

# Chapter 2 - The First Program: Little Crab

# Announcement


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

# 2.1 Little Crab Scenario

Greenfoot: little-crab

Scenario Edit Controls Help

crabWorld

 **Inheritance**  
**The Arrows Denote Hierarchy**

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

Scenario Information

World classes

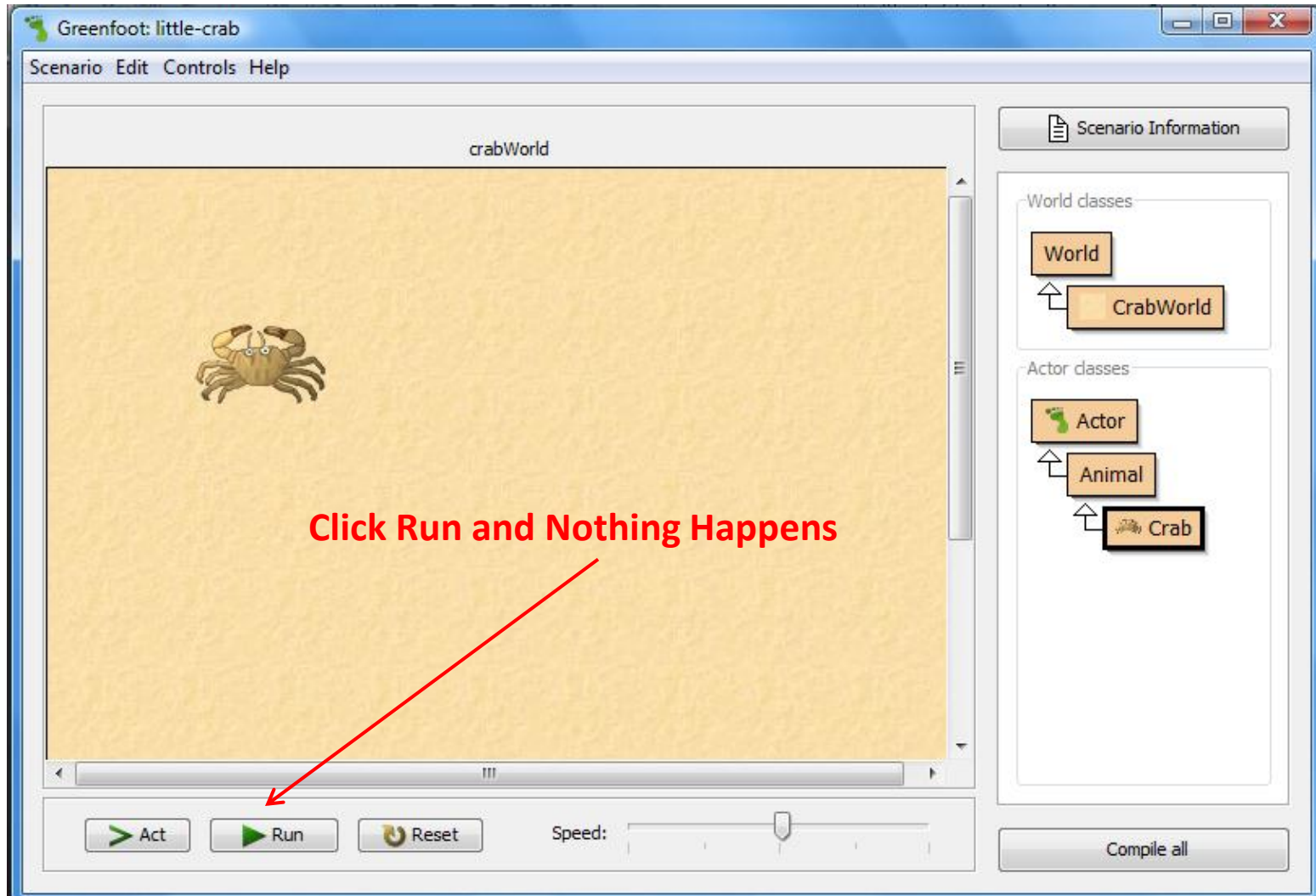
- World
- CrabWorld

Actor classes

- Actor
- Animal
- Crab

Act Run Reset Speed:  Compile all

# Exercise 2.1

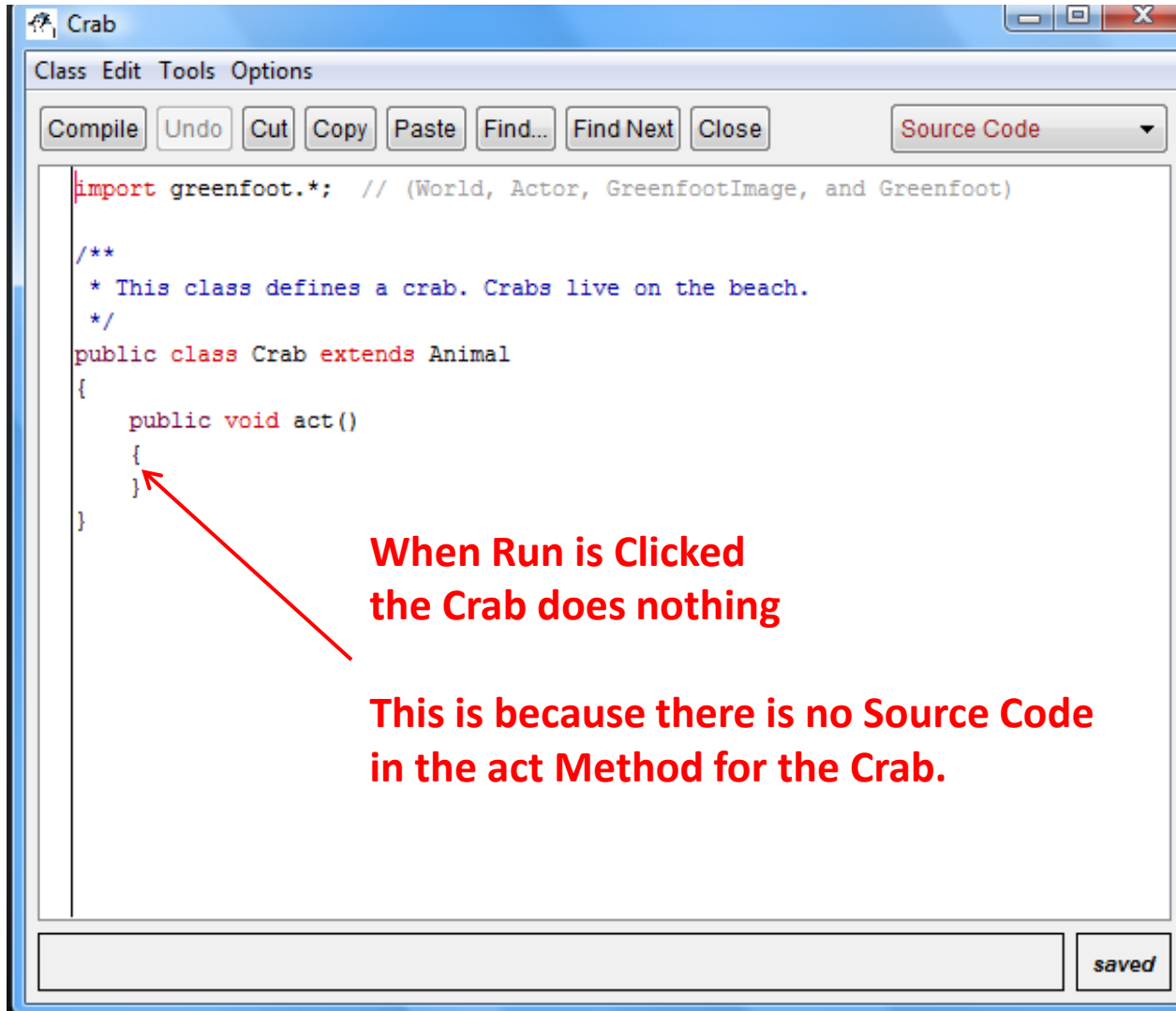


# Exercise 2.1

The screenshot shows the Greenfoot IDE interface. The main window is titled "Greenfoot: little-crab" and contains a "Scenario Edit Controls Help" menu. The central area is a "crabWorld" with a yellow background and a crab actor. The right sidebar shows "Scenario Information" and two class hierarchies: "World classes" (World, CrabWorld) and "Actor classes" (Actor, Animal, Crab). A context menu is open over the Crab class, listing options: "new Crab()", "Open editor", "Set image...", "Inspect", "Remove", and "New subclass...". A red arrow points from the text "Click On Open Editor" to the "Open editor" option in the menu. The bottom of the window features "Act", "Run", and "Reset" buttons, a "Speed" slider, and a "Compile all" button.

**Right Click the Crab Class**  
**Click On Open Editor**

# Exercise 2.1



Crab

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Find Next Close Source Code

```
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.**

saved

## 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

