fUML - Java

User-Guide

Author: Maximilian Hammer

Release Date: 2025-02-04

Document Version: 2.0

Version History:

Version	Release Date	Author(s)
1.0	2025-01-31	Maximilian Hammer
2.0	2025-02-04	Maximilian Hammer

Table of Contents

1 Prerequisites

This section contains software that is required for building and using fUML-Java and optional software.

1.1 Required software

This section covers required software that has to be set up on your system before being able to work with fUML-Java.

1.1.1 Java

A valid Java installation is required to run Eclipse (see section 1.1.2).

1.1.2 Eclipse IDE for Java Developers

Eclipse is the one and only IDE used with fUML-Java. The standard Java IDE version of Eclipse is suitable.

1.2 Optional software

This section covers recommended but optional software that is not necessarily required to run fUML-Java but will contribute to a faster and more productive workflow.

1.2.1 Eclipse Modeling Tools

Eclipse Modeling Tools is set of plugins for the Eclipse IDE, which provides tools for creating and processing UML models and profiles. There is a pre-packed release that already contains these plugins, see Eclipse Modeling Framework.

1.2.2 Papyrus

Eclipse Papyrus is a much more sophisticated software for UML modeling than the plain Eclipse Modeling Tools. It also provides graphical modeling functionality and is highly recommended as the main modeling tool when working with fUML-Java.

1.2.3 Acceleo

By now, fUML-Java does not provide any XMI-parsing functionality to execute UML models. As described in detail in section 4, there are two possible ways to create executable models in fUML-Java. On the one hand you can create an executable model using source code. As soon as models get slightly larger, this is a rather time-consuming task. That's why fUML-Java provides a code-generation facility, which produces compilable Java source code from UML models.

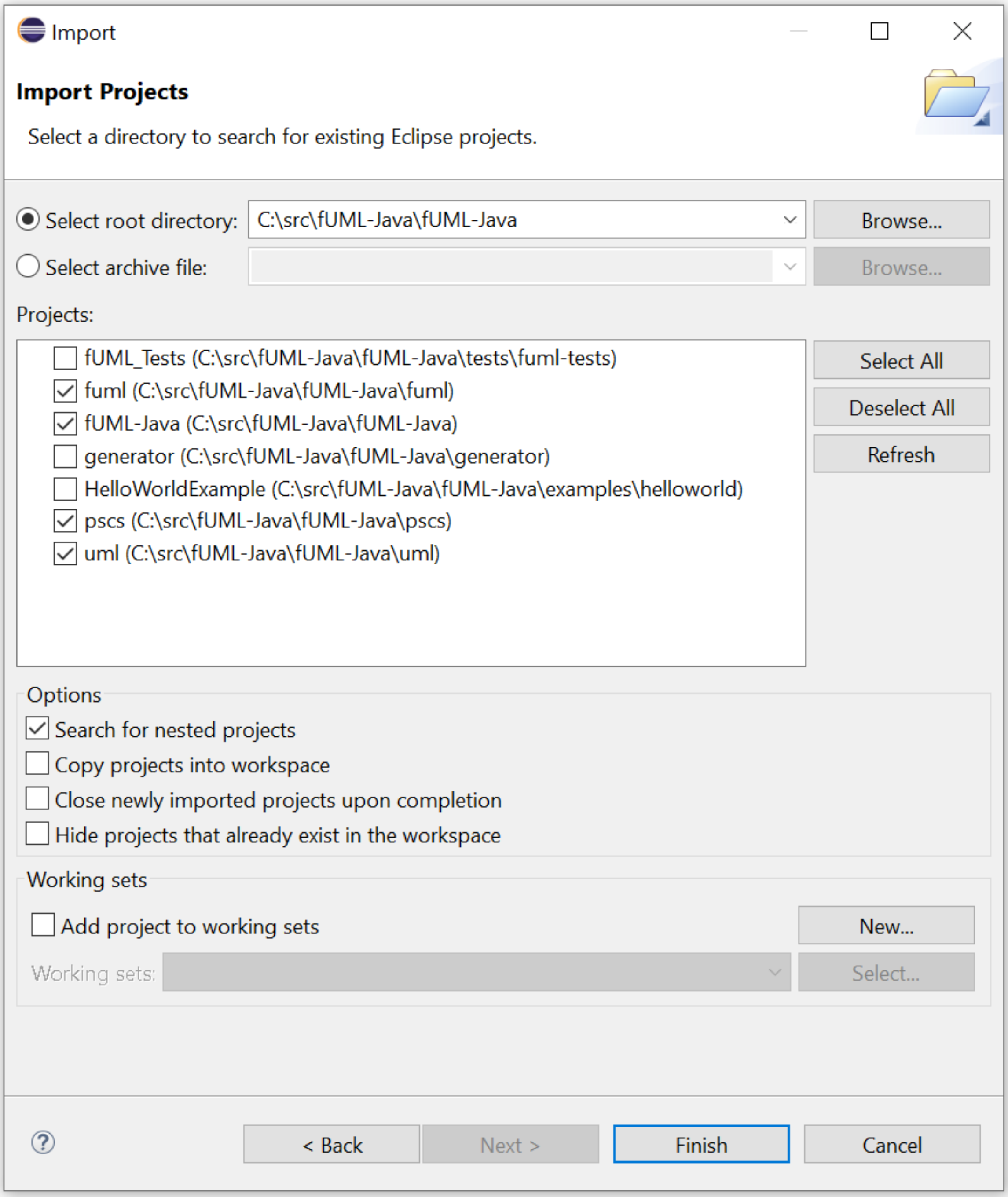
Acceleo is yet another Eclipse tool which provides functionalities to develop and run model-to-text transformation tools. As the code generator of fUML-Java was in fact built with Acceleo, a valid Acceleo installation within your Eclipse IDE is required if you want to use the generator.

