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
- You can define new methods to simplify logic, save time, and use fewer lines of code
  - Define (code) a new method for an action below the act method
  - Call (use) the new method in the act method of a world or actor, or within another method
  - Define (code) the method in the superclass if you want its subclasses to automatically inherit the method

Using methods allows us to take advantage of the features supplied by the Greenfoot development team.

Defining our own methods allows us to expand the functionality of our objects.
Your defined methods can be used the same as the inherited methods supplied by Greenfoot.

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 a class's source code below the act method.

Defining your own methods may not improve the game performance, but can lead to many other advantages such as more readable code and quicker development time.

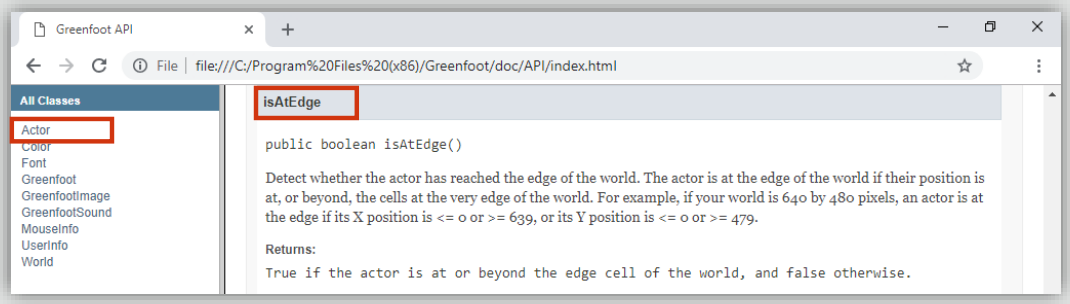
Turn at the Edge of the World

- Problem:
  - Instances stop and are unable to move when they reach the edge of the world
  - Instances should turn and move in the opposite direction 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
  - Create subclasses of the Actor subclass that should inherit the method
  - Call (use) the new method in the subclasses that should be able to turn and move at the edge of the world

It is not always obvious when we should create a subclass of our subclass. Planning your game helps you to spot these patterns early.

Test if an Actor is at the edge of the World

- Greenfoot has a method in the Actor class called isAtEdge()
- This returns true if the Actor is at one of the edges
- We can use this to detect and then turn actors around rather than them hover at one of the edges



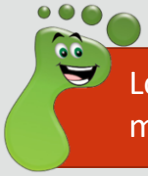
Test an Object's Position in the World

- To test if an object is near the edge of the world:
  - Use an if statement with the boolean isAtEdge() method to test if the condition is true or false
  - Example: We can rotate an instance by 180 degrees if its at the edge of the world

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


Rotating an image by 180 degrees means it will turn exactly half a full rotation. i.e. turn it in the opposite direction.

Logic Operators



Logic operators can be added to if/else statements 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

To use Boolean logic, it is common to use variables to test a condition. These are variables that the object instance does not store and are associated at the method level. So when a method is finished the value of the local variable is lost. It is important to name these variables appropriately as they can make reading your Boolean expressions a lot easier.

Logic Operators - Examples

- If a Spider is in the top right half of the scenario, move the Spider forward:

```
public void act()
{
    if((getX() > getWorld().getWidth()/2) && (getY()< getWorld().getHeight()/2))
    {
        move(2);
    }//endif
}
```

- If the Actor's horizontal position is greater than the center of the horizontal world, and the Actor's vertical position is less than the center of the vertical world…
  - getX() returns the horizontal position of the Actor
  - getWorld().getWidth() returns the horizontal size of the World
  - getY() returns the vertical position of the Actor
  - getWorld().getHeight() returns the vertical size of the World

Logic Operators - Examples

```
public void act()
{
    if((getX() > getWorld().getWidth()/2) && (getY()< getWorld().getHeight()/2))
    {
        move(2);
    }//endif
}
```

- Substitute || for the && to test if a Spider is in the top half OR the right half of the scenario

```
public void act()
{
    if((getX() > getWorld().getWidth()/2) || (getY()< getWorld().getHeight()/2))
    {
        move(2);
    }//endif
}
```

Logic Operators - Examples

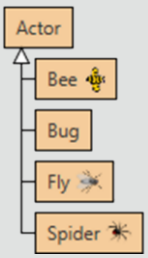
```
public void act()
{
    if((getX() > getWorld().getWidth()/2) || (getY()< getWorld().getHeight()/2))
    {
        move(2);
    }//endif
}
```

