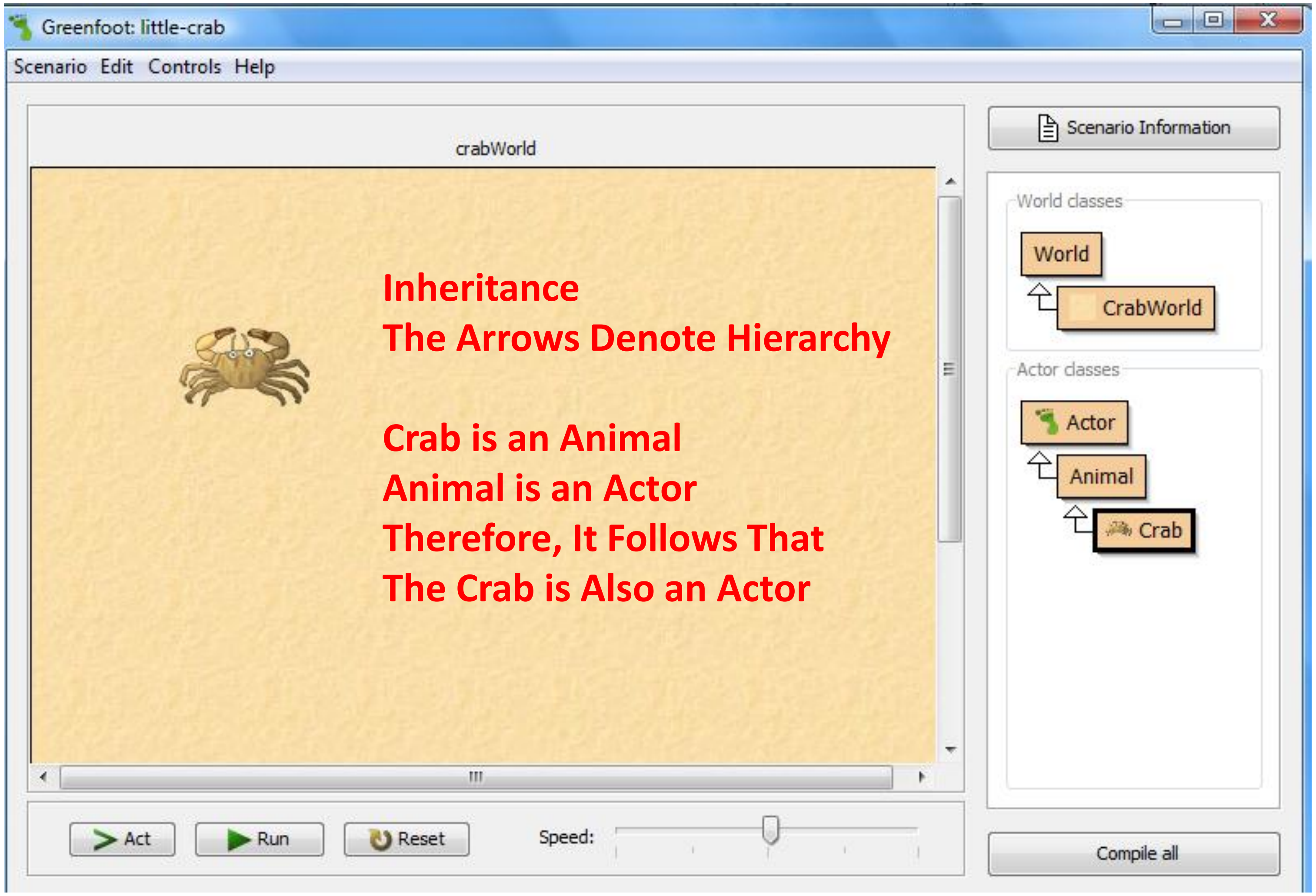
Chapter 2 - The First Program: Little Crab

Acknowledgement: Michael Kolling, Bruce Chittenden

Announcement

- Written Exams
  - Exam 1 : July 11th 3:00 PM – 4:00 PM
  - Exam 2 : July 18th 4:00 PM – 4:50 PM
  - Exam 3 : July 30th 4:00 PM – 4:50 PM
- Practical Exams
  - Exam 1 : July 11th 4:00 PM – 6:00 PM
  - Exam 2 : Aug 1st 2:00 PM – 5:00 PM
- Lab 1 posted on course website and UBLearns

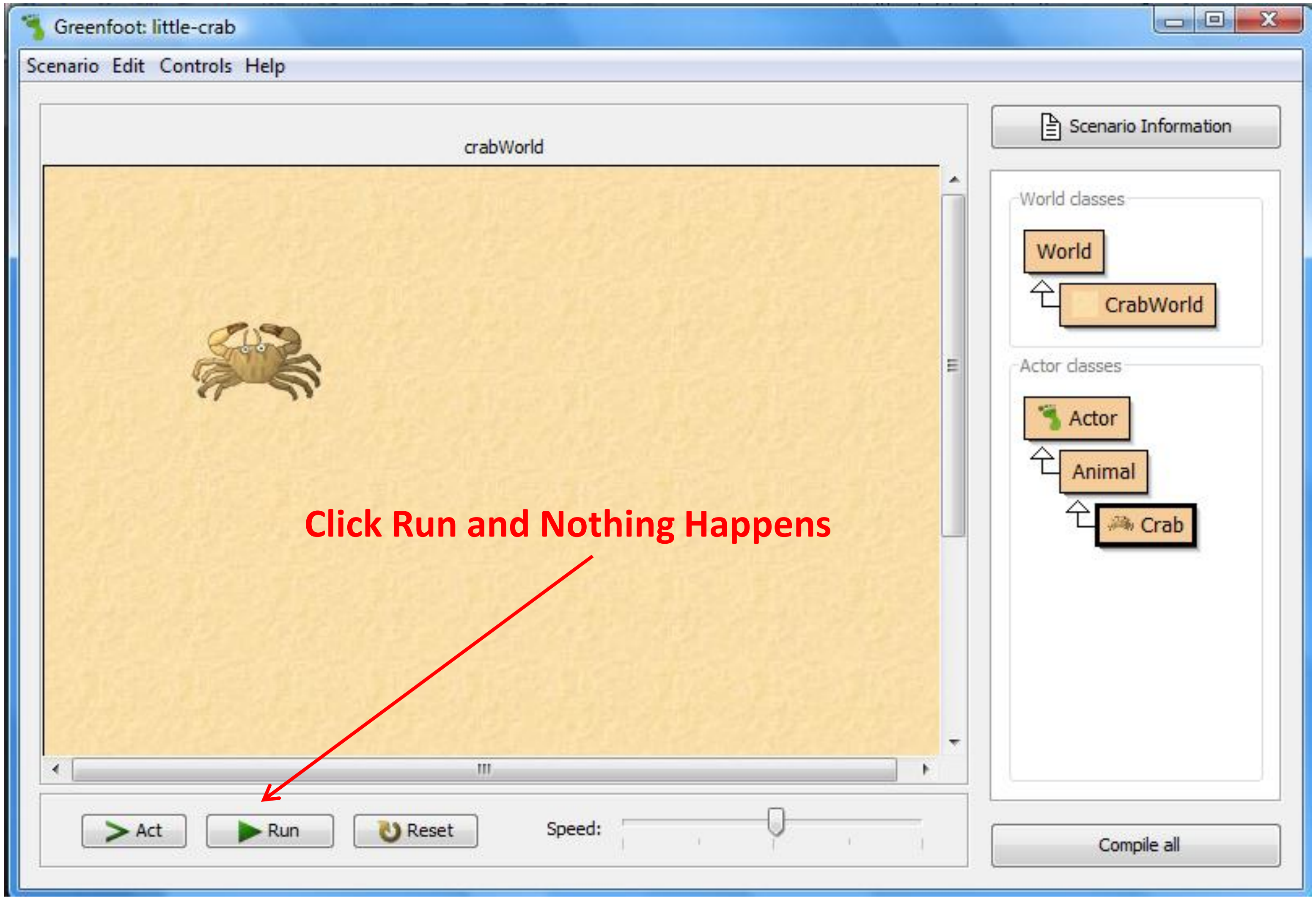
2.1 Little Crab Scenario



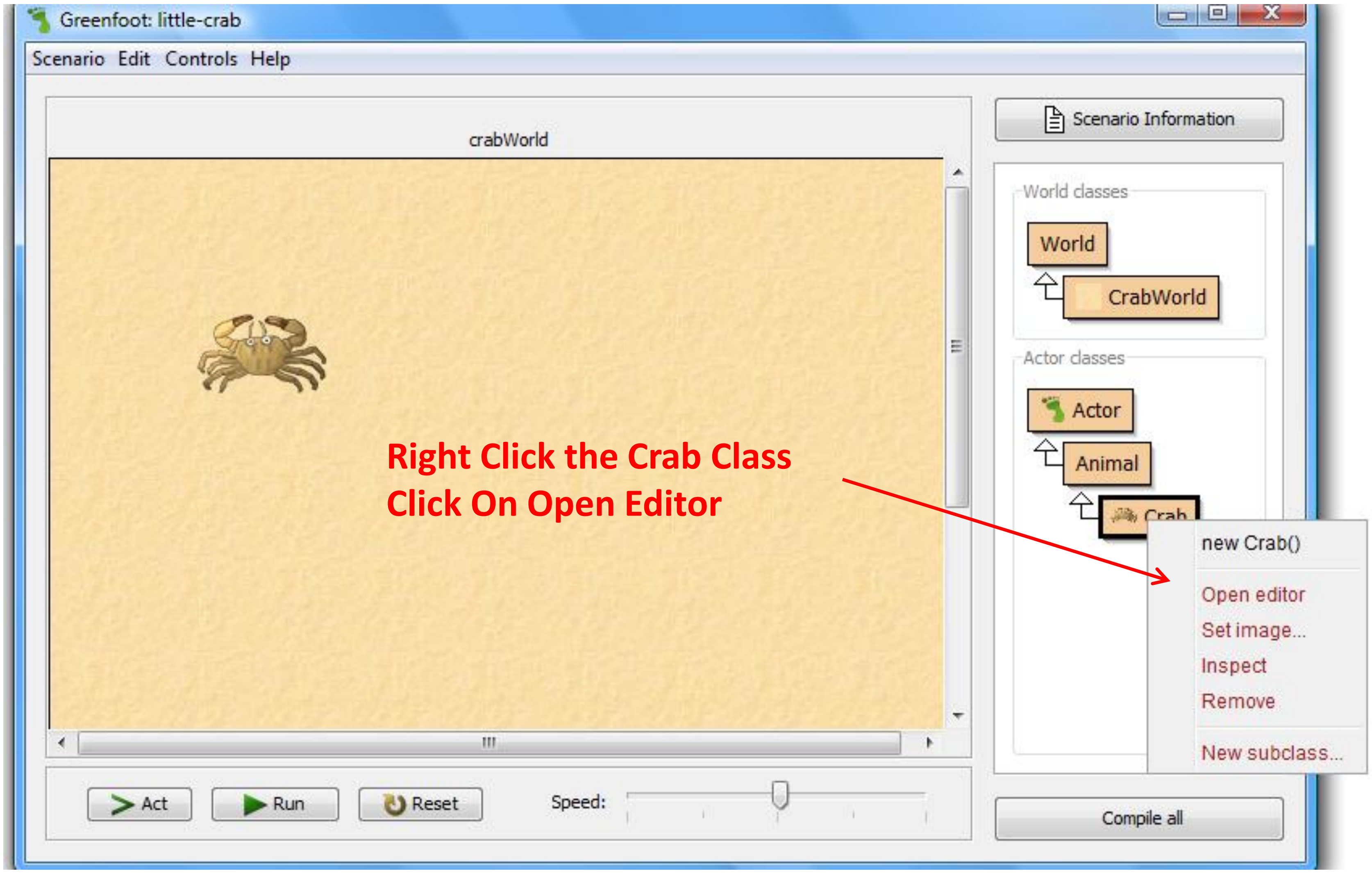
Inheritance
The Arrows Denote Hierarchy

Crab is an Animal
Animal is an Actor
Therefore, It Follows That
The Crab is Also an Actor

Exercise 2.1

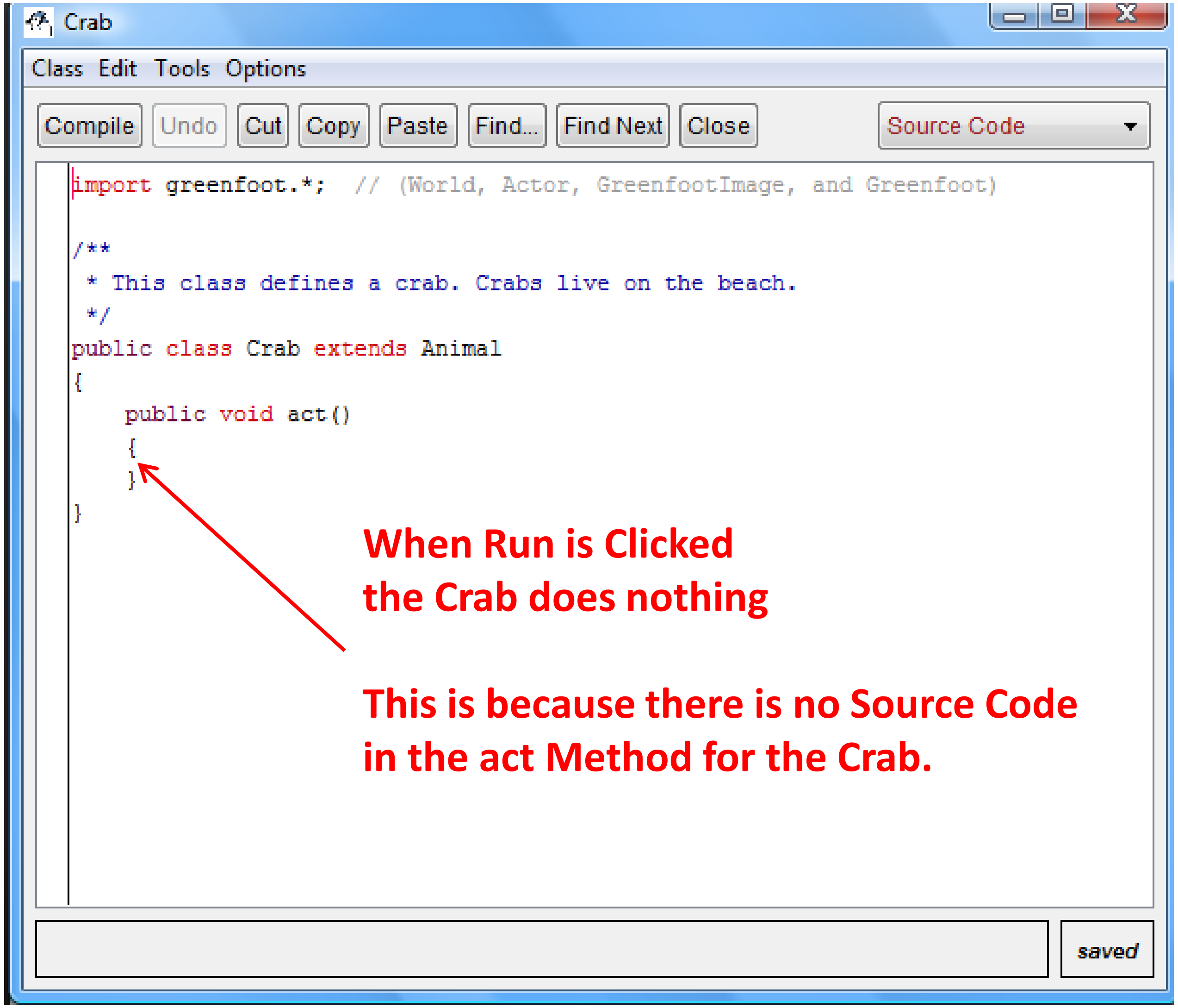

Greenfoot: little-crab
Scenario Edit Controls Help
crabWorld
Scenario Information
World classes
World
CrabWorld
Actor classes
Actor
Animal
Crab
Click Run and Nothing Happens
Act
Run
Reset
Speed:
Compile all

Exercise 2.1



Right Click the Crab Class
Click On Open Editor

Exercise 2.1



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

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
    }
}
```

When Run is Clicked the Crab does nothing

This is because there is no Source Code in the act Method for the Crab.

2.2 Making the Crab Move

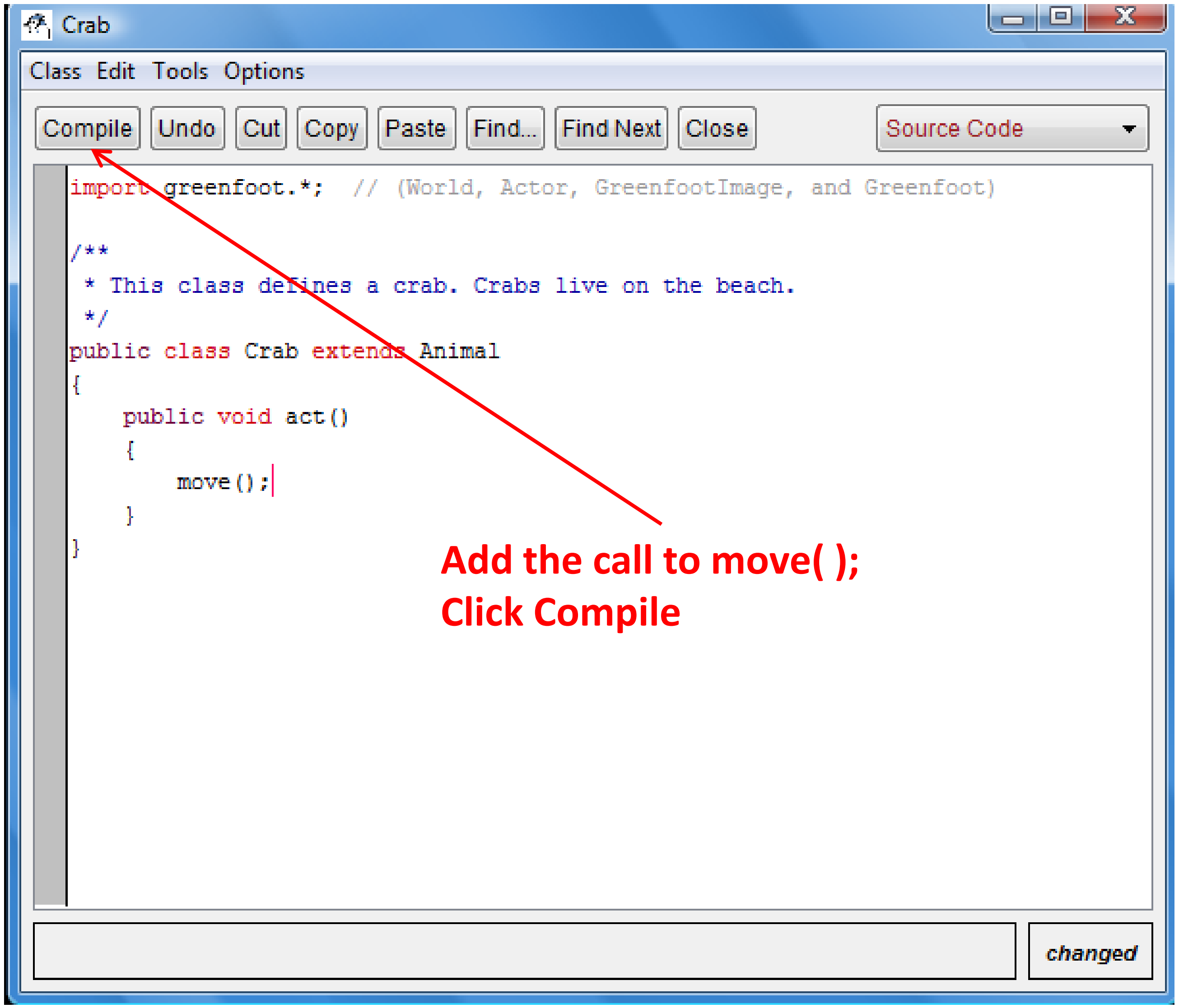
```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/*
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
        public void act()
        {
                // Add your action code here
        }
}
```

Replace the comment with move();

Code 2.1

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

/*
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
        public void act()
        {
                move();
        }
}
```

