assembly configurations for the standalone application.

12.4. MODIFYING THE EMPLOYEE ROSTERING STARTER APPLICATION

To modify the employee rostering starter application to suit your needs, you must change the rules that govern the optimization process. You must also ensure that the data structures include the required data and provide the required calculations for the rules. If the required data is not present in the user interface, you must also modify the user interface.

The following procedure outlines the general approach to modifying the employee rostering starter application.

Prerequisites

- You have a build environment that successfully builds the application.

- You can read and modify Java code.

Procedure

1. Plan the required changes. Answer the following questions:

   - What are the additional scenarios that *must* be avoided? These scenarios are *hard constraints*.

   - What are the additional scenarios that the optimizer must *try to avoid* when possible? These scenarios are *soft constraints*.

   - What data is required to calculate if each scenario is happening in a potential solution?

   - Which of the data can be derived from the information that the user enters in the existing version?

   - Which of the data can be hardcoded?

   - Which of the data must be entered by the user and is not entered in the current version?

2. If any required data can be calculated from the current data or can be hardcoded, add the calculations or hardcoding to existing view or utility classes. If the data must be calculated on the server side, add REST API endpoints to read it.

3. If any required data must be entered by the user, add the data to the classes representing the data entities (for example, the **Employee** class), add REST API endpoints to read and write the data, and modify the user interface to enter the data.

4. When all of the data is available, modify the rules. For most modifications, you must add a new rule. The rules are located in the

src/main/java/org/optaweb/employeerostering/service/solver/EmployeeRosteringConstrai ntProvider.java file of the **optaweb-employee-rostering-backend** module.

5. After modifying the application, build and run it.

CHAPTER 13. DEPLOYING AND USING RED HAT BUILD OF OPTAPLANNER IN RED HAT OPENSHIFT CONTAINER PLATFORM: AN EMPLOYEE ROSTERING STARTER EXAMPLE

As a business rules developer, you can test and interact with the Red Hat build of OptaPlanner functionality by quickly deploying the **optaweb-employee-rostering** starter project included in the Red Hat Decision Manager distribution to OpenShift.

Prerequisites

- You have access to a deployed OpenShift environment. For details, see the documentation for the OpenShift product that you use.

13.1. OVERVIEW OF THE EMPLOYEE ROSTERING STARTER APPLICATION

The employee rostering starter application assigns employees to shifts on various positions in an organization. For example, you can use the application to distribute shifts in a hospital between nurses, guard duty shifts across a number of locations, or shifts on an assembly line between workers.

Optimal employee rostering must take a number of variables into account. For example, different skills can be required for shifts in different positions. Also, some employees might be unavailable for some time slots or might prefer a particular time slot. Moreover, an employee can have a contract that limits the number of hours that the employee can work in a single time period.

The Red Hat build of OptaPlanner rules for this starter application use both hard and soft constraints. During an optimization, the planning engine may not violate hard constraints, for example, if an employee is unavailable (out sick), or that an employee cannot work two spots in a single shift. The planning engine tries to adhere to soft constraints, such as an employee's preference to not work a specific shift, but can violate them if the optimal solution requires it.

13.2. INSTALLING AND STARTING THE EMPLOYEE ROSTERING STARTER APPLICATION ON OPENSHIFT

Use the **runOnOpenShift.sh** script to deploy the Employee Rostering starter application to Red Hat OpenShift Container Platform. The **runOnOpenShift.sh** shell script is available in the **Red Hat Decision Manager 7.11.0 Kogito and OptaPlanner 8 Decision Services Quickstarts** distribution.

The **runOnOpenShift.sh** script builds and packages the application source code locally and uploads it to the OpenShift environment for deployment. This method requires Java Development Kit, Apache Maven, and a bash shell command line.

13.2.1. Deploying the application using the provided script

You can deploy the Employee Rostering starter application to Red Hat OpenShift Container Platform using the provided script. The script builds and packages the application source code locally and uploads it to the OpenShift environment for deployment.

Prerequisites

- You are logged in to the target OpenShift environment using the **oc** command line tool. For more information about this tool, see the OpenShift Container Platform CLI Reference.

- OpenJDK 11 or later is installed. Red Hat build of Open JDK is available from the Software Downloads page in the Red Hat Customer Portal (login required).

- Apache Maven 3.6 or higher is installed. Maven is available from the Apache Maven Project website.

- A **bash** shell environment is available on your local system.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required), and select the product and version from the drop-down options:

   - Product: Decision Manager

   - Version: 7.11

2. Download the **Red Hat Decision Manager 7.11 Maven Repository Kogito and OptaPlanner 8 Maven Repository** (**rhdm-7.11.0-kogito-maven-repository.zip**) file.

3. Extract the **rhdm-7.11.0-kogito-maven-repository.zip** file.

4. Copy the contents of the **rhdm-7.11.0-kogito-maven-repository/maven-repository** subdirectory into the **~/.m2/repository** directory.

5. Download the **rhdm-7.11.0-decision-services-quickstarts.zip** file from the Software Downloads page of the Red Hat Customer Portal.

6. Extract the downloaded archive.

7. Navigate to the **optaweb-employee-rostering** folder.

8. To build the Employee Rostering application, run the following command:

   ```
   mvn clean install -DskipTests -DskipITs
   ```

9. Log in to an OpenShift account or a Red Hat Code Ready Container instance. In the following example, **<account-url>** is the URL for an OpenShift account or Red Hat Code Ready Container instance and **<login-token>** is the login token for that account:

   ```
   oc login <account-url> --token <login-token>
   ```

10. Create a new project to host Employee Rostering:

    ```
    oc new-project optaweb-employee-rostering
    ```

11. Run the provision script to build and deploy the application:

    ```
    ./runOnOpenShift.sh
    ```

    Compilation and packaging might take up to 10 minutes to complete. These processes continually show progress on the command line output.

    When the operation completes, the following message is displayed, where **<URL>** is the URL for the deployment:

> You can access the application at <URL> once the deployment is done.

12. Enter the URL that you used earlier in the procedure, for either an OpenShift account or Red Hat Code Ready Container instance, to access the deployed application. The first startup can take up to a minute because additional building is completed on the OpenShift platform.
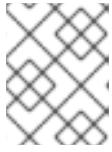
> **NOTE**
>
> If the application does not open a minute after clicking the link, perform a hard refresh of your browser page.

13.3. USING THE EMPLOYEE ROSTERING STARTER APPLICATION

You can use the web interface to use the Employee Rostering starter application. The interface is developed in ReactJS. You can also access the REST API to create a custom user interface as necessary.

13.3.1. The draft and published periods

At any particular moment, you can use the application to create the roster for a time period, called a *draft* period. By default, the length of a draft period is three weeks.

When the roster is final for the first week of the draft period, you can *publish* the roster. At this time, the roster for the first week of the current draft period becomes a *published* period. In a published period, the roster is fixed and you can no longer change it automatically (however, emergency manual changes are still possible). This roster can then be distributed to employees so they can plan their time around it. The draft period is shifted a week later.

For example, assume that a draft period of September 1 to September 21 is set. You can automatically create the employee roster for this period. Then, when you publish the roster, the period up to September 7 becomes published. The new draft period is September 8 to September 28.

For instructions about publishing the roster, see Section 13.3.12, "Publishing the shift roster" .

13.3.2. The rotation pattern

The employee rostering application supports a *rotation pattern* for shifts and employees.

