

1.1 The FABLES – Java Bridge

Fables is a Java-oriented language, and these languages have several connections. With a minimal modification Fables is able to call any native Java code, and the data of the simulation may be accessed and/or manipulated from the context of the Java code.

The following section contains a step-by-step tutorial that demonstrates this feature through one of the built-in example models, Ants, which is an implementation of the simple one-dimensional random walk problem.

1.1.1 Creating a Java class

1. First we have to create the [Ants](#) example project. Click on *New* → *Fables Example Project*, and select the Ants model.
2. The model created is the following one:

```
model Ants {
    param antNum = 100, worldSize = 100;

    norm(x) = x mod worldSize;

    class Ant {
        var pos ;

        move( x ) = pos := norm( pos + x ) ;

        schedule Stepper cyclic 1 {
            1 : move( discreteUniform(-1, 0, 1) ) ;
        }
    };

    startUp() {
        for each i in [1..antNum] do
            create Ant[ pos := worldSize div 2 ] ;
    };
};
```

3. After the first compilation, Fables creates interfaces for the model and for all of the declared agents (these interfaces are placed into the [Import](#) directory). In this case, an interface is generated for the model ([AntsInterface](#)), and another one is generated for the agents ([AntInterface](#)).

These interfaces provide a convenient way to access any part of the model. As a general rule, the interfaces contain:

- A getter/setter method for each variable, constant or parameter (e.g. [get_x\(\)](#), [set_x\(\)](#) for a variable named [x](#)).
- A delegate call for each function (e.g. [call_x\(\)](#) for a function named [x](#)).
- Models contain functions returning a list for each of their declared classes containing all of the existing agent instances. Moreover, they allow access to simulation events ([setup\(\)](#), [begin\(\)](#), [stop\(\)](#), [pause\(\)](#)), and to the native Repast scheduler.

For the current model, the [AntInterface](#) generated for the agents is as follows:

```
package Import;

public interface AntInterface {
    public int get_pos(); // Variable pos
    public void set_pos(int pos);

    public void call_move( int x ); // Function move( x )
}
```

and the `AntsInterface` generated for the model is:

```
package Import;

import java.util.List;
import fables.runtime.FablesSchedule;

public interface AntsInterface {
    public int get_antNum(); // Constant antNum
    public void set_antNum(int antNum);

    public int get_worldSize(); // Constant worldSize
    public void set_worldSize(int worldSize);

    public int call_norm(int x); // Function norm( x )

    public List<AntInterface> get_AntList(); // The list of Ant agents

    public void setup(); // Simulation components
    public void begin();
    public void pause();
    public void stop();
    public FablesSchedule getSchedule();
}
```

4. We have to create a Java class to write some Java code that will be used from our Fables model. Native Java code has to be placed into the `Import` directory as well, which is considered as a Java package.

Right-click on the `Import` directory and select *New* → *Other...* from the context menu. You'll notice the *IME* has a full support for *Java* development.

- a. Create a new *Java* class by selecting *Class* from the list of available items (it is the second element in the list) and click on *Next*.