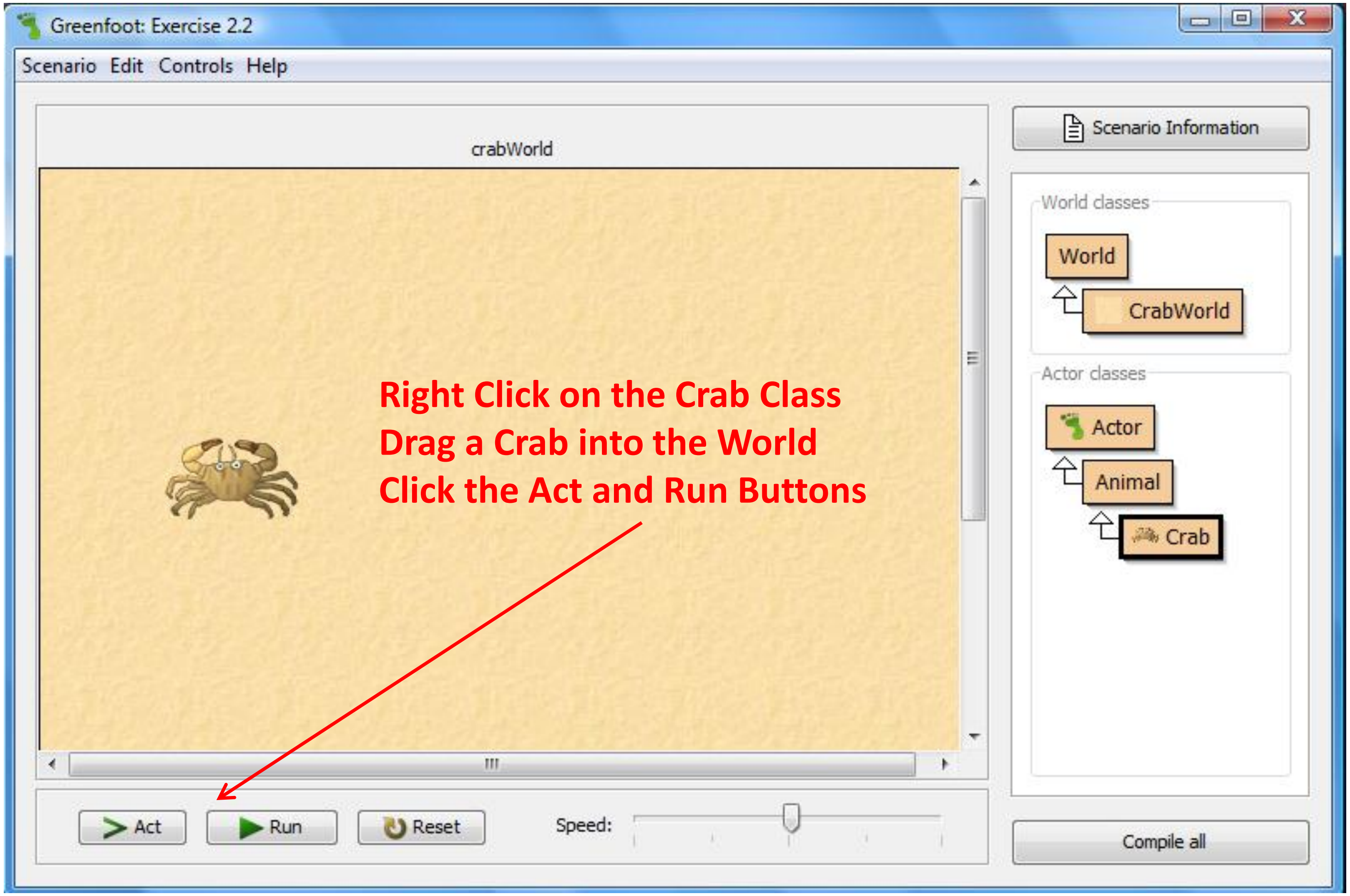
Exercise 2.2



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

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        move();
    }
}
```

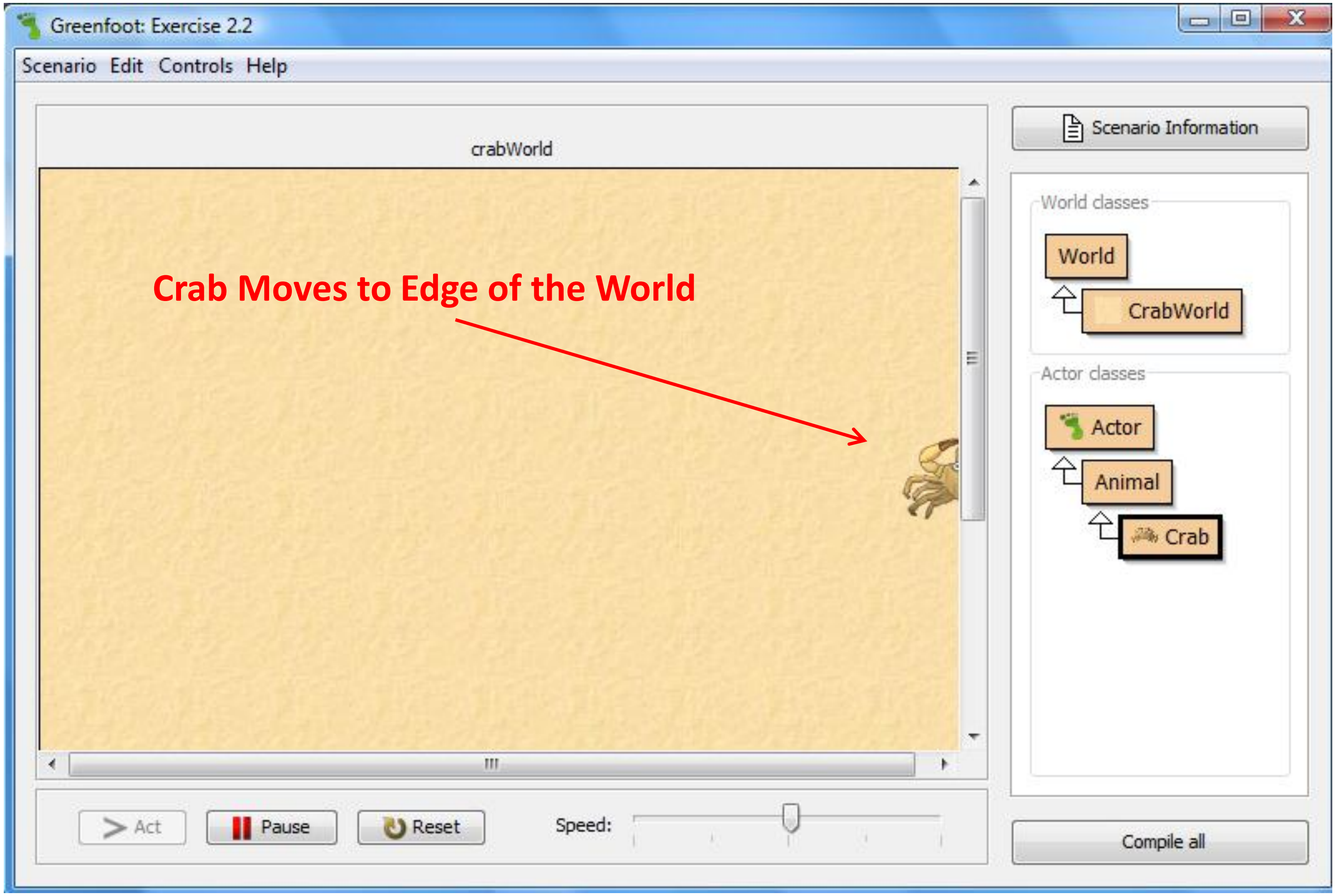
Add the call to move( );
Click Compile

Exercise 2.2

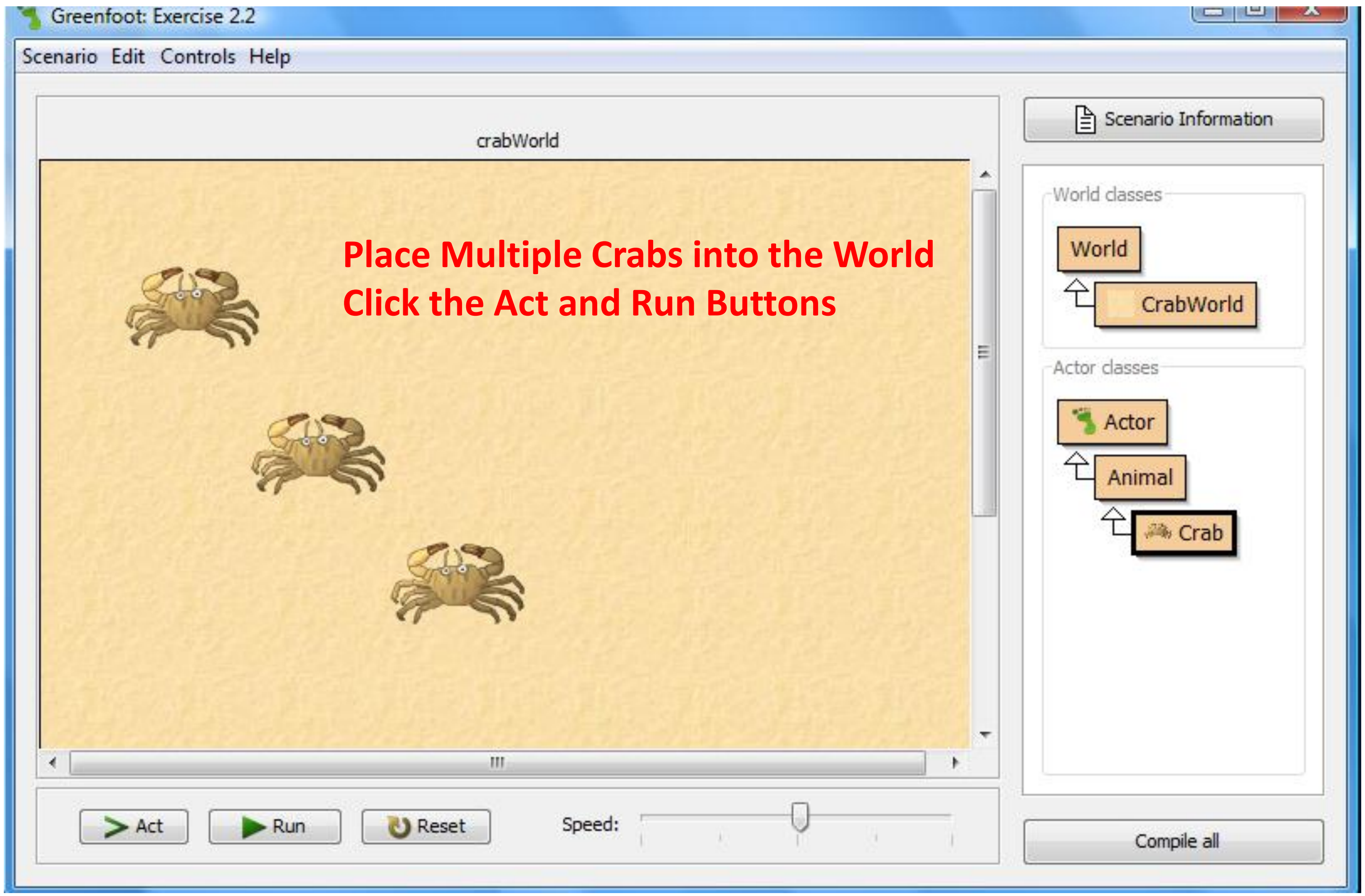


Right Click on the Crab Class
Drag a Crab into the World
Click the Act and Run Buttons

Exercise 2.2

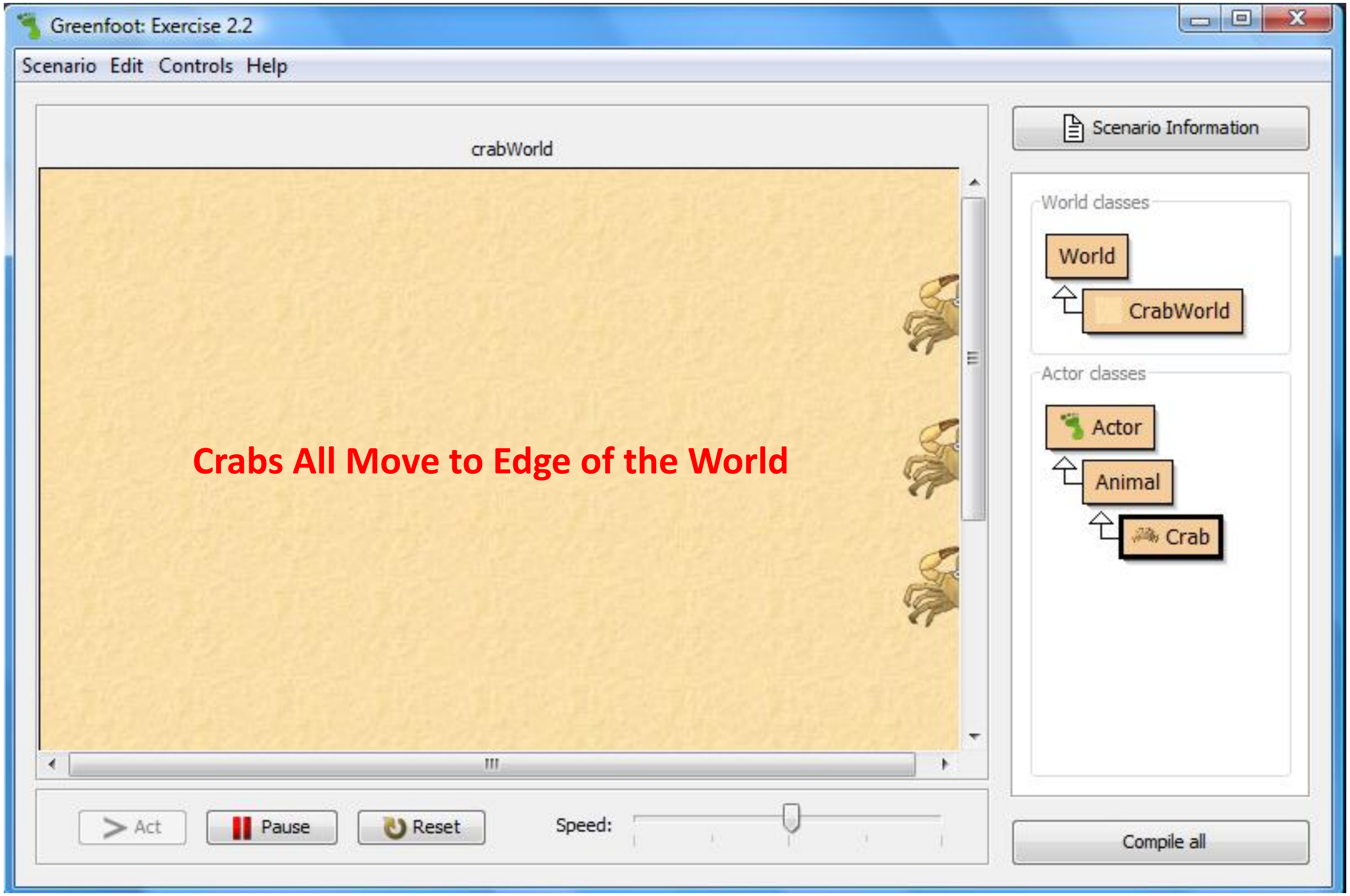

Greenfoot: Exercise 2.2
Scenario Edit Controls Help
crabWorld
Crab Moves to Edge of the World
Scenario Information
World classes
World
CrabWorld
Actor classes
Actor
Animal
Crab
Act
Pause
Reset
Speed:
Compile all

Exercise 2.3



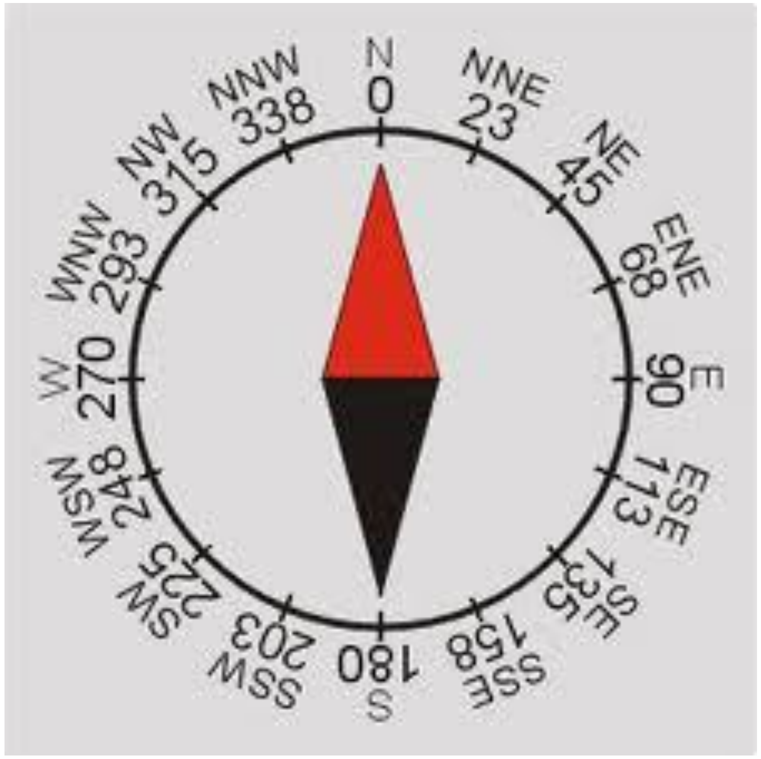
Place Multiple Crabs into the World
Click the Act and Run Buttons

Exercise 2.3



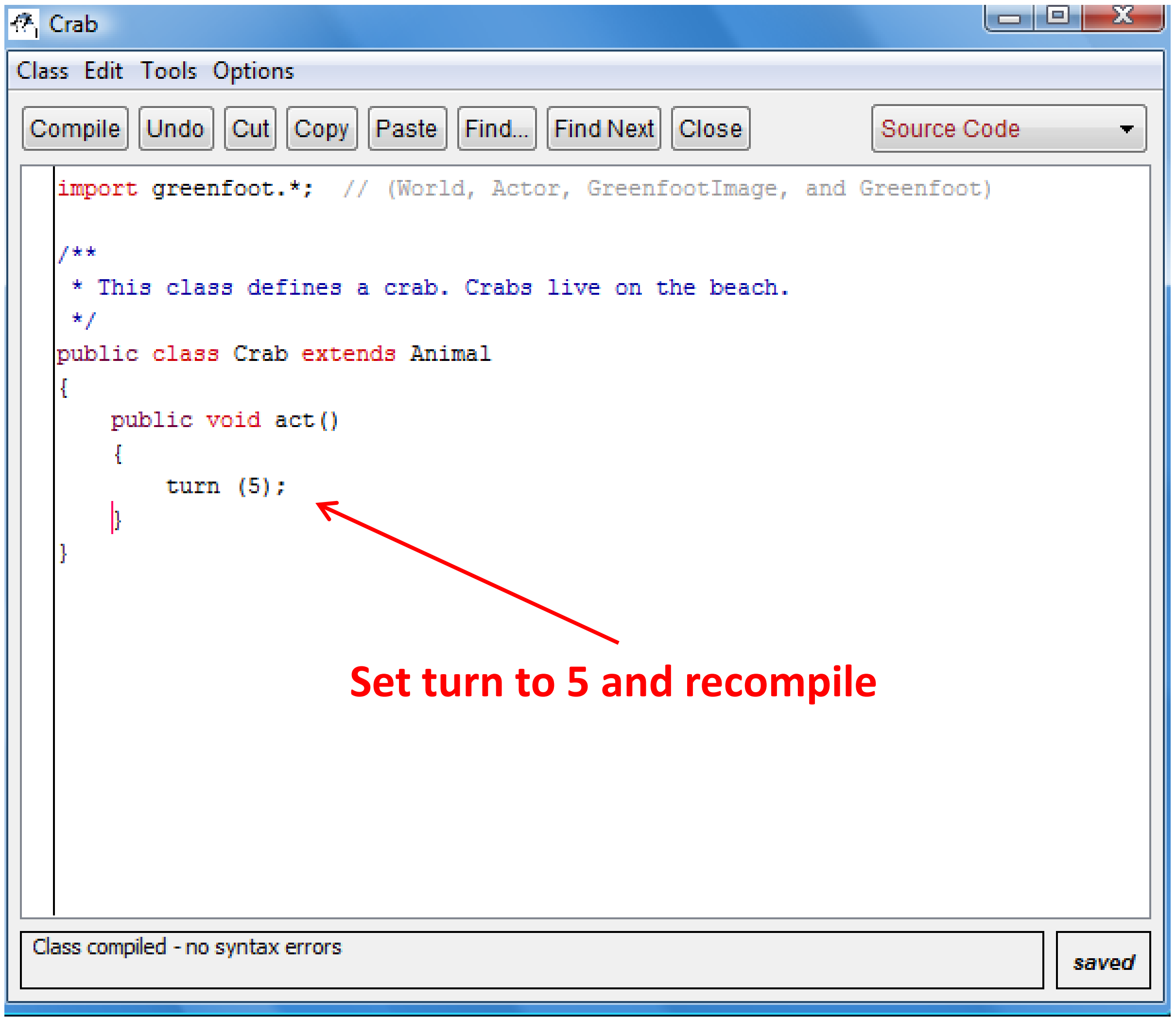
Crabs All Move to Edge of the World

2.3 Turning