The rotation pattern is a "model" period of any time starting from two days. The pattern is not tied to a particular date.

You can create time buckets for every day of the rotation. Every time bucket sets the time of a shift. Optionally, the template can include the name of the default employee for the shift.

When you publish the roster, the application adds a new week to the draft period. At this time, the shifts and, if applicable, default employee names are copied from the rotation pattern to the new part of the draft period.

When the end of the rotation pattern is reached, it is automatically restarted from the beginning.

If weekend shift patterns in your organization are different from weekday shift patterns, use a rotation pattern of one week or a whole number of weeks, for example, 14, 21, or 28 days. The default length is 28 days. Then the pattern is always repeated on the same weekdays and you can set the shifts for different weekdays.

For instructions about editing the rotation pattern, see Section 13.3.13, "Viewing and editing the rotation pattern".

13.3.3. Employee Rostering tenants

The Employee Rostering application supports multiple *tenants*. Each tenant is an independent set of data, including inputs and roster outputs. Changing data for one tenant does not affect other tenants. You can switch between tenants to use several independent data sets, for example, to prepare employee rosters for different locations.

Several sample tenants are present after installation, representing several typical enterprise types such as a factory or hospital. You can select any of these tenants and modify them to suit your needs. You can also create a new tenant to enter data from a blank slate.

13.3.3.1. Changing an Employee Rostering tenant

You can change the current tenant. After you select a different tenant, all of the displayed information refers to this tenant and any changes you make affect only this tenant.

Procedure

1. In the Employee Rostering application web interface, in the top right part of the browser window, click the **Tenant** list.

2. Select a tenant from the list.

13.3.3.2. Creating a tenant

You can create a new tenant to enter data from a blank slate. When creating a tenant, you can set several parameters that determine how the application prepares the output for this tenant.
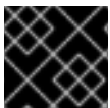
IMPORTANT

You cannot change tenant parameters after you create the tenant.

Procedure

1. To create a new tenant in the Employee Rostering application web interface, in the top right corner of the browser window click the settings (gear) icon then click **Add**.

2. Set the following values:

   - **Name**: The name of the new tenant. This name is displayed in the list of tenants.

   - **Schedule Start Date**: The start date of the initial draft period. After you publish the roster, this date becomes the start date of the published period. The weekday of this date always remains the weekday that starts the draft period, any particular published period, and the first use of the rotation pattern. So it is usually most convenient to set the start date to the start of a week (Sunday or Monday).

   - **Draft Length (days)**: The length of the draft period. The draft period stays the same length for the lifetime of the tenant.

   - **Publish Notice (days)**: The length of the publish notice period. Aspire to publish the final roster for any day at least this time in advance, so employees have enough notice to plan

their personal life around their shift times. In the current version, this setting is not enforced in any way.

- **Publish Length (days)**: The length of the period that becomes published (fixed) every time you publish the roster. In the current version, this setting is fixed at 7 days.

- **Rotation Length (days)**: The length of the rotation pattern.

- **Timezone**: The timezone of the environment to which the roster applies. This timezone is used to determine the "current" date for user interface display.

3. Click **Save**.

The tenant is created with blank data.

13.3.4. Entering skills

You can set all *skills* that are required in any position within the roster. For example, a 24-hour diner can require cooking, serving, bussing, and hosting skills, in addition to skills such as general human resources and restaurant operations.

Procedure

1. In the Employee Rostering application web interface, click the **Skills** tab.
   You can see the numbers of currently visible skills in the top right part of the browser window, for example, **1-15 of 34**. You can use the ‹ and › buttons to display other skills in the list.

   You can enter any part of a skill name in the **Search** box to search for skills.

2. Complete the following steps to add a new skill:

   a. Click **Add**.

   b. Enter the name of the new skill in the text field under **Name**.

   c. Click the Save icon.

3. To edit the name of a skill, click the **Edit Skill** icon (pencil shape) next to the skill.

4. To delete a skill, click the **Delete Skill** icon (trashcan shape) next to the skill.

> **NOTE**
>
> Within each tenant, skill names must be unique. You cannot delete a skill if the skill is associated with an employee or spot.

13.3.5. Entering spots

You must enter the list of *spots*, which represent various positions at the business. For a diner, spots include the bar, the bussing stations, the front counter, the various kitchen stations, the serving areas, and the office.

For each spot, you can select one or more required skills from the list that you entered in the **Skills** tab. The application rosters only employees that have all of the required skills for a spot into that spot. If the spot has no required skill, the application can roster any employee into the spot.

Procedure

1. To enter or change spots in the Employee Rostering application web interface, click the **Spots** tab. You can enter any part of a spot name in the **Search** box to search for spots.

2. Complete the following steps to add a new spot:

   a. Click **Add Spot**.

   b. Enter the name of the new spot in the text field under **Name**.

   c. Optional: Select one or more skills from the drop-down list under **Required skill set**.

   d. Click the Save icon.

3. To edit the name and required skills for a spot, click the **Edit Spot** icon (pencil shape) next to the spot.

4. To delete a spot, click the **Delete Spot** icon (trashcan shape) next to the spot.

> **NOTE**
>
> Within each tenant, spot names must be unique. You cannot delete a spot when any shifts are created for it.

13.3.6. Entering the list of contracts

You must enter the list of all of the types of contracts that the business uses for employees.

A contract determines the maximum time that the employee can work in a day, calendar week, calendar month, or calendar year.

When creating a contract, you can set any of the limitations or none at all. For example, a part-time employee might not be allowed to work more than 20 hours in a week, while a full-time employee might be limited to 10 hours in a day and 1800 hours in a year. Another contract might include no limitations on worked hours.

You must enter all work time limits for contracts in minutes.

Procedure

1. To enter or change the list of contracts in the Employee Rostering application web interface, click the **Contracts** tab.
   You can see the numbers of currently visible contracts in the top right part of the browser window, for example, **1-15 of 34**. You can use the **<** and **>** buttons to display other contracts in the list.

   You can enter any part of a contract name in the **Search** box to search for contracts.

2. Complete the following steps to add a new contract:

   a. Click **Add**.

   b. Enter the name of the contract in the text field under **Name**.

   c. Enter the required time limits under **Maximum minutes**:

- If the employee must not work more than a set time per day, enable the check box at **Per Day** and enter the amount of minutes in the field next to this check box.

- If the employee must not work more than a set time per calendar week, enable the check box at **Per Week** and enter the amount of minutes in the field next to this check box.

- If the employee must not work more than a set time per calendar month, enable the check box at **Per Month** and enter the amount of minutes in the field next to this check box.

- If the employee must not work more than a set time per calendar year, enable the check box at **Per Year** and enter the amount of minutes in the field next to this check box.

    d. Click the Save icon.

3. To edit the name and time limits for a contract, click the **Edit Contract** icon (pencil shape) next to the name of the contract.

4. To delete a contract, click the **Delete Contract** icon (trashcan shape) next to the name of the contract.
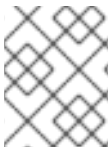
> **NOTE**
>
> Within each tenant, contract names must be unique. You cannot delete a contract if it is assigned to any employee.

13.3.7. Entering the list of employees

You must enter the list of all employees of the business, the skills they possess, and the contracts that apply to them. The application rosters these employees to spots according to their skills and according to the work time limits in the contracts.

