
ORACLE
ACADEMY

Java Fundamentals

3-6
Defining Methods

```
private void catchFly() {
    if (isTouching(Fly.class)) {
        removeTouching(Fly.class);
    }
}
```

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- Describe effective placement of methods in a super or subclass
- Simplify programming by creating and calling defined methods
- Handling collisions

Efficient Placement of Methods

- At times, many lines of code are required to program a behavior.
- For example, you may want to program an instance to eat other objects, or turn when it reaches the edge of the world.
- Define new methods to save time and lines of code.
  - Define a new method for an action below the act method.
  - Call the new method in the act method or within another method.
  - Define the method in the superclass if you want its subclasses to automatically inherit the method.

Defined Methods

- Defined methods are new methods created by the programmer.
- These methods:
  - Can be executed immediately, or stored and called later.
  - Do not change the behavior of the class when stored.
  - Separate code into shorter methods, making it easier to read.

Defined methods create a new method that a class did not already possess. These methods are written in the class's source code below the act method.

Steps to Define a New Method

- Select a name for the method.
- Open the Code editor for the class that will use the method.
- Add the code for the method definition below the act method.
- Call this new method from the act method, or store it for use later.

Steps to Define a New Method

- We could for example add our Bee movement code in act() to a new method.
- This helps keep the code in act to a minimum and keep its actions clearer.

```
public void act()
{
    move(3);
    if (Greenfoot.isKeyDown("left")) {
        turn(-2);
    }
    if (Greenfoot.isKeyDown("right")) {
        turn(2);
    }
}
```

```
public void act()
{
    handleMovement();
}
public void handleMovement() {
    move(3);
    if (Greenfoot.isKeyDown("left")) {
        turn(-2);
    }
    if (Greenfoot.isKeyDown("right")) {
        turn(2);
    }
}
```

Turn at the Edge of the World

- Problem:
  - Instances stop and are unable to move when they reach the edge of the world.
  - Instances should turn and move when they reach the edge of the world.
- Solution:
  - Create a subclass of Actor that defines a method that can detect if the object is at the edge of the world and to turn appropriately.
  - Call the new methods in the subclasses that should be able to turn and move at the edge of the world.

Test if at edge of World

- Greenfoot has a method in the Actor class called isAtEdge().
- This returns true if the actor is at one of the edges.
- We could use this to detect and then turn actors around rather than them hover at one of the edges.
- If our program required to know which edge an actor was at we would have to either define a method to return the side we are touching, or 4 separate methods, one for each side.

Test an Object's Position in the World

- To test if an object is near the edge of the world, this requires:
  - Boolean expressions to express if a condition is true or false.
  - Example – We could rotate an instance by 180 degrees if its at the edge of the world.

```
public void act()
{
    move(1);
    if (isAtEdge()) {
        turn(180);
    }
}
```

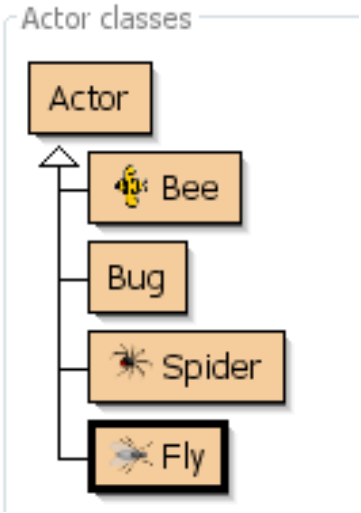
Logic Operators

Logic operators can be used to combine multiple boolean expressions into one boolean expression.

Logic Operator	Means	Definition
Exclamation Mark (!)	NOT	Reverses the value of a boolean expression (if b is true, !b is false. If b is false, !b is true).
Double ampersand (&&)	AND	Combines two boolean values, and returns a boolean value which is true if and only if both of its operands are true.
Two lines (II)	OR	Combines two boolean variables or expressions and returns a result that is true if either or both of its operands are true.

