

Chapter 5 – Creating and Using Jeroo Methods

The Jeroo language contains a set of fundamental methods that define the basic behaviors of every Jeroo. These include the basic actions (table 4.3) and the sensor methods (table 6.3). Some problems are such that it would be convenient if we could extend the basic behaviors of the Jeroos. The Jeroo language allows us to write programmer-defined Jeroo methods that extend the behavior of every Jeroo.

5.1 Creating and Using a Jeroo Method

The concepts of behavior and method were defined in section 2.7 and are repeated here. A **behavior** is an action that an object can take or a task that it can perform in response to a request from an external source. A **method** is a collection of statements that are written in some programming language to describe a specific behavior.

These definitions imply that the creation of a method is a two-part process. First, we need to define and name the new behavior. Second, we need to write the source code for the method.

Defining a Behavior

The first question we must ask is “How do I decide on a good behavior?” There is no fixed answer to this question, but there are some guidelines to follow.

1. Examine the high-level algorithm. Any complex, but well-defined, step is a candidate for a new behavior, especially if two or more Jeroos need to perform that step.
2. Examine the detailed-algorithm. Any sequence of steps that occur several times is a candidate for a new behavior.

These guidelines serve as a starting point, but experience is a good teacher. Examine your own programs and those of others. A good behavior has a very clear definition and is used more than once in the program.

Using a Jeroo Method

A Jeroo method is used just like any other method. In the main method, we send a message to a specific Jeroo object, requesting that Jeroo to perform the task associated with the method.

EXAMPLE

Let's have Jeroo Ali plant two rows of flowers, south and east of (5,5)

```
method main()  
{  
    Jeroo Ali = new Jeroo(5,5,8);  
  
    Ali.plantRowsOfFour();  
  
} //===== end method main() =====
```

5.2 Preconditions and Postconditions

We should always define a behavior carefully before we write the code for the corresponding method. A complete definition for a behavior must include a statement of the preconditions and the postconditions.

A precondition for a method is something that is assumed to be true before the method is invoked. The portion of the code that invokes the method is responsible for ensuring that all preconditions are satisfied before the method is invoked.

A postcondition for a method is something that is true after the method has been executed. The code within the method is responsible for ensuring that all postconditions are met.

The process of determining good preconditions and postconditions can be difficult, but it is easier if we remember a few characteristics of objects and methods.

1. All work is done by sending messages to objects.
2. Exactly one object executes a method in response to a message.
3. A method can modify the attributes of the object that executes the method, but cannot directly modify the attributes of any other object.
4. One method can send messages to several different objects, and those messages can lead to modifications in their receivers.

It is possible for one Jeroo method to use another, or even itself.

EXAMPLE

New Behaviors: Plant four flowers in a row

Plant two adjacent rows with four flowers per row

```
//*****
// This method plants four flowers in a row.
// starting at the current location
//*****
method plantFour()
{
    plant();    //-- one ---

    hop();
    plant();    //-- two ---

    hop();
    plant();    //-- three ---

    hop();
    plant();    //-- four ---

} //==== end method plantFour() ====

//*****
// This method plants two adjacent rows of flowers.
//*****
method plantRowsOfFour()
{
    //-- Plant first row ---
    plantFour();

    //-- Move into position for next row ---
    turn(RIGHT);
    hop();
    turn(RIGHT);

    //-- Plant second row (in opposite direction) ---
    plantFour();

} //==== end method plantRowsOfFour() =====
```

Writing a Jeroo Method

A Jeroo method contains the source code that describes what an arbitrary Jeroo needs to do to carry out the corresponding behavior. The form of a Jeroo method is shown in figure 5.1 for Jeroo's Java/C++/C#-style language. The *method_identifier* on the first line (the header line) is a name that the programmer chooses for the method. The name should indicate the corresponding behavior. The rules for creating an identifier for a method are the same as those given in section 4.3. As with the main method, we should indent every line between the opening and closing braces in every Jeroo method.