Procedure

1. To enter or change the list of employees in the Employee Rostering application web interface, click the **Employees** tab.
   You can see the numbers of currently visible employees in the top right part of the browser window, for example, **1-15 of 34**. You can use the **<** and **>** buttons to display other employees in the list.

   You can enter any part of an employee name in the **Search** box to search for employees.

2. Complete the following steps to add a new employee:

    a. Click **Add**.

    b. Enter the name of the employee in the text field under **Name**.

    c. Optional: Select one or more skills from the drop-down list under **Skill set**.

    d. Select a contract from the drop-down list under **Contract**.

    e. Click the Save icon.

3. To edit the name and skills for an employee, click the **Edit Employee** icon (pencil shape) next to the name of the employee.

4. To delete an employee, click the **Delete Employee** icon (trashcan shape) next to the name of the employee.

> **NOTE**
>
> Within each tenant, employee names must be unique. You cannot delete employees if they are rostered to any shifts.

13.3.8. Setting employee availability

You can set the availability of employees for particular time spans.

If an employee is *unavailable* for a particular time span, the employee can never be assigned to any shift during this time span (for example, if the employee has called in sick or is on vacation). *Undesired* and *desired* are employee preferences for particular time spans; the application accommodates them when possible.

Procedure

1. To view and edit employee availability in the Employee Rostering application web interface, click the **Availability Roster** tab.
   In the top left part of the window, you can see the dates for which the roster is displayed. To view other weeks, you can use the **<** and **>** buttons next to the **Week of** field. Alternatively, you can click the date field and change the date to view the week that includes this date.

2. To create an availability entry for an employee, click empty space on the schedule and then select an employee. Initially, an **Unavailable** entry for the entire day is created.

3. To change an availability entry, click the entry. You can change the following settings:

   - **From** and **To** date and time: The time span to which the availability entry applies.

   - Status: you can select **Unavailable**, **Desired**, or **Undesired** status from a drop-down list. To save the entry, click **Apply**.

4. To delete an availability entry, click the entry, then click **Delete availability**.
   You can also change or delete an availability entry by moving the mouse pointer over the entry and then clicking one of the icons displayed over the entry:

   - Click the 🔴 icon to set the status of the entry to **Unavailable**.

   - Click the ⚠️ icon to set the status of the entry to **Undesired**.

   - Click the 🟢 icon to set the status of the entry to **Desired**.

   - Click the 🗑️ icon to delete the entry.
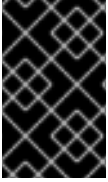
> **IMPORTANT**
>
> If an employee is already assigned to a shift and then you create or change an availability entry during this shift, the assignment is not changed automatically. You must create the employee shift roster again to apply new or changed availability entries.

13.3.9. Viewing and editing shifts in the shift roster

The Shift Roster is a table of all spots and all possible time spans.

If an employee must be present in a spot during a time span, a *shift* must exist for this spot and this time span. If a spot requires several employees at the same time, you can create several shifts for the same spot and time span.

Each shift is represented by a rectangle at the intersection of a spot (row) and time span (column).

When new time is added to the draft period, the application copies the shifts (and default employees, if present) from the rotation pattern into this new part of the draft period. You can also manually add and edit shifts in the draft period.

Procedure

1. To view and edit the shift roster in the Employee Rostering application web interface, click the **Shift** tab.
   In the top left part of the window, you can see the dates for which the roster is displayed. To view other weeks, you can use the < and > buttons next to the **Week of** field. Alternatively, you can click the date field and change the date to view the week that includes this date.

2. To add a shift, click an open area of the schedule. The application adds a shift, determining the slot and time span automatically from the location of the click.

3. To edit a shift, click the shift. You can set the following values for a shift:

   - **From** and **To** date and time: The exact time and duration of the shift.

   - **Employee**: The employee assigned to the shift.

   - **Pinned**: Whether the employee is *pinned* to the shift. If an employee is pinned, automatic employee rostering cannot change the assignment of the employee to the shift. A pinned employee is not automatically replicated to any other shift.
     To save the changes, click **Apply**.

4. To delete a shift, click the shift, and then click **Delete shift**

13.3.10. Creating and viewing the employee shift roster

You can use the application to create and view the optimal shift roster for all employees.

Procedure

1. To view and edit the shift roster in the Employee Rostering application web interface, click the **Shift** tab.

2. To create the optimal shift roster, click **Schedule**. The application takes 30 seconds to find the optimal solution.

Result

When the operation is finished, the Shift Roster view contains the optimal shift roster. The new roster is created for the draft period. The operation does not modify the published periods.

In the top left part of the window, you can see the dates for which the roster is displayed. To view other weeks, you can use the **<** and **>** buttons next to the **Week of** field. Alternatively, you can click the date field and change the date to view the week that includes this date.

In the draft period, the borders of boxes that represent shifts are dotted lines. In the published periods, the borders are unbroken lines.

The color of the boxes that represent shifts shows the constraint status of every shift:

- Strong green: Soft constraint matched; for example, the shift is in a "desired" timeslot for the employee.

- Pale green: No constraint broken.

- Grey: Soft constraint broken; for example, the shift is in an "undesired" timeslot for the employee.

- Yellow: Medium constraint broken; for example, no employee is assigned to the shift.

- Red: Hard constraint broken; for example, an employee has two shifts at the same time.

13.3.11. Viewing employee shifts

You can view the assigned shifts for particular employees in an employee-centric table. The information is the same as the Shift Roster, but the viewing format might be more convenient for informing employees about their assigned shifts.

Procedure

To view a table of employees and shifts in the Employee Rostering application web interface, click the **Availability Roster** tab.

In the top left part of the window, you can see the dates for which the roster is displayed. To view other weeks, you can use the **<** and **>** buttons next to the **Week of** field. Alternatively, you can click the date field and change the date to view the week that includes this date.

You can see the numbers of currently visible employees in the top right part of the browser window, for example, **1-10 of 34**. You can use the **<** and **>** buttons next to the numbers to display other employees in the list.

In the draft period, the borders of boxes representing shifts are dotted lines. In the published periods, the borders are unbroken lines.

13.3.12. Publishing the shift roster

When you publish the shift roster, the first week of the draft period becomes published. Automatic employee rostering no longer changes any shift assignments in the published period, though emergency manual changing is still possible. The draft period is shifted one week later. For details about draft and published periods, see Section 13.3.1, "The draft and published periods".

Procedure

1. To view and edit the shift roster in the Employee Rostering application web interface, click the **Shift** tab.

2. Review the shift roster for the first week of the draft period to ensure that it is acceptable.

3. Click **Publish**.

13.3.13. Viewing and editing the rotation pattern

The rotation pattern enables you to add, move, and delete shifts so you can manage your employee resources efficiently. It is defined by time buckets and seats.

- A time bucket describes a time slot (for example, 9:00 a.m. to 5:00 p.m.) for a particular spot or location **(A)** (for example, Anaesthetics), over two or more days, and any skills that are required (for example, firearm training).

- A seat **(B)** is an employee assignment for a particular day in a specific time bucket.

- An employee stub is an icon that represents an employee that is available to be assigned to a time bucket. Employee stubs are listed in the **Employee Stub List**.

For more information about the rotation pattern, see Section 13.3.2, "The rotation pattern".