- Use ! to move a Spider forward if it is NOT in the right half of the scenario

```
public void act()
{
    if(!(getX() > getWorld().getWidth()/2) || (getY()< getWorld().getHeight()/2))
    {
        move(2);
    }//endif
}
```

Creating a Superclass

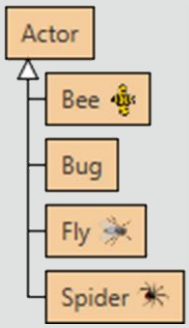
- To allow both the Spider and the Fly classes to inherit the turning behavior we place the code in a superclass
- The name of this superclass should be descriptive of the commonality of the classes that will inherit from it
- We will create a Bug class that has no image and will not have instances that act in the scenario, but will hold defined methods that its subclasses will inherit
  - Right+click Actor, choose New subclass
  - Name the new class “Bug”
  - Do not assign an image
  - Click Ok



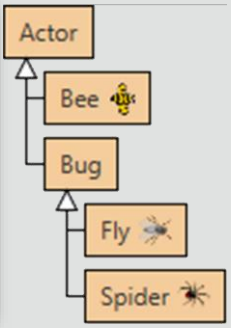
The diagram shows that all of our classes are direct subclasses of Actor.

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




```
public class Spider extends Bug
{
    /**
     * Act - do whatever the Spider wants to do. Thi
     * the 'Act' or 'Run' button gets pressed in the
     */

    public void act()
    {
        if(!(getX() > getWorld().getWidth()/2) || (g
        {
```



ORACLE Academy

JF 3-6
Defining Methods

14

The class is defined as:
public class Spider extends Actor

Change to :

public class Spider extends Bug

This changes the Spider's superclass, and works fine as we stay in the same inheritance path.

Define turnAtEdge() Method in Superclass

- Open the Code editor for the Bug class
- Write the code for the turnAtEdge() method, below the act() method

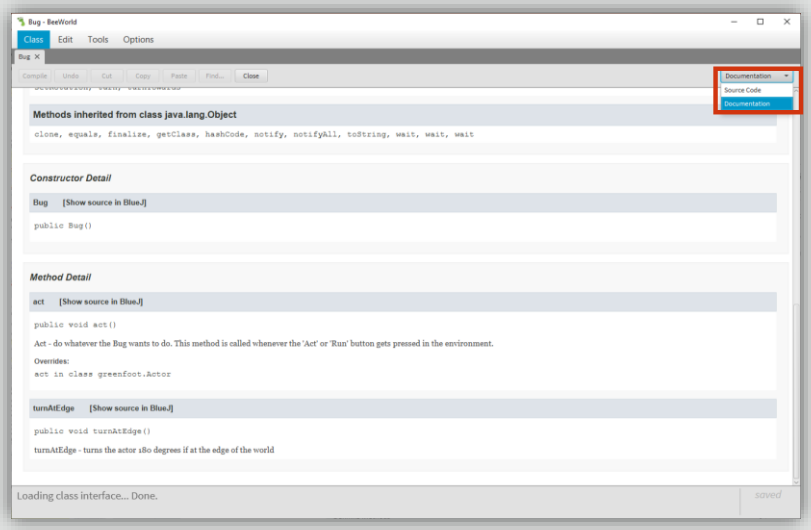
```
public class Bug extends Actor
{
    /**
     * Act - do whatever the Bug wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }

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

ORACLE Academy

We have made it public because we want subclasses to use this method.
It is void because it doesn't return a value.

Class Documentation

- The Bug class documentation shows the new method after its defined



Change the view in the editor to show the documentation from the drop down box at the top right of the window.

Call turnAtEdge() Method in Subclass

- Open the Code editor for the Fly subclass from the previous lesson
- Add a call to the method turnAtEdge() within the act() method
- Repeat this for the Spider subclass

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


Notice we can call (use) a superclass's method (depending on its visibility/accessibility) if it is public from a subclass.

Define at[left, right, top, bottom]Edge() Methods in Bee class

- For the Bee class, we will write methods that will
  - determine the edge where the Bee is approaching
  - move the Bee to the opposite edge (appears to wrap around)
- For example, if the Bee is moving toward the right, and approaching the right edge:
  - it will disappear from the right edge
  - re-appear on the left edge
  - continue moving toward the right edge again

- To know which edge an actor is touching, we will define 4 separate methods, one for each side

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

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


getWorld() returns a reference to the current world.
getWorld().getWidth() returns the current world's width.
We could have used a value here like 800, but if we changed the world's width then this methods functionality would break. Calling the worlds getWidth() method produces a flexible solution.

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

```
}//end method act

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

Methods called from atRightEdge() and atBottomEdge()

- The methods used in atRightEdge() 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 World class method that returns the height of the world
  - getWidth():
    - A World class method that returns the width of the world

It is always best to use the getHeight() and getWidth() methods rather than typing a value.