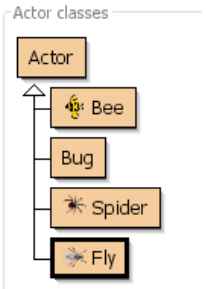
Create the Bug Superclass

- Before creating the defined methods, create a new subclass of the Actor class named Bug.
- This class has no image and will not have instances that act in the scenario, but will hold some defined methods that other subclasses will inherit.

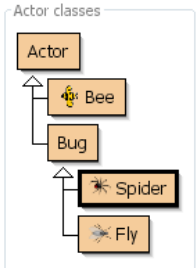


Create the Bug Subclasses

- We could recreate our Spider and Fly by right clicking on Bug, and selecting new Subclass.
- But as we have previously created them we can modify our Spider and Fly source code to extend from Bug rather than Actor.
- When you click compile Greenfoot will update the class structure.




```
public class Spider extends Bug
{
    public void act()
    {
        // Add your action code here.
    }
}
```



Define turnAtEdge Method in Superclass

- Open the Code editor for the Bug class. Write the code for the turnAtEdge method, below the act method.
- Compile the code and then close the Code editor.

```
/**
 * turnAtEdge - turn the actor 180 degrees if at the edge of the world.
 */
public void turnAtEdge()
{
    if (isAtEdge()) {
        turn(180);
    }
}
```

Call turnAtEdge Method in Subclass

- Open the Code editor for the Fly subclass.
- Add a call to the method turnAtEdge within the Act method.

```
public void act()
{
    move(1);
    if (Greenfoot.getRandomNumber(100) < 10) {
        turn(Greenfoot.getRandomNumber(90)-45);
    }
    turnAtEdge();
}
```

Define atRightEdge Method in Bee class

- Open the Code editor for the Bee class.
- Write the code for the atRightEdge method, below the act method.
- Compile the code and then close the Code editor.

```
/**
 * Test if we are close to the right edge of the world. Return true if the object is.
 */
public boolean atRightEdge()
{
    if(getX() > getWorld().getWidth() - 20 )
        return true;
    else
        return false;
}
```

Define atBottomEdge Method in Bee Class

- Open the Code editor for the Bee class.
- Write the code for the atBottomEdge method, below the act method.
- Compile the code and then close the Code editor.

```
/**
 * Test if we are close to the bottom edge of the world. Return true if the object is.
 */
public boolean atBottomEdge()
{
    if(getY() > getWorld().getHeight() - 20 )
        return true;
    else
        return false;
}
```

Methods in atRightEdge and atBottomEdge

- The methods used in atRightEdge and atBottomEdge include:
  - getX: An Actor method that returns the x-coordinate of the actor's current location.
  - getY: An Actor method that returns the y-coordinate of the actor's current location.
  - getWorld: An Actor method that returns the world that this actor lives in.
  - getHeight: A GreenfootImage class method that returns the height of the image.
  - getWidth: A GreenfootImage class method that returns the width of the image.

Call Method in Class

- Open the Code editor for the Bee class.
- Create an IF statement that calls the atRightEdge and atLeftEdge method as a condition in act.
- If the Bee is at the left it will re-appear on the right and vice versa.