Procedure

1. Click the **Rotation** tab to view and edit the rotation pattern.

2. Select a spot from the **Rotation** menu.

3. Click **Add New Time Bucket**. The **Creating Working Time Bucket** dialog is displayed.

4. Specify a start and end time, select any additional required skills, select the days for this time bucket, and click **Save**. The unassigned seats for that time bucket appears on the **Rotation** page organized by time ranges.

5. To create an employee stub list so that you can add employees to the rotation, click **Edit Employee Stub List**.

6. In the **Edit Employee Stub List** dialog, click **Add Employee** and select an employee from the list.

7. Add all of the employees required for this stub list and click **Save**. The employees appear above the time buckets on the **Rotation** page.

8. Click an employee icon to select an employee from the employee stub list.

9. Click and drag the mouse over the seats of a time bucket to assign the selected employee to those seats. The seat is populated with the employee icon.

> **NOTE**
>
> A time bucket can only have one employee assigned to it for each day. To add multiple employees to the same time bucket, copy the time bucket and change the employee name as required.

10. To provision the schedule, click **Scheduling** and select the spot that you created the rotation for.

11. Click **Provision** and specify the date range.

12. Deselect the spots that you do not want to include in this schedule.

13. Click the arrow next to the selected spot and deselect any time buckets that you do not want to use in your schedule.

14. Click **Provision Shifts**. The calendar is populated with shifts generated from the time buckets.

15. To modify a shift, click a generated shift on the calendar.

CHAPTER 14. DEPLOYING AND USING THE RED HAT BUILD OF OPTAPLANNER VEHICLE ROUTE PLANNING STARTER APPLICATION

As a developer, you can use the OptaWeb Vehicle Routing starter application to optimize your vehicle fleet deliveries.

Prerequisites

- OpenJDK (JDK) 11 is installed. Red Hat build of Open JDK is available from the Software Downloads page in the Red Hat Customer Portal (login required).

- Apache Maven 3.6 or higher is installed. Maven is available from the Apache Maven Project website.

14.1. WHAT IS OPTAWEB VEHICLE ROUTING?

The main purpose of many businesses is to transport various types of cargo. The goal of these businesses is to deliver a piece of cargo from the loading point to a destination and use its vehicle fleet in the most efficient way. One of the main objectives is to minimize travel costs which are measured in either time or distance.

This type of optimization problem is referred to as the vehicle routing problem (VRP) and has many variations.

Red Hat build of OptaPlanner can solve many of these vehicle routing variations and provides solution examples. OptaPlanner enables developers to focus on modeling business rules and requirements instead of learning constraint programming theory. OptaWeb Vehicle Routing expands the vehicle routing capabilities of OptaPlanner by providing a starter application that answers questions such as these:

- Where do I get the distances and travel times?

- How do I visualize the solution on a map?

- How do I build an application that runs in the cloud?

OptaWeb Vehicle Routing uses OpenStreetMap (OSM) data files. For information about OpenStreetMap, see the OpenStreetMap web site.

Use the following definitions when working with OptaWeb Vehicle Routing:

Region: An arbitrary area on the map of Earth, represented by an OSM file. A region can be a country, a city, a continent, or a group of countries that are frequently used together. For example, the DACH region includes Germany (DE), Austria (AT), and Switzerland (CH).

Country code: A two-letter code assigned to a country by the ISO-3166 standard. You can use a country code to filter geosearch results. Because you can work with a region that spans multiple countries (for example, the DACH region), OptaWeb Vehicle Routing accepts a list of country codes so that geosearch filtering can be used with such regions. For a list of country codes, see *ISO 3166 Country Codes*

Geosearch: A type of query where you provide an address or a place name of a region as the search keyword and receive a number of GPS locations as a result. The number of locations returned depends on how unique the search keyword is. Because most place names are not unique, filter out nonrelevant

results by including only places in the country or countries that are in your working region.

14.2. DOWNLOAD AND BUILD THE OPTAWEB VEHICLE ROUTING DEPLOYMENT FILES

You must download and prepare the deployment files before building and deploying OptaWeb Vehicle Routing.

Procedure

1. Navigate to the Software Downloads page in the Red Hat Customer Portal (login required), and select the product and version from the drop-down options:

   - Product: Decision Manager

   - Version: 7.11.0

2. Download **Red Hat Decision Manager 7.11.0 Kogito and OptaPlanner 8 Decision Services Quickstarts** (**rhdm-7.11.0-decision-services-quickstarts.zip**).

3. Extract the **rhdm-7.11.0-decision-services-quickstarts.zip** file.

4. Download the **Red Hat Decision Manager 7.11 Maven Repository Kogito and OptaPlanner 8 Maven Repository** (**rhdm-7.11.0-kogito-maven-repository.zip**) file.

5. Extract the **rhdm-7.11.0-kogito-maven-repository.zip** file.

6. Copy the contents of the **rhdm-7.11.0-kogito-maven-repository/maven-repository** subdirectory into the **~/.m2/repository** directory.

7. Navigate to the **optaweb-8.5.0.Final-redhat-00004/optaweb-vehicle-routing** directory.

8. Enter the following command to build OptaWeb Vehicle Routing:

   ```
   mvn clean package -DskipTests
   ```

14.3. RUN OPTAWEB VEHICLE ROUTING LOCALLY USING THE RUNLOCALLY.SH SCRIPT

Linux users can use the **runLocally.sh** Bash script to run OptaWeb Vehicle Routing.

> **NOTE**
>
> The **runLocally.sh** script does not run on macOS. If you cannot use the **runLocally.sh** script, see Section 14.4, "Configure and run OptaWeb Vehicle Routing manually" .

The **runLocally.sh** script automates the following setup steps that otherwise must be carried out manually:

- Create the data directory.

- Download selected OpenStreetMap (OSM) files from Geofabrik.

- Try to associate a country code with each downloaded OSM file automatically.

- Build the project if the standalone JAR file does not exist.

- Launch OptaWeb Vehicle Routing by taking a single region argument or by selecting the region interactively.

See the following sections for instructions about executing the **runLocally.sh** script:

14.3.1. Run the OptaWeb Vehicle Routing runLocally.sh script in quick start mode

The easiest way to get started with OptaWeb Vehicle Routing is to run the **runLocally.sh** script without any arguments.

Prerequisites

- OptaWeb Vehicle Routing has been successfully built with Maven as described in .

- Internet access is available.

Procedure

1. Enter the following command in the **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-vehicle-routing** directory.

   ```
   ./runLocally.sh
   ```

2. If prompted to create the **.optaweb-vehicle-routing** directory, enter **y**. You are prompted to create this directory the first time you run the script.

3. If prompted to download an OSM file, enter **y**. The first time that you run the script, OptaWeb Vehicle Routing downloads the Belgium OSM file.
   The application starts after the OSM file is downloaded.

4. To open the OptaWeb Vehicle Routing user interface, enter the following URL in a web browser:

   ```
   http://localhost:8080
   ```

> **NOTE**
>
> The first time that you run the script, it will take a few minutes to start because the OSM file must be imported by GraphHopper and stored as a road network graph. The next time you run the **runlocally.sh** script, load times will be significantly faster.

Next steps

14.3.2. Run the OptaWeb Vehicle Routing runLocally.sh script in interactive mode

Use interactive mode to see the list of downloaded OSM files and country codes assigned to each region. You can use the interactive mode to download additional OSM files from Geofabrik without visiting the website and choosing a destination for the download.

