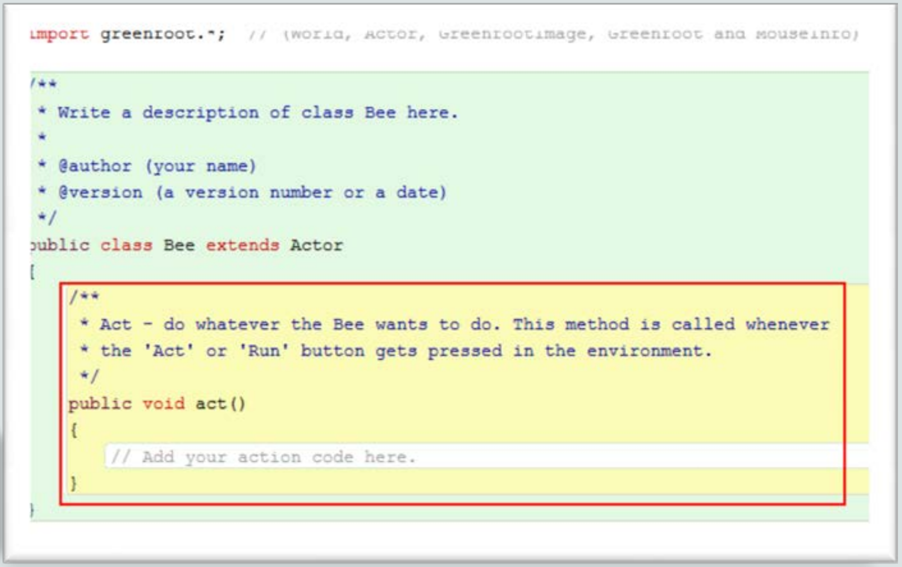
ORACLE® ACADEMY

Java Fundamentals

3-3
Source Code and Documentation



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Bee here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Bee extends Actor
{
    /**
     * Act - do whatever the Bee wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

ORACLE ACADEMY

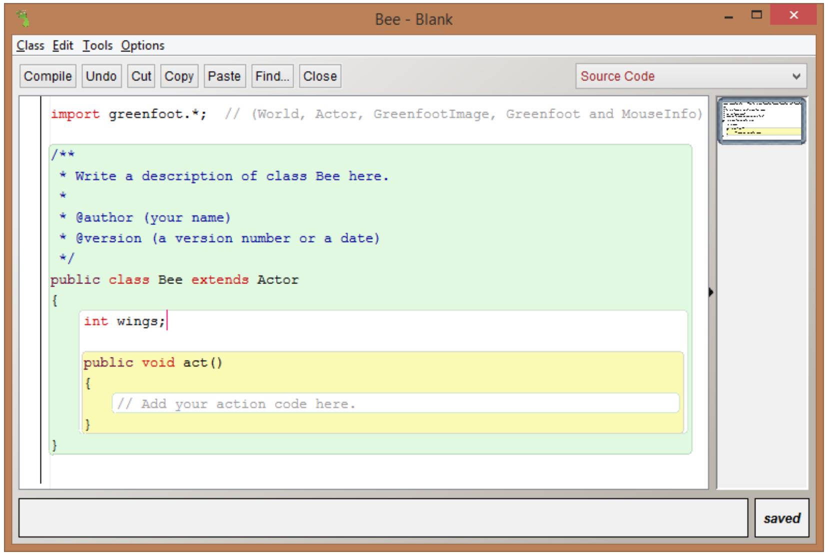
Objectives

This lesson covers the following objectives:

- Demonstrate source code changes to invoke methods programmatically
- Demonstrate source code changes to write an if decision statement
- Describe a method to display object orientation

Source Code

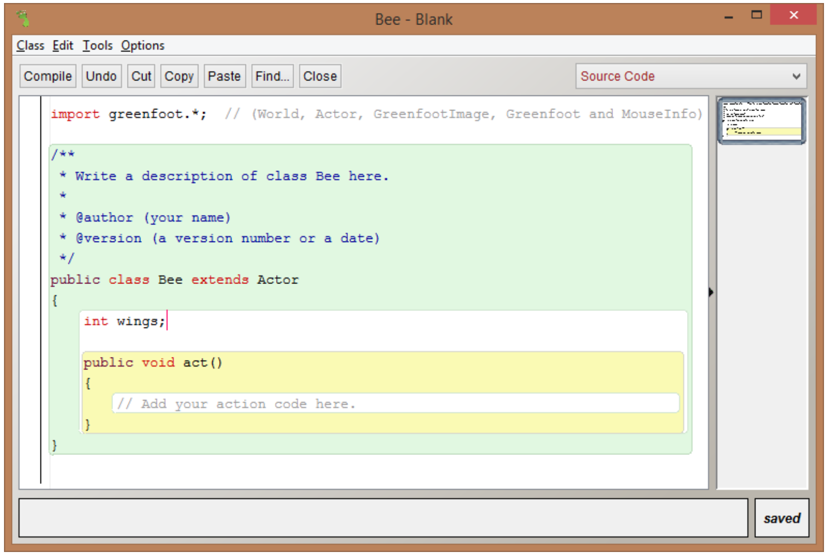
- Source code is the blueprint or map that defines how your objects and program function.
- It commands the objects in your scenario to move and interact.