```
public void act ()
{
        turn (5);
}
```

Exercise 2.4



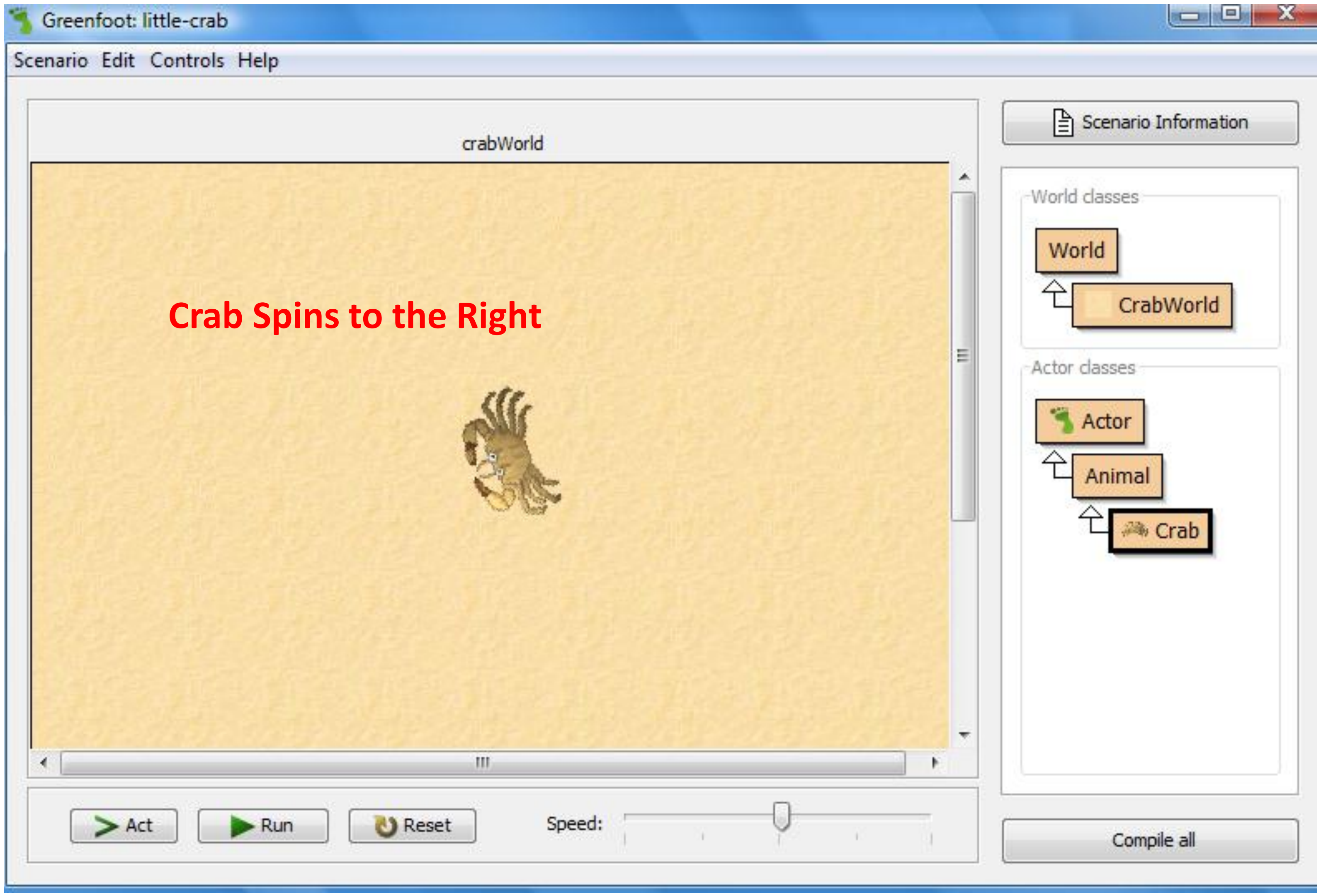
```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        turn (5);
    }
}
```

Set turn to 5 and recompile

Class compiled - no syntax errors

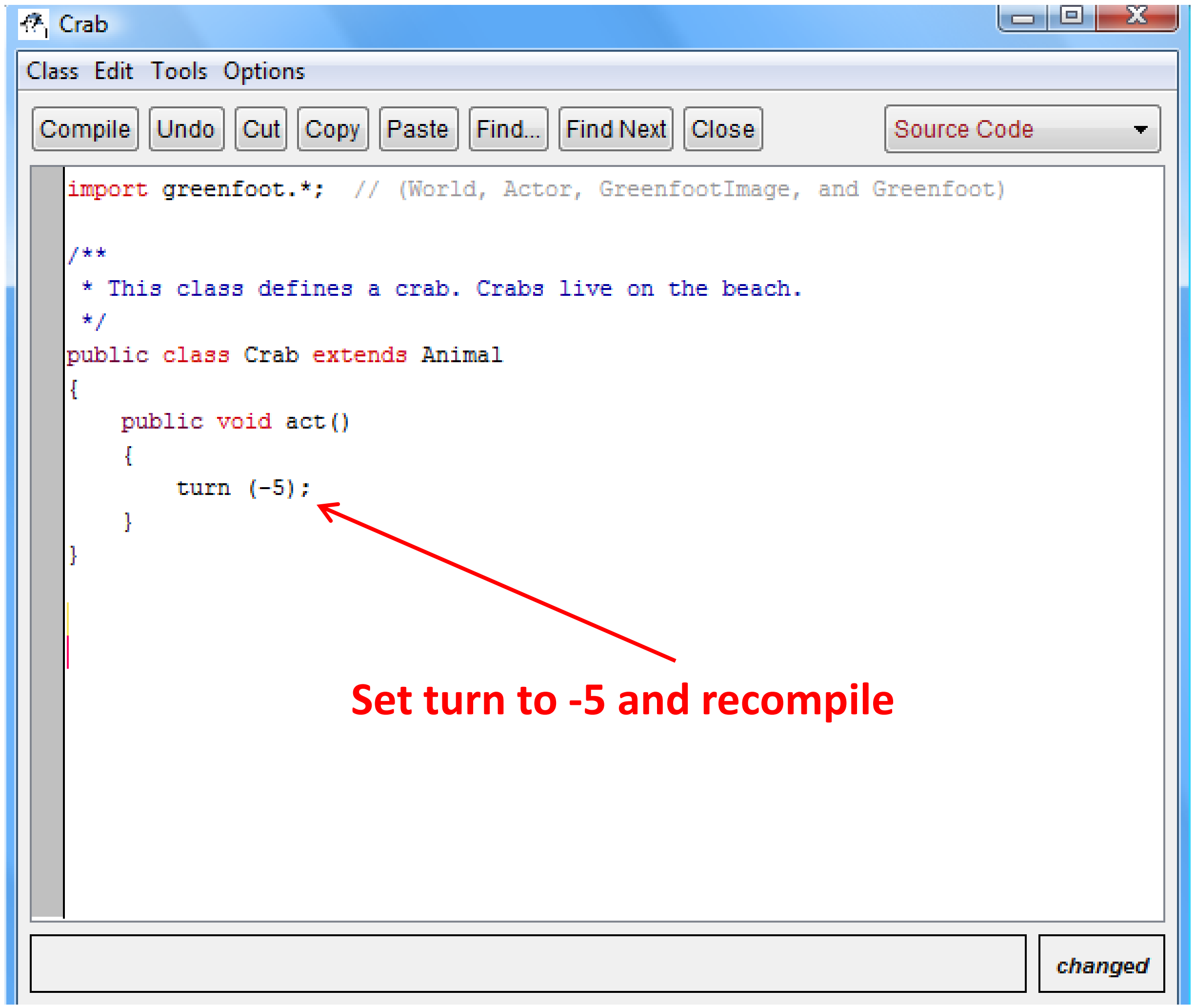
saved

Exercise 2.4



Crab Spins to the Right

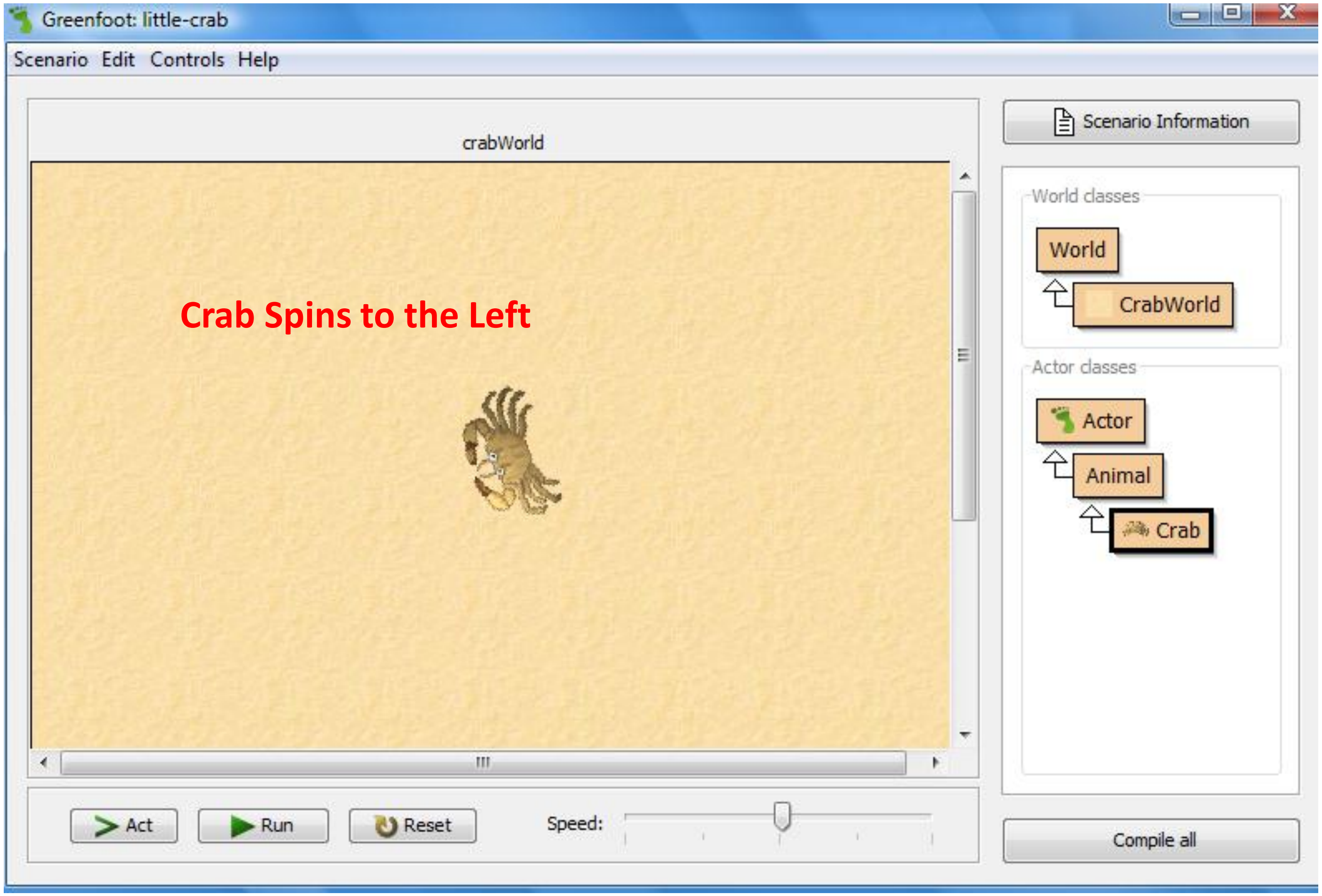
Exercise 2.5



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

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        turn (-5);
    }
}
```

Set turn to -5 and recompile

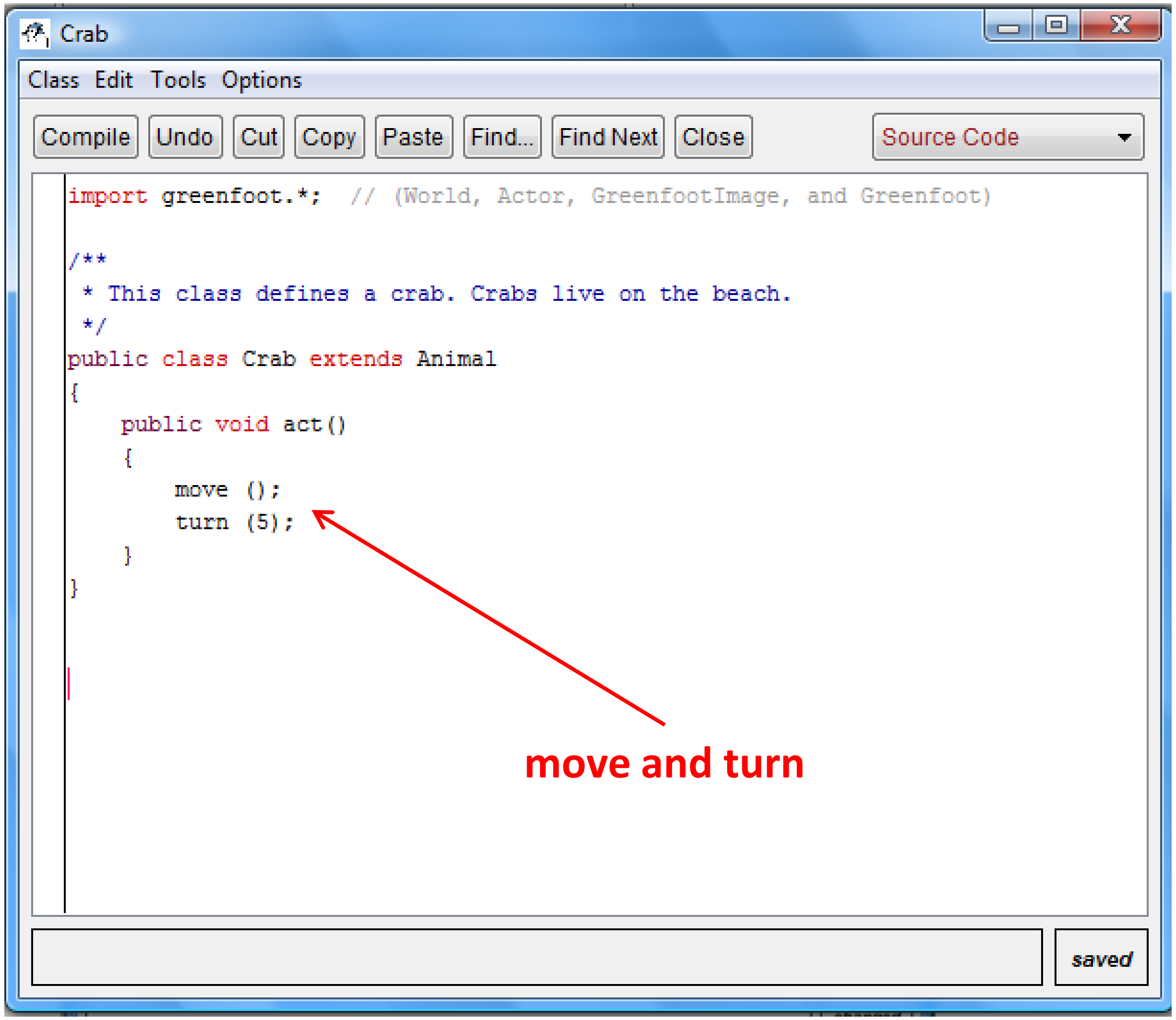
Exercise 2.5


Greenfoot: little-crab
Scenario
Edit
Controls
Help
crabWorld
Crab Spins to the Left
Scenario Information
World classes
World
CrabWorld
Actor classes
Actor
Animal
Crab
Act
Run
Reset
Speed:
Compile all