2 Setup

This section covers additional steps that are necessary before building and using fUML-Java.

2.1 Setting up Eclipse

First, prepare the Eclipse IDE for the build process. Start Eclipse and choose a workspace directory. Import the fUML-Java root project called *fUML-Java*, which is located at "<fUML-Java-rootdir>\fUML-Java\".
Within the *"Import"* dialog, check the *"Search for nested projects"* option and also import the nested projects for UML, fUML and PSCS (for PSCS-compatibility) as shown below.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

(● Select root directory:) C:\src\fUML-Java\fUML-Java

○ Select archive file:

Projects:

- ☐ fUML_Tests (C:\src\fUML-Java\fUML-Java\tests\fuml-tests)
- ☑ fuml (C:\src\fUML-Java\fUML-Java\fuml)
- ☑ fUML-Java (C:\src\fUML-Java\fUML-Java)
- ☐ generator (C:\src\fUML-Java\fUML-Java\generator)
- ☐ HelloWorldExample (C:\src\fUML-Java\fUML-Java\examples\helloworld)
- ☑ pscs (C:\src\fUML-Java\fUML-Java\pscs)
- ☑ uml (C:\src\fUML-Java\fUML-Java\uml)

Options

- ☑ Search for nested projects
- ☐ Copy projects into workspace
- ☐ Close newly imported projects upon completion
- ☐ Hide projects that already exist in the workspace

Working sets

- ☐ Add project to working sets

Working sets:

3 Build

Eclipse will compile the *.java* files automatically in the background for you. No further user action is required.

NOTE: Instructions for external build, i.e., without any IDE, will be added in the future.