Prerequisites

- OptaWeb Vehicle Routing has been successfully built with Maven as described in Section 14.2, "Download and build the OptaWeb Vehicle Routing deployment files".

- Internet access is available.

Procedure

1. Change directory to **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-vehicle-routing**.

2. Enter the following command to run the script in interactive mode:

   ./runLocally.sh -i

3. At the **Your choice** prompt, enter **d** to display the download menu. A list of previously downloaded regions appears followed by a list of regions that you can download.

4. Optional: Select a region from the list of previously downloaded regions:

   a. Enter the number associated with a region in the list of downloaded regions.

   b. Press the Enter key.

5. Optional: Download a region:

   a. Enter the number associated with the region that you want to download. For example, to select the map of Europe, enter **5**.

   b. To download the map, enter **d** then press the Enter key.

   c. To download a specific region within the map, enter **e** then enter the number associated with the region that you want to download, and press the Enter key.

   > **USING LARGE OSM FILES**
   >
   > For the best user experience, use smaller regions such as individual European or US states. Using OSM files larger than 1 GB will require significant RAM size and take a lot of time (up to several hours) for the initial processing.

   The application starts after the OSM file is downloaded.

6. To open the OptaWeb Vehicle Routing user interface, enter the following URL in a web browser:

   http://localhost:8080

Next steps

14.3.3. Run the OptaWeb Vehicle Routing runLocally.sh script in non-interactive mode

Use OptaWeb Vehicle Routing in non-interactive mode to start OptaWeb Vehicle Routing with a single command that includes an OSM file that you downloaded previously. This is useful when you want to switch between regions quickly or when doing a demo.

Prerequisites

- OptaWeb Vehicle Routing has been successfully built with Maven as described in Section 14.2, "Download and build the OptaWeb Vehicle Routing deployment files".

- The OSM file for the region that you want to use has been downloaded. For information about downloading OSM files, see Section 14.3.2, "Run the OptaWeb Vehicle Routing runLocally.sh script in interactive mode".

- Internet access is available.

Procedure

1. Change directory to **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-vehicle-routing**.

2. Execute the following command where **<OSM_FILE_NAME>** is an OSM file that you downloaded previously:

   ```
   ./runLocally.sh <OSM_FILE_NAME>
   ```

Next steps

14.3.4. Update the data directory

You can update the data directory that OptaWeb Vehicle Routing uses if you want to use a different data directory. The default data directory is **$HOME/.optaweb-vehicle-routing**.

Prerequisites

- OptaWeb Vehicle Routing has been successfully built with Maven as described in Section 14.2, "Download and build the OptaWeb Vehicle Routing deployment files".

Procedure

- To use a different data directory, add the directory's absolute path to the **.DATA_DIR_LAST** file in the current data directory.

- To change country codes associated with a region, edit the corresponding file in the **country_codes** directory, in the current data directory.
  For example, if you downloaded an OSM file for Scotland and the script fails to guess the country code, set the content of **country_codes/scotland-latest** to GB.

- To remove a region, delete the corresponding OSM file from **openstreetmap** directory in the data directory and delete the region's directory in the **graphhopper** directory.

14.4. CONFIGURE AND RUN OPTAWEB VEHICLE ROUTING MANUALLY

The easiest way to run OptaWeb Vehicle Routing is to use the **runlocally.sh** script. However, if Bash is not available on your system you can manually complete the steps that the **runlocally.sh** script performs.

Prerequisites

- OptaWeb Vehicle Routing has been successfully built with Maven as described in Section 14.2, "Download and build the OptaWeb Vehicle Routing deployment files".

- Internet access is available.

Procedure

1. Download routing data.
   The routing engine requires geographical data to calculate the time it takes vehicles to travel between locations. You must download and store OpenStreetMap (OSM) data files on the local file system before you run OptaWeb Vehicle Routing.

   > **NOTE**
   >
   > The OSM data files are typically between 100 MB to 1 GB and take time to download so it is a good idea to download the files before building or starting the OptaWeb Vehicle Routing application.

   a. Open **http://download.geofabrik.de/** in a web browser.

   b. Click a region in the **Sub Region** list, for example **Europe**. The subregion page opens.

   c. In the **Sub Regions** table, download the OSM file (**.osm.pbf**) for a country, for example Belgium.

2. Create the data directory structure.
   OptaWeb Vehicle Routing reads and writes several types of data on the file system. It reads OSM (OpenStreetMap) files from the **openstreetmap** directory, writes a road network graph to the **graphhopper** directory, and persists user data in a directory called **db**. Create a new directory dedicated to storing all of these data to make it easier to upgrade to a newer version of OptaWeb Vehicle Routing in the future and continue working with the data you created previously.

   a. Create the **$HOME/.optaweb-vehicle-routing** directory.

   b. Create the **openstreetmap** directory in the **$HOME/.optaweb-vehicle-routing** directory:

   ```
   $HOME/.optaweb-vehicle-routing
   └── openstreetmap
   ```

   c. Move all of your downloaded OSM files (files with the extension **.osm.pbf**) to the **openstreetmap** directory.

The rest of the directory structure is created by the OptaWeb Vehicle Routing application when it runs for the first time. After that, your directory structure is similar to the following example:

```
$HOME/.optaweb-vehicle-routing

├── db
│   └── vrp.mv.db
├── graphhopper
│   └── belgium-latest
└── openstreetmap
    └── belgium-latest.osm.pbf
```

3. Change directory to **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-vehicle-routing/optaweb-vehicle-routing-standalone/target**.

4. To run OptaWeb Vehicle Routing, enter the following command:

```
java \
-Dapp.demo.data-set-dir=$HOME/.optaweb-vehicle-routing/dataset \
-Dapp.persistence.h2-dir=$HOME/.optaweb-vehicle-routing/db \
-Dapp.routing.gh-dir=$HOME/.optaweb-vehicle-routing/graphhopper \
-Dapp.routing.osm-dir=$HOME/.optaweb-vehicle-routing/openstreetmap \
-Dapp.routing.osm-file=<OSM_FILE_NAME> \
-Dapp.region.country-codes=<COUNTRY_CODE_LIST> \
-jar quarkus-app/quarkus-run.jar
```

In this command, replace the following variables:

- **<OSM_FILE_NAME>**: The OSM file for the region that you want to use and that you downloaded previously

- **<COUNTRY_CODE_LIST>**: A comma-separated list of country codes used to filter geosearch queries. For a list of country codes, see *ISO 3166 Country Codes*.
  The application starts after the OSM file is downloaded.

  In the following example, OptaWeb Vehicle Routing downloads the OSM map of Central America (**central-america-latest.osm.pbf**) and searches in the countries Belize (BZ) and Guatemala (GT).

```
java \
-Dapp.demo.data-set-dir=$HOME/.optaweb-vehicle-routing/dataset \
-Dapp.persistence.h2-dir=$HOME/.optaweb-vehicle-routing/db \
-Dapp.routing.gh-dir=$HOME/.optaweb-vehicle-routing/graphhopper \
-Dapp.routing.osm-dir=$HOME/.optaweb-vehicle-routing/openstreetmap \
-Dapp.routing.osm-file=entral-america-latest.osm.pbf \
-Dapp.region.country-codes=BZ,GT \
-jar quarkus-app/quarkus-run.jar
```

5. To open the OptaWeb Vehicle Routing user interface, enter the following URL in a web browser:

```
http://localhost:8080
```

Next steps

Section 14.6, "Using OptaWeb Vehicle Routing"

14.5. RUN OPTAWEB VEHICLE ROUTING ON RED HAT OPENSHIFT CONTAINER PLATFORM

Linux users can use the **runOnOpenShift.sh** Bash script to install OptaWeb Vehicle Routing on Red Hat OpenShift Container Platform.
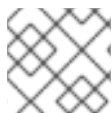
> **NOTE**
>
> The **runOnOpenShift.sh** script does not run on macOS.