Code 2.2

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

/*
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
	public void act()
	{
		move ();
		turn (5);
	}
}
```

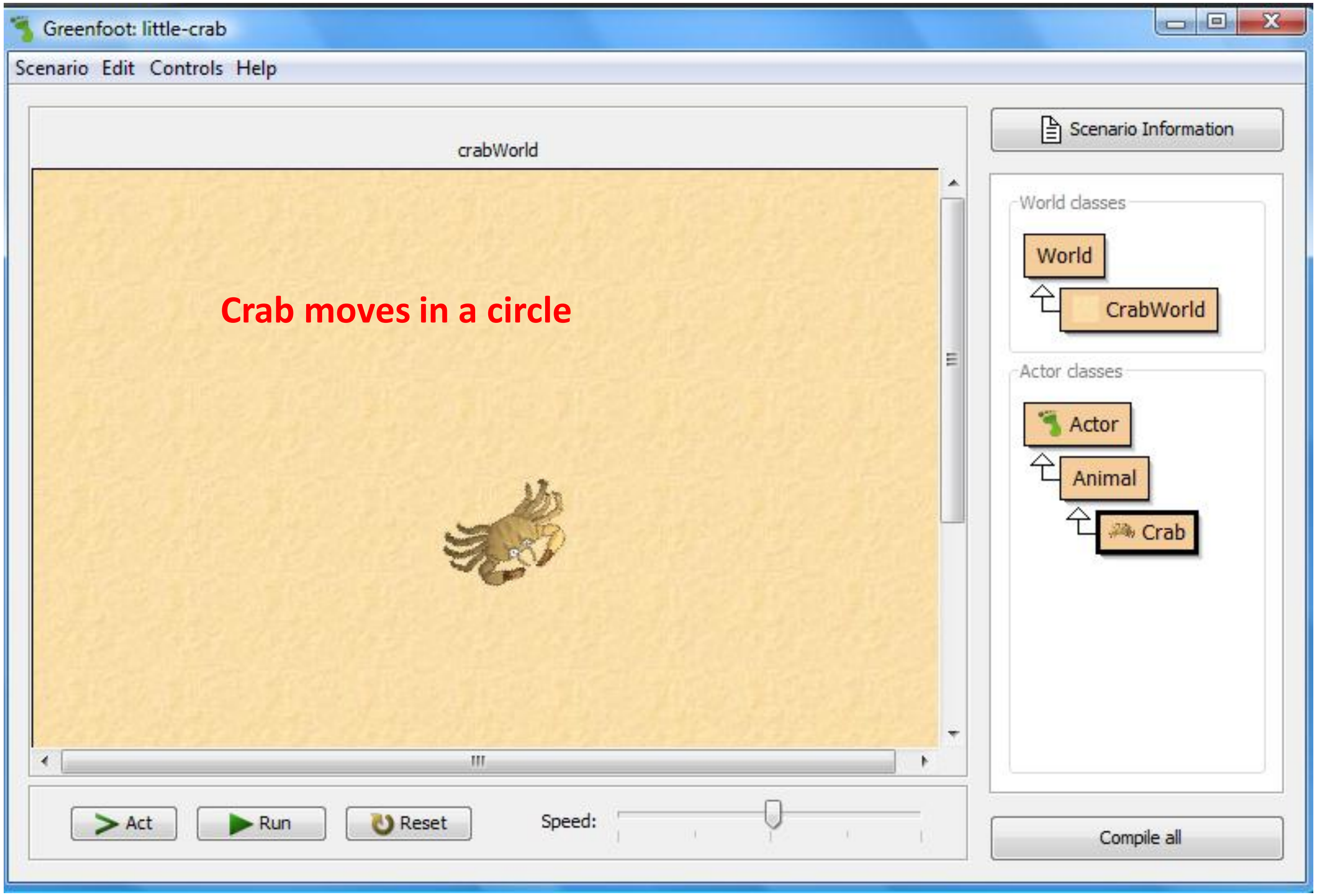
Exercise 2.6



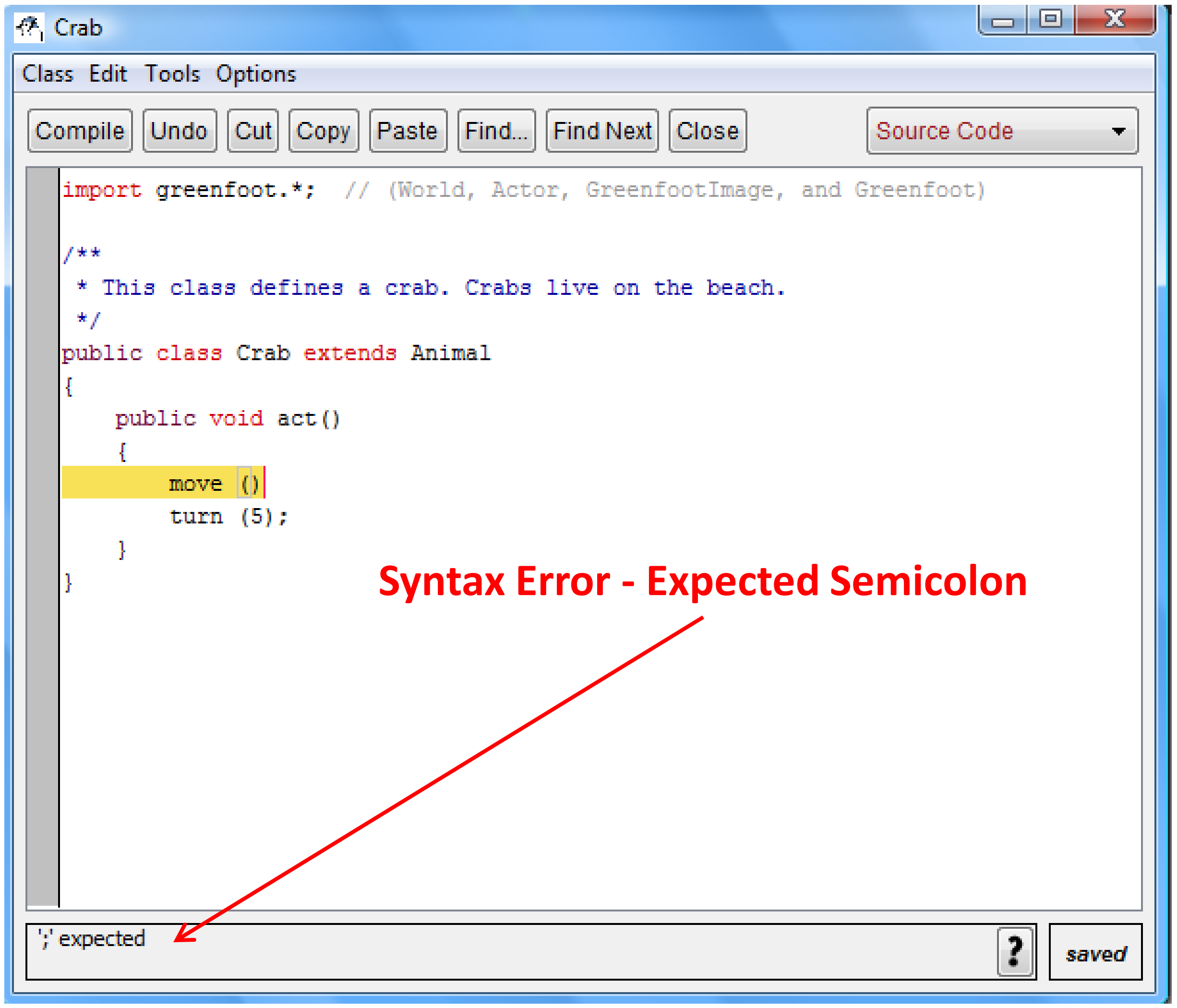
```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        move ();
        turn (5);
    }
}
```

move and turn

Exercise 2.6


Greenfoot: little-crab
Scenario
Edit
Controls
Help
crabWorld
Crab moves in a circle
Scenario Information
World classes
World
CrabWorld
Actor classes
Actor
Animal
Crab
Act
Run
Reset
Speed:
Compile all

Exercise 2.7



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

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        move ()
        turn (5);
    }
}
```

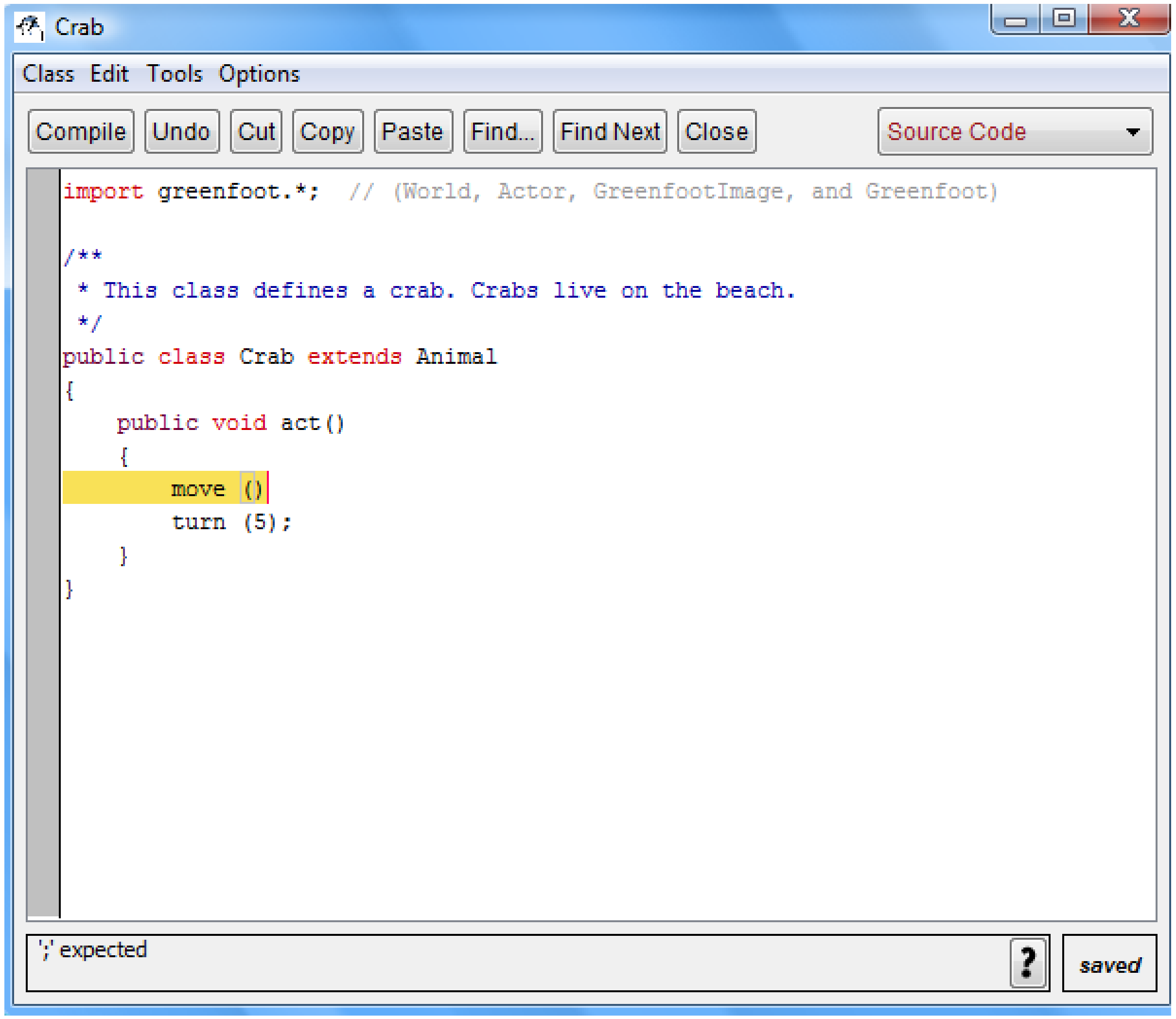
';' expected

Syntax Error - Expected Semicolon

Exercise 2.8

1. ‘;’ expected
2. cannot find symbol - method Move()
3. turn(int) in Animal cannot be applied to (int,int)
4. turn(int) in Animal cannot be applied to (double)
5. act() in Crab cannot override act() in Animal; attempting to use incompatible return type
6. cannot find symbol - class voids
7. unclosed comment
8. class, interface, or enum expected
9. illegal start of type
10. illegal start of expression

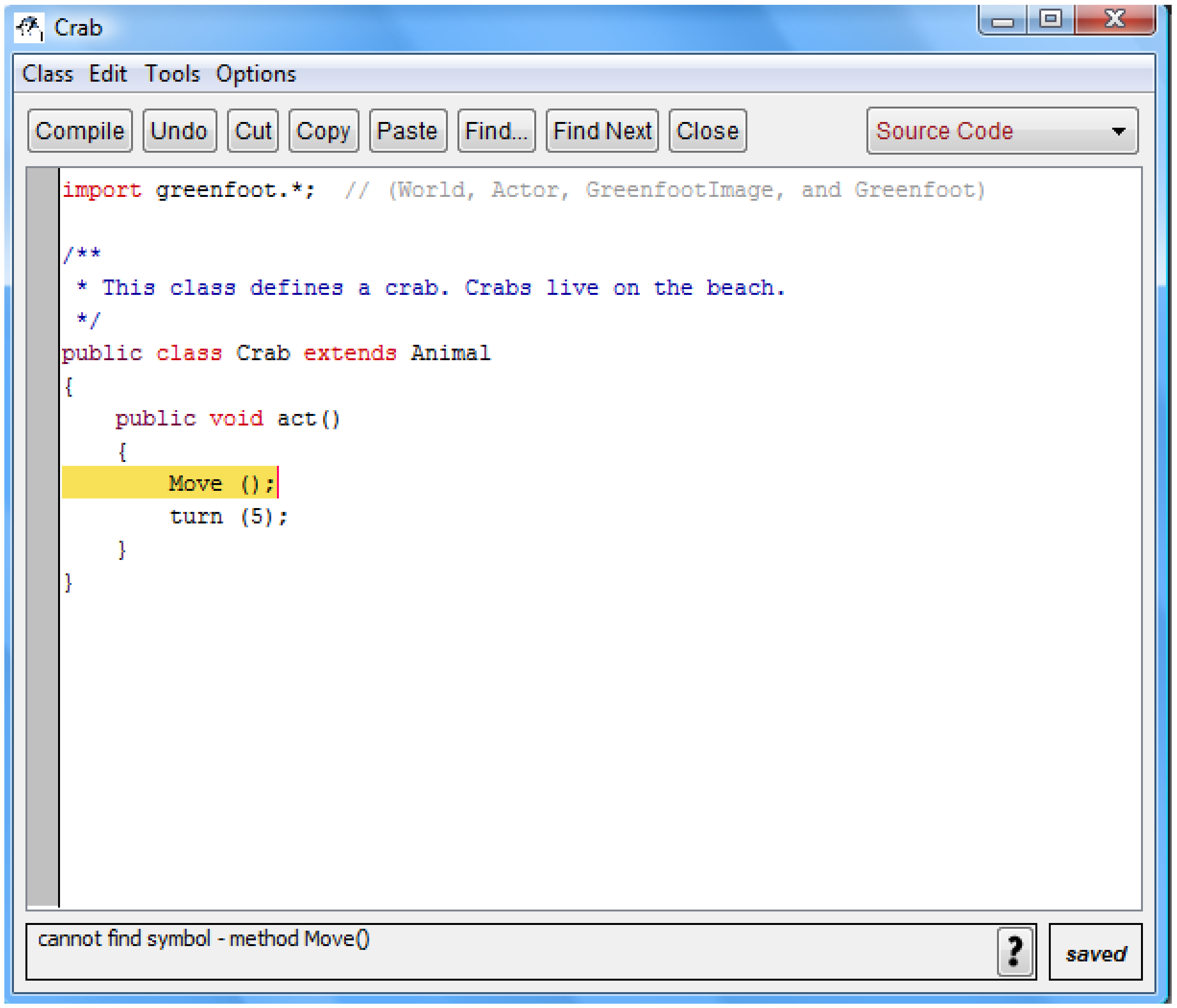
Exercise 2.8



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

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        move ()
        turn (5);
    }
}
```

';' expected

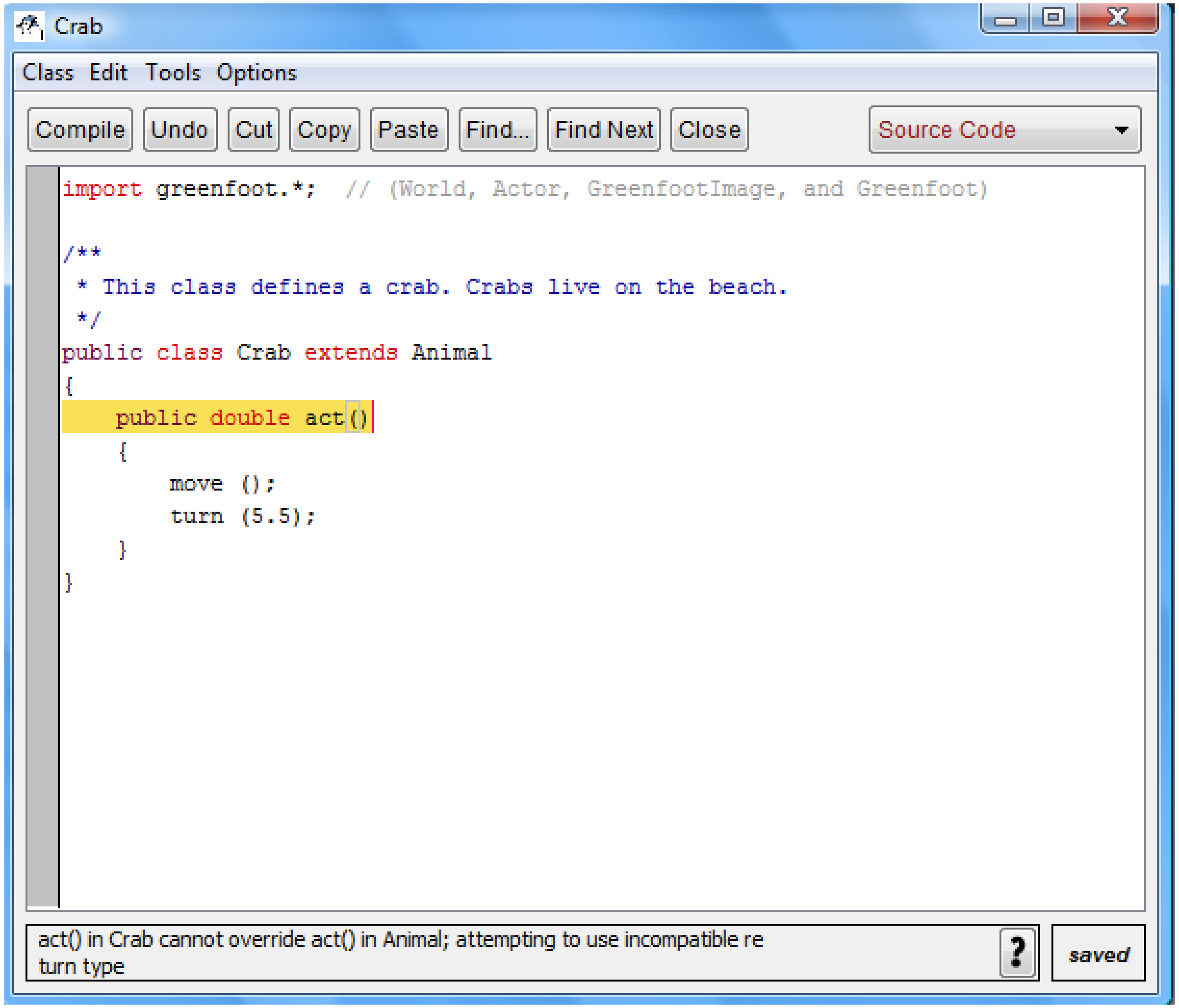
Exercise 2.8



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

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        Move ();
        turn (5);
    }
}
```

cannot find symbol - method Move()