Call Methods in Bee Class

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

Move will always execute

If the Bee is near the right edge, the x co-ordinate will be set to 6. The current Y position is kept using the getY() method.

If the Bee is near the bottom edge, the Y co-ordinate will be set to 6. The current X position is kept using the getX() method.

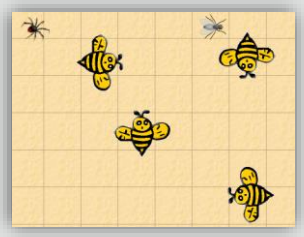
```
public void act()
{
    move(1);
    if(atRightEdge()){
        setLocation(6, getY());
    }else
        if(atBottomEdge()){
            setLocation(getX(), 6);
        }//endif
    //endif
}//end method act
```

ORACLE Academy

We use 6 as part of the new location so that we are not detected immediately as being at the edge again.

Call Methods in Class

- Open the Code editor for the myWorld class
- If necessary,
  - add four Bee objects to your world
  - select Controls -> Save the World



- In the prepare() method, add the following:
  - bee2.turn(90);
  - bee3.turn(-90);
  - bee4.turn(180);

```
private void prepare()
{
    Fly fly = new Fly();
    addObject(fly,334,42);
    Spider spider = new Spider();
    addObject(spider,46,48);
    Bee bee = new Bee();
    addObject(bee, 150, 100);
    Bee bee2 = new Bee();
    addObject(bee2,388,87);
    Bee bee3 = new Bee();
    addObject(bee3,220,214);
    Bee bee4 = new Bee();
    addObject(bee4,396,312);
    bee2.turn(90);
    bee3.turn(-90);
    bee4.turn(180);
```

Call Method in Class

- Complete the IF statement for the atLeftEdge() and atTopEdge()
- Test your program to make sure the Bee instances move off the world edges to the opposite edge

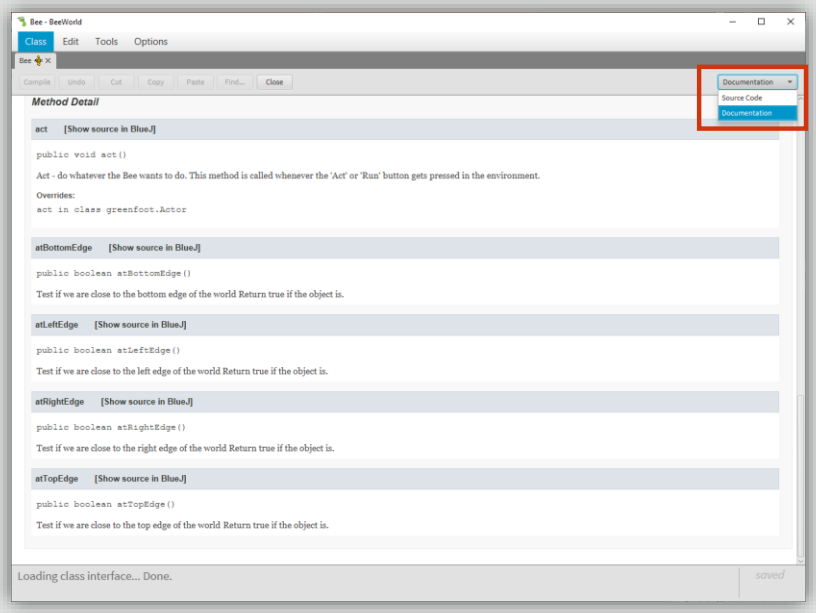
```
}//end method act

/**
 * Test if we are close to the top edge of the world
 * Return true if the object is.
 */
private boolean atTopEdge()
{
    if(getY() < 6)
       return true;
    else
       return false;
}//end method atTopEdge

/**
 * Test if we are close to the left edge of the world
 * Return true if the object is.
 */
private boolean atLeftEdge()
{
    if(getX() < 6)
       return true;
    else
       return false;
}//end method atLeftEdge
```

```
/**
 * 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()
{
    move(1);
    if(atRightEdge()){
        setLocation(6, getY());
    }
    else if(atBottomEdge()){
            setLocation(getX(), 6);
    }
    else if(atLeftEdge()){
            setLocation(getWorld().getWidth()-20, getY());
    }
    else if(atTopEdge()){
            setLocation(getX(), getWorld().getHeight()-20);
    }//endif
}//end method act
```


Our bees should now fly off one edge of the screen and reappear on the opposite one.