Prerequisites

- You have access to an OpenShift cluster and the OpenShift command-line interface (**oc**) has been installed. For information about Red Hat OpenShift Container Platform, see *Installing OpenShift Container Platform*.

- OptaWeb Vehicle Routing has been successfully built with Maven as described in Section 14.2, "Download and build the OptaWeb Vehicle Routing deployment files".

- Internet access is available.

Procedure

1. Log in to or start a Red Hat OpenShift Container Platform cluster.

a. Enter the following command where **<PROJECT_NAME>** is the name of your new project:

```
oc new-project <PROJECT_NAME>
```

b. If necessary, change directory to **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-vehicle-routing**.

c. Enter the following command to execute the **runOnOpenShift.sh** script and download an OpenStreetMap (OSM) file:

```
./runOnOpenShift.sh <OSM_FILE_NAME> <COUNTRY_CODE_LIST>
<OSM_FILE_DOWNLOAD_URL>
```

In this command, replace the following variables:

- **<OSM_FILE_NAME>**: The name of a file downloaded from **<OSM_FILE_DOWNLOAD_URL>**.

- **<COUNTRY_CODE_LIST>**: A comma-separated list of country codes used to filter geosearch queries. For a list of country codes, see *ISO 3166 Country Codes*.

- **<OSM_FILE_DOWNLOAD_URL>**: The URL of an OSM data file in PBF format accessible from OpenShift. The file will be downloaded during backend startup and saved as **/deployments/local/<OSM_FILE_NAME>**.
  In the following example, OptaWeb Vehicle Routing downloads the OSM map of Central America (**central-america-latest.osm.pbf**) and searches in the countries Belize (BZ) and Guatemala (GT).

```
./runOnOpenShift.sh central-america-latest.osm.pbf BZ,GT
http://download.geofabrik.de/europe/central-america-latest.osm.pbf
```

> **NOTE**
>
> For help with the **runOnOpenShift.sh** script, enter **./runOnOpenShift.sh --help**.

14.5.1. Updating the deployed OptaWeb Vehicle Routing application with local changes

After you deploy your OptaWeb Vehicle Routing application on Red Hat OpenShift Container Platform, you can update the back end and front end.

Prerequisites

- OptaWeb Vehicle Routing has been successfully built with Maven and deployed on OpenShift.

Procedure

- To update the back end, perform the following steps:

  1. Change the source code and build the back-end module with Maven.

  2. Change directory to **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-vehicle-routing**.

  3. Enter the following command to start the OpenShift build:

     ```
     oc start-build backend --from-dir=. --follow
     ```

- To update the front end, perform the following steps:

  1. Change the source code and build the front-end module with the **npm** utility.

  2. Change directory to **sources/optaweb-vehicle-routing-frontend**.

  3. Enter the following command to start the OpenShift build:

     ```
     oc start-build frontend --from-dir=docker --follow
     ```

Next steps

Section 14.6, "Using OptaWeb Vehicle Routing"

14.6. USING OPTAWEB VEHICLE ROUTING

In the OptaWeb Vehicle Routing application, you can mark a number of locations on the map. The first location is assumed to be the depot. Vehicles must deliver goods from this depot to every other location that you marked.

You can set the number of vehicles and the carrying capacity of every vehicle. However, the route is not guaranteed to use all vehicles. The application uses as many vehicles as required for an optimal route.

The current version has certain limitations:

- Every delivery to a location is supposed to take one point of vehicle capacity. For example, a vehicle with a capacity of 10 can visit up to 10 locations before returning to the depot.

- Setting custom names of vehicles and locations is not supported.

14.6.1. Creating a route

To create an optimal route, use the **Demo** tab of the OptaWeb Vehicle Routing user interface.

Prerequisites

- OptaWeb Vehicle Routing is running and you have access to the user interface.

Procedure

1. In OptaWeb Vehicle Routing, click **Demo** to open the **Demo** tab.

2. Use the blue minus and plus buttons above the map to set the number of vehicles. Each vehicle has a default capacity of 10.

3. Use the plus button in a square on the map to zoom in as required.
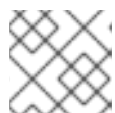
> **NOTE**
>
> Do not double-click to zoom in. A double click also creates a location.

4. Click a location for the depot.

5. Click other locations on the map for delivery points.

6. If you want to delete a location:

   a. Hover the mouse cursor over the location to see the location name.

   b. Find the location name in the list in the left part of the screen.

   c. Click the **X** icon next to the name.

Every time you add or remove a location or change the number of vehicles, the application creates and displays a new optimal route. If the solution uses several vehicles, the application shows the route for every vehicle in a different color.

14.6.2. Viewing and setting other details

You can use other tabs in the OptaWeb Vehicle Routing user interface to view and set additional details.

Prerequisites

- OptaWeb Vehicle Routing is running and you have access to the user interface.

Procedure

- Click the **Vehicles** tab to view, add, and remove vehicles, and also set the capacity for every vehicle.

- Click the **Visits** tab to view and remove locations.

- Click the **Route** tab to select each vehicle and view the route for the selected vehicle.

14.6.3. Creating custom data sets with OptaWeb Vehicle Routing

There is a built-in demo data set consisting of a several large Belgian cities. If you want to have more demos available in the **Load demo** menu, you can prepare your own data sets.

Procedure

1. In OptaWeb Vehicle Routing, add a depot and one or more of visits by clicking on the map or using geosearch.

2. Click **Export** and save the file in the data set directory.

   NOTE

   The data set directory is the directory specified in the **app.demo.data-set-dir** property.

   If the application is running through the **runLocally.sh** script, the data set directory is set to **$HOME/.optaweb-vehicle-routing/dataset**.

   Otherwise, the property is taken from the **application.properties** file and defaults to **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-vehicle-routing/optaweb-vehicle-routing-standalone/target/local/dataset**.

   You can edit the **app.demo.data-set-dir** property to specify a diffent data directory.

3. Edit the YAML file and choose a unique name for the data set.

4. Restart the back end.

After you restart the back end, files in the data set directory appear in the **Load demo** menu.

14.6.4. Troubleshooting OptaWeb Vehicle Routing

If the OptaWeb Vehicle Routing behaves unexpectedly, follow this procedure to trouble-shoot.

Prerequisites

- OptaWeb Vehicle Routing is running and behaving unexpectedly.