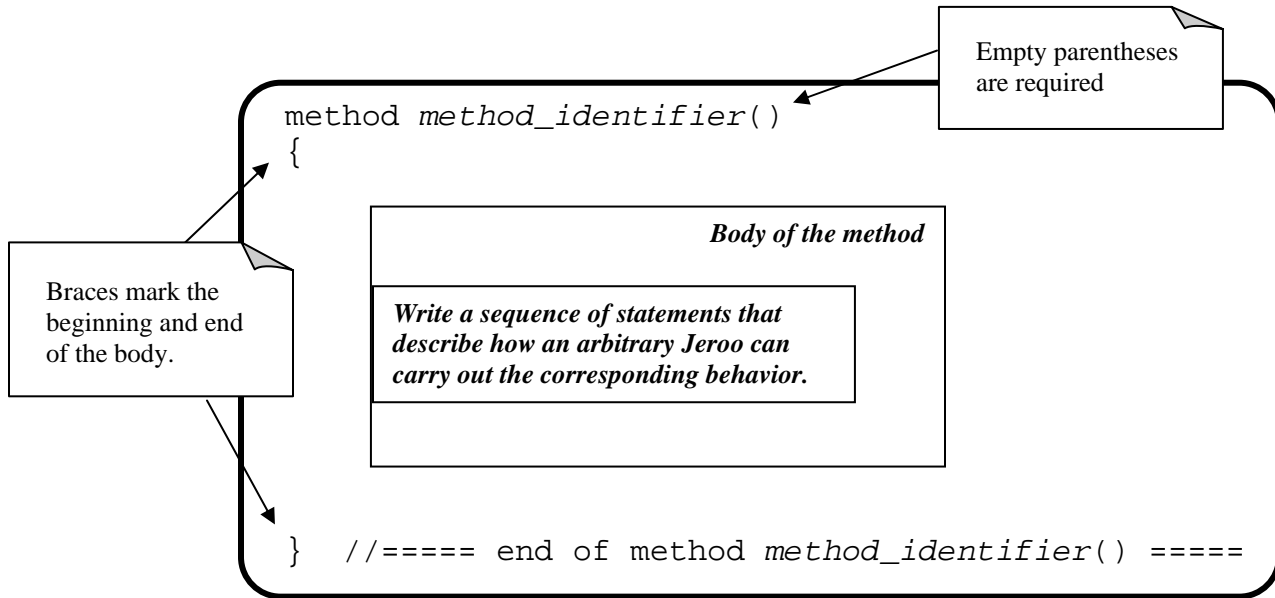


Figure 5.1 – Form of a Java-Style Jeroo Method

There is one important difference between a Jeroo method and the main method. Since a Jeroo method defines a behavior that applies to every Jeroo, we cannot send a message to a specific Jeroo. Instead, we simply write the name of the method. When the program is running, the messages are sent to the Jeroo that is performing the entire method.

EXAMPLE

New Behavior: Turn around

```
//*****
// This method makes a Jeroo turn 180 degrees
//*****
method turnAround()
{
    turn(LEFT);
    turn(LEFT);
} //==== end method turnAround() =====
```

Using the previous list of characteristics as a guide, we can use the following questions as a basis for writing preconditions and postconditions. When we are working with the Jeroo language, we only have to consider two classes of objects, Jeroos and Island cells. We need to consider how a method can change the attributes of the Jeroo object that executes the method. The Island cells are less evident because we cannot send them messages, but the Jeroo actions *pick*, *plant*, and *toss* can change the contents of a cell. Behind the scenes, the *pick*, *plant*, and *toss* methods send appropriate messages to the island cells.

Precondition Questions	Postcondition Questions
Do any of the attributes of the receiving object need to have specific values? Location Direction Flowers	How does this method affect the attributes of the receiving object? Location Direction Flowers
Are the contents of certain island cells important?	Have the contents of any island cells changed?

The preconditions and postconditions can be created rather informally, but the final versions should be stated in a comment block at the beginning of the source code for the method.

EXAMPLE

New Behavior: Plant four flowers in a row

```

//*****
// This method plants four flowers in a row.
// starting at the current location
//
//PRECONDITIONS
// 1. The three spaces directly ahead of
//    the Jeroo are clear.
// 2. The Jeroo has at least four flowers.
//POSTCONDITIONS
// 1. The Jeroo has planted four flowers,
//    starting at its current location and
//    proceeding straight ahead.
// 2. The Jeroo is standing on the last flower,
//    and facing in its original direction.
//*****

```