```
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()
```

```
    {
```

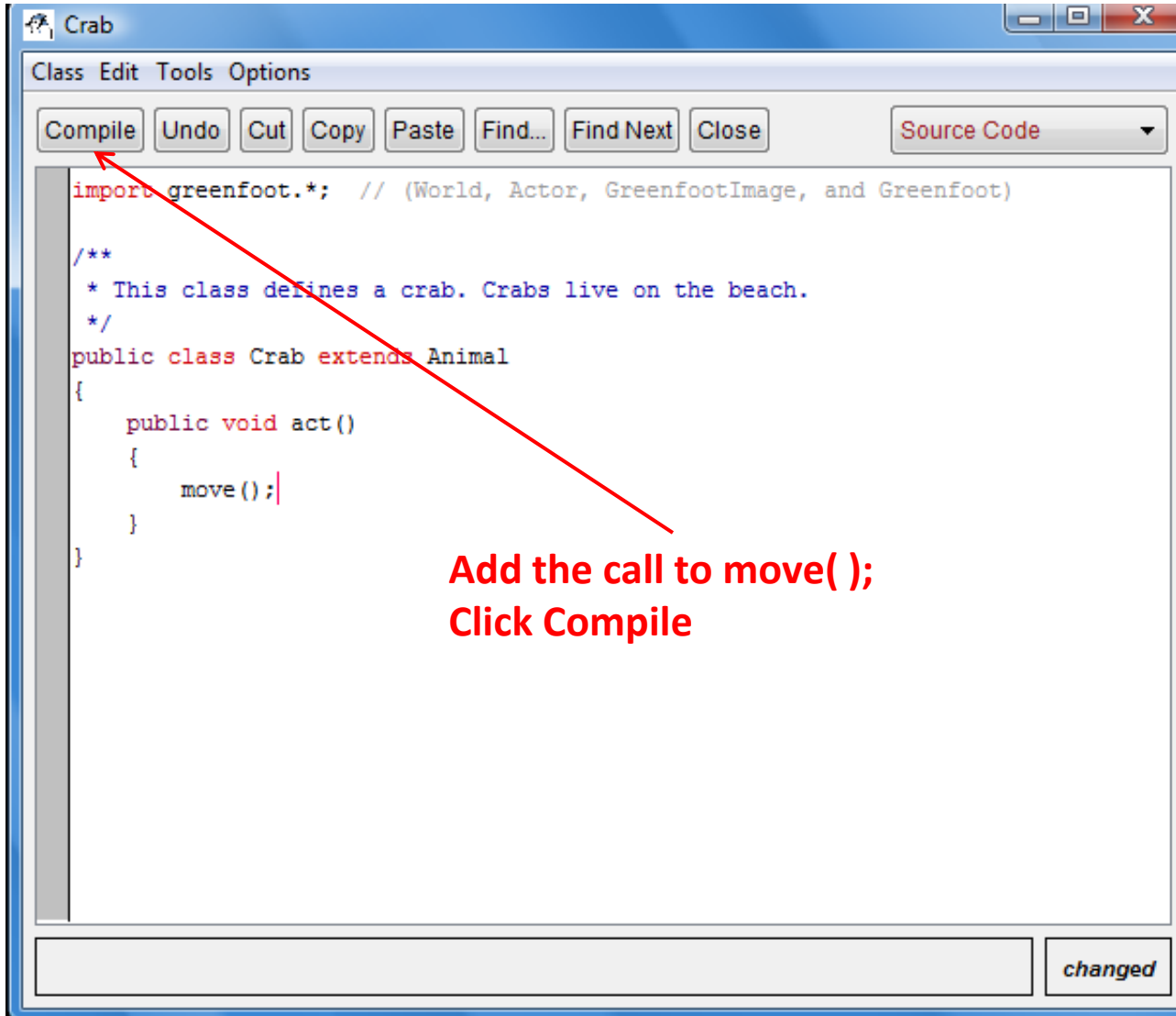
```
        move();
```

```
    }
```

```
}
```



# Exercise 2.2



# Exercise 2.2

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

The screenshot shows the Greenfoot IDE interface for 'Exercise 2.2'. The main window displays a 'crabWorld' scene with a crab actor. A red arrow points from the 'Run' button in the bottom control panel to the 'Crab' class in the 'Actor classes' panel. The 'Actor classes' panel shows a hierarchy: Actor (base class), Animal (subclass of Actor), and Crab (subclass of Animal). The 'World classes' panel shows a hierarchy: World (base class) and CrabWorld (subclass of World). The bottom control panel includes buttons for 'Act', 'Run', and 'Reset', a 'Speed' slider, and a 'Compile all' button.

# Exercise 2.2

The screenshot shows the Greenfoot IDE interface for a scenario named "crabWorld". The main workspace displays a yellowish-brown world with a crab on the right edge. A red arrow points from the text "Crab Moves to Edge of the World" to the crab. The interface includes a menu bar (Scenario, Edit, Controls, Help), a toolbar with "Act", "Pause", and "Reset" buttons, a "Speed" slider, and a "Compile all" button. On the right, there is a "Scenario Information" panel with class diagrams for "World classes" and "Actor classes".

**World classes:**