Procedure

1. To identify issues, review the back-end terminal output log.

2. To resolve issues, remove the back-end database:

   a. Stop the back end by pressing Ctrl+C in the back-end terminal window.

b. Remove the **optaweb-vehicle-routing/optaweb-vehicle-routing-backend/local/db** directory.
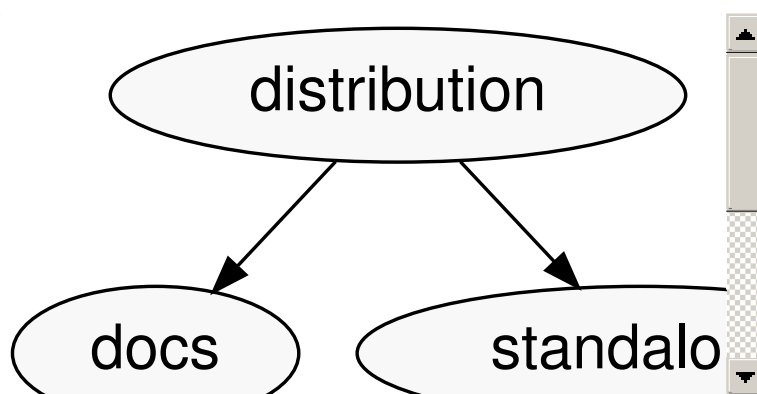
c. Restart OptaWeb Vehicle Routing.

14.7. OPTAWEB VEHICLE ROUTING DEVELOPMENT GUIDE

This section describes how to configure and run the back-end and front-end modules in development mode.

14.7.1. OptaWeb Vehicle Routing project structure

The OptaWeb Vehicle Routing project is a multi-module Maven project.

Figure 14.1. Module dependency tree diagram

The back-end and front-end modules are at the bottom of the module tree. These modules contain the application source code.

The standalone module is an assembly module that combines the back end and front end into a single executable JAR file.

The distribution module represents the final assembly step. It takes the standalone application and the documentation and wraps them in an archive that is easy to distribute.

The back end and front end are separate projects that you can build and deploy separately. In fact, they are written in completely different languages and built with different tools. Both projects have tools that provide a modern developer experience with fast turn-around between code changes and the running application.

The next sections describe how to run both back-end and front-end projects in development mode.

14.7.2. The OptaWeb Vehicle Routing back-end module

The back-end module contains a server-side application that uses Red Hat build of OptaPlanner to optimize vehicle routes. Optimization is a CPU-intensive computation that must avoid any I/O operations in order to perform to its full potential. Because one of the chief objectives is to minimize travel cost, either time or distance, OptaWeb Vehicle Routing keeps the travel cost information in RAM memory. While solving, OptaPlanner needs to know the travel cost between every pair of locations entered by the user. This information is stored in a structure called the *distance matrix*.

When you enter a new location, OptaWeb Vehicle Routing calculates the travel cost between the new location and every other location that has been entered so far, and stores the travel cost in the distance matrix. The travel cost calculation is performed by the GraphHopper routing engine.

The back-end module implements the following additional functionality:

- Persistence

- WebSocket connection for the front end

- Data set loading, export, and import

To learn more about the back-end code architecture, see Section 14.8, "OptaWeb Vehicle Routing back-end architecture".

The next sections describe how to configure and run the back end in development mode.

14.7.2.1. Running the OptaWeb Vehicle Routing back-end module

You can run the back-end module in Quarkus development mode.

Prerequisites

- OptaWeb Vehicle Routing has been configured as described in Section 14.4, "Configure and run OptaWeb Vehicle Routing manually".

Procedure

1. Change directory to **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-vehicle-routing/optaweb-vehicle-routing-backend**.

2. To run the back end in development mode, enter the following command:

   ```
   mvn compile quarkus:dev
   ```

14.7.2.2. Running the OptaWeb Vehicle Routing back-end module from IntelliJ IDEA Ultimate

You can use IntelliJ IDEA Ulitmate to run the OptaWeb Vehicle Routing back-end module to make it easier to develop your project. IntelliJ IDEA Ultimate includes a Quarkus plug-in that automatically creates run configurations for modules that use the Quarkus framework.

Procedure

Use the **optaweb-vehicle-routing-backend** run configuration to run the back end.

Additional resources

For more information, see Run the Quarkus application.

14.7.2.3. Quarkus development mode

In development mode, if there are changes to the back-end source code or configuration and you refresh the browser tab where the front end runs, the back-end automatically restarts.

Learn more about Quarkus development mode.

14.7.2.4. Changing OptaWeb Vehicle Routing back-end module system property values

You can temporarily or permanently override the default system property values of the OptaWeb Vehicle Routing back-end module.

The OptaWeb Vehicle Routing back-end module system properties are stored in the **/src/main/resources/application.properties** file. This file is under version control. Use it to permanently store default configuration property values and to define Quarkus profiles.

Prerequisites

- The OptaWeb Vehicle Routing starter application has been downloaded and extracted. For information, see Section 14.2, "Download and build the OptaWeb Vehicle Routing deployment files".

Procedure

- To temporarily override a default system property value, include the **-D<PROPERTY>= <VALUE>** argument when you run the **mvn** or **java** command, where **<PROPERTY>** is the name of the property that you want to change and **<VALUE>** is the value that you want to temporarily assign to that property. The following example shows how to temporarily change the value of the **quarkus.http.port** system property to **8181** when you use Maven to compile a Quarkus project in **dev** mode:

  ```
  mvn compile quarkus:dev -Dquarkus.http.port=8181
  ```

  This temporarily changes the value of the property stored in the **/src/main/resources/application.properties** file.

- To change a configuration value permanently, for example to store a configuration that is specific to your development environment, copy the contents of the **env-example** file to the **optaweb-vehicle-routing-backend**/**.env** file.
  This file is excluded from version control and therefore it does not exist when you clone the repository. You can make changes in the **.env** file without affecting the Git working tree.

Additional resources

For a complete list of OptaWeb Vehicle Routing configuration properties, see Section 14.9, "OptaWeb Vehicle Routing back-end configuration properties".

14.7.2.5. OptaWeb Vehicle Routing backend logging

OptaWeb Vehicle Routing uses the SLF4J API and Logback as the logging framework. For more information, see Quarkus - Configuring Logging.

14.7.3. Working with the OptaWeb Vehicle Routing front-end module

The front-end project was bootstrapped with Create React App. Create React App provides a number of scripts and dependencies that help with development and with building the application for production.

Prerequisites

- The OptaWeb Vehicle Routing starter application has been downloaded and extracted. For information, see Section 14.2, "Download and build the OptaWeb Vehicle Routing deployment files".

Procedure

1. On Fedora, enter the following command to set up the development environment:

   ```
   sudo dnf install npm
   ```

   See Downloading and installing Node.js and npm for more information about installing npm.

2. Change directory to **rhdm-7.11.0-decision-services-quickstarts/optaweb-8.5.0.Final-redhat-00004/optaweb-vehicle-routing/optaweb-vehicle-routing-frontend**.

3. Install **npm** dependencies:

   ```
   npm install
   ```

   Unlike Maven, the **npm** package manager installs dependencies in **node_modules** under the project directory and does that only when you execute **npm install**. Whenever the dependencies listed in **package.json** change, for example when you pull changes to the master branch, you must execute **npm install** before you run the development server.

4. Enter the following command to run the development server:

   ```
   npm start
   ```

5. If it does not open automatically, open **http://localhost:3000/** in a web browser.
   By default, the **npm start** command attempts to open this URL in your default browser.

   > **NOTE**
   >
   > If you do not want the **npm start** command to open a new browser tab each time you run it, export the **BROWSER=none** environment variable. You can use **.env.local** file to make this preference permanent. To do that, enter the following command:
   >
   > ```
   > echo BROWSER=none >> .env.local
   > ```

   The browser refreshes the page whenever you make changes in the front-end source code. The development server process running in the terminal picks up the changes as well and prints compilation and lint errors to the console.