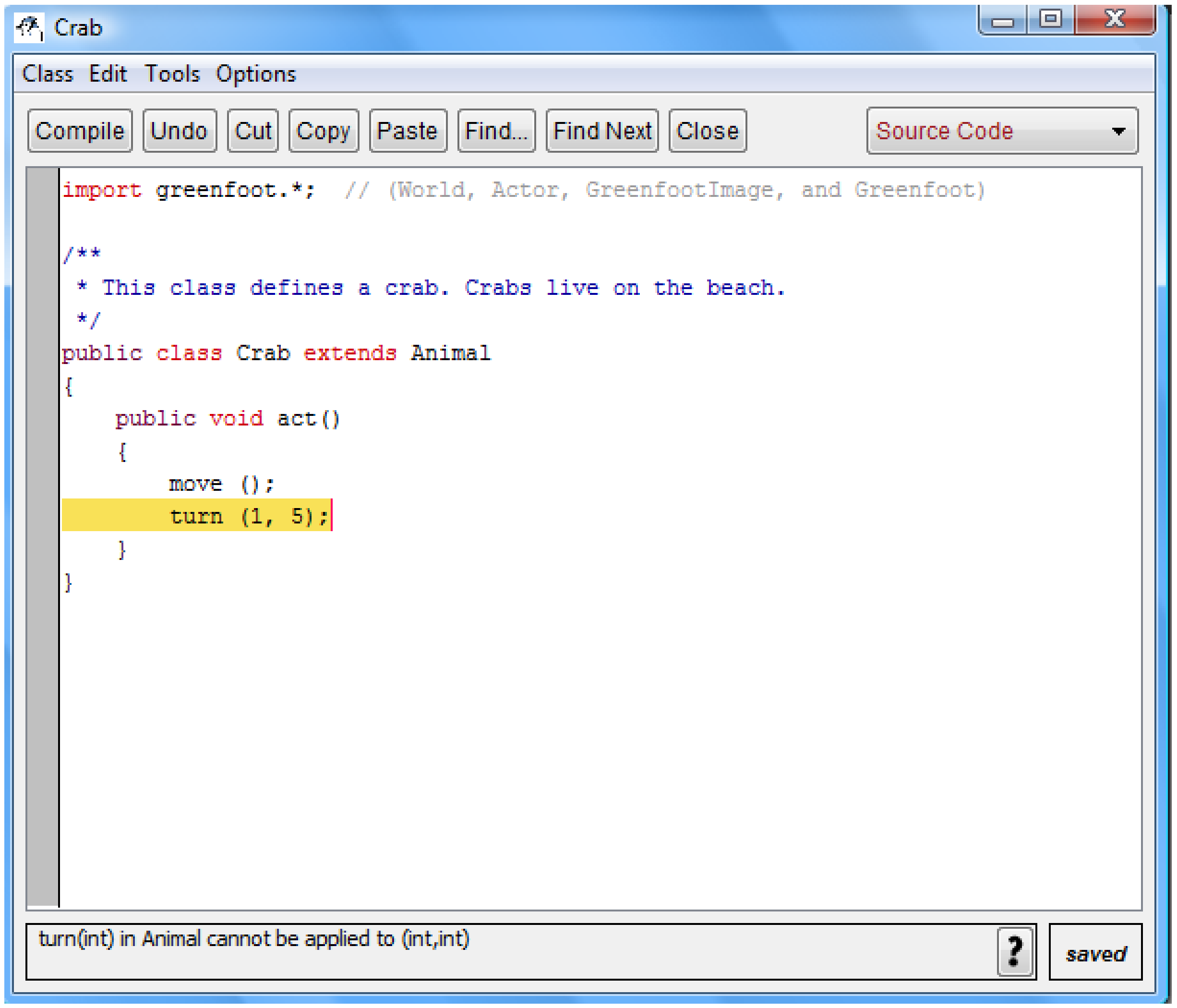
Exercise 2.8



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

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public double act()
    {
        move ();
        turn (5.5);
    }
}
```

act() in Crab cannot override act() in Animal; attempting to use incompatible re
turn type

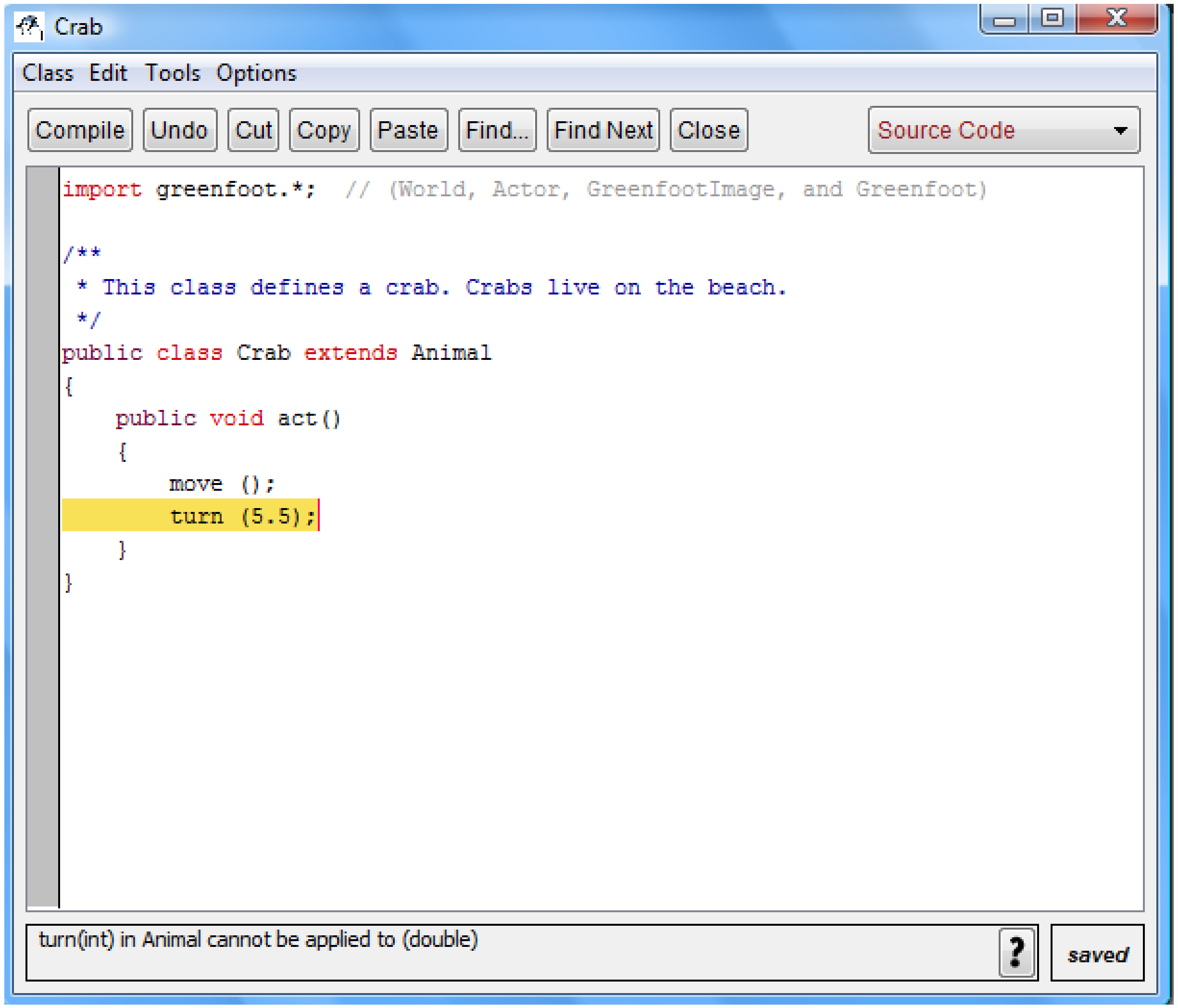
Exercise 2.8



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

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        move ();
        turn (1, 5);
    }
}
```

```
turn(int) in Animal cannot be applied to (int,int)
```

Exercise 2.8

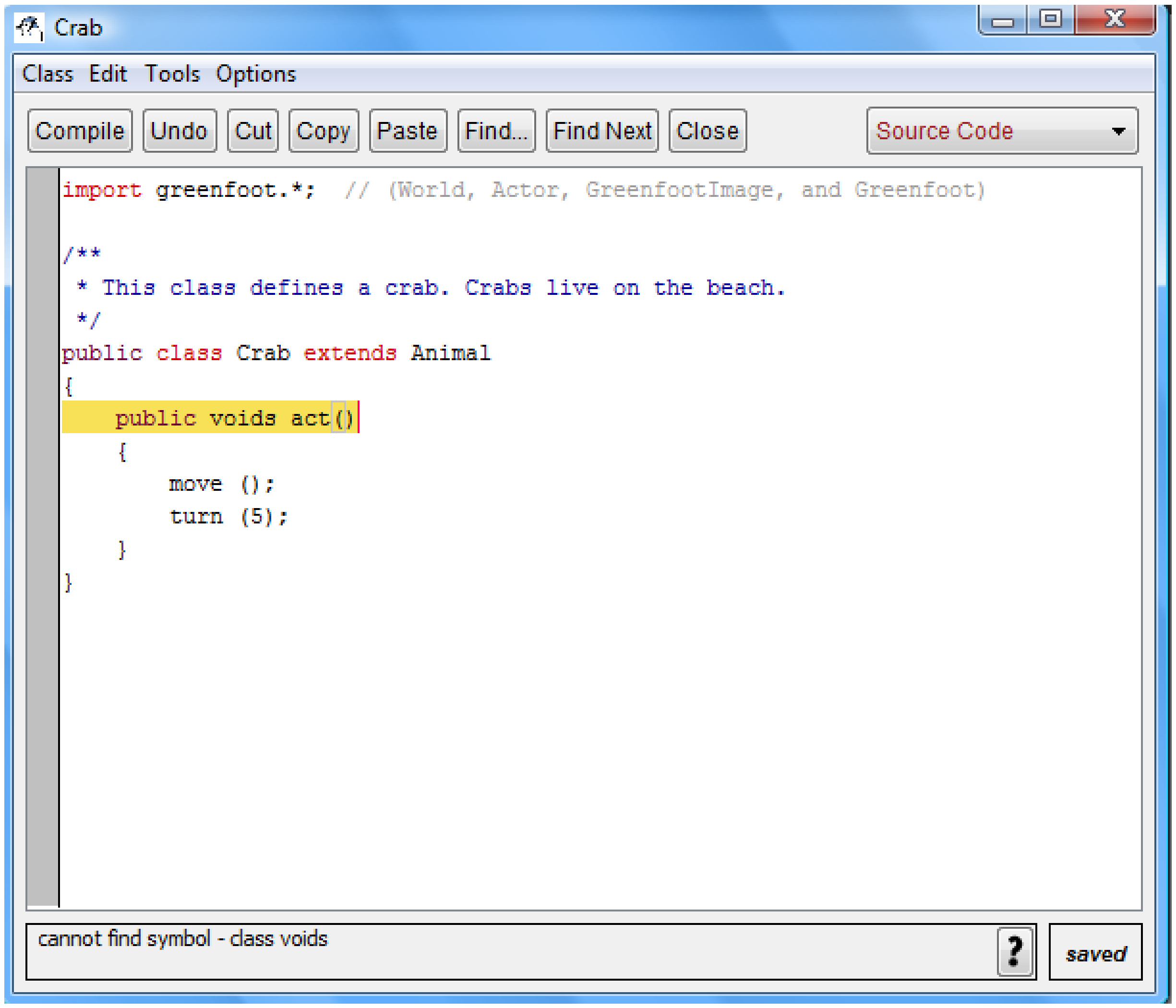


```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        move ();
        turn (5.5);
    }
}
```

turn(int) in Animal cannot be applied to (double)

saved

Exercise 2.8

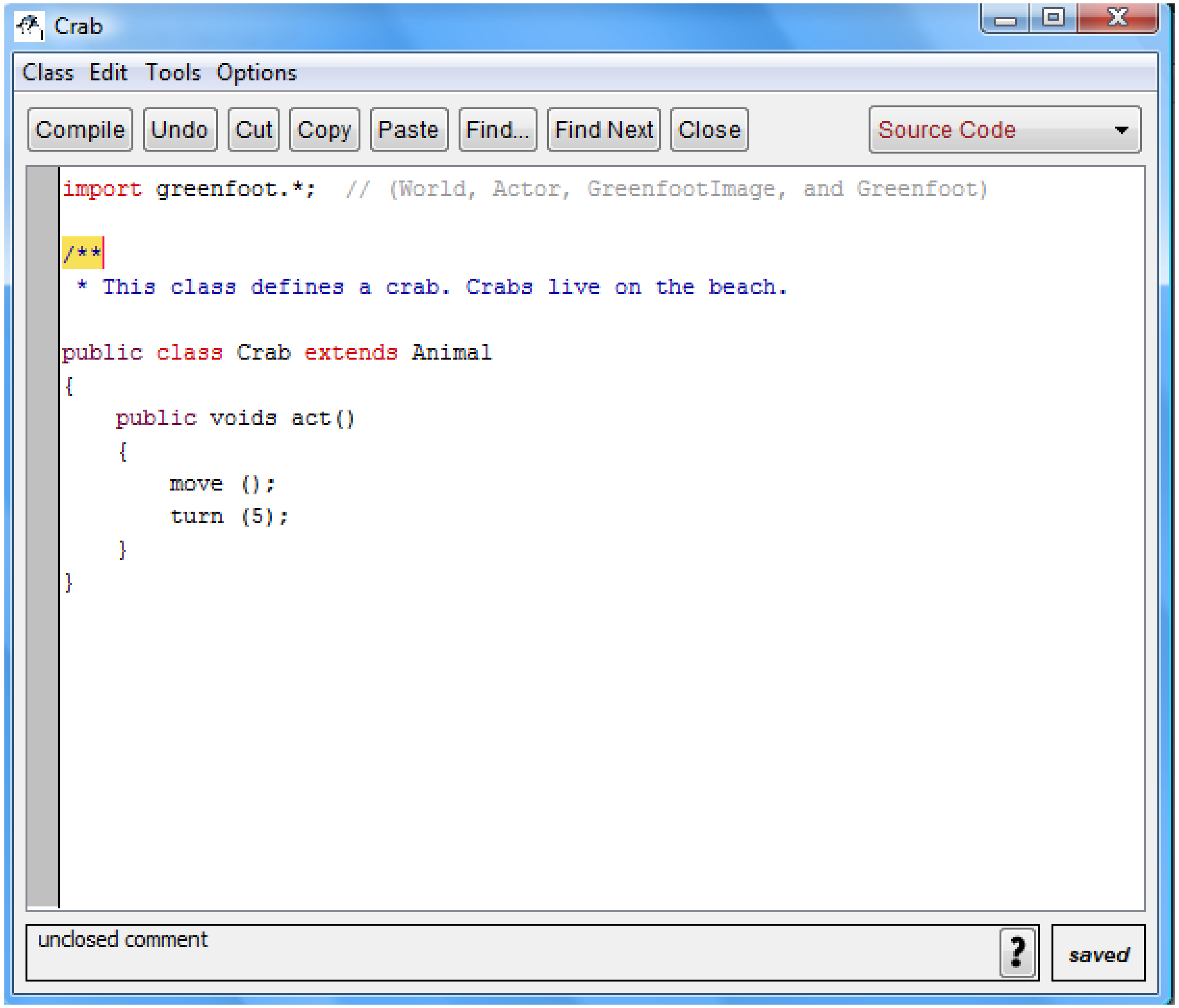


```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public voids act()
    {
        move ();
        turn (5);
    }
}
```

cannot find symbol - class voids

saved

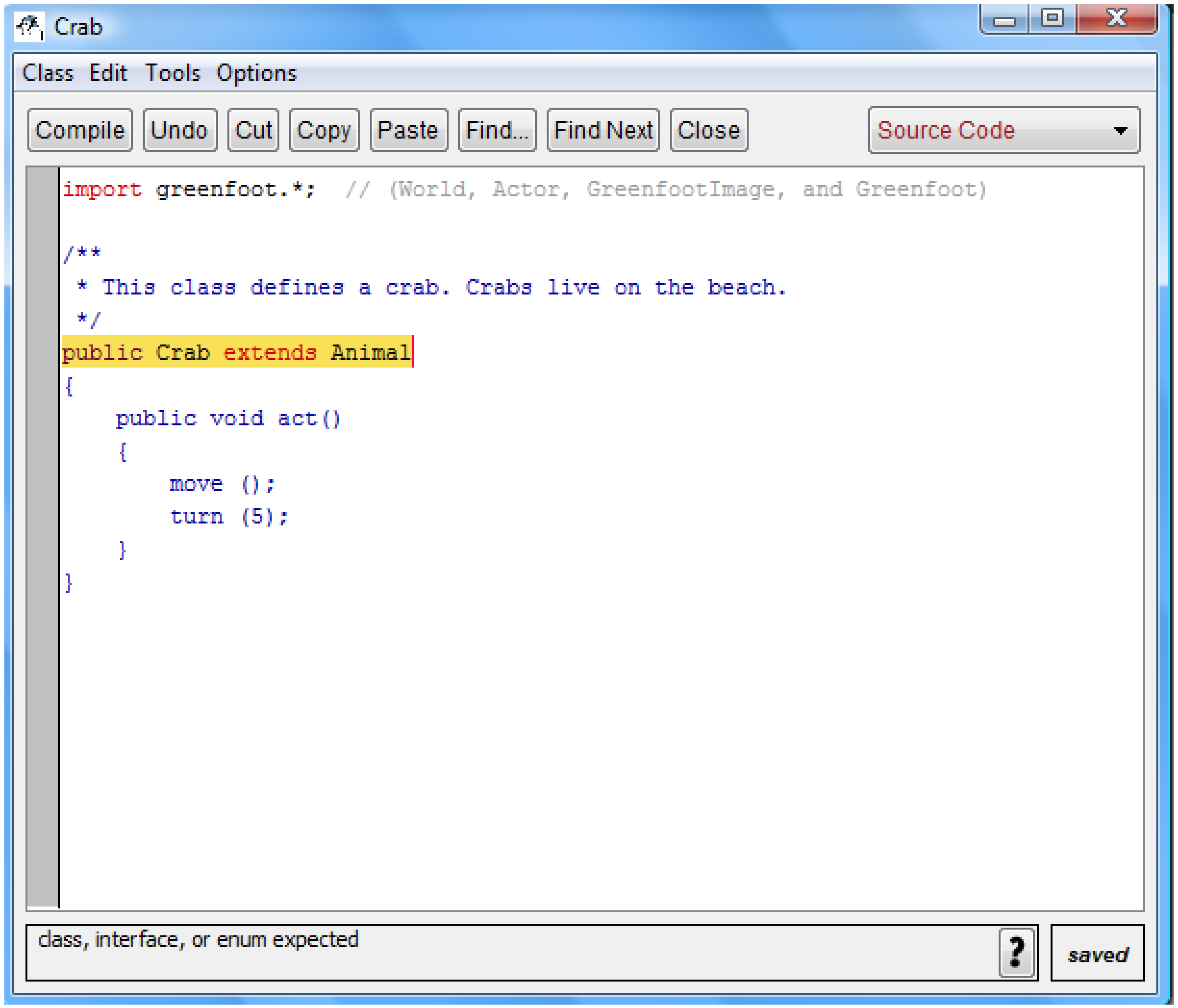
Exercise 2.8



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

/**
 * This class defines a crab. Crabs live on the beach.

public class Crab extends Animal
{
    public voids act()
    {
        move ();
        turn (5);
    }
}
```

unclosed comment

Exercise 2.8

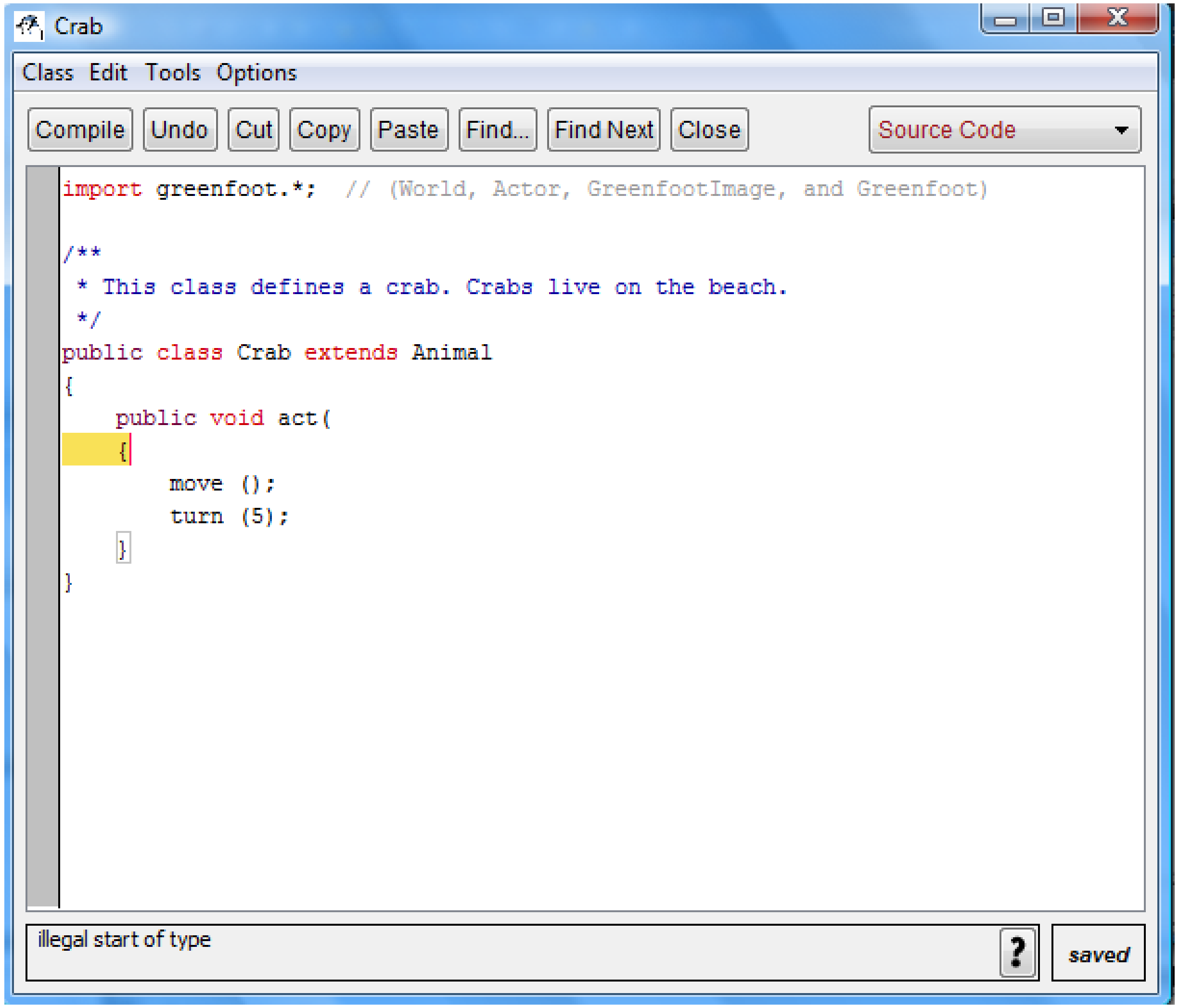