```
graph BT; World --> CrabWorld;
```

**Actor classes:**

```
graph BT; Actor --> Animal; Animal --> Crab;
```

Crab Moves to Edge of the World

Act Pause Reset Speed: [Slider] Compile all

# Exercise 2.3

Greenfoot: Exercise 2.2

Scenario Edit Controls Help

crabWorld

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

Scenario Information

World classes

- World
- CrabWorld

Actor classes

- Actor
- Animal
- Crab

Act Run Reset Speed: [Slider] Compile all

# Exercise 2.3

The screenshot shows the Greenfoot IDE interface for 'Exercise 2.2'. The main window displays a 'crabWorld' with a light brown background. Three crabs are visible on the right edge of the world. The text 'Crabs All Move to Edge of the World' is overlaid in red. The right sidebar contains a 'Scenario Information' panel with two class diagrams. The 'World classes' diagram shows 'CrabWorld' inheriting from 'World'. The 'Actor classes' diagram shows 'Crab' inheriting from 'Animal', which inherits from 'Actor'. The bottom control bar includes 'Act', 'Pause', 'Reset', and 'Speed' controls, along with a 'Compile all' button.

Greenfoot: Exercise 2.2

Scenario Edit Controls Help

crabWorld

**Crabs All Move to Edge of the World**

Scenario Information

World classes

- World
