Creating Java Programs with Greenfoot

Defining Methods

Overview

This lesson covers the following topics:

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

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.
  - 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

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

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:
  - Define a method in the Animal superclass to give all Animal subclasses the ability to turn when they reach the edge of the world.
  - Call the new method in the subclasses that should be able to turn and move at the edge of the world.

Test an Object's Position in the World

- To test if an object is near the edge of the world, this requires:
  - Multiple boolean expressions to express if one or both conditions are true or false.
  - Logic operators to connect the boolean expressions used to evaluate a condition.

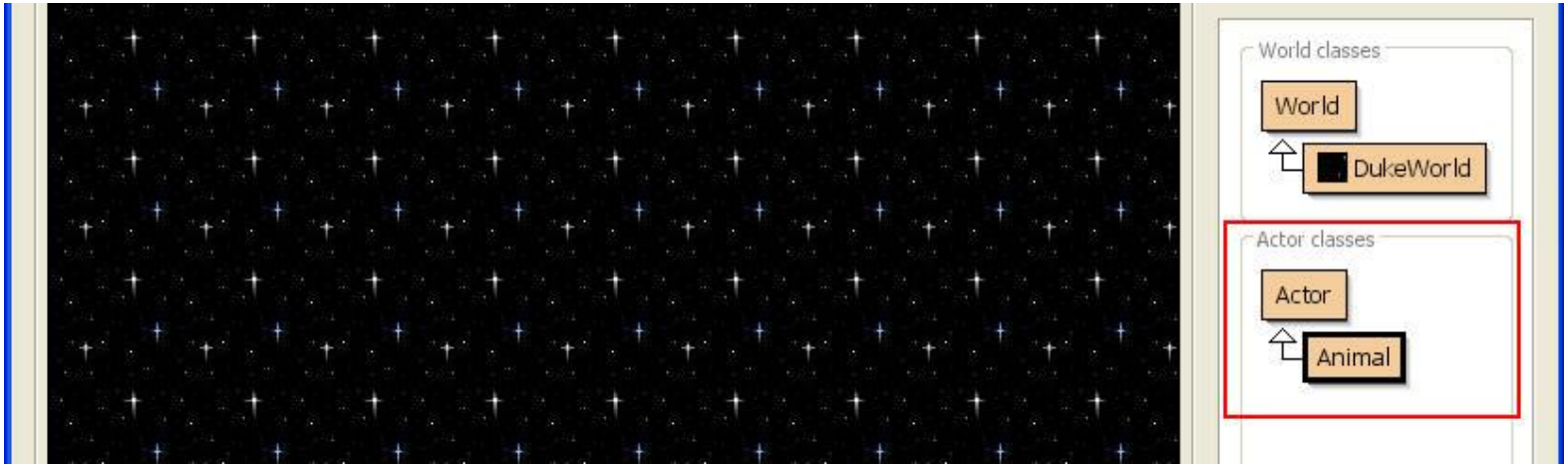
Logic Operators

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.

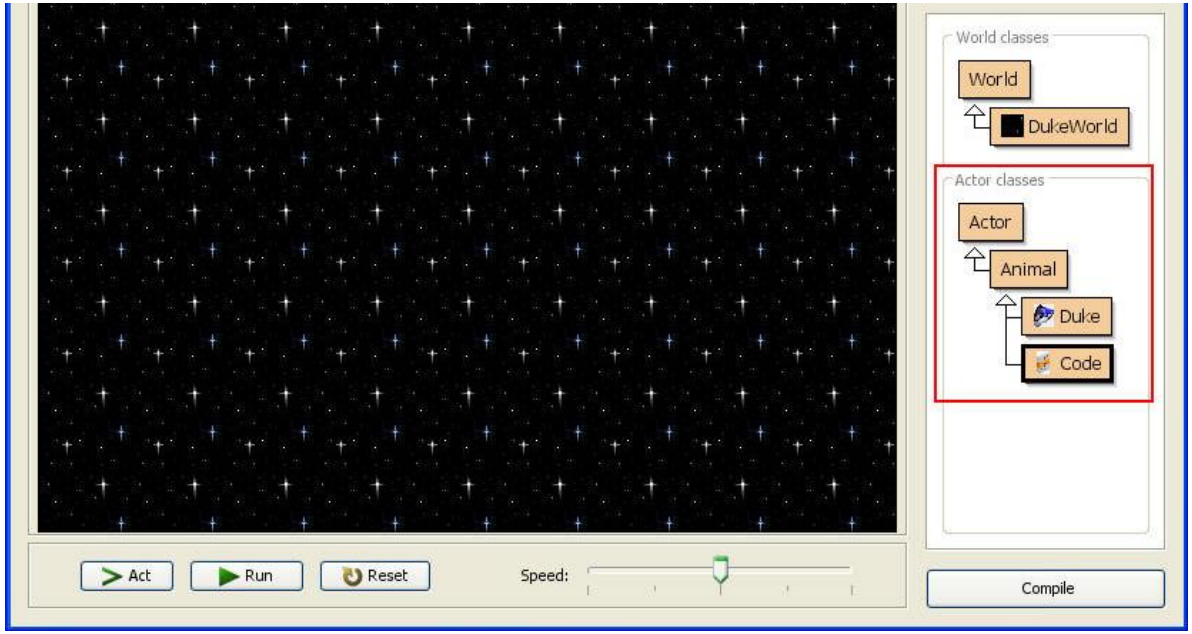
Create the Animal Superclass

Before creating the defined methods, create a new subclass of the Actor class named Animal. 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 Animal Subclasses

Create subclasses of the Animal superclass that will have instances that act in the scenario. In this example, a Duke subclass and Code subclass were added for a game where Duke, the Java mascot, eats Code objects.