```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * This class defines a crab. Crabs live on the beach.
 */
public Crab extends Animal
{
    public void act()
    {
        move ();
        turn (5);
    }
}
```

class, interface, or enum expected

saved

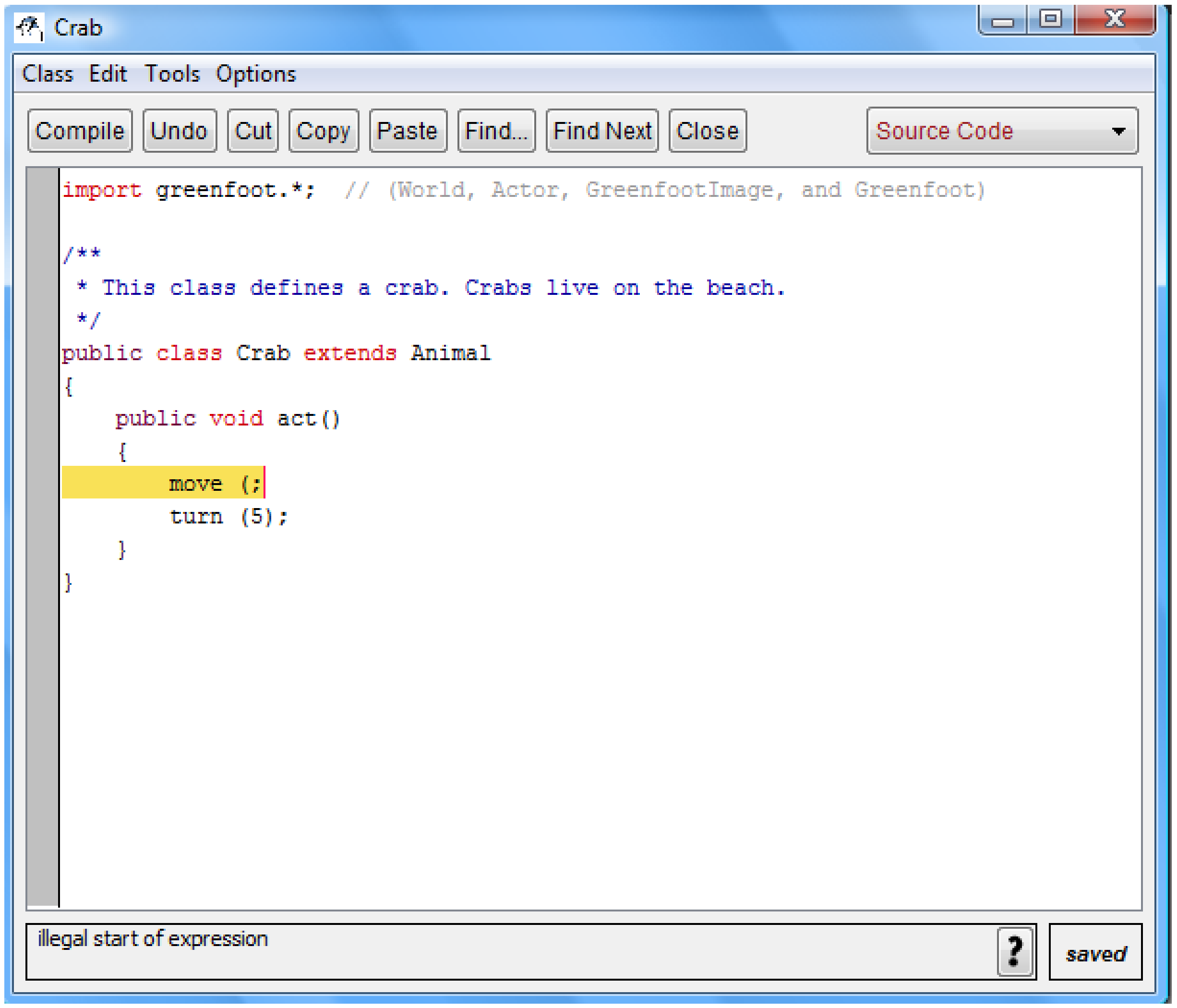
Exercise 2.8



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

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act(
    {
        move ();
        turn (5);
    }
}
```

illegal start of type

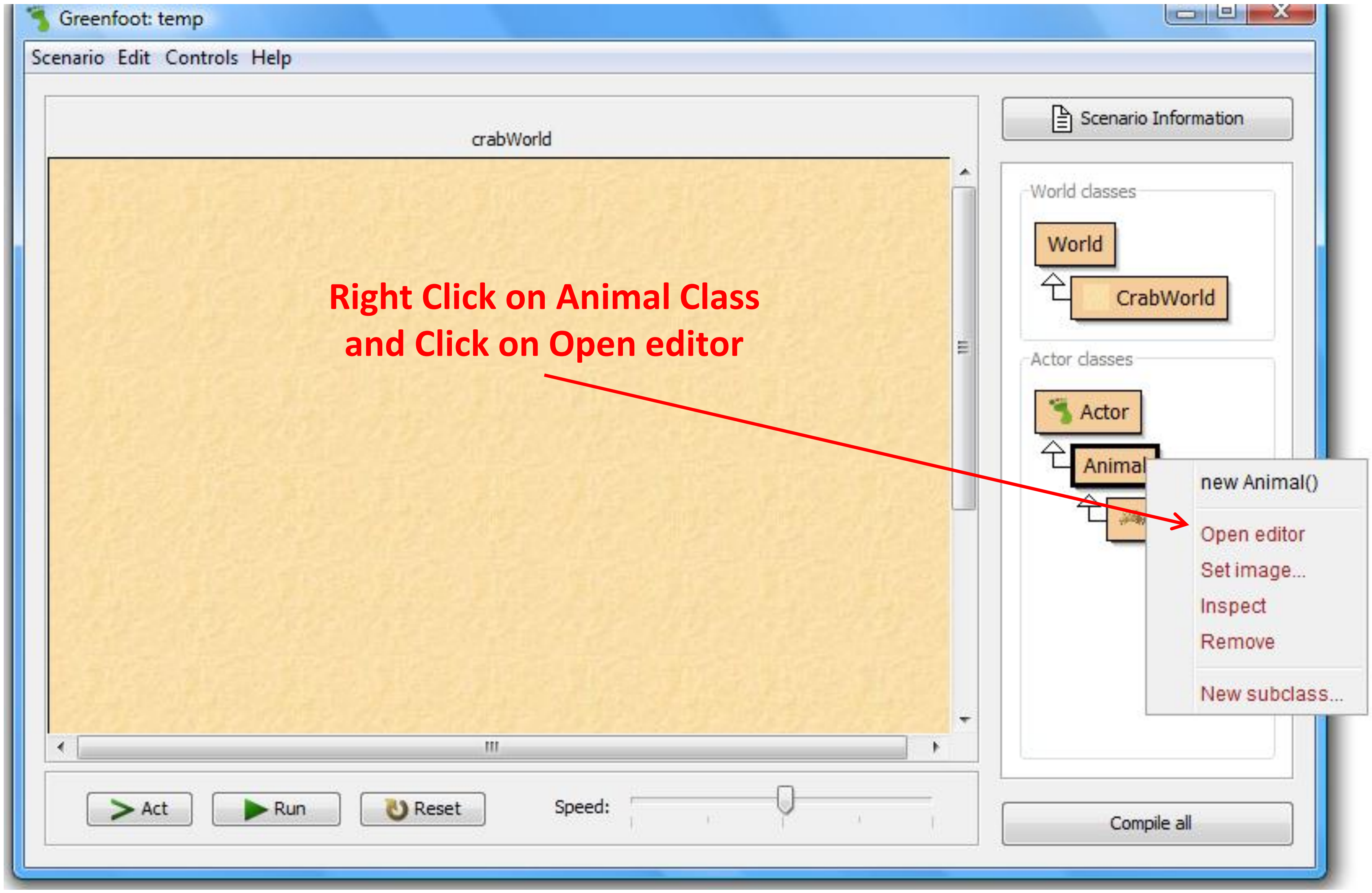
Exercise 2.8



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

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        move (;
        turn (5);
    }
}
```

```
illegal start of expression
```

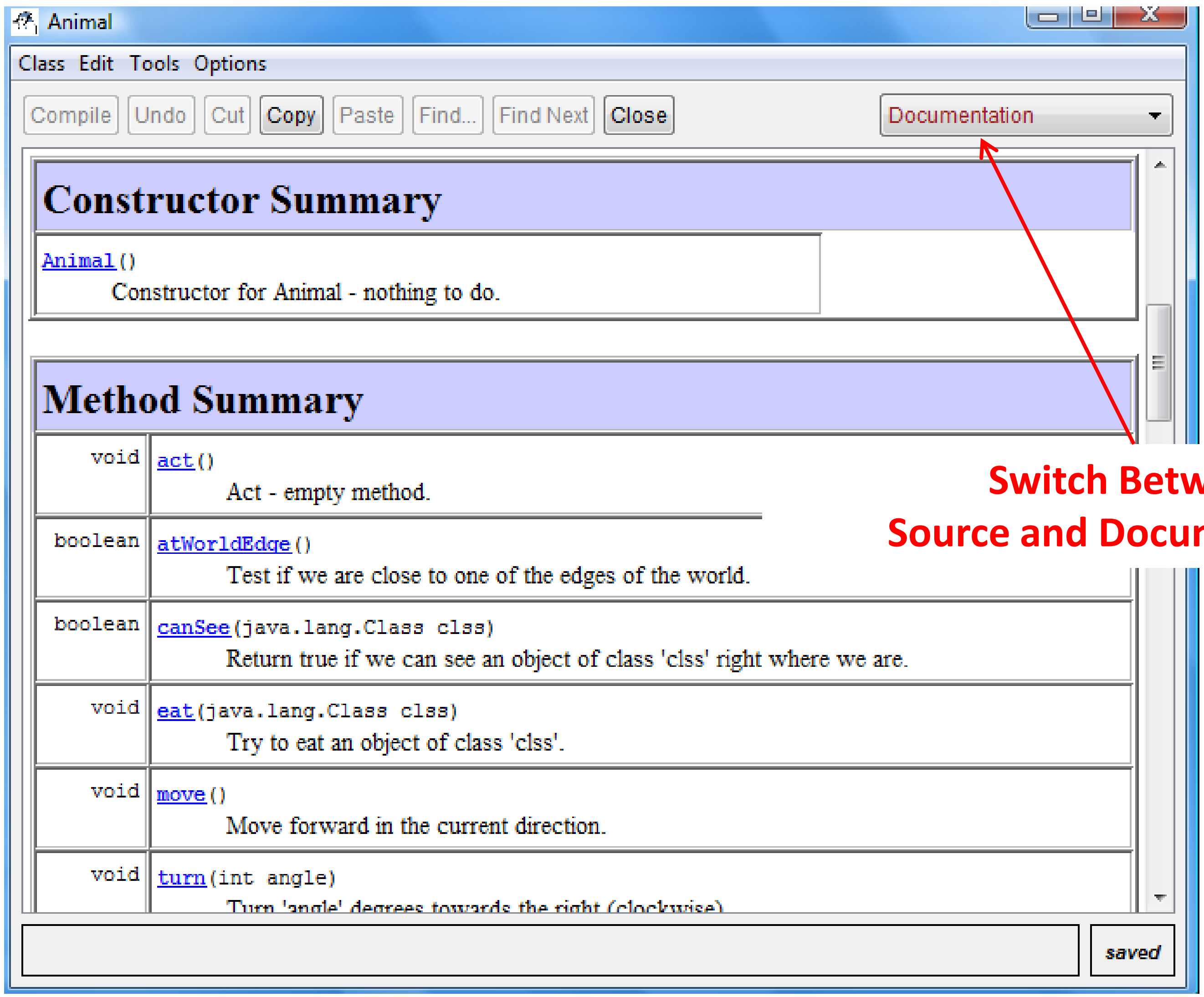
2.4 Dealing with Screen Edges



Right Click on Animal Class
and Click on Open editor

Documentation View

Method Summary



Constructor Summary

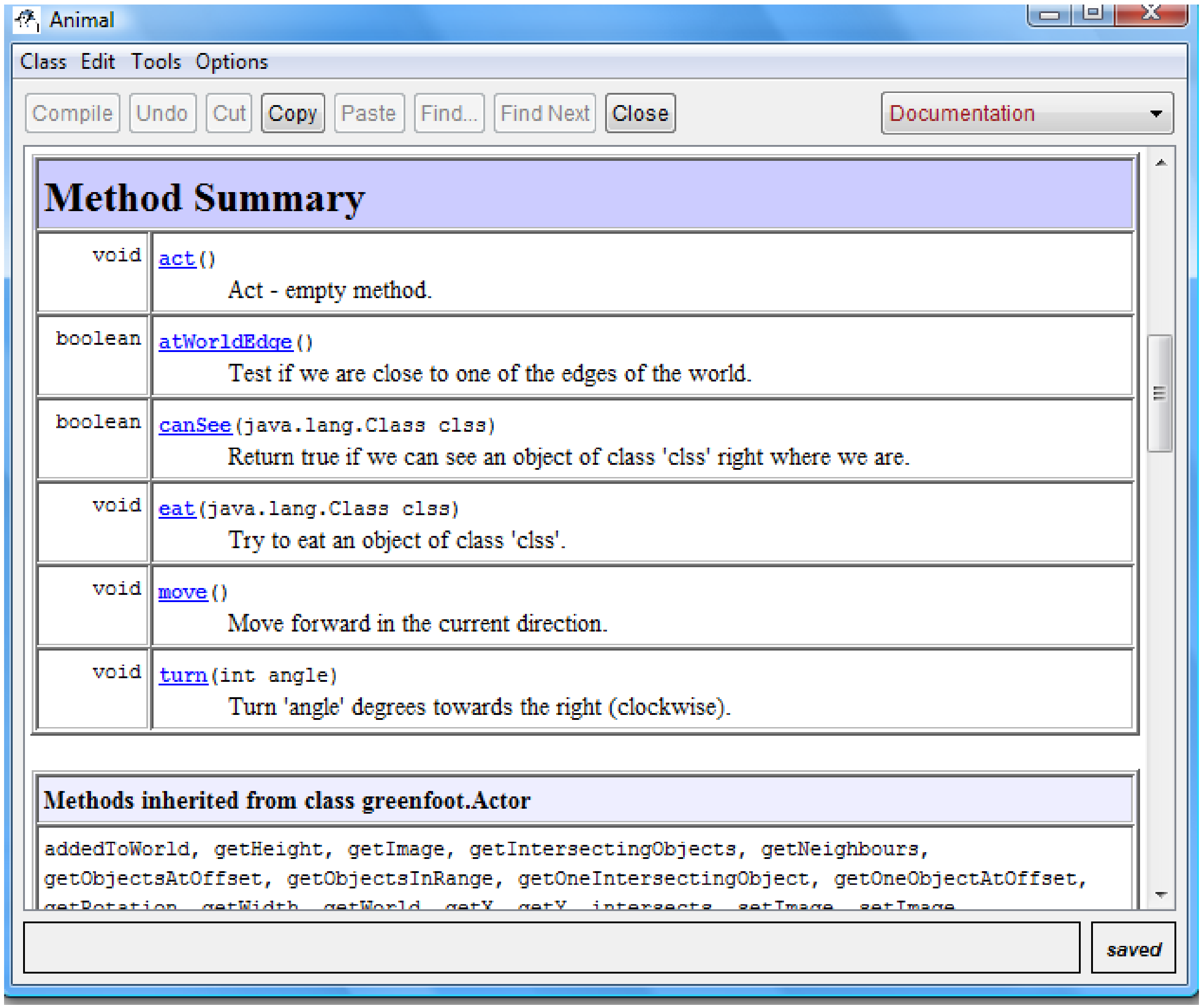
Animal()   Constructor for Animal - nothing to do.

Method Summary

void	act() Act - empty method.
boolean	atWorldEdge() Test if we are close to one of the edges of the world.
boolean	canSee(java.lang.Class clss) Return true if we can see an object of class 'clss' right where we are.
void	eat(java.lang.Class clss) Try to eat an object of class 'clss'.
void	move() Move forward in the current direction.
void	turn(int angle) Turn 'angle' degrees towards the right (clockwise).

Switch Between Source and Documentation

Exercise 2.9



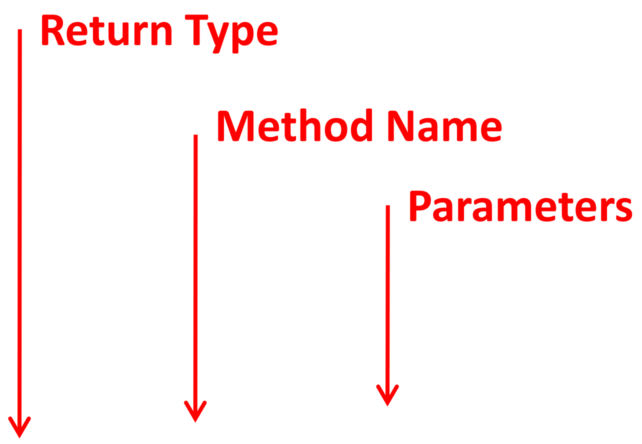
Method Summary

void	act() Act - empty method.
boolean	atWorldEdge() Test if we are close to one of the edges of the world.
boolean	canSee(java.lang.Class clss) Return true if we can see an object of class 'clss' right where we are.
void	eat(java.lang.Class clss) Try to eat an object of class 'clss'.
void	move() Move forward in the current direction.
void	turn(int angle) Turn 'angle' degrees towards the right (clockwise).

Methods inherited from class greenfoot.Actor

```
addedToWorld, getHeight, getImage, getIntersectingObjects, getNeighbours,
getObjectsAtOffset, getObjectsInRange, getOneIntersectingObject, getOneObjectAtOffset,
getRotation, getWidth, getWorld, getX, getY, intersects, setImage, setImage
```

Method Signatures

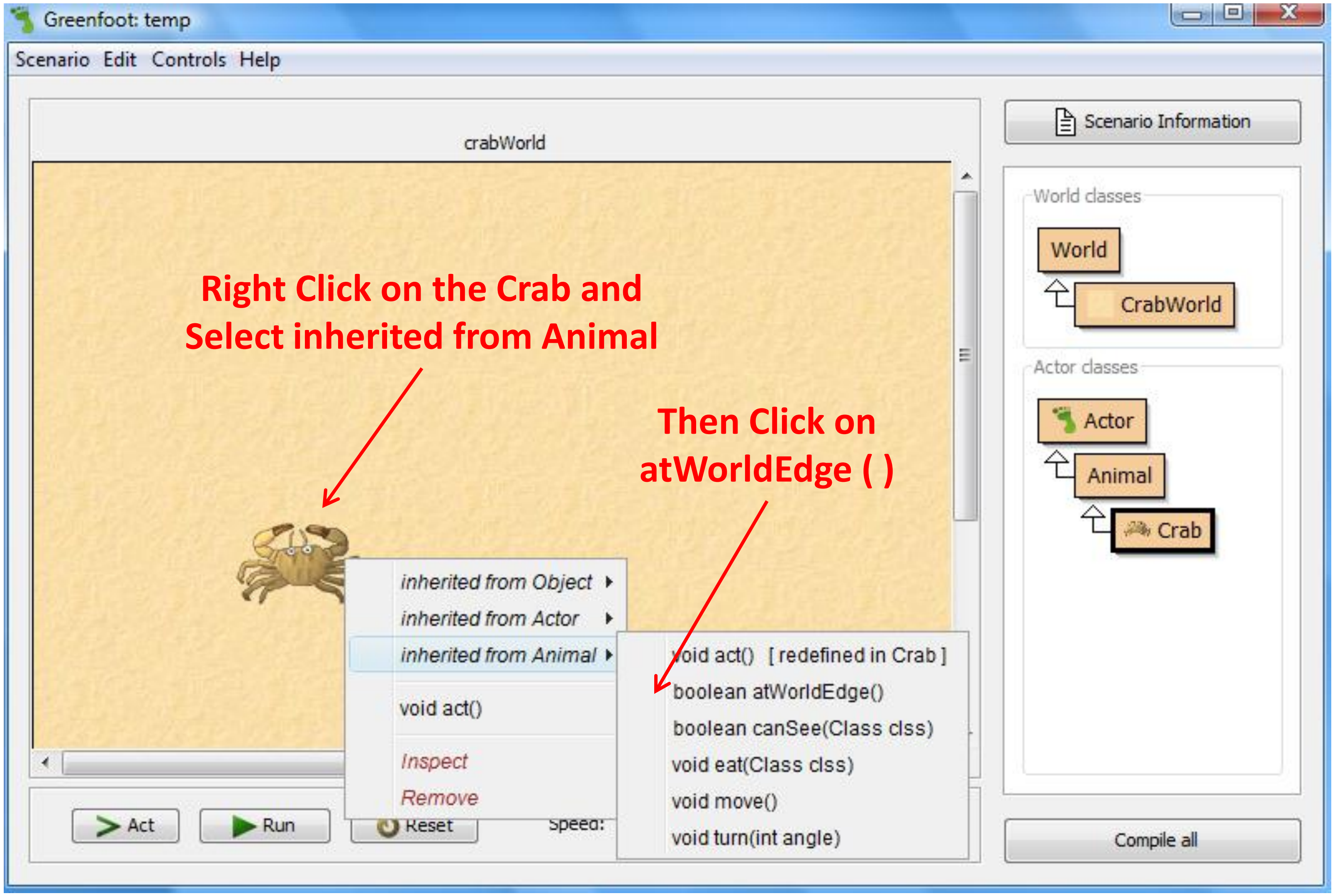