- CrabWorld (inherits from World)

Actor classes

- Actor
- Animal (inherits from Actor)
- Crab (inherits from Animal)

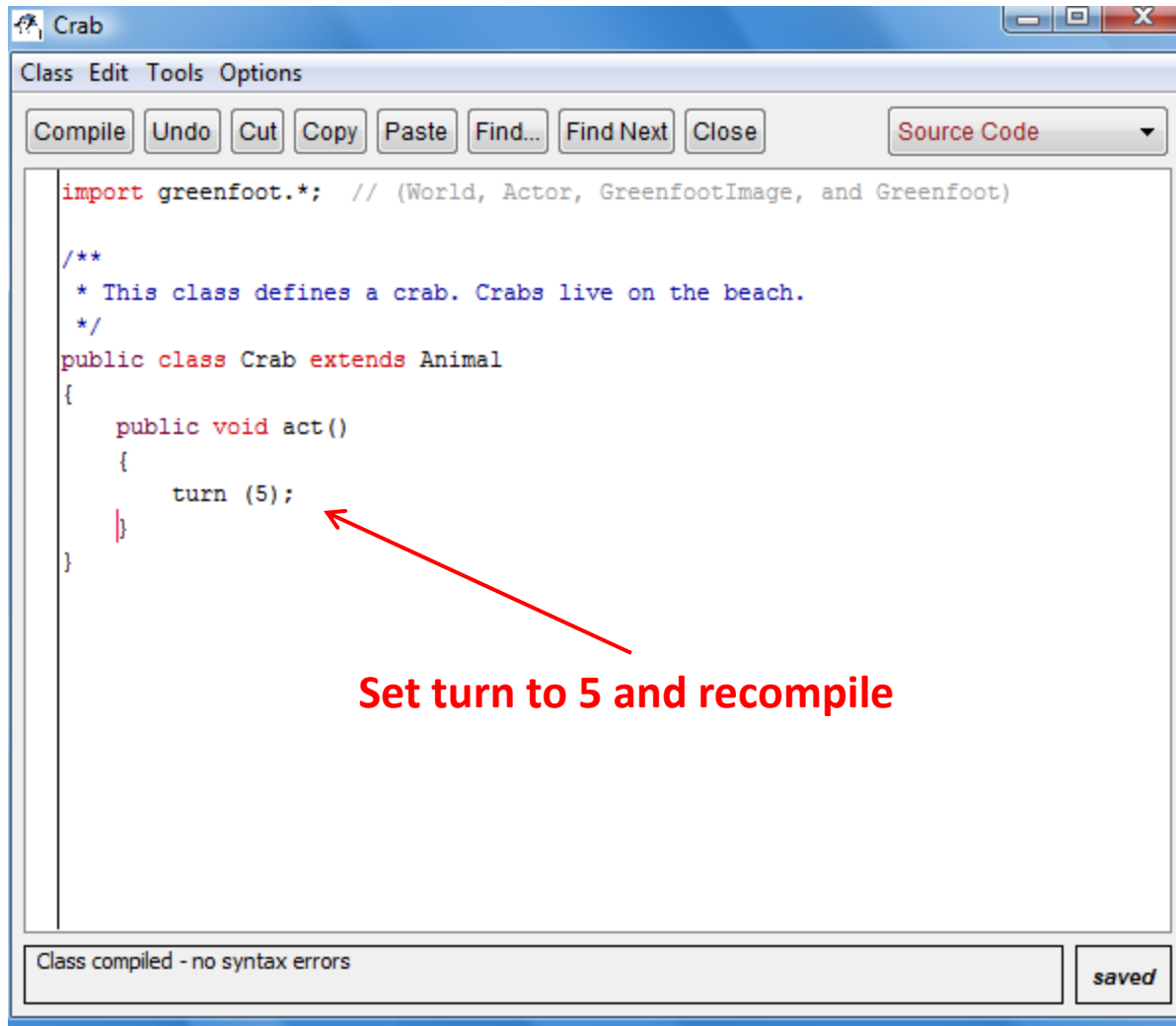
> Act   || Pause   ↺ Reset   Speed: [Slider]   Compile all

## 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

The screenshot shows the Greenfoot IDE interface for a project named "little-crab". The main workspace displays a world named "crabWorld" with a light brown background. A crab is positioned in the center, and the text "Crab Spins to the Right" is written in red above it. The interface includes a menu bar (Scenario, Edit, Controls, Help), a toolbar with "Act", "Run", and "Reset" buttons, and a "Speed" slider. On the right side, there is a "Scenario Information" panel showing a class hierarchy: "World" is the superclass of "CrabWorld", "Actor" is the superclass of "Animal", and "Crab" is the superclass of "Crab". The "Crab" class is highlighted with a black border. A "Compile all" button is located at the bottom right.

Greenfoot: little-crab

Scenario Edit Controls Help

crabWorld

**Crab Spins to the Right**

World classes

- World
- CrabWorld

Actor classes

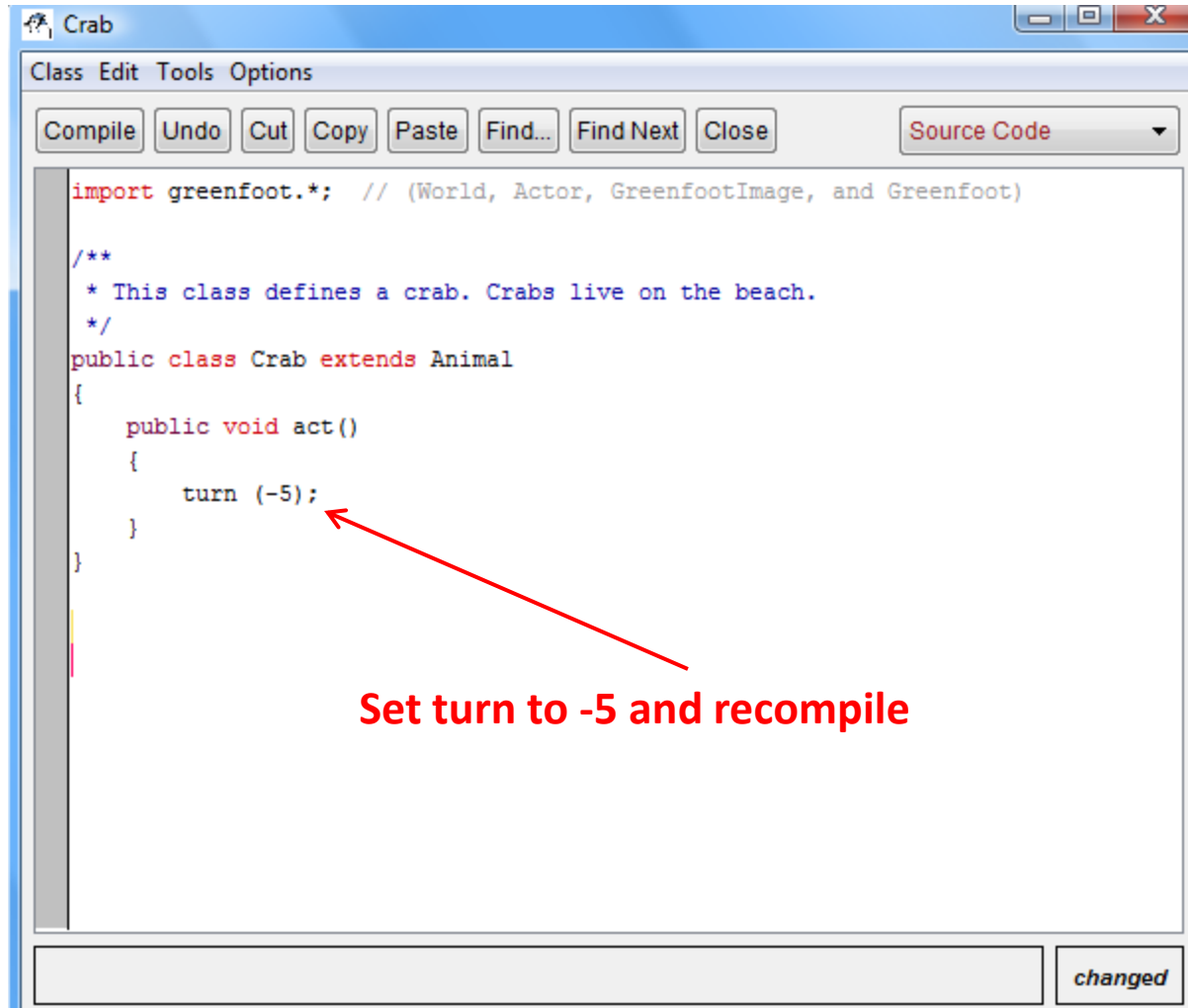
- Actor
- Animal
- Crab

Act Run Reset Speed: [Slider]

Compile all



# Exercise 2.5