4 Usage

In fUML-Java there are two possible ways to create executable models:

- By creating a user-defined Java-project and implement the model to be executed by hand
- Automatically generating a model-specific Java-project from a user-defined UML model using the generator component

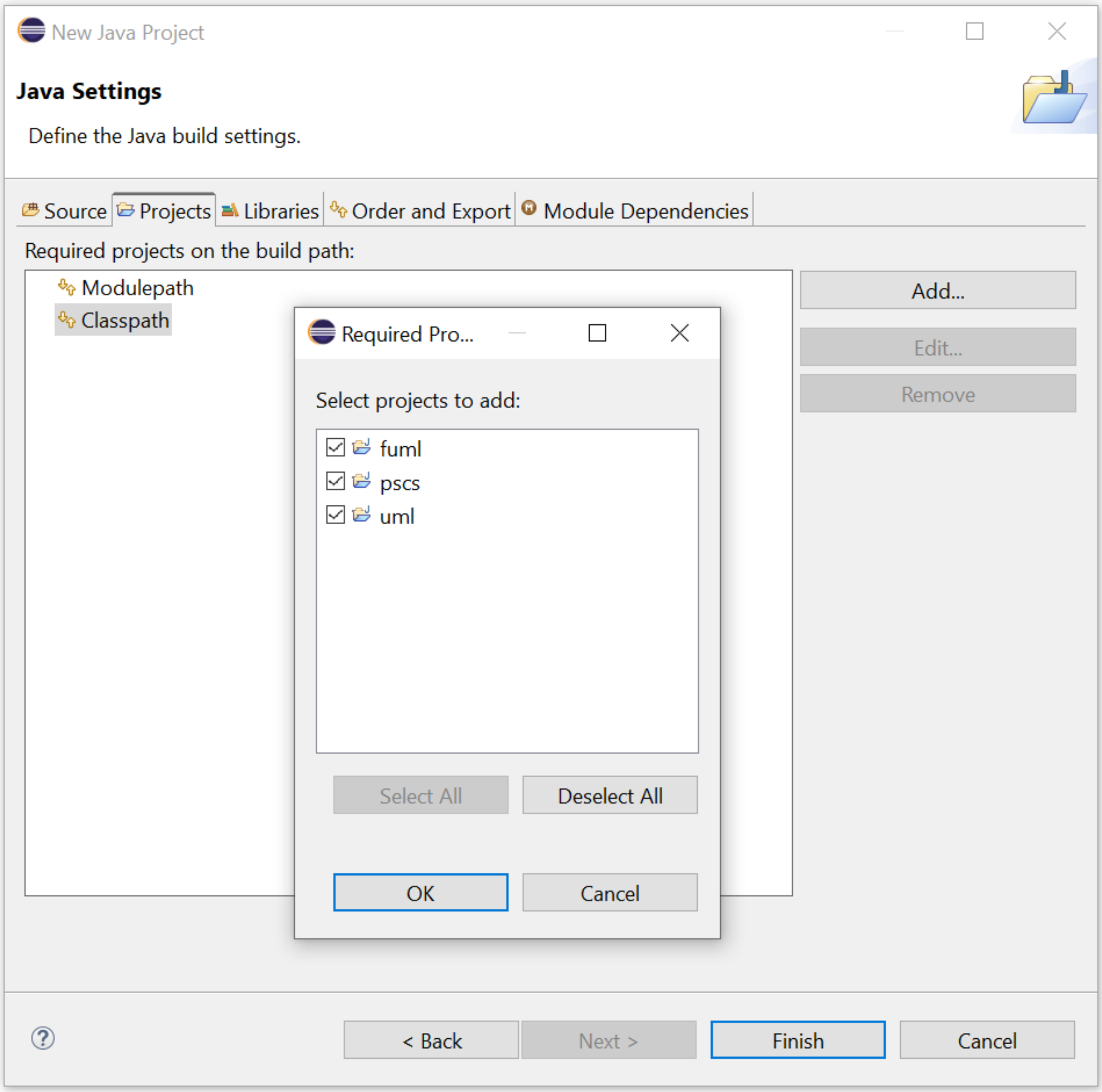
Both methods are described below.