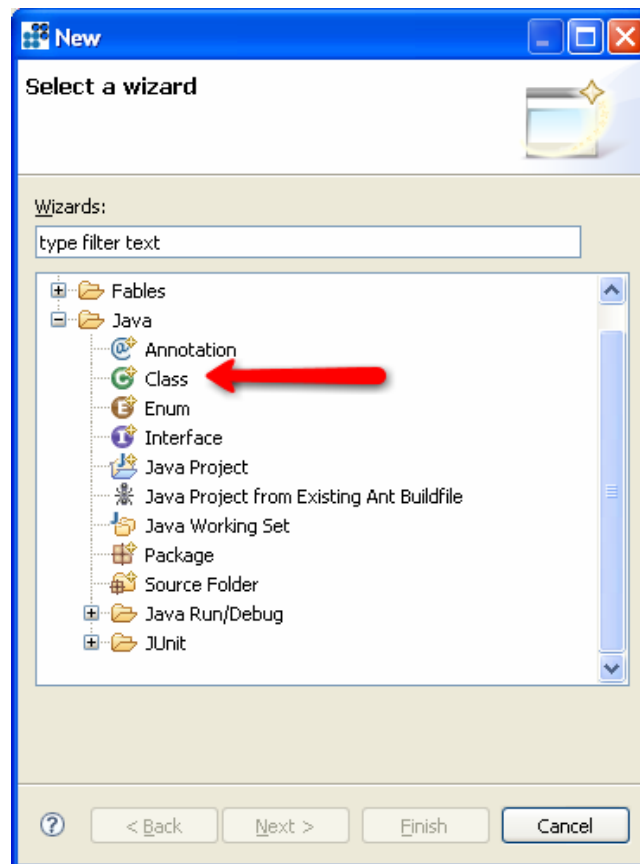


Figure 1 - Creating a new Java class, Step 1

b. Provide the necessary information for the new *Java* class on the next dialogue (see figure Figure 2):

- **Source Folder:** *Ants*

This value should always be the name of the current project.

- **Package:** *Import*

This value should always be *Import*. If the *Java* source file is misplaced to somewhere else, it won't be available from the *Fables* source!

- **Name:** the name of the new class. In the example we'll go on with a new class named *Statistics*.

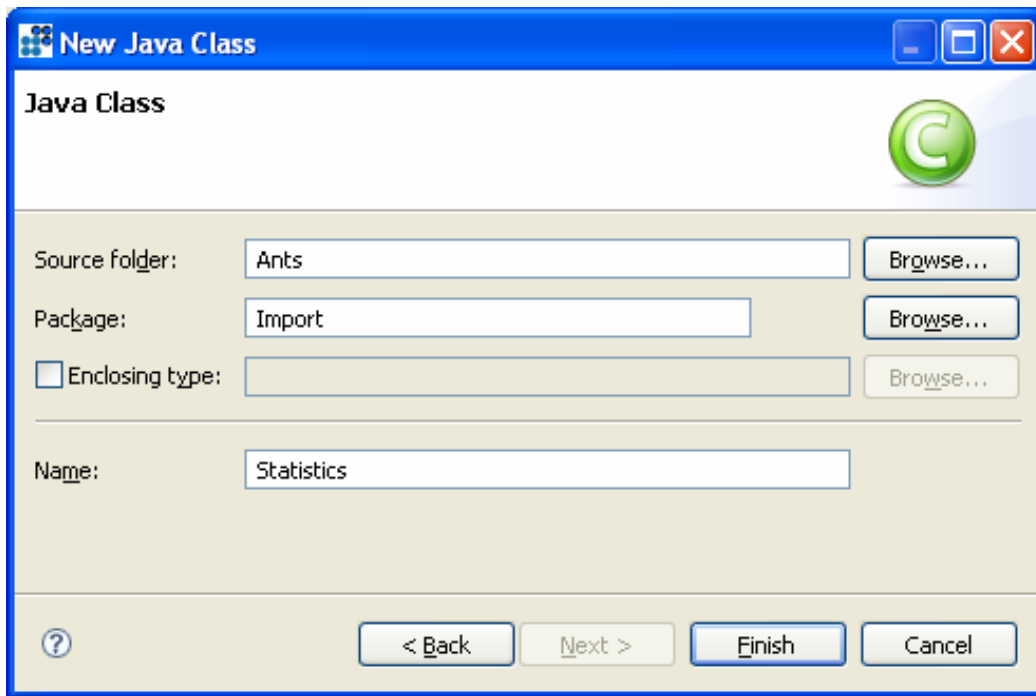


Figure 2 - Creating a new Java class, Step 2

Note: This is not an actual screenshot; several unnecessary settings are edited out from it. In *Fables IME* there are some other options that have no importance in this example.

c. When done, press *Finish*.

Notice that the workbench changes. A new *Java Editor* appears containing a minimal *Java* source file describing the new *Statistics* class. Also, the Outline changes and now contains the elements of the *Java* source file.