```
Crab
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
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**

*changed*

# Exercise 2.5

The screenshot shows the Greenfoot IDE interface for a project named "little-crab". The main workspace displays a world named "crabWorld" with a yellow background. A crab is positioned in the center, and the text "Crab Spins to the Left" is written in red above it. The interface includes a menu bar (Scenario, Edit, Controls, Help), a toolbar with "Act", "Run", and "Reset" buttons, and a "Speed" slider. On the right side, there is a "Scenario Information" panel showing class hierarchies. The "World classes" section shows "World" as a base class for "CrabWorld". The "Actor classes" section shows "Actor" as a base class for "Animal", which is a base class for "Crab". A "Compile all" button is located at the bottom right.

Greenfoot: little-crab

Scenario Edit Controls Help

crabWorld

**Crab Spins to the Left**

World classes

- World
- CrabWorld

Actor classes

- Actor
- Animal
- Crab

Act Run Reset Speed: [Slider]

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

```
Crab
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
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

saved

# Exercise 2.6

The screenshot shows the Greenfoot IDE interface for a project named "little-crab". The main workspace displays a world named "crabWorld" with a yellow background. A crab is positioned in the center, and the text "Crab moves in a circle" is written in red. The right-hand panel shows the class hierarchy:

- World classes:** World (base class), CrabWorld (subclass).
- Actor classes:** Actor (base class), Animal (subclass), Crab (subclass).

The bottom control bar includes buttons for "Act", "Run", and "Reset", a "Speed:" slider, and a "Compile all" button.

# 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);
    }
}
```