Define atWorldEdge Method in Superclass

Open the Code editor for the Animal superclass. Write the code for the atWorldEdge method, below the act method. Compile the code and then close the Code editor.

```
/**
 * Test if we are close to one of the edges of the world. Return true is we are.
 */
public boolean atWorldEdge()
{
    if(getX() < 20 || getX() > getWorld().getWidth() - 20)
        return true;
    if(getY() < 20 || getY() > getWorld().getHeight() - 20)
        return true;
    else
        return false;
}
```

Methods Used in atWorldEdge Method

The methods used in atWorldEdge 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.
- ||: Symbol for a conditional statement that means “OR”.

Call atWorldEdge Method in Subclass

- Open the Code editor for an Animal subclass.
- Create an IF statement that calls the atWorldEdge method as a condition. The condition tells the instance how many degrees to turn if the condition is true.
- Compile the code and run the scenario to test it.

```
public class Bee extends Animal
{
    /**
     * 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()
    {
        if (atWorldEdge())
        {
            turn(19);
        }
        move(2);
    }
}
```

Class Documentation

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

Method Summary	
void	act() Act - do whatever the Animal wants to do.
boolean	atWorldEdge() Test if object is close to one of the edges of the world.

Methods inherited from class
addedToWorld, getImage, getIntersectingObjects, getNeighbours, getObjectsAtOffset, getObjectsInRange, getOneIntersectingObject, getOneObjectAtOffset, getRotation, getWorld, getX, getY, intersects, move, setImage, setImage, setLocation, setRotation, turn

Methods inherited from class
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Defined Method to Eat 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 Animal superclass called canSee to enable objects of Animal subclasses to eat other objects.
- To create this defined method:
  - Declare a variable for the prey.
  - Use an assignment operator to set the value of the variable equal to the return value of the getOneObjectAtOffset method.

Define canSee Method

Define the canSee Method in the Animal superclass. This method returns true if the predator object (Duke) lands on a prey object (Code).

```
public class Animal extends Actor
{
    /**
     * Act - do whatever the Animal 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.
    }
    /**
     * Return true if Duke can see an object of the Code class 'clss'. Return false if there is no Code object here.
     */
    public boolean canSee(Class clss)
    {
        Actor actor = getOneObjectAtOffset(0, 0, clss);
        return actor != null;
    }
}
```

getOneObjectAtOffset Method

- The canSee Method uses the getOneObjectAtOffset method.
  - This method returns one object that is located at the specified cell (relative to this objects location).
  - Review this method in the Actor documentation.

```
    /**
     * Return true if Duke can see an object of the Code class 'clss'. Return false if there is no Code object here.
     */
    public boolean canSee(Class clss)
    {
        Actor actor = getOneObjectAtOffset(0, 0, clss);
        return actor != null;
    }
}
```

Define eat Method

Create a defined method in the Animal superclass called eat. This method will instruct Duke to eat the Code object if he lands on it.

```
/**
 * Eat an object of class 'clss' if there is such an object in our current location.
 * Otherwise do nothing.
 */
public void eat(Class clss)
{
    Actor actor = getOneObjectAtOffset(0, 0, clss);
    if(actor != null) {
        getWorld().removeObject(actor);
    }
}
```

Define lookForCode Method

Create a defined method in the animal subclass (Duke class) called lookForCode that checks if Duke has landed on a Code object. If so, Duke will eat the Code object.

```
public class Duke extends Animal
{
    /**
     * Act - do whatever the Duke 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.
    }
    /**
     * Look for code. If Duke finds code, he eats it. Otherwise, he does nothing.
     */
    public void lookForCode()
    {
        if (canSee(Code.class))
        {
            eat(Code.class);
        }
    }
}
```

Call lookForCode in Act Method

Call the new lookForCode method in Duke's act method. Run the animation to test the code.

```
public class Duke extends Animal
{
    /**
     * Act - do whatever the Duke wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        move(3);
        if (Greenfoot.isKeyDown("left"))
        {
            turn(-2);
        }
        if (Greenfoot.isKeyDown("right"))
        {
            turn(2);
        }
        lookForCode();
    }
```

Terminology

Key terms used in this lesson included:

- Defined methods

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

Practice

The exercises for this lesson cover the following topics:

- Examining and simplifying programming by creating and calling defined methods