5. When creating a *Java* class that is used from *Fables*, the *Java* class has to fulfill some minimal requirements. The first important one is that the class has to contain a constructor that accepts a model interface as its parameter. When the class is instantiated, this reference will contain the running *Fables* model:

```
package Import;
import utils.FablesType;

public class Statistics {
    private final AntsInterface model;

    public Statistics(final AntsInterface model) {
        this.model = model;
    }
}
```

Assigning the model reference to a class variable ensures that the model may be accessed from each of the class' functions.

6. In this example, we are going to evaluate the standard deviation of ant positions.

We create a function that returns the mean of agent positions (this is only a utility function, and we will not contribute it to Fables), and a function that returns the standard deviation of ant positions. Create the following Java functions in the `Statistics` class:

```
private double meanOfAgentPositions() { // non-visible
    final int N = model.get_antNum();
    double mean = 0;
    for (final AntInterface ant : model.get_AntList())
        mean += ant.get_pos();
    return mean / N;
}

@FablesType("real")
public double stdDev() {
    final int N = model.get_antNum();
    double mean = meanOfAgentPositions();
    double sum = 0.0;
    for (final AntInterface ant : model.get_AntList())
        sum += Math.pow(ant.get_pos() - mean, 2);
    return Math.sqrt(sum / ( N - 1 )) ;
}
```

To make a function visible from Fables, it has to be annotated with the `FablesType` annotation. This annotation may be found in the `utils` package, and accepts a `String` parameter that describes the return type of the function. It is required to allow the compiler to verify the semantics of the source code. For the primitive types there are built-in constant expressions to avoid typos, but for composite and agent types always a well-formed string has to be specified. The following table enumerates the usable expressions:

Function's return value	String	Built-in constant
boolean	"boolean"	<code>FablesType.BOOLEAN</code>
String	"string"	<code>FablesType.STRING</code>
int	"int"	<code>FablesType.INTEGER</code>
double	"real"	<code>FablesType.REAL</code>
void	"void"	<code>FablesType.VOID</code>
Schedule	"schedule"	<code>FablesType.SCHEDULE</code>
an array of <A>	"seq(A) "	-
IndexableSet<A>	"set(A) "	-
agent interface	"Agent "	-

Note Array and set types could be composed. For example, a 2 dimensional array of integers may be described with the `"seq(seq(int))"` string, and a set containing integer arrays with the `"set(seq(int))"` string. For set types, the functions must always return an instance of `utils.IndexableSet`.

Agents may be declared as return types with their own names used in the Fables source code. For instance, if the Fables source has a class declaration `class Agent { ... }`, the string `"Agent "` must be used. These functions must return a reference to an instance of the generated agent interface.

At the end of the section you can find some minor examples demonstrating the usage of these variables.

- After implementing the desired function we will make it accessible from Fables. In order to achieve this, only a simple import statement is required before the model declaration:

```
import Statistics; // Import statement

model Ants {
    param antNum = 100, worldSize = 100;
    ...
}
```

After importing a native Java class all of the functions annotated with `FablesType` become visible in the Fables source. The following new schedule in Fables prints out the standard deviation of agent positions in each simulation tick:

```
schedule cyclic 1 {
    1 : println( "Standard deviation of ant positions: " ++ stdDev() );
};
```

- As an additional example, we create a Java function that returns an agent whose position is one of the closest to the mean of agent positions. Add the following definition to the `Statistics` class:

```
@FablesType("Ant")
public AntInterface getAgentClosestToMean() {
    final double mean = meanOfAgentPositions();
    System.out.println("[Java] Mean of agents is: " + mean);

    AntInterface ret = model.get_AntList().get(0);
    double best = Math.abs(ret.get_pos() - mean);

    for (final AntInterface ant : model.get_AntList()) {
        final double delta = Math.abs(ant.get_pos() - mean);
        if ( delta < best) {
            ret = ant;
            best = delta;
        }
    }

    return ret;
}
```

The code above iterates through the agent instances (agent instances can be accessed through the `model` reference via the `get_<Agent name>List` functions), and searches for the first one whose position is the closest to the `mean` of positions (note that the function uses the standard output stream to print a message, and this message is displayed in the Fables Console during runtime).