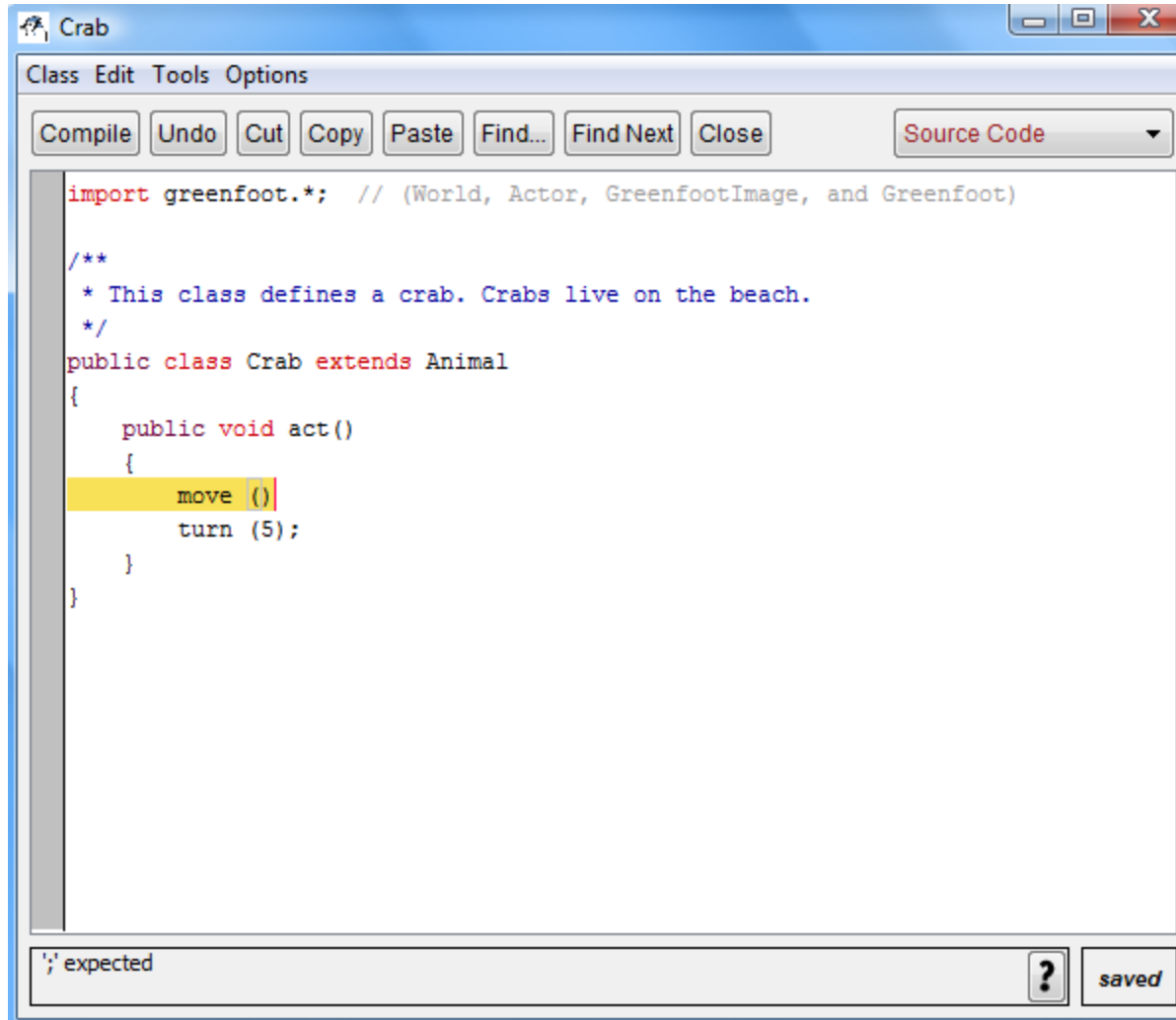
**Syntax Error - Expected Semicolon**

; expected

# 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



The screenshot shows a Java IDE window titled "Crab". The window has a menu bar with "Class", "Edit", "Tools", and "Options". Below the menu bar is a toolbar with buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", "Find Next", and "Close". There is also a "Source Code" dropdown menu. The main area is a code editor containing the following Java code:

```
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);
    }
}
```

The code is highlighted in a light blue color. The "move ();" line is highlighted in yellow. At the bottom of the window, there is a status bar with the text "'; expected" on the left, a question mark icon in the middle, and the word "saved" on the right.

