For loop

There are two forms of `for` loop and two forms of `while` loop in Java. We will begin with the easiest to use form of the `for` loop, the so-called "enhanced" `for` loop:

```
for( <element type> <name> : <collection> ) {
    <statements> //the loop body executed for each element
}
```

This form is used to iterate through arrays and collections. Whenever you need to do something with each element of a collection, we recommend to use this loop because it is more compact and easy to read, and is supported by all collection types (unlike index-based iteration).

Example: in an agent-based model, a firm's product portfolio is modeled as a replicated `products` object (remember that a replicated object is a collection). The following code goes through all products in the portfolio and kills those, whose estimated ROI is less than some allowed minimum:

```
for( Product p : products ) {
    if( p.getEstimatedROI() < minROI )
        p.kill();
}
```

Another example: the loop counts the number of sold seats in a movie theater. The seats are modeled as the `seats` Java array with the elements of the `boolean` type (`true` means sold):

```
boolean[] seats = new boolean[600]; //array declaration
…
int nsold = 0;
for( boolean sold : seats )
    if( sold )
        nsold++;
```

Note that if the body of the loop contains only one statement, the braces `{…}` can be dropped. In the code above, braces are dropped both in the `for` and the `if` statements.

Another more general form of the `for` loop is typically used for index-based iterations. In the header of the loop you can put the initialization code, for example, the declaration of the index variable, the condition that is tested before each iteration to determine whether the loop should continue, and the code to be executed after each iteration that can be used, say, to increment the index variable:

```
for( <initialization>; <continue condition>; <increment> ) {
    <statements>
}
```

The following loop finds all circles in a group of shapes and sets their fill color to red:

```
for( int i=0; i<group.size(); i++ ) { //index-based loop
    Object obj = group.get( i ); //get the i-th element of the group
    if( obj instanceof ShapeOval ) { //test if it is a ShapeOval - AnyLogic class for ovals
        ShapeOval ov = (ShapeOval)obj; //if it is oval, "cast" it to ShapeOval
```

```
            ov.setFillColor( red ); //set the fill color to red
        }
    }
```

As long as there is no other way to iterate through the `ShapeGroup` contents than accessing the shapes by index, only this form of loop is applicable here.

Many Process Modeling Library objects also offer index-based iterations. For example, this code goes through all agents in the queue from the end to the beginning and removes the first one that does not possess any resource units:

```
    for( int i=queue.size()-1; i>=0; i-- ) { //the index goes from queue.size()-1 down to 0
        Agent a = queue.get(i); //obtain the i-th agent
        if( a.resourceUnits().isEmpty() ) { //test
            queue.remove( a ); //remove the agent from the queue
            break; //exit the loop
        }
    }
```

Note that in this loop the index is decremented after each iteration, and correspondingly the continue condition tests if it has reached `0`. Once we have found the agent that satisfies our condition, we remove it and do not need to continue. The `break` statement is used to exit the loop immediately. If the agent is not found, the loop will finish in its natural way when the index after a certain iteration becomes `-1`.