4.1 Creating executable models using source code

NOTE: See "<fUML-Java-rootdir>\examples\helloworld" as a reference project for creating executable models directly from source code.

First, create a new Java-project within the fUML-Java root project. It is suggested to use the common directory "<fUML-Java-rootdir>\usersrc" for user-defined source-code projects. Within this directory, you can organize your projects in an arbitrary structure of nested subdirectories.

Upon project creation, don't forget to add the projects for UML, fUML and PSCS (for PSCS-compatibility) to your new project's classpath as shown below.



- [✓] fuml
- [✓] pscs
- [✓] uml

fUML-Java executable models require at least two specific classes to be implemented: A model-specific *Environment* class and a model-specific *Model* class. fUML-Java provides two abstract base classes that must be derived in user-defined executable models.

Create a <model-name>*Environment* class by deriving from class *utils.environmentfuml.Environment* (or *utils.environmentpscs.Environment* for PSCS-compatibility). A user-defined *Environment* class must at least provide:

- a static *instance()* method (as *Environment* classes are singleton classes)

See "<fUML-Java-rootdir>\examples\helloworld\src\examples\HelloWorldExample\HelloWorldExampleEnvironment.java" to see how a valid *instance*() method should be implemented.

Next, create a <model-name>*Model* class by deriving from class *utils.environment.InMemoryModel*. A user-defined *Model* class must at least provide:

- a static *instance()* method (as *Model* classes are singleton classes)
- a *void initializeInMemoryModel()* method (as this method should be called from the *instance()* method of the <model-name>*Model* class)

Additionally, the *Model* class should contain all of the desired model elements as public members (they should be public so that multiple models can potentially access each other's elements). Within the *initializeInMemoryModel()* method all model elements must be instantiated and all of their properties must be set and initialized.
See "<fUML-Java-rootdir>\examples\helloworld\src\examples\HelloWorldExample\HelloWorldExampleModel.java" to see how a valid *initializeInMemoryModel()* method should be implemented.

Last, create a main module (e.g., "<model-name>.java") that contains a main function. Within the main function, call <model -name>*Environment.instance().execute("<behavior-name>");* for each behavior you want to execute in subsequent order.

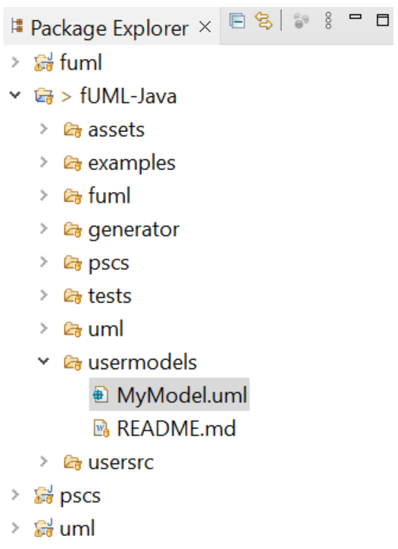
To execute the model, right click the project folder of your Java project and choose "*Run as*" → "*Java Application*".

4.2 Generating executable models from .uml models

A more convenient method to create executable models within fUML-Java is to automatically generate the corresponding Java-project from a UML model using the built-in code generator.

NOTE: To be able to use the code generator, the Acceleo plugin must be installed in your Eclipse IDE (see section 1.2.3).

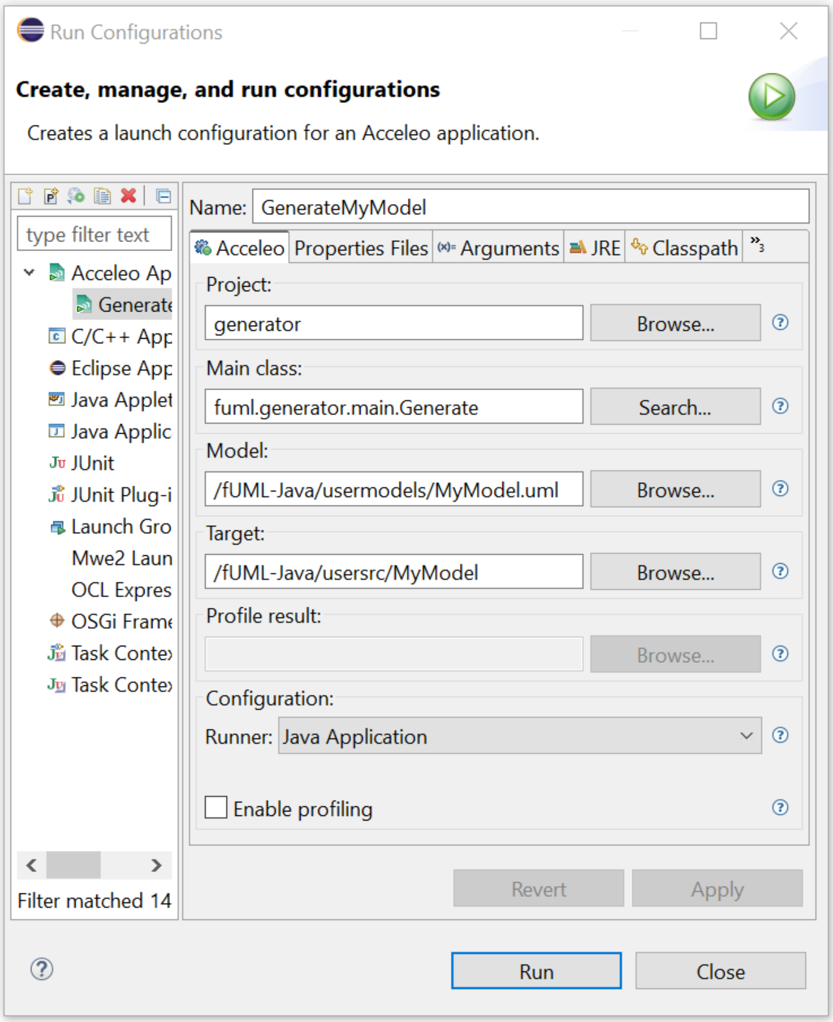
Create a new UML model with a modeling tool of your choice (Eclipse Papyrus is suggested, see section 1.2.2). It is suggested to store user-defined models in the common directory "<fUML-Java-rootdir>\usermodels". Within this directory, you can organize your models in an arbitrary structure of nested subdirectories.



Import the nested *generator* project within the fUML-Java root project. Navigate to file "\generator\src\fuml.generator.main\generate.mtl" in the package explorer. Right click *generate.mtl* and choose "*Run as*" → "*Run configurations...*". Configure a new Acceleo application as shown below:

- define a name for the generator application
- choose *fuml.generator.main.Generate* as the application's main class
- choose your model file
- choose a target directory for the generated source code ("<fUML-Java-rootdir>\usersrc\<model-name>" is suggested)

Run the generator application.


Run Configurations

Create, manage, and run configurations

Creates a launch configuration for an Acceleo application.

Name: GenerateMyModel

Project: generator

Main class: fuml.generator.main.Generate

Model: /fUML-Java/usermodels/MyModel.uml

Target: /fUML-Java/usersrc/MyModel

Profile result:

Configuration:

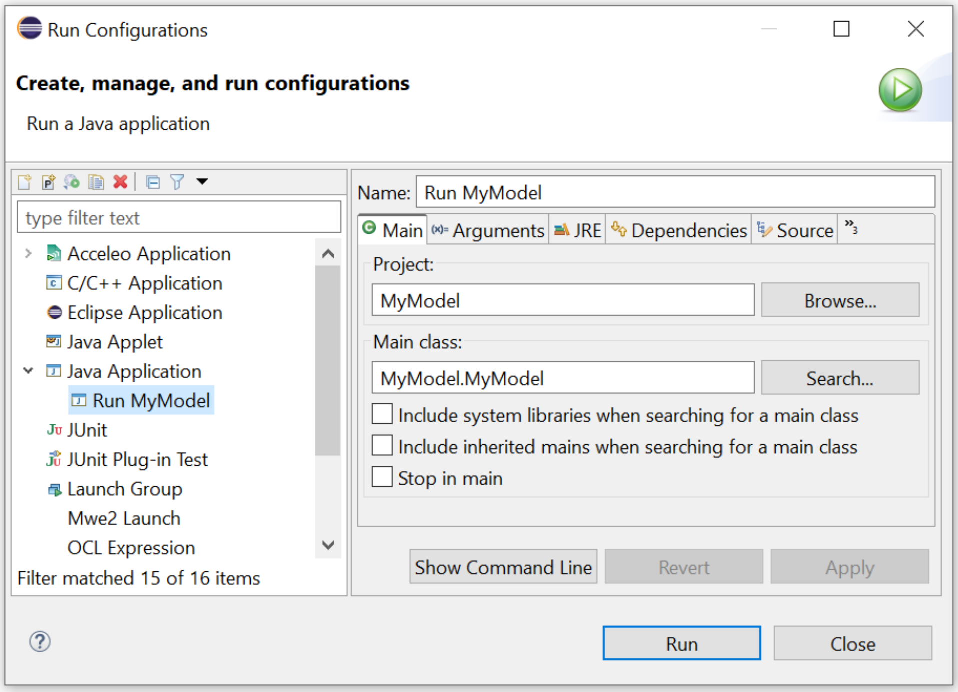
Runner: Java Application

Enable profiling

Now import the newly generated Java-project within the fUML-Jave root project. To execute the generated model, select the corresponding project folder. Right click and choose "*Run as*" → "*Run configurations*…". Create a new Java application as shown below:

- define a name for the Java application
- choose <*model-name*>.<*model-name*> as the application's main class
- Within the *Arguments* tab, provide the behaviors that should be executed in subsequent order as *Program arguments* in the form of: <behavior-name> [<behavior-name> <behavior-name> <behavior-name> <...>]

Click "*Run*" to launch the application.



Run Configurations

Create, manage, and run configurations

Run a Java application

type filter text

- Acceleo Application
- C/C++ Application
- Eclipse Application
- Java Applet
- Java Application
  - Run MyModel
- JUnit
- JUnit Plug-in Test
- Launch Group
- Mwe2 Launch
- OCL Expression

Filter matched 15 of 16 items

Name: Run MyModel

Main | Arguments | JRE | Dependencies | Source | »3

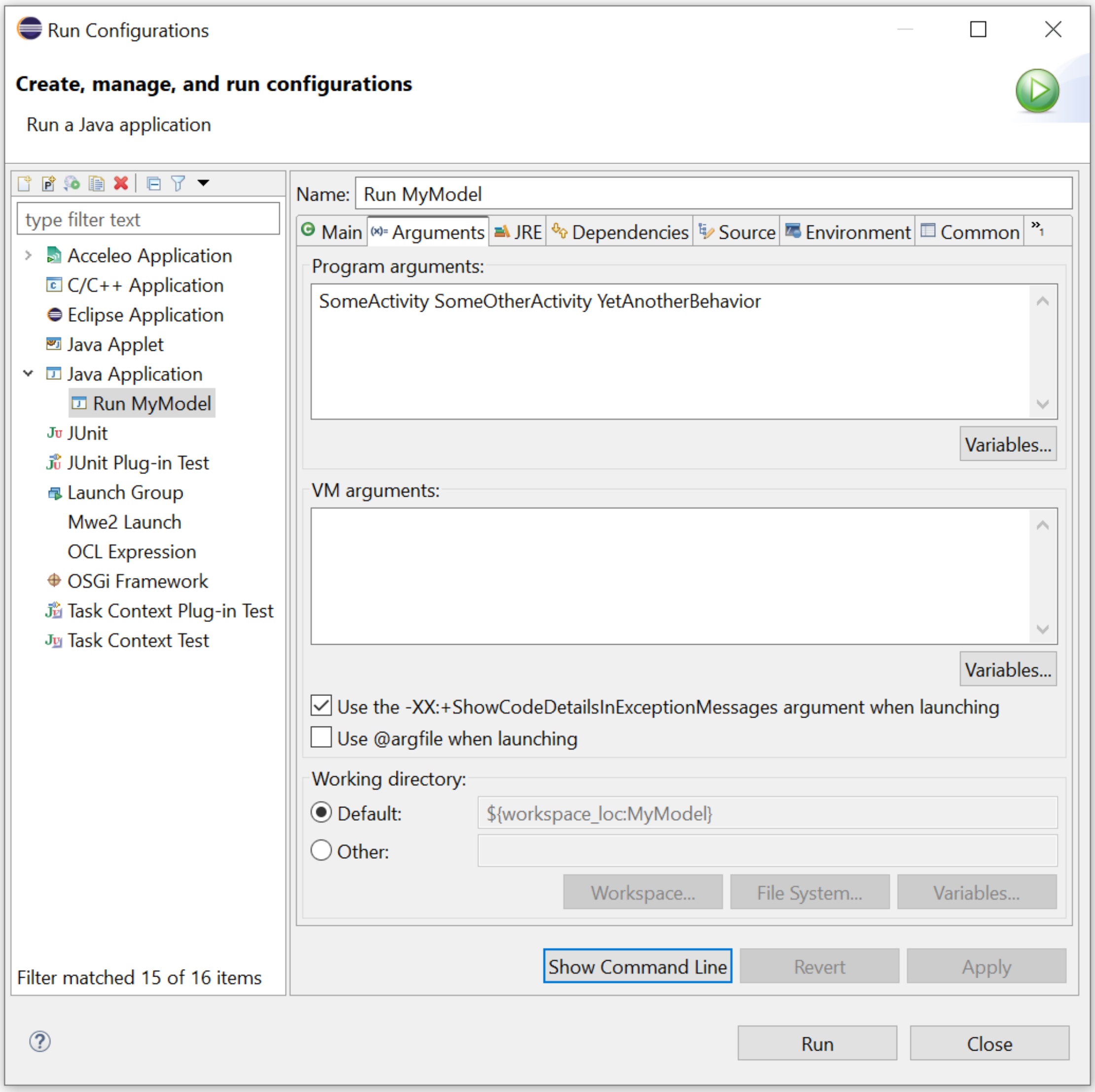
Project: MyModel [Browse...]

Main class: MyModel.MyModel [Search...]

- [ ] Include system libraries when searching for a main class
- [ ] Include inherited mains when searching for a main class
- [ ] Stop in main

Show Command Line | Revert | Apply

Run | Close



Run Configurations

Create, manage, and run configurations

Run a Java application

type filter text

- Acceleo Application
- C/C++ Application
- Eclipse Application
- Java Applet
- Java Application
  - Run MyModel
- JUnit
- JUnit Plug-in Test
- Launch Group
- Mwe2 Launch
- OCL Expression
- OSGi Framework
- Task Context Plug-in Test
- Task Context Test

Filter matched 15 of 16 items

Name: Run MyModel

Main | Arguments | JRE | Dependencies | Source | Environment | Common | »1

Program arguments:

```
SomeActivity SomeOtherActivity YetAnotherBehavior
```

[Variables...]

VM arguments:

[Variables...]

- [✓] Use the -XX:+ShowCodeDetailsInExceptionMessages argument when launching
- [ ] Use @argfile when launching

Working directory:

[✓ Default:] ${workspace_loc:MyModel}

Other:

Workspace... | File System... | Variables...

Show Command Line | Revert | Apply

Run | Close