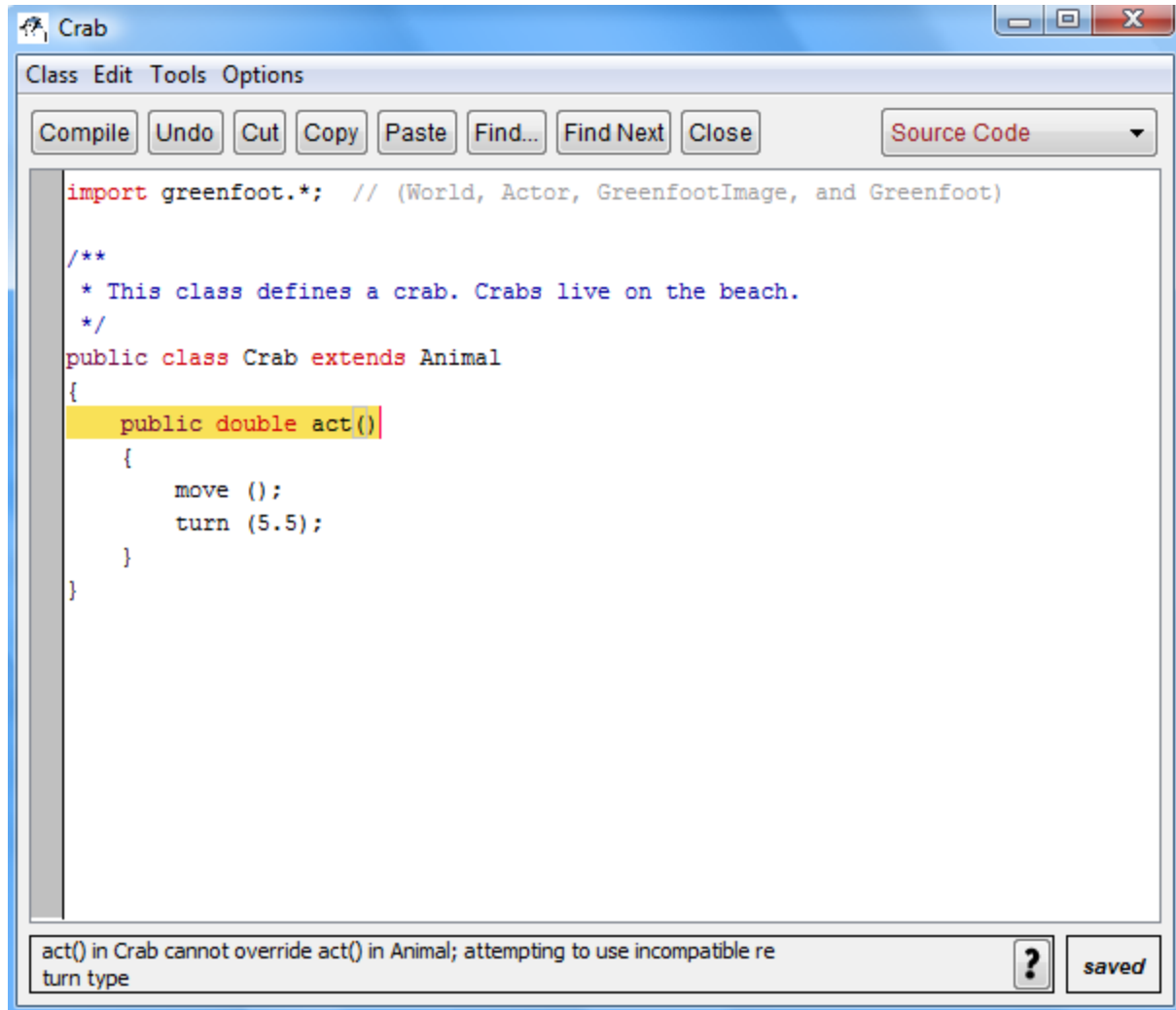


# Exercise 2.8

```
Crab
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
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() ? saved

# Exercise 2.8

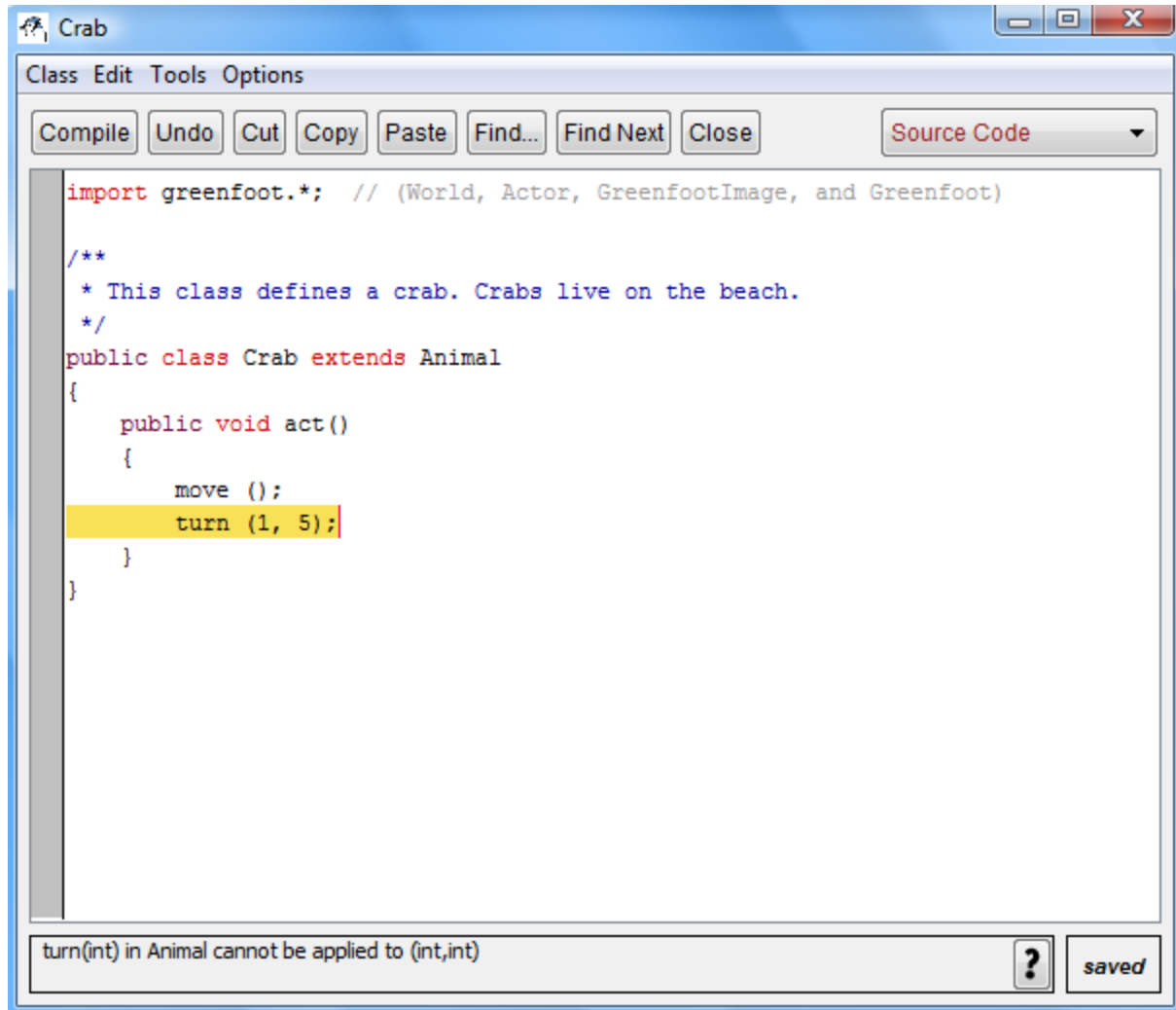