Class Documentation

- The Bee class documentation shows the new method after its defined

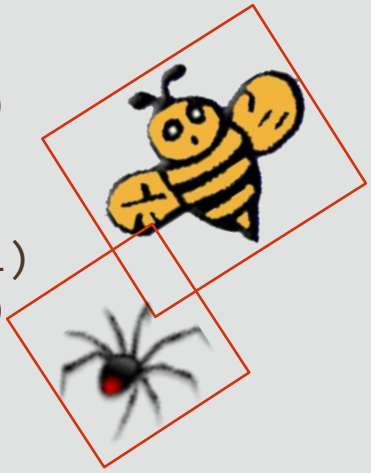


ORACLE Academy

Remember to change the view in the editor to show the documentation we change the drop down box at the top right of the window.

Collisions

- Most game projects will have to detect when two actors touch, which is 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)`



Although we define a collision as when 2 actors touch, we can also modify our code so that we detect a collision if 2 actors are within close proximity.
The last two methods above return a list. We would then have to process this list to find out what we want.

Collisions

Method	When To Use
isTouching()	When you want to detect a collision with an object
getOneIntersectingObject()	When you want to return a reference to the object you have collided with Use 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

isTouching() returns a boolean value (true or false). The other two methods return a reference to an actor.

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 Bee class called catchfly() to enable us to remove the flies that the Bee catches
- To create this defined method we are going to use the simplest collision detection – isTouching()
- This method detects a Bee's collision with a Fly, and then removes it

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

ORACLE Academy

The code for this could be written multiple ways.

If we had used isTouching() (with nothing in the parenthesis), then the method would return true if we touch any other actor, including an instance of another Bee.

Define catchfly() Method - Alternative

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

```
}//end method act

/**
 * catchFly2 - if the Bee touches a fly the fly is removed
 */
private void catchFly2(){
    Actor fly = getOneIntersectingObject(Fly.class);
    if(fly != null){
        getWorld().removeObject(fly);
    }//endif
}

/**
 * catchFly - if the Bee touches a fly the fly is removed
```


getOneIntersectingObject() returns a reference to an actor. The type of actor in this example is restricted to only being a Fly class instance. If it doesn't find a collision with a Fly, then the reference variable fly is given the null value.

We then test if fly is null and if it isn't we remove it.

Call catchfly() in act() Method

- Call the new catchfly() method in the Bee's act() method
- Be careful to add the code outside the if statement as we want the bee to always try to catch a fly
- Run the scenario to test the code

```
public void act()
{
    move(1);
    if(atRightEdge()){
        setLocation(6, getY());
    }
    else if(atBottomEdge()){
            setLocation(getX(), 6);
    }
    else if(atLeftEdge()){
            setLocation(getWorld().getWidth()-20, getY());
    }
    else if(atTopEdge()){
            setLocation(getX(), getWorld().getHeight()-20);
    }//endif
    catchFly();
}//end method act
```

Call catchfly() in act() Method

- Additional changes recommended for the act() method:
  - re-locate the move() statement from the act() method to a new method, called handleMovement()
  - re-locate the if statements from the act() method that check if the Bee is at the edge by adding a turnAtEdge() method that tests if the Bee is at the edge

```
public void act()
{
    handleMovement();
    turnAtEdge();
    catchFly();
}//end method act
```

```
private void handleMovement(){
    move(1);
}//end method handleMovement
```

```
private void turnAtEdge(){
    if(atRightEdge()){
        setLocation(6, getY());
    }
    else if(atBottomEdge()){
            setLocation(getX(), 6);
    }
    else if(atLeftEdge()){
            setLocation(getWorld().getWidth()-20, getY());
    }
    else if(atTopEdge()){
            setLocation(getX(), getWorld().getHeight()-20);
    }//endif
}//end method turnAtEdge
```


Best practice is to create methods for Actor actions, and then call those methods from the act() method.

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