6. Enter the following command to run tests:

   ```
   npm test
   ```

7. Change the value of the **REACT_APP_BACKEND_URL** environment variable to specify the location of the back-end project to be used by **npm** when you execute **npm start** or **npm run build**, for example:

   ```
   REACT_APP_BACKEND_URL=http://10.0.0.123:8081
   ```

> **NOTE**
>
> Environment variables are hard coded inside the JavaScript bundle during the **npm** build process, so you must specify the back-end location before you build and deploy the front end.

To learn more about the React environment variables, see *Adding Custom Environment Variables*.

8. To build the front end, enter one of the following commands:

```
./mvnw install
```

```
mvn install
```

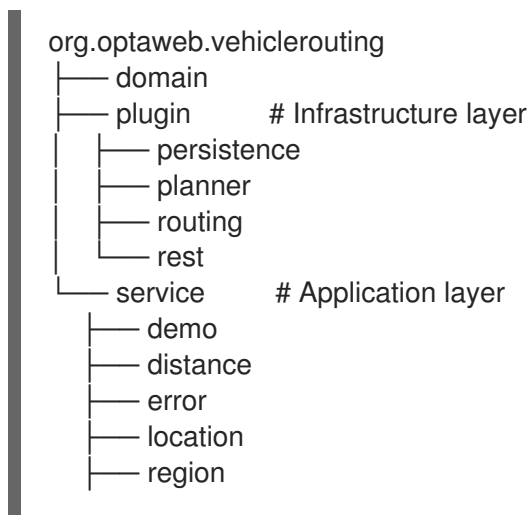## 14.8. OPTAWEB VEHICLE ROUTING BACK-END ARCHITECTURE

Domain model and use cases are essential for the application. The OptaWeb Vehicle Routing domain model is at the center of the architecture and is surround by the application layer that embeds use cases. Functions such as route optimization, distance calculation, persistence, and network communication are considered implementation details and are placed at the outermost layer of the architecture.
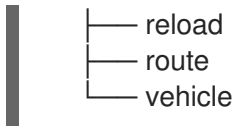
Figure 14.2. Diagram of application layers
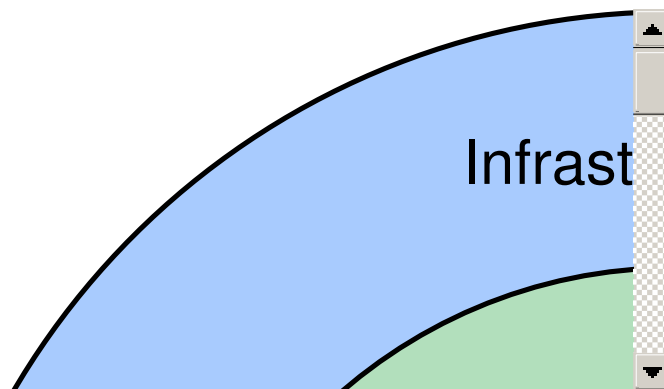
14.8.1. Code organization

The back-end code is organized in three layers, illustrated in the preceding graphic.

```
org.optaweb.vehiclerouting
├── domain
├── plugin        # Infrastructure layer
│   ├── persistence
│   ├── planner
│   ├── routing
│   └── rest
└── service       # Application layer
    ├── demo
    ├── distance
    ├── error
    ├── location
    ├── region
```

```
    ├── reload
    ├── route
    └── vehicle
```

The **service** package contains the application layer that implements use cases. The **plugin** package contains the infrastructure layer.

Code in each layer is further organized by function. This means that each service or plug-in has its own package.

14.8.2. Dependency rules

Compile-time dependencies are only allowed to point from outer layers towards the center. Following this rule helps to keep the domain model independent of underlying frameworks and other implementation details and models the behavior of business entities more precisely. With presentation and persistence being pushed out to the periphery, it is easier to test the behavior of business entities and use cases.

The domain has no dependencies.

Services only depend on the domain. If a service needs to send a result (for example to the database or to the client), it uses an output boundary interface. Its implementation is injected by the contexts and dependency injection (CDI) container.

Plug-ins depend on services in two ways. First, they invoke services based on events such as a user input or a route update coming from the optimization engine. Services are injected into plug-ins which moves the burden of their construction and dependency resolution to the IoC container. Second, plug-ins implement service output boundary interfaces to handle use case results, for example persisting changes to the database or sending a response to the web UI.

14.8.3. The domain package

The **domain** package contains *business objects* that model the domain of this project, for example **Location**, **Vehicle**, **Route**. These objects are strictly business-oriented and must not be influenced by any tools and frameworks, for example object-relational mapping tools and web service frameworks.

14.8.4. The service package

The **service** package contains classes that implement *use cases*. A use case describes something that you want to do, for example adding a new location, changing vehicle capacity, or finding coordinates for an address. The business rules that govern use cases are expressed using the domain objects.

Services often need to interact with plug-ins in the outer layer, such as persistence, web, and optimization. To satisfy the dependency rules between layers, the interaction between services and plug-ins is expressed in terms of interfaces that define the dependencies of a service. A plug-in can satisfy a dependency of a service by providing a bean that implements the boundary interface of the service. The CDI container creates an instance of the plug-in bean and injects it to the service at runtime. This is an example of the inversion of control principle.

14.8.5. The plugin package

The **plugin** package contains infrastructure functions such as optimization, persistence, routing, and network.

14.9. OPTAWEB VEHICLE ROUTING BACK-END CONFIGURATION PROPERTIES

You can set the OptaWeb Vehicle Routing application properties listed in the following table.

Property	Type	Example	Description
app.demo.data-set-dir	Relative or absolute path	**/home/user/.optaweb-vehicle-routing/dataset**	Custom data sets are loaded from this directory. Defaults to **local/dataset**.
app.persistence.h2-dir	Relative or absolute path	**/home/user/.optaweb-vehicle-routing/db**	The directory used by H2 to store the database file. Defaults to **local/db**.
app.region.country-codes	List of **ISO 3166-1 alpha-2** country codes	**US**, **GB,IE**, **DE,AT,CH**, may be empty	Restricts geosearch results.
app.routing.engine	Enumeration	**air**, **graphhopper**	Routing engine implementation. Defaults to **graphhopper**.
app.routing.gh-dir	Relative or absolute path	**/home/user/.optaweb-vehicle-routing/graphhopper**	The directory used by GraphHopper to store road network graphs. Defaults to **local/graphhopper**.
app.routing.osm-dir	Relative or absolute path	**/home/user/.optaweb-vehicle-routing/openstreetmap**	The directory that contains OSM files. Defaults to **local/openstreetmap**.
app.routing.osm-file	File name	**belgium-latest.osm.pbf**	Name of the OSM file to be loaded by GraphHopper. The file must be placed under **app.routing.osm-dir**.
optaplanner.solver.termination.spent-limit	**java.time.Duration**	**1m** **150s** **P2dT21h** (**PnDTnHnMn.nS**)	How long the solver should run after a location change occurs.

Property	Type	Example	Description
server.address	IP address or hostname	**10.0.0.123**, **my-vrp.geo-1.openshiftapps.com**	Network address to which to bind the server.
server.port	Port number	**4000**, **8081**	Server HTTP port.

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Monday, July 26, 2021.

APPENDIX B. CONTACT INFORMATION

Red Hat Decision Manager documentation team: brms-docs@redhat.com