```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)

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

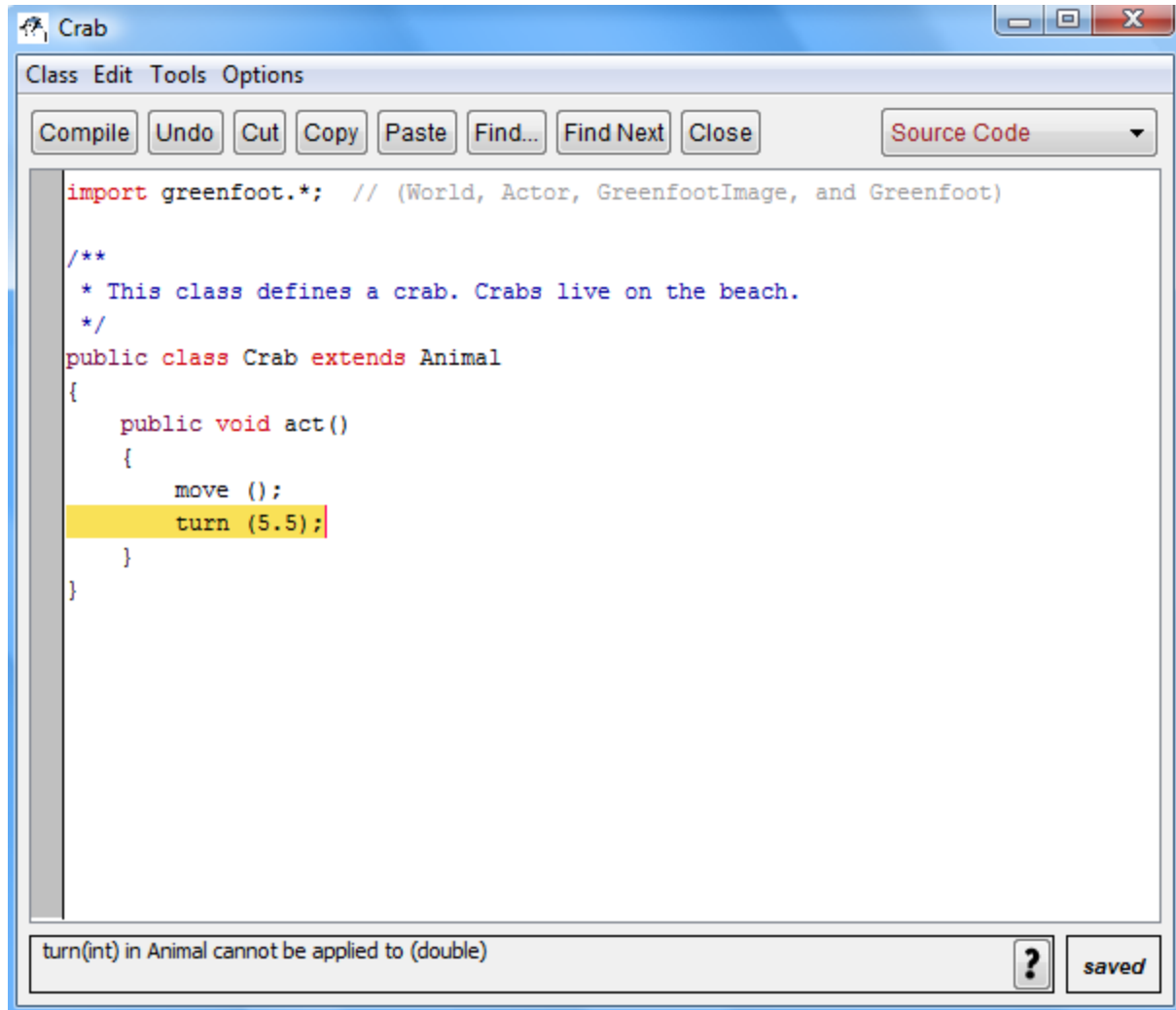
act() in Crab cannot override act() in Animal; attempting to use incompatible re  
turn type

# Exercise 2.8



```
Crab
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
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) ? saved
```

# Exercise 2.8