```
void turn (int angle)
boolean atWorldEdge ( )
void move ( )
```

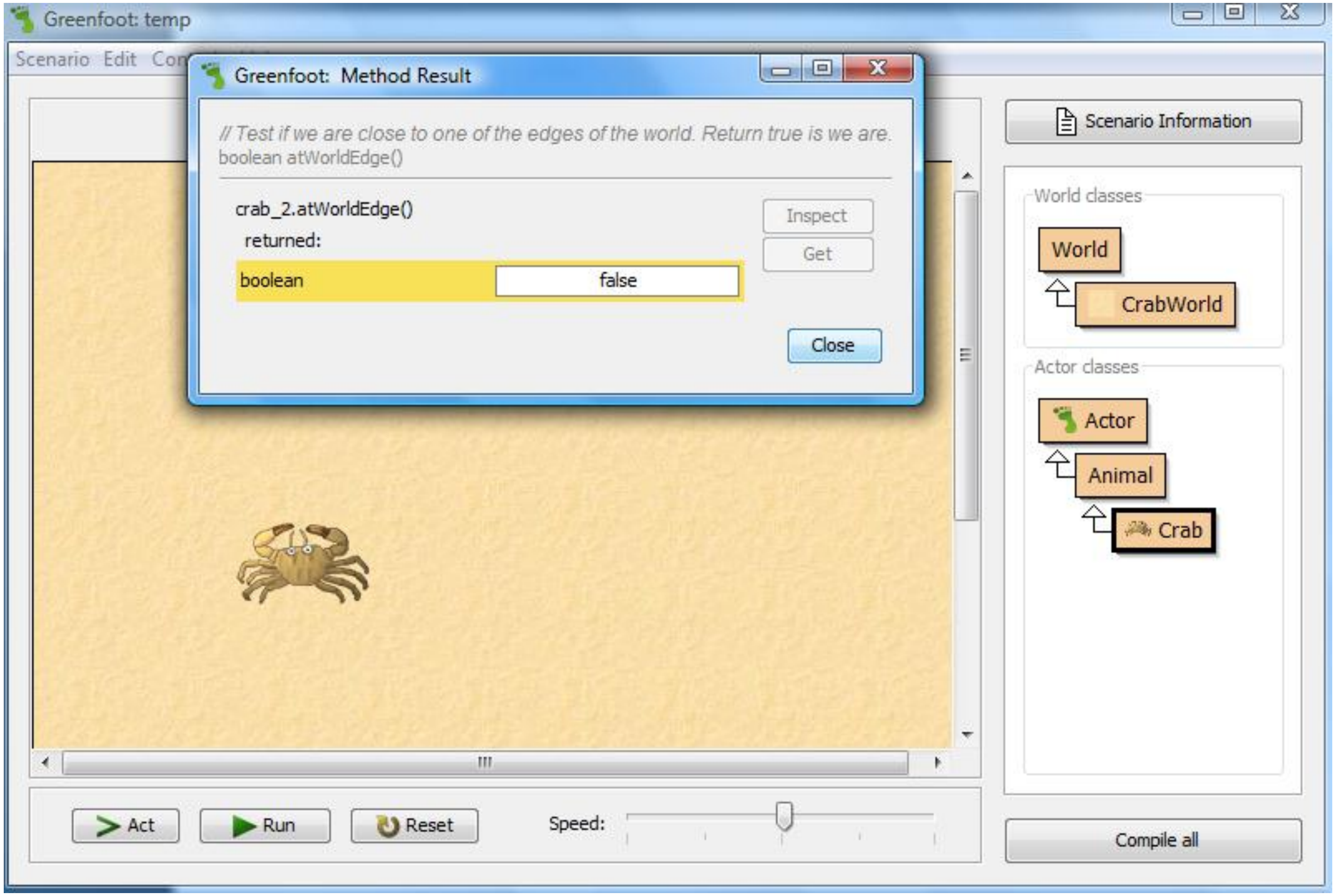
Source for atWorldEdge ( )

```
/*
 * 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;
}
```

Exercise 2.10



Exercise 2.10

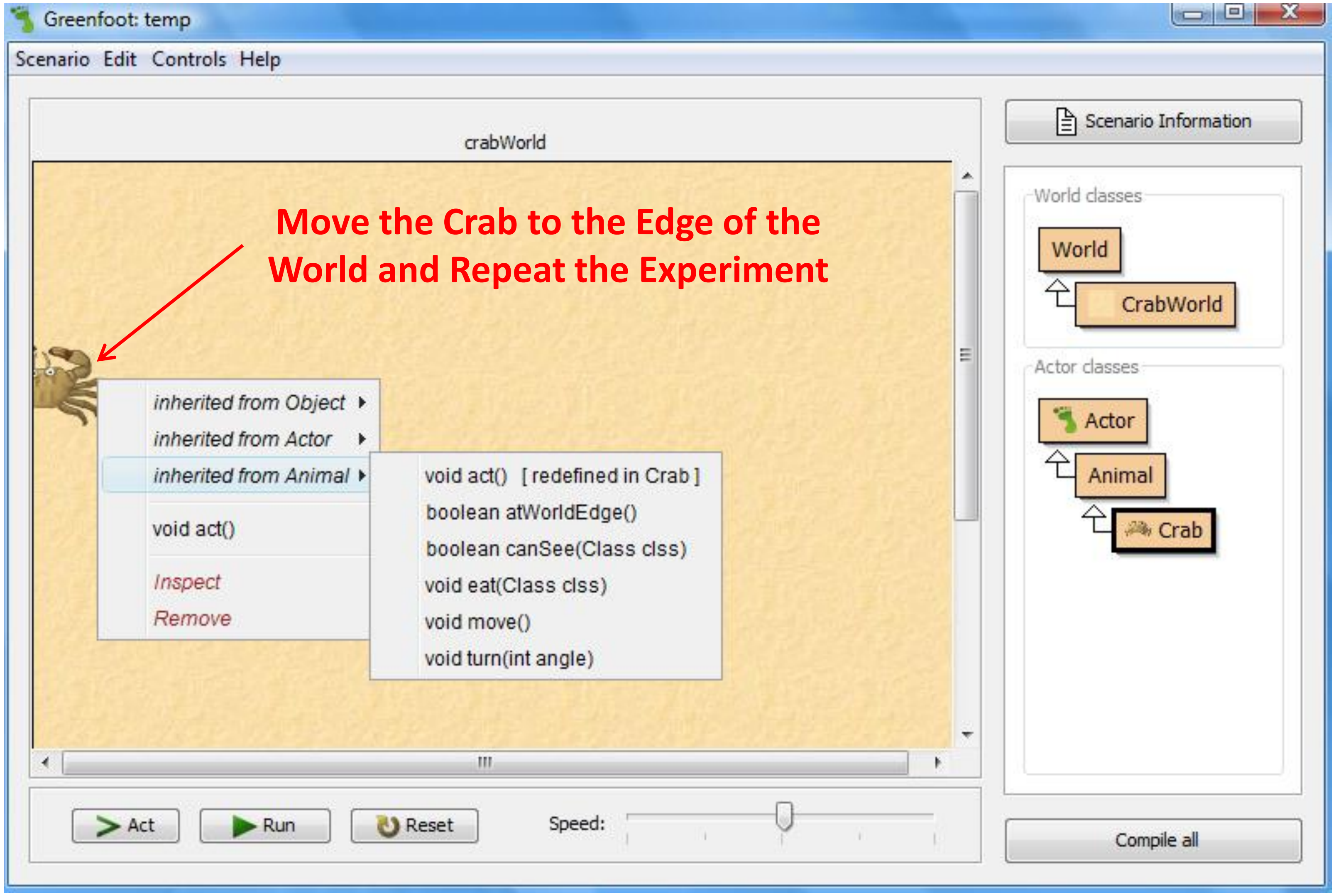


// Test if we are close to one of the edges of the world. Return true is we are.
boolean atWorldEdge()

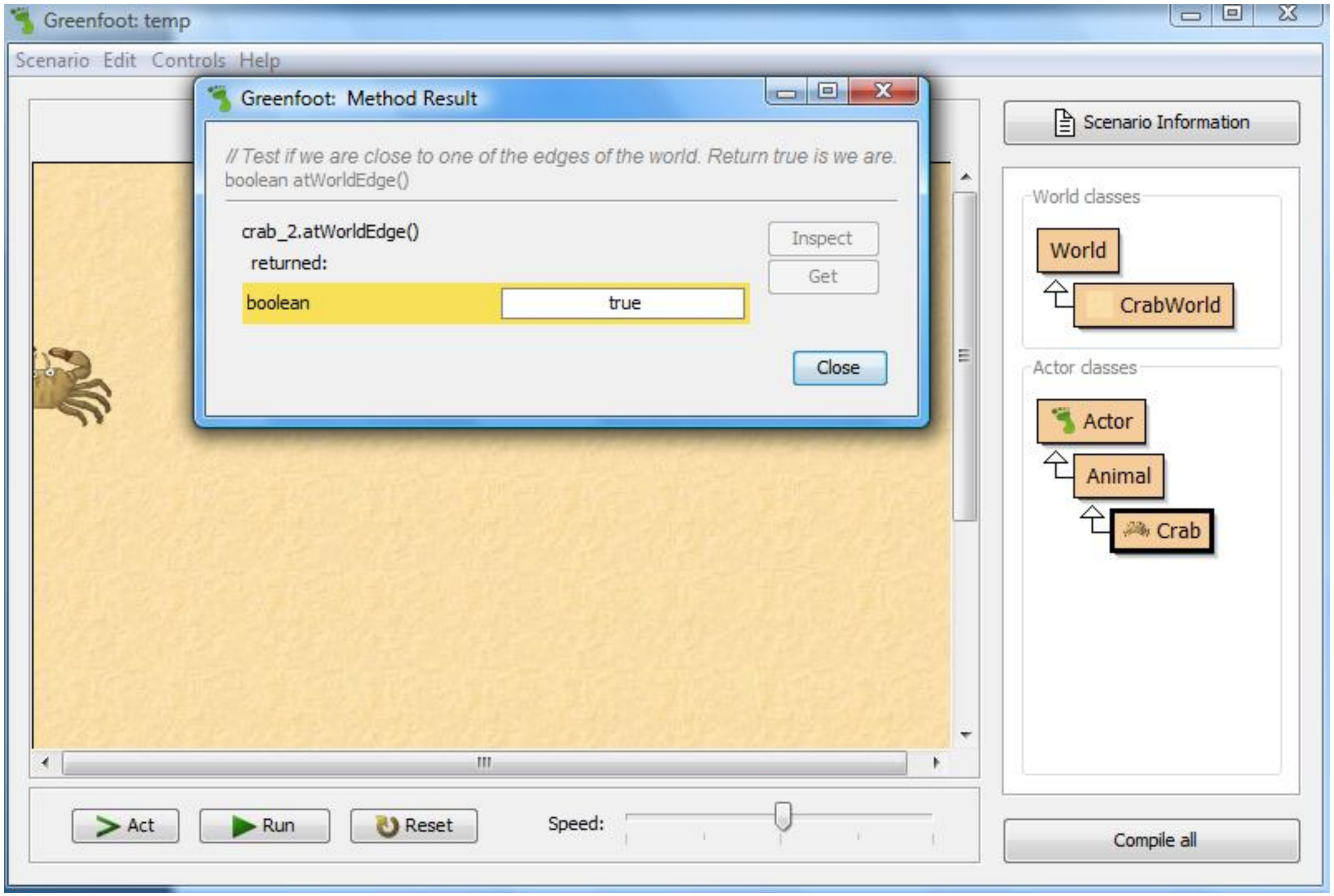
crab_2.atWorldEdge()
returned:

boolean false

Exercise 2.11



Exercise 2.11



// Test if we are close to one of the edges of the world. Return true is we are.
boolean atWorldEdge()

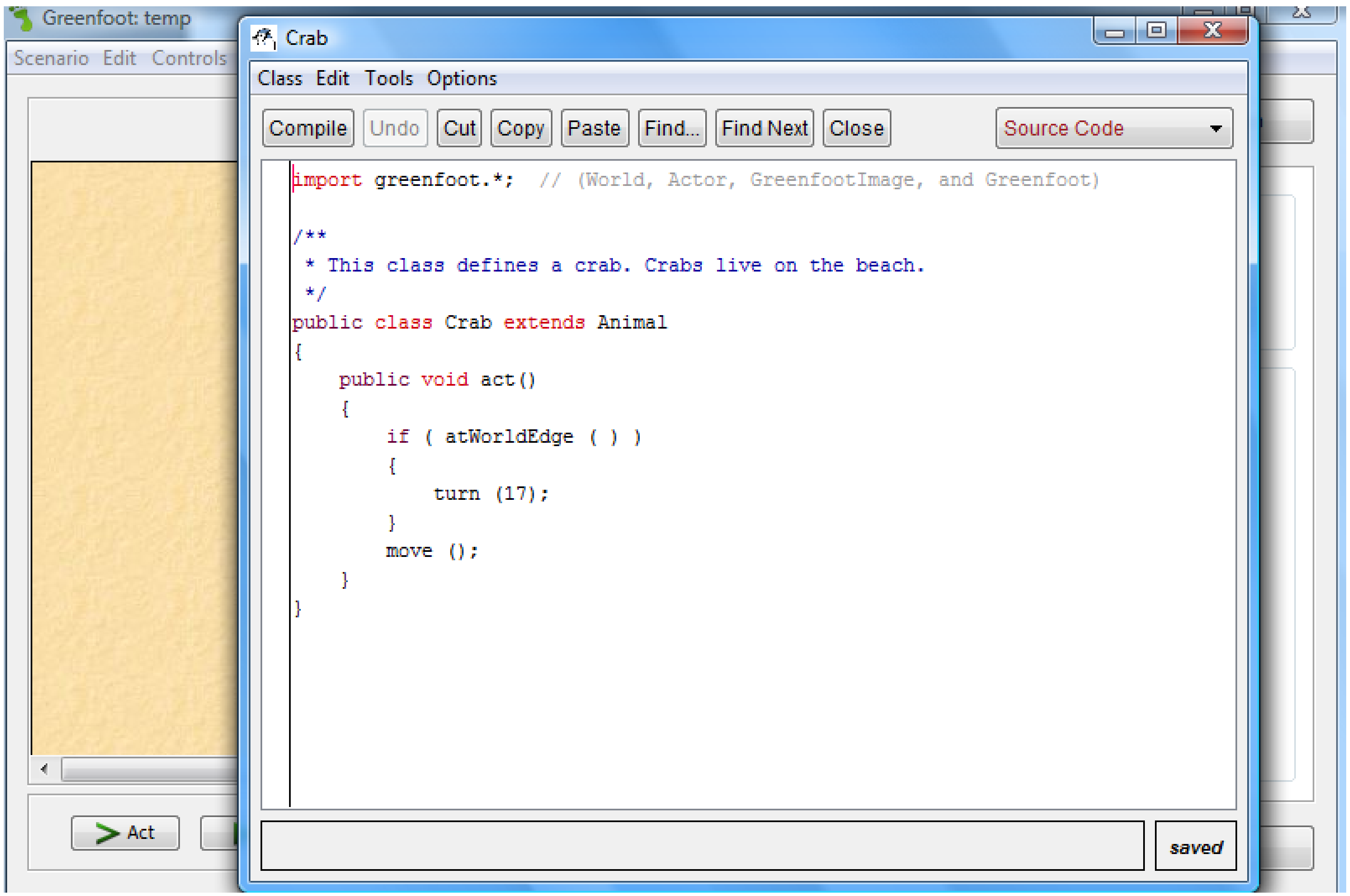
crab_2.atWorldEdge()
returned:

boolean true

Code 2.3

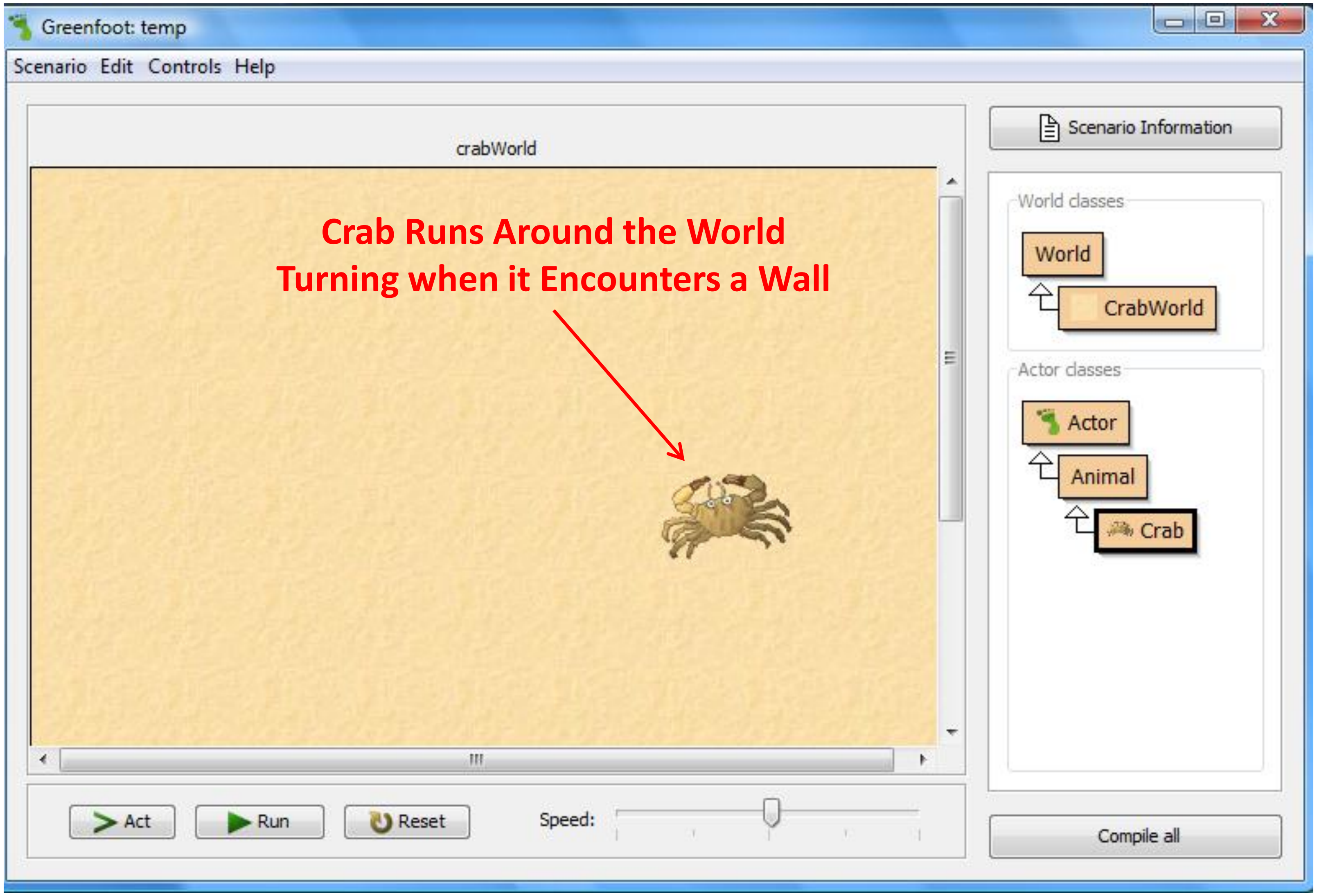
```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/*
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
        public void act()
        {
                if ( atWorldEdge ( )  )
                {
                        turn (17);
                }
                move ( );
        }
}
```

Exercise 2.12



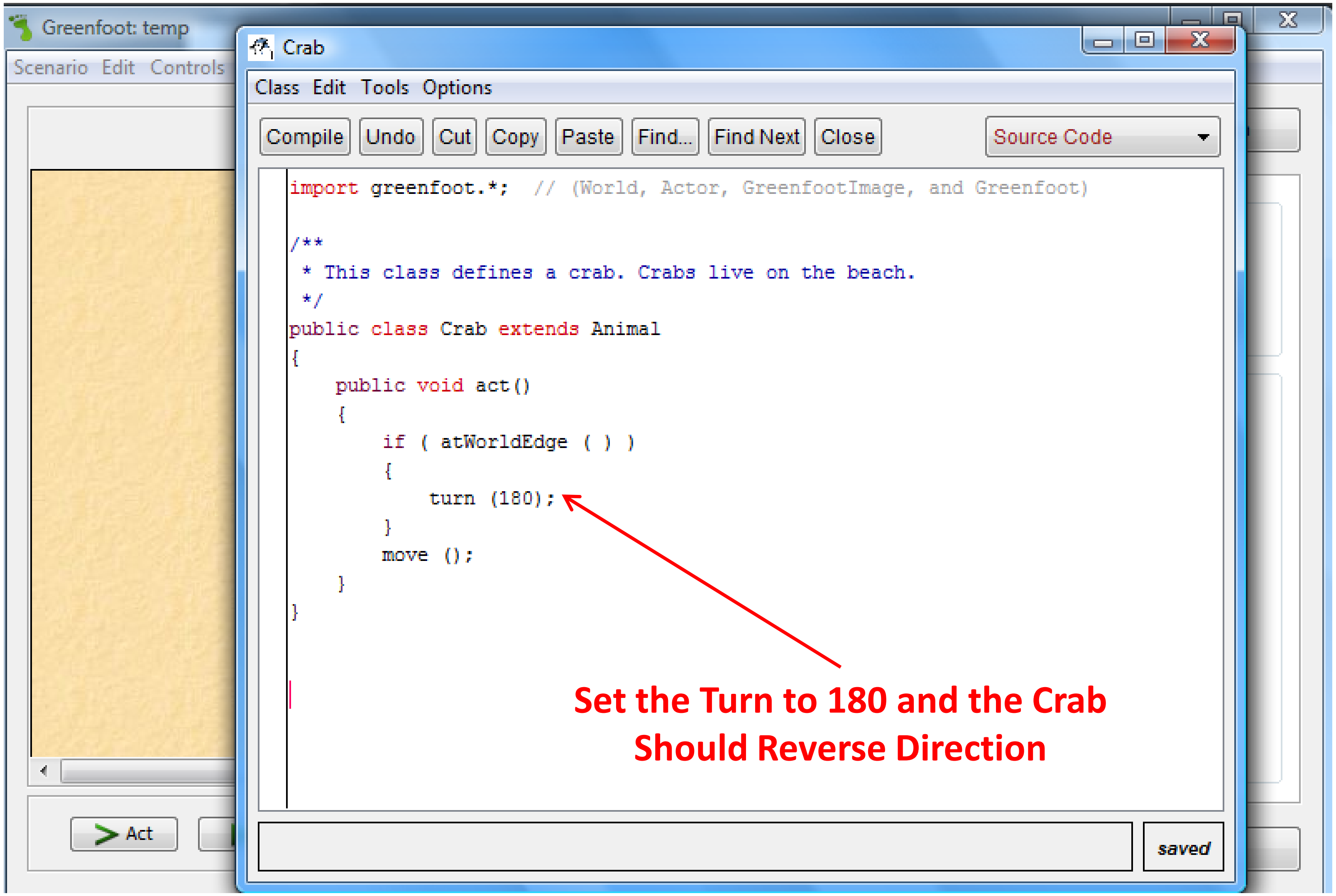
```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        if ( atWorldEdge ( ) )
        {
            turn (17);
        }
        move ();
    }
}
```

Exercise 2.12



Crab Runs Around the World Turning when it Encounters a Wall

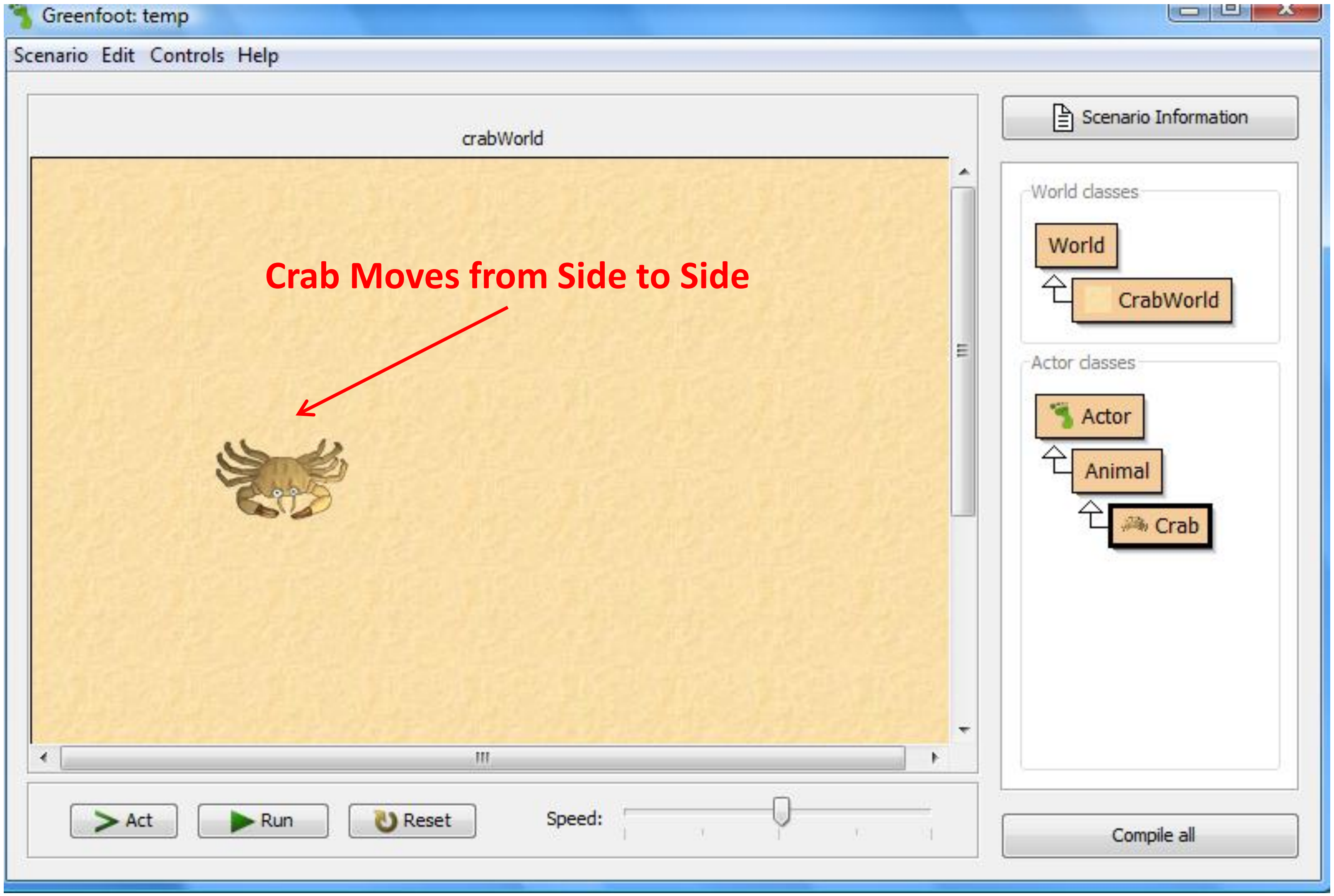
Exercise 2.13



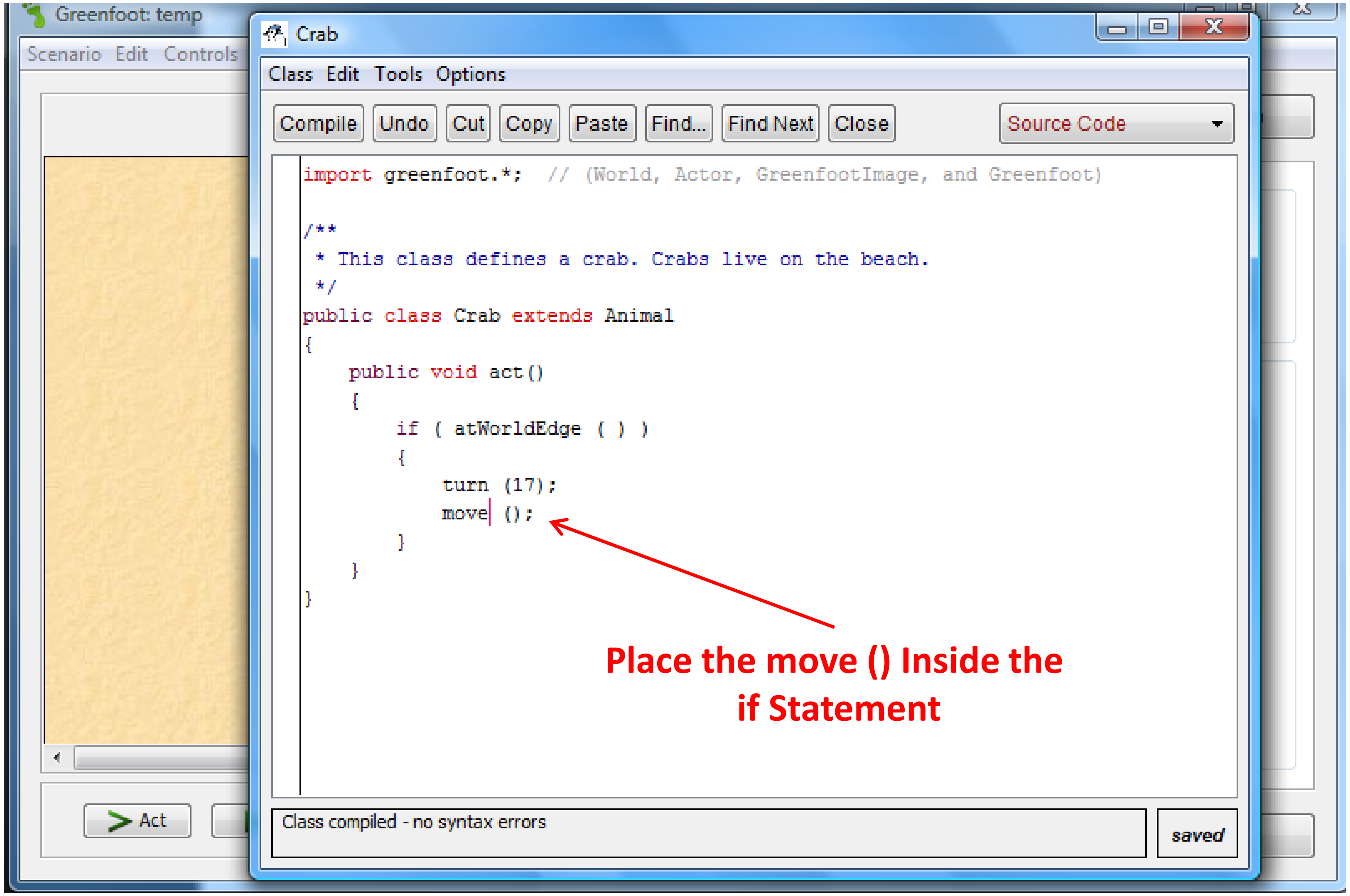
```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        if ( atWorldEdge ( ) )
        {
            turn (180);
        }
        move ();
    }
}
```

Set the Turn to 180 and the Crab Should Reverse Direction

Exercise 2.13



Exercise 2.14

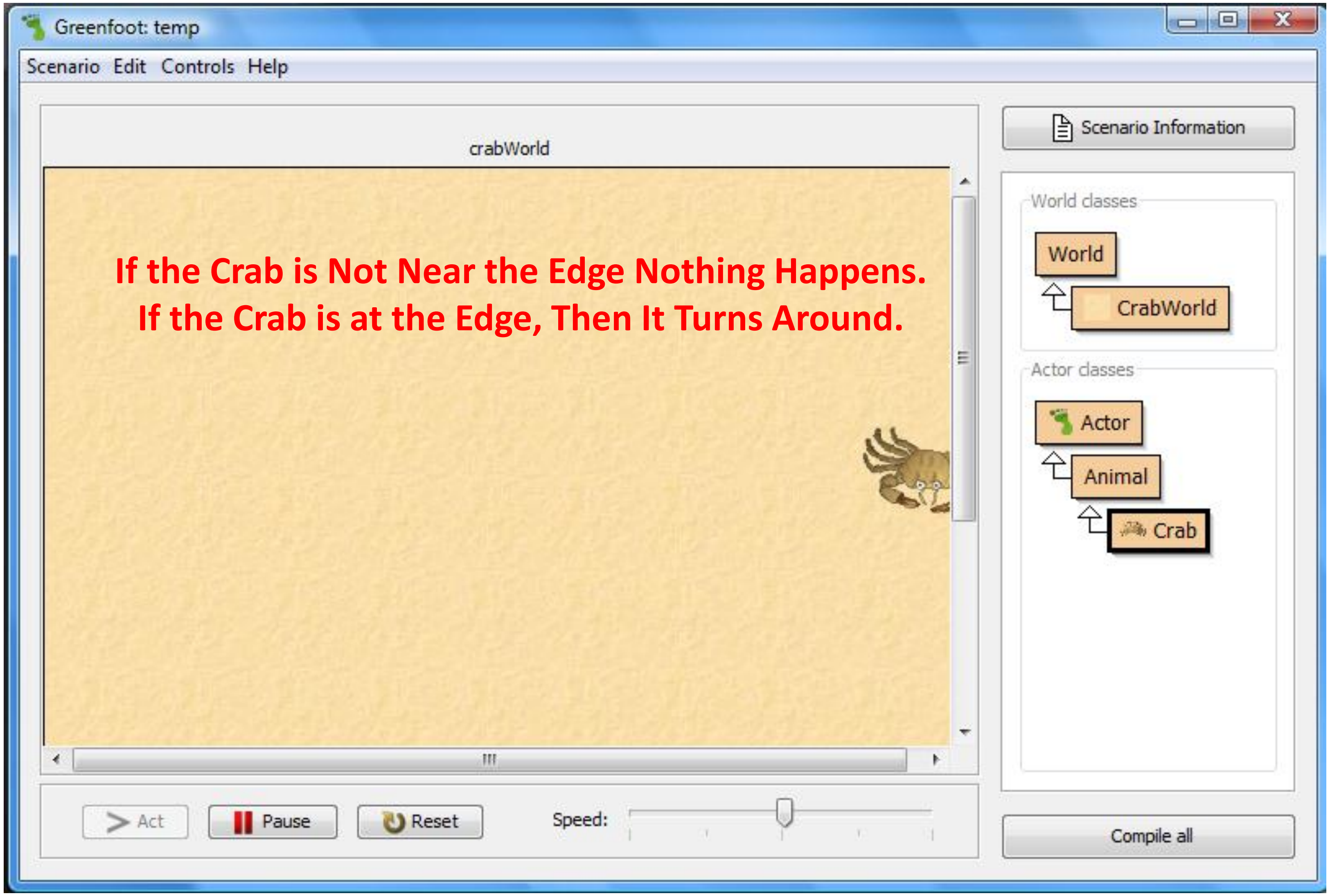