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Bee here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Bee extends Actor
{
    int wings;

    public void act()
    {
        // Add your action code here.
    }
}
```

Code Editor

- Source code is managed in the Code editor.
- To view the Code editor, right click on any class in the environment, then select Open editor from the menu.



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Bee here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Bee extends Actor
{
    int wings;

    public void act()
    {
        // Add your action code here.
    }
}
```

Functions of the Code Editor

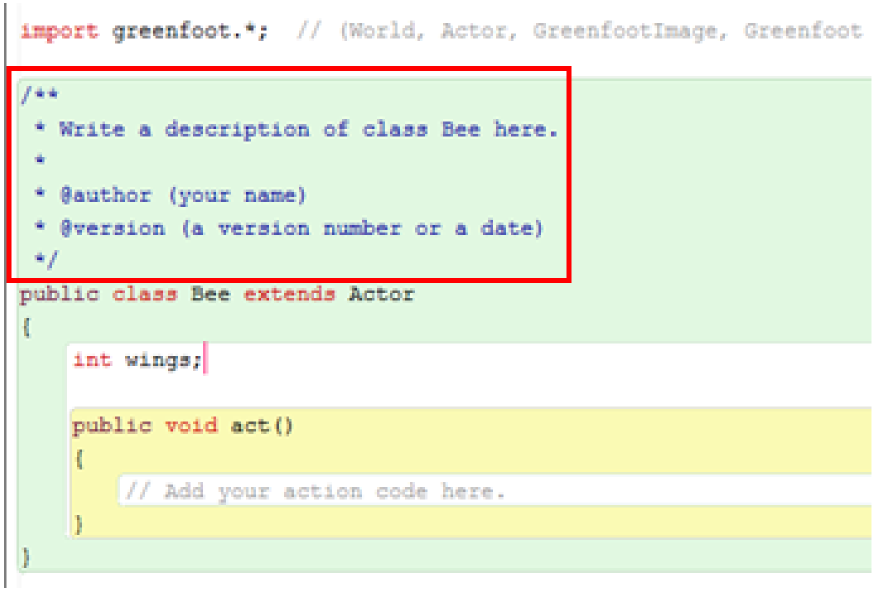
- In the Code editor, you can:
  - Write source code to program instances of the class to act.
  - Modify source code to change an instance's behavior.
  - Review the class's inherited methods and properties.
  - Review methods created specifically for the class by the programmer who wrote the source code.

Components of Source Code

1. Class Description
2. act() Method
3. Method Signature
4. Method Body
5. Comments
6. Documentation
7. Class Definition

Class Description

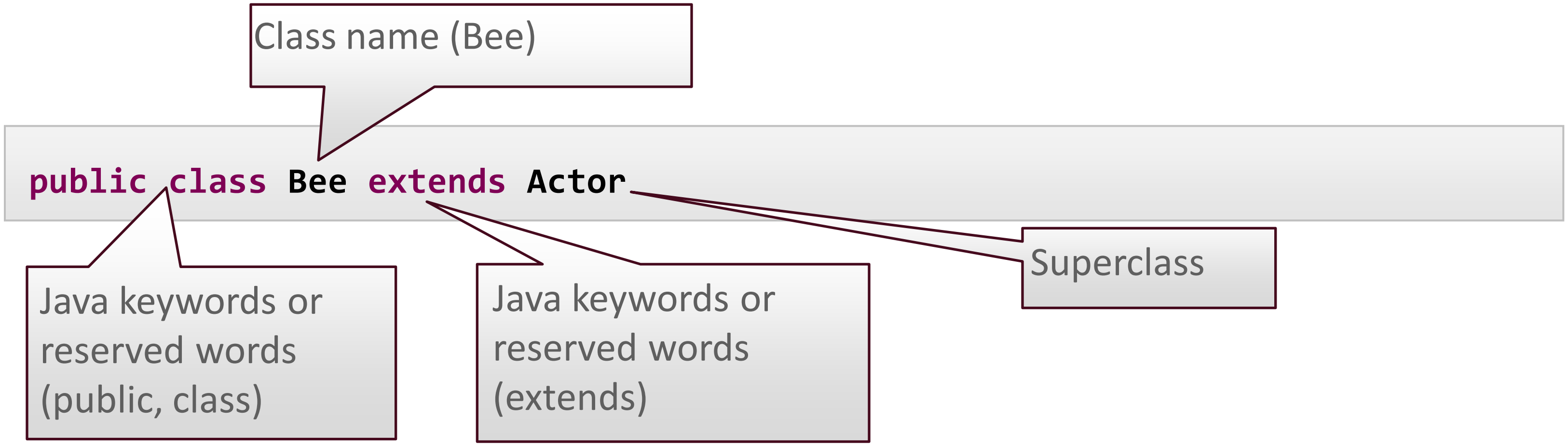
- The class description is a set of comments that can be modified to describe the class.
- This includes:
  - A description of what the class does.
  - The name of the person who authored the code.
  - The date the source code was last modified.



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot

/**
 * Write a description of class Bee here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Bee extends Actor
{
    int wings;

    public void act()
    {
        // Add your action code here.
    }
}
```

Class Definition Components

- The class definition includes:
  - Java keywords or reserved words.
  - The name of the class as defined by the programmer.
  - The name of the superclass that the subclass extends from.



Class name (Bee)

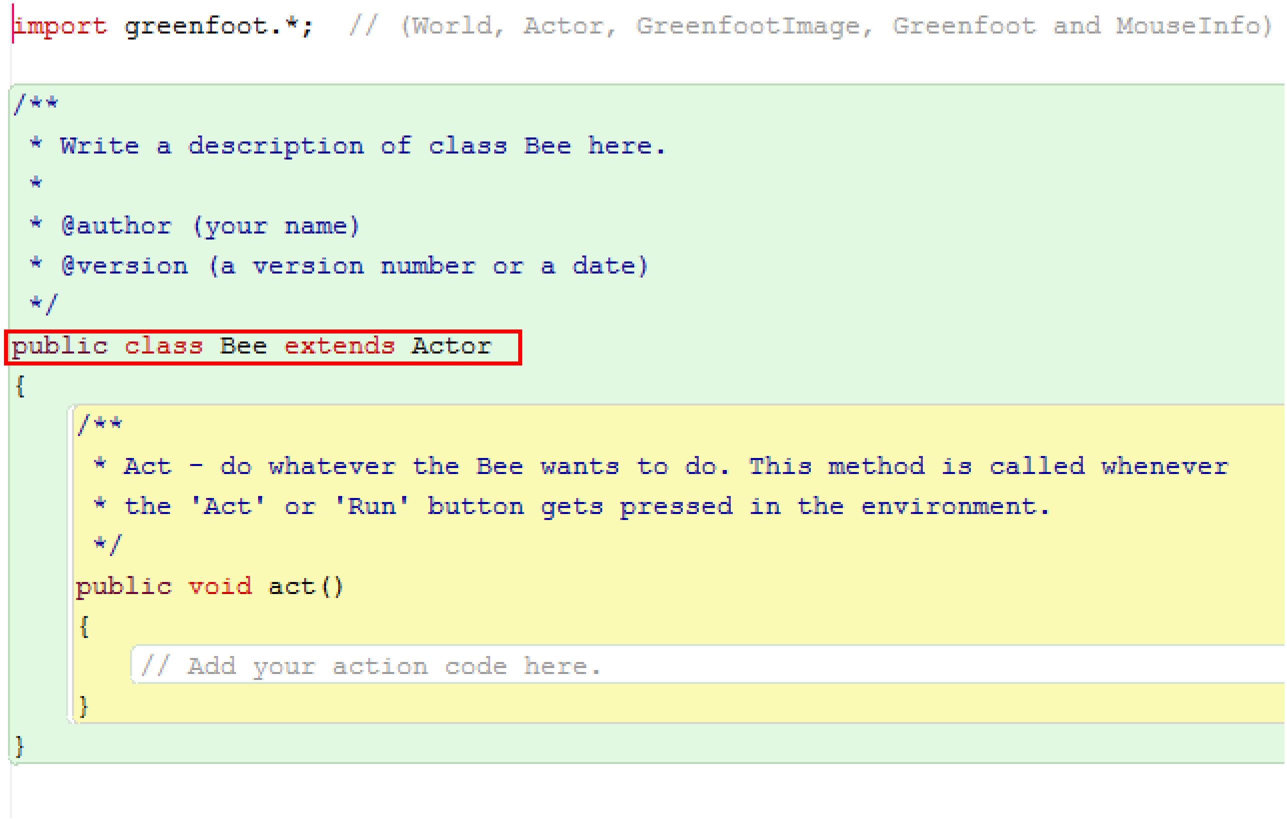
```
public class Bee extends Actor
```

Java keywords or reserved words (public, class)

Java keywords or reserved words (extends)

Superclass

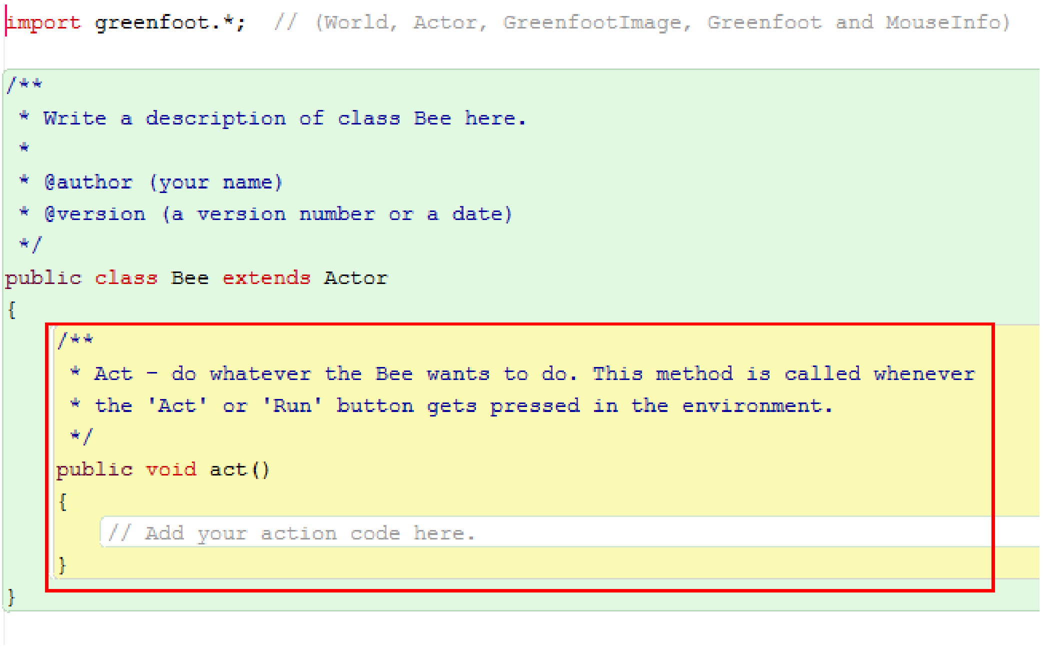
Class Definition Example



```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Bee here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Bee extends Actor
{
    /**
     * Act - do whatever the Bee wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

act() Method

- The act() method is the part of the class definition that tells objects which methods to perform when the Act or Run execution controls are clicked in the environment.



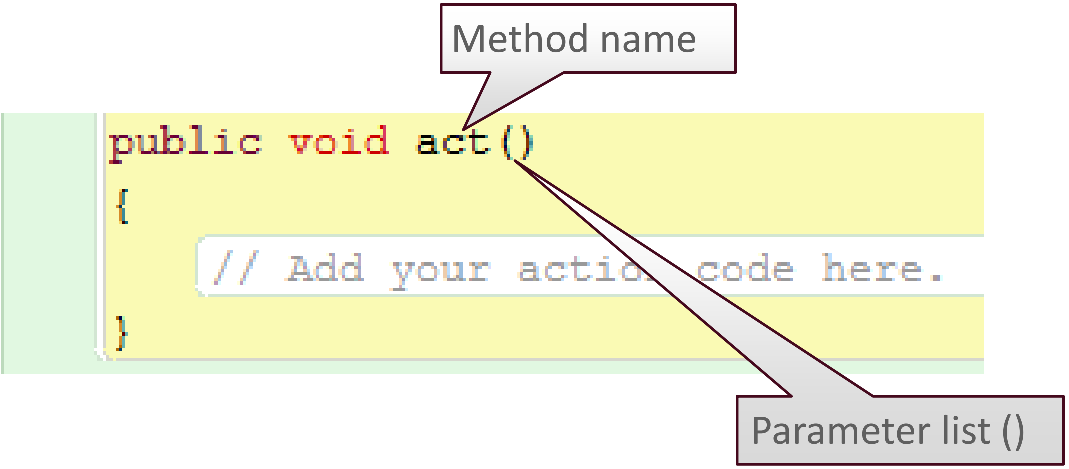
```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Bee here.
 * 
 * @author (your name) 
 * @version (a version number or a date)
 */
public class Bee extends Actor
{
    /**
     * Act - do whatever the Bee wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act() 
    {
        // Add your action code here.
    }    
}
```

Defining Classes

- The class definition defines:
  - Variables (or fields) that store data persistently within an instance.
  - Constructors that initially set up an instance.
  - Methods that provide the behaviors for an instance.
- Use a consistent format when you define a class.
  - For example, define variables first, constructors second, and methods third.

Method Signature

- The method signature describes what the method does.
- The signature contains a method name and parameter list.



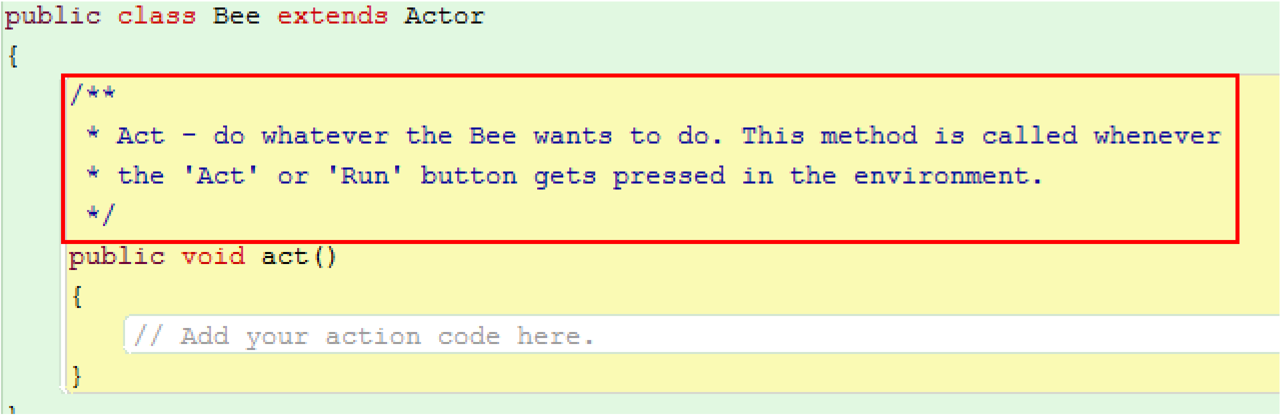
```
public void act()
{
    // Add your action code here.
}
```

Method name

Parameter list ()

Comments

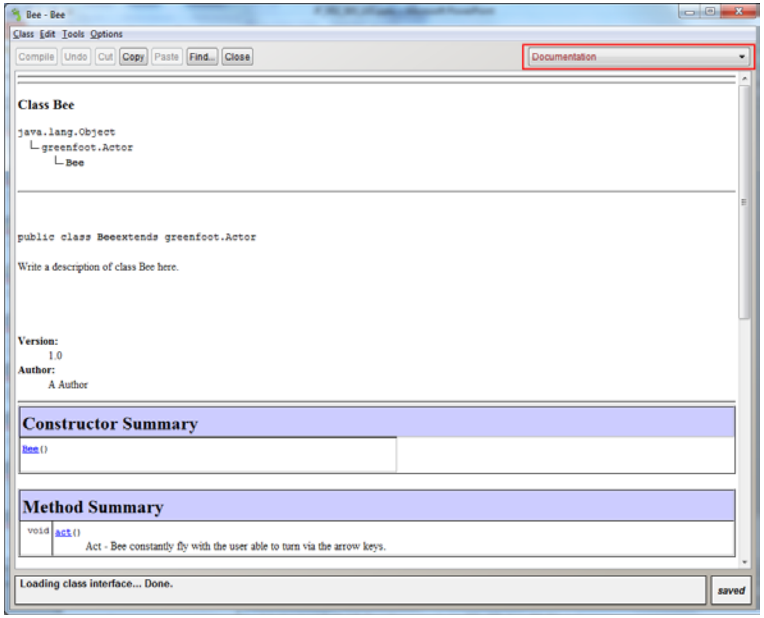
- Comments describe what the source code does.
  - Do not impact the functionality of the program.
  - Start with a forward slash and two asterisks /** or simply a double forward slash.
  - End /** comments with */
  - Written in blue font (in Greenfoot).



```
public class Bee extends Actor
{
    /**
     * Act - do whatever the Bee wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

Documentation

- Documentation describes the properties of the class.
- To view, select Documentation from the drop-down menu at the top right of the Code editor.



Class Bee

```
java.lang.Object
  └greenfoot.Actor
      └Bee
```

```
public class Beeextends greenfoot.Actor
```

Write a description of class Bee here.

Version:
1.0
Author:
A Author

Constructor Summary

Bee ()	

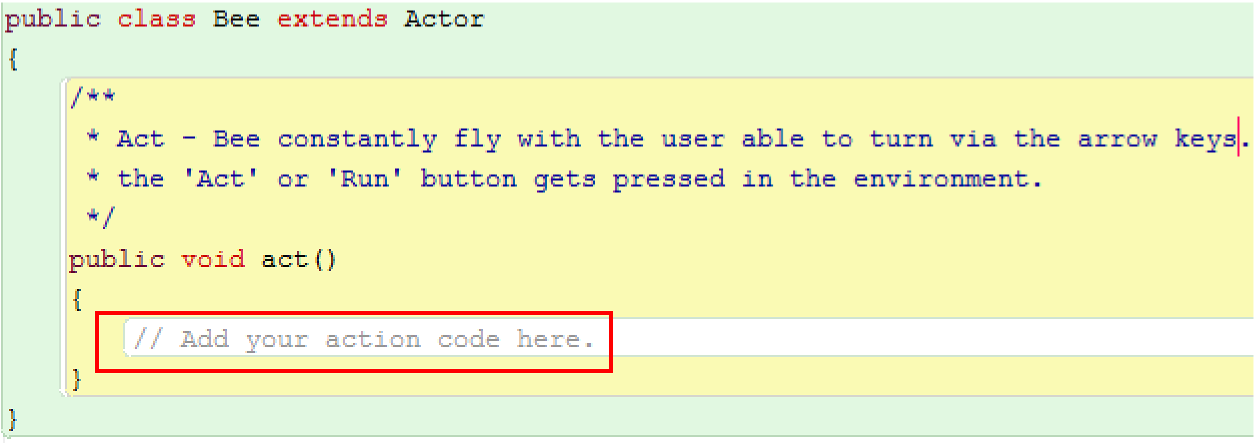
Method Summary

void	act () Act - Bee constantly fly with the user able to turn via the arrow keys.

Loading class interface... Done.

Invoke Methods Programmatically

- Methods must be invoked to command instances to act in your game.
- Invoke methods programmatically by writing them in the body of the act() method in the space between the curly brackets.



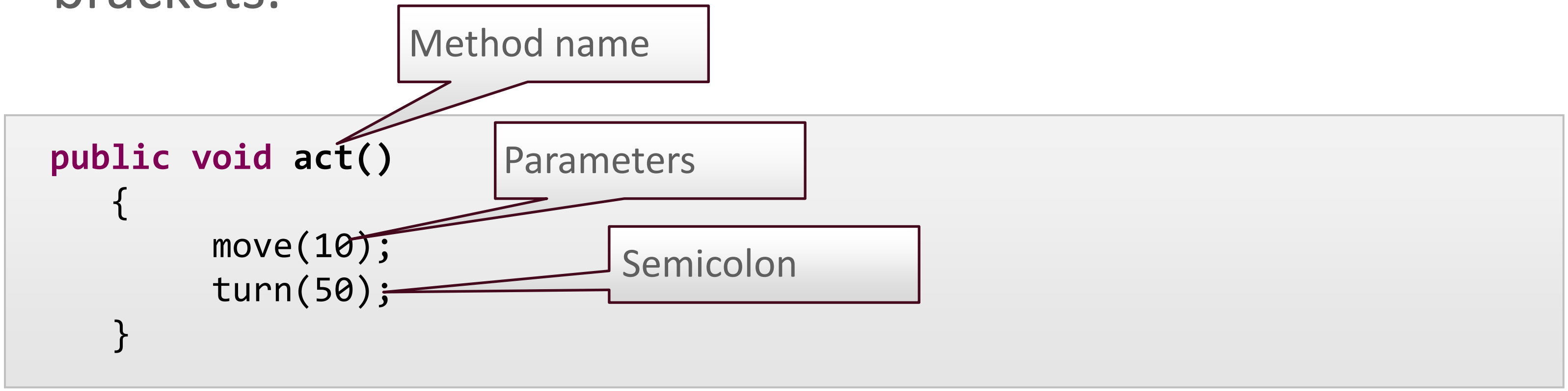
```
public class Bee extends Actor
{
    /**
     * Act - Bee constantly fly with the user able to turn via the arrow keys.
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

Method Call Components

- Method call components:
  - Return type
    - Data type of return value
    - Void return types do not require variables nor return data.
  - Method name
  - Parameter list to indicate the type of arguments to invoke, if required.
  - Semicolon to mark the end of the method call.

Invoking Methods Example 1

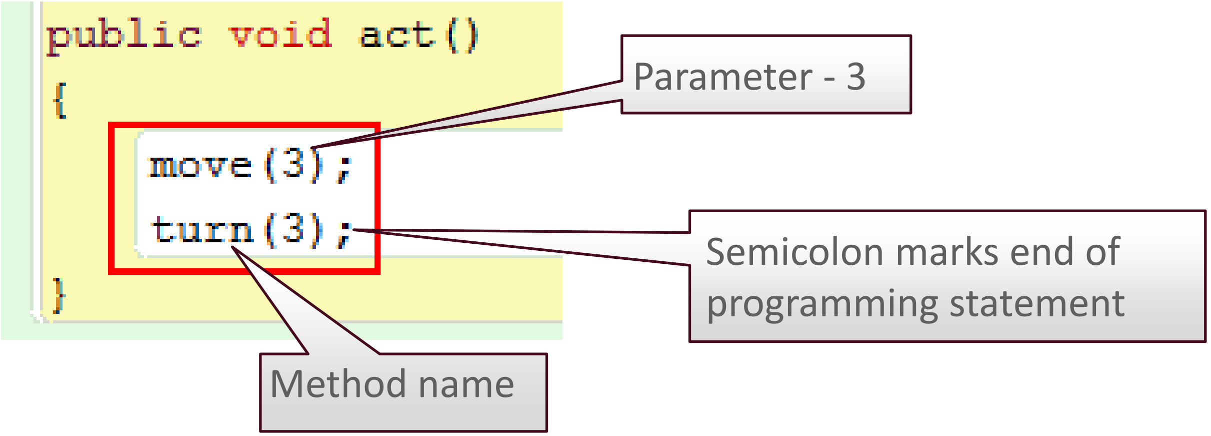
- Each method is written in the space between the curly brackets.



```
public void act()
   {
          move(10);
          turn(50);
   }
```

Invoking Methods Example 2

- The first method call is written into the body of the act() method, ending with a semicolon.
- Each additional method call is typed directly underneath, until all methods are entered in the space between the curly brackets.



```
public void act()
{
    move(3);
    turn(3);
}
```

Parameter - 3

Semicolon marks end of programming statement

Method name

Methods that Instruct Objects to Perform Actions

Method Name	Description
void move(int distance)	Assigns the object a number of steps to move, or the command to simply move when the Act or Run buttons are clicked.
void turn(int amount)	Assigns the object a number of degrees to turn.
void act()	Gives the object the opportunity to perform an action in the scenario. Method calls are inserted into this method.
void setLocation(int x, int y)	Assigns a new location for this object.
void setRotation(int rotation)	Sets a new rotation for this object.

Ways to View a Class's Inherited Methods

- View the Greenfoot Class Documentation.
  - Open Greenfoot.
  - Select Help.
  - Select Greenfoot Class Documentation.
- View the Java Library Documentation.
  - Open Greenfoot.
  - Select Help.
  - Select Java Library Documentation.

Sequential Tasks

- A single task, such as going to school, requires multiple sub-tasks:
  - Wake up
  - Take a shower
  - Brush your teeth
  - Get dressed…
- Within a sub-task, there could be more sub-tasks (walking to school requires the left leg and right legs to move forward, in order).

Sequential Methods

- Sequential methods are multiple methods executed by Greenfoot in the order in which they are written in the program.
- These methods make it possible for an object to perform sequential tasks, such as run and then jump, or play a sound after something explodes.
- Objects can be programmed to perform sequential methods whenever the Act button is clicked.

if-then Relationships

- Many things around us have a cause and effect relationship, or "if-then" relationship.
  - If your cell phone rings, then you answer it. If it doesn't ring, then you do not answer it.
  - If a flower starts to wilt, then you give it water. If the flower looks healthy, then you do not give it water.

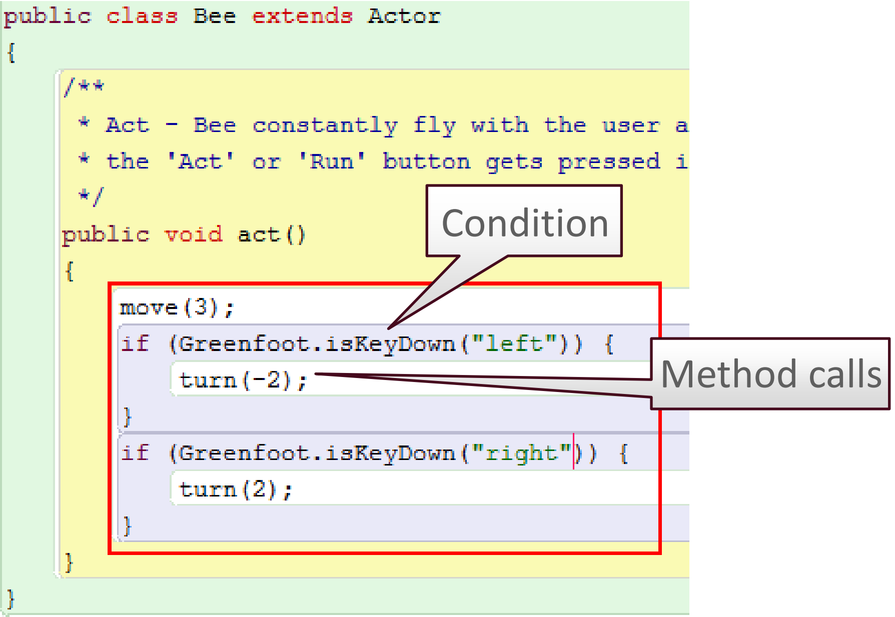
if Decision Statements

- An IF statement is written to tell your program to execute a set of programming statements only if and when a certain condition is true.

```
if (condition)
   {
       instruction;
       instruction;
       …
   }
```

if Decision Statement Components

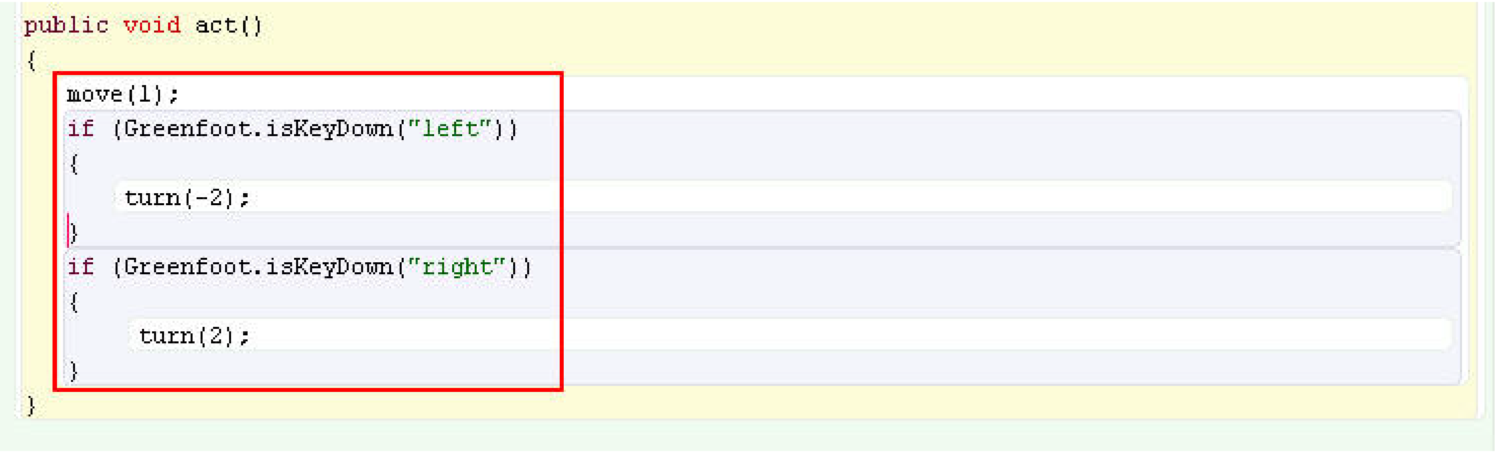
- The if statement contains a condition, which is a true or false expression, and one or more method calls that are executed if the condition is met.



```
public class Bee extends Actor
{
    /**
     * Act - Bee constantly fly with the user a
     * the 'Act' or 'Run' button gets pressed i
     */
    public void act()
    {
        move(3);
        if (Greenfoot.isKeyDown("left")) {
            turn(-2);
        }
        if (Greenfoot.isKeyDown("right")) {
            turn(2);
        }
    }
}
```

if Decision Statement Example

- In the following example:
  - The left and right arrow keys on the keyboard make the object turn left and right.
  - If the condition is false, the method calls defined in the IF statement are not executed.
  - The move method is executed regardless of the IF statement.



```
public void act()
{
    move(1);
    if (Greenfoot.isKeyDown("left"))
    {
        turn(-2);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(2);
    }
}
```

isKeyDown Method

- The isKeyDown method is a pre-existing Greenfoot method that listens to determine if a keyboard key is pressed during program execution.
- This method is called in a class using dot notation.

When a method is not in the class or inherited by the class you are programming, specify the class or object that has the method before the method name, then a dot, then the method name. This technique is called dot notation.

Object Orientation in the Real World

- As we move about the world we live in, it's important for us to know our orientation, or sense of direction.
  - When you drive a car, you always need to know if your car is in the correct lane of the road.
  - When a plane flies through the air, it needs to know where it's located relative to other planes, so a collision doesn't occur.
  - When you enter your location on a map in a cell phone, you receive coordinates that tell you where you are, and the address.

Display an Object's Orientation

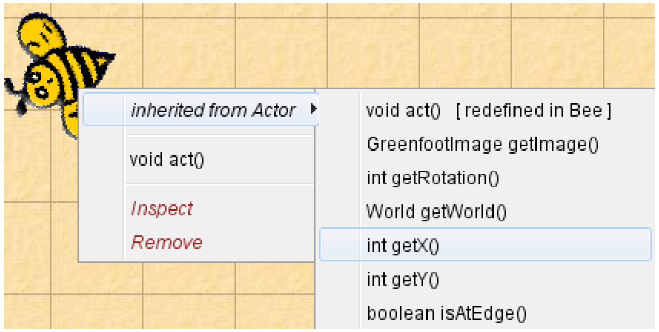
- Methods can tell us how an object is positioned in the world, relative to itself and other objects.
- You can invoke a method:
  - With a specific data type, such as boolean, to ask the object a question about it's orientation.
  - In the environment to learn how the object is oriented in the scenario.

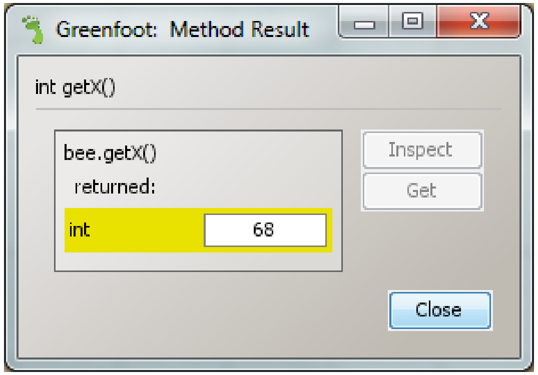
Methods that Return Information About an Object's Orientation

Method Name	Description
int getRotation()	Returns the current rotation of the object.
World getWorld()	Returns the world that the object is currently in.
int getX()	Returns the x-coordinate of the object's current location.
int getY()	Returns the y-coordinate of the object's current location.

Steps to Invoke a Method that Displays an Object's Orientation

- Right click on the instance in the world.
- Select Inherited from Actor to view its methods.
- Invoke (select) a method with a specific data type to ask the object a question about its orientation.
- The method result will display. Note the value returned, then click Close.





int getX()

bee.getX()
returned:
int 68

Terminology

Key terms used in this lesson included:

- Class description
- Comments
- if decision statements
- Invoking a method
- Object oriented analysis
- Sequential methods

Summary

In this lesson, you should have learned how to:

- Demonstrate source code changes to invoke methods programmatically
- Demonstrate source code changes to write an if decision statement
- Describe a method to display object orientation

ORACLE® ACADEMY