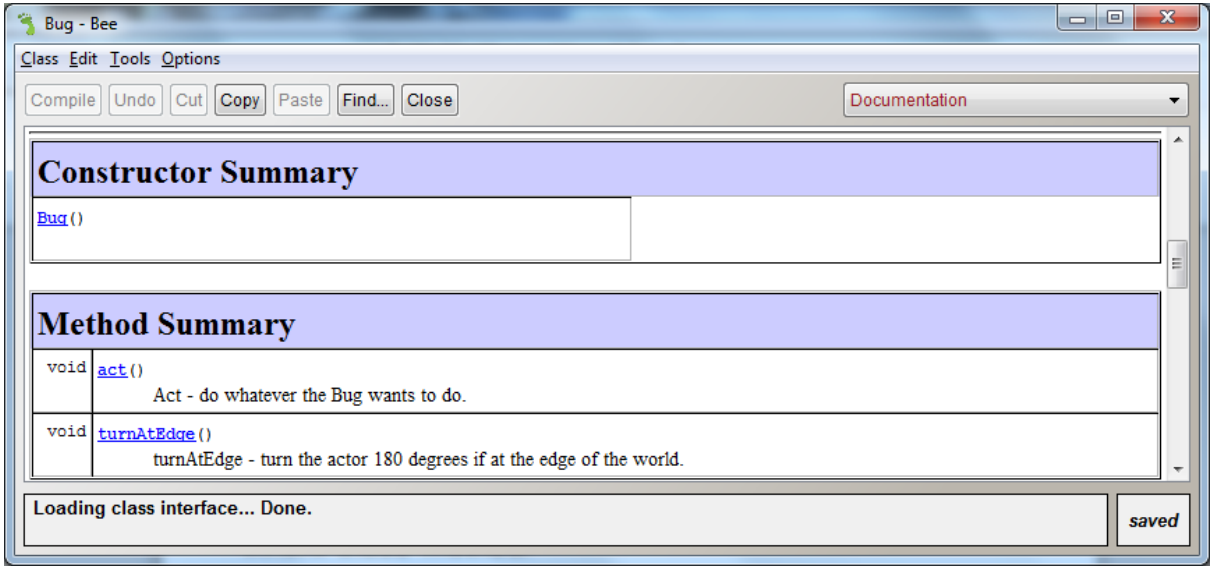
```
public void act()
{
    handleMovement();
    if (isAtLeft()) {
        setLocation(getWorld().getWidth() - 6, getY());
    }
    else if (isAtRight()) {
        setLocation(6,getY());
    }
}
```

Call Method in Class

- Compile the code and run the scenario to test it.
- Complete the IF statement for the atTopEdge and atBottomEdge.

Class Documentation

- The Bug class documentation shows the new method after its defined.
- All subclasses of the Bug superclass inherit this method.



Collisions

- Most game projects will usually have to detect when two actors touch which is usually called a collision.
- In GreenFoot there are multiple ways to detect this.
- Some of these are
  - isTouching()
  - getOneIntersectingObject(Class)
  - getOneObjectAtOffset(Class)
  - getIntersectingObjects(Class)
  - getNeighbours(distance,diagonal)
  - getObjectsAtOffset(dx,dy,Class)

Collisions

Method	When To Use
isTouching	When you simply want to detect a collision with an object
getOneIntersectingObject	When you want to return a reference to the object you have collided with. This is normally used if you wish to perform an action on the collided object.
getOneObjectAtOffset	Same as getOneIntersectingObject except that you can change where the collision will be detected relative to the current object. So you could have the collision detected before it happens. i.e. to stop an actor walking into a wall.

Defined Method to Remove Objects

- You can write code in your game so a predator object is able to eat prey objects.
- Create a defined method in the act method of the Bee superclass called catchFly to enable us to remove flies that we catch.
- To create this defined method we are going to use the simplest collision detection – isTouching.

Define catchFly Method

- Define the catchFly Method in the Bee class.
- This method detects a collision with a fly and then removes it.

```
private void catchFly() {
    if (isTouching(Fly.class)) {
        removeTouching(Fly.class);
    }
}
```

Define catchFly Method - Alternative

- Alternatively we could have used getOneIntersectingObject and accessed a reference to the actor before deleting it.

```
private void catchFly2() {
    Actor fly = getOneIntersectingObject(Fly.class);
    if (fly != null) {
        getWorld().removeObject(fly);
    }
}
```

Call catchFly in Act Method

- Call the new catchFly method in the Bee's act method.
- Run the scenario to test the code.

```
public void act()
{
    handleMovement();
    if (isAtLeft()) {
        setLocation(getWorld().getWidth() - 6, getY());
    }
    else if (isAtRight()) {
        setLocation(6,getY());
    }
    catchFly();
}
```

Terminology

Key terms used in this lesson included:

- Defined methods
- Collisions

Summary

In this lesson, you should have learned how to:

- Describe effective placement of methods in a super or subclass
- Simplify programming by creating and calling defined methods


ORACLE
ACADEMY