```
import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)

/**
 * This class defines a crab. Crabs live on the beach.
 */
public class Crab extends Animal
{
    public void act()
    {
        if ( atWorldEdge ( ) )
        {
            turn (17);
            move ();
        }
    }
}
```

Class compiled - no syntax errors

Place the move () Inside the if Statement

Exercise 2.14



If the Crab is Not Near the Edge Nothing Happens.
If the Crab is at the Edge, Then It Turns Around.

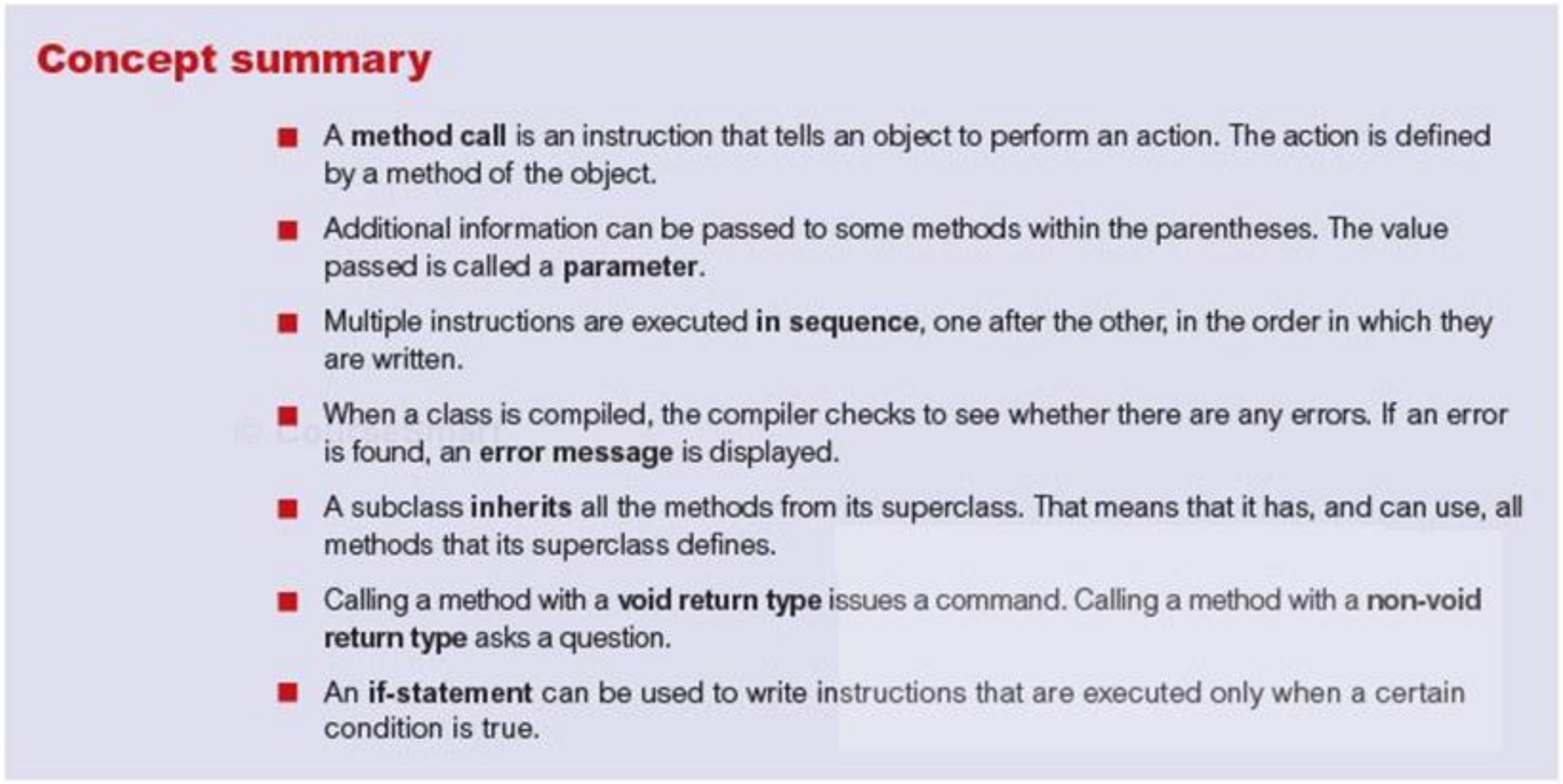
2.5 Summary of Programming Techniques

In this chapter, we have seen how to call methods such as move( ), with and without parameters. This will form the basis for all further Java Programming.

We have encountered a glimpse of inheritance. Classes inherit the methods from their superclasses.

And, very important we have seen how to make decisions. We have used an if-statement for conditional execution.

Concept Summary



Concept summary

- A **method call** is an instruction that tells an object to perform an action. The action is defined by a method of the object.
- Additional information can be passed to some methods within the parentheses. The value passed is called a **parameter**.
- Multiple instructions are executed **in sequence**, one after the other, in the order in which they are written.
- When a class is compiled, the compiler checks to see whether there are any errors. If an error is found, an **error message** is displayed.
- A subclass **inherits** all the methods from its superclass. That means that it has, and can use, all methods that its superclass defines.
- Calling a method with a **void return type** issues a command. Calling a method with a **non-void return type** asks a question.
- An **if-statement** can be used to write instructions that are executed only when a certain condition is true.