- Extending the new schedule we created we can print out the agent whose position is to the closest to the mean of positions as well in each simulation step:

```
schedule cyclic 1 {
    1 : println("Standard deviation of ant positions: " ++ stdDev());
        println("Agent closest to the standard deviation: " ++
            getAgentClosestToMean());
};
```

1.1.2 Source Code of the Example

The Fables source:

```
import Statistics; // Import statement

model Ants {
  param antNum = 100, worldSize = 100;

  norm(x) = x mod worldSize;

  class Ant {
    var pos ;

    move( x ) = pos := norm( pos + x ) ;

    schedule Stepper cyclic 1 {
      1 : move( discreteUniform(-1, 0, 1) ) ;
    }
  };

  startUp() {
    for each i in [1..antNum] do
      create Ant[ pos := worldSize div 2 ] ;
  };

  schedule cyclic 1 {
    1 : println("Standard deviation of ant positions: " ++ stdDev());
      println("Agent closest to the standard deviation: " ++
        getAgentClosestToMean());
  };
};
```

The Statistics Java class:

```
package Import;

import utils.FablesType;

public class Statistics {
  private final AntsInterface model;

  public Statistics(final AntsInterface model) {
    this.model = model;
  }

  private double meanOfAgentPositions() { // Non-visible (no FablesType noted)
    final int N = model.get_antNum();
    double mean = 0;
    for (final AntInterface ant : model.get_AntList())
      mean += ant.get_pos();
    return mean / N;
  }

  @FablesType("real")
  public double stdDev() {
    final int N = model.get_antNum();
    double mean = meanOfAgentPositions();
    double sum = 0.0;
    for (final AntInterface ant : model.get_AntList())
      sum += Math.pow(ant.get_pos() - mean, 2);
    return Math.sqrt(sum / ( N - 1 ) );
  }

  @FablesType("Ant")
  public AntInterface getAgentClosestToMean() {
    final double mean = meanOfAgentPositions();
    System.out.println("[Java] Mean of agents is: " + mean);
  }
}
```

```

    AntInterface ret = model.get_AntList().get(0);
    double best = Math.abs(ret.get_pos() - mean);

    for (final AntInterface ant : model.get_AntList()) {
        final double delta = Math.abs(ant.get_pos() - mean);
        if ( delta < best) {
            ret = ant;
            best = delta;
        }
    }

    return ret;
}
}

```

1.1.3 Annotation Examples

These examples are created to show the correct way of using the `FablesType` annotations:

```

//@FablesType("boolean")           // Identical notations
@FablesType(FablesType.BOOLEAN)
public boolean retBoolean() { return true; }

//@FablesType("string")           // Identical notations
@FablesType(FablesType.STRING)
public String retString() { return "string"; }

//@FablesType("int")              // Identical notations
@FablesType(FablesType.INTEGER)
public int retInt() { return 0; }

//@FablesType("real")            // Identical notations
@FablesType(FablesType.REAL)
public double retReal() { return 0.0; }

//@FablesType("void")            // Identical notations
@FablesType(FablesType.VOID)
public void retVoid() { ; }

//@FablesType("schedule")        // Identical notations
@FablesType(FablesType.SCHEDULE)
public Schedule retSchedule() { return model.getSchedule(); }

@FablesType("Ant")
public AntInterface retAnt() { return model.get_AntList().get(0); }

@FablesType("seq(int)")
public int[] retSeqInt() { return new int[] {1, 2, 3}; }

@FablesType("set(int)")
public IndexableSet<Integer> retSetInt() {
    return new IndexableSet<Integer>(Arrays.asList(4,5,6));
}

@FablesType("set(seq(int))")
public IndexableSet<int[]> retSetSeqInt() {
    IndexableSet<int[]> ret = new IndexableSet<int[]>();
    ret.add(new int[] {1,2});
    ret.add(new int[] {2,3});
    ret.add(new int[] {3,4});
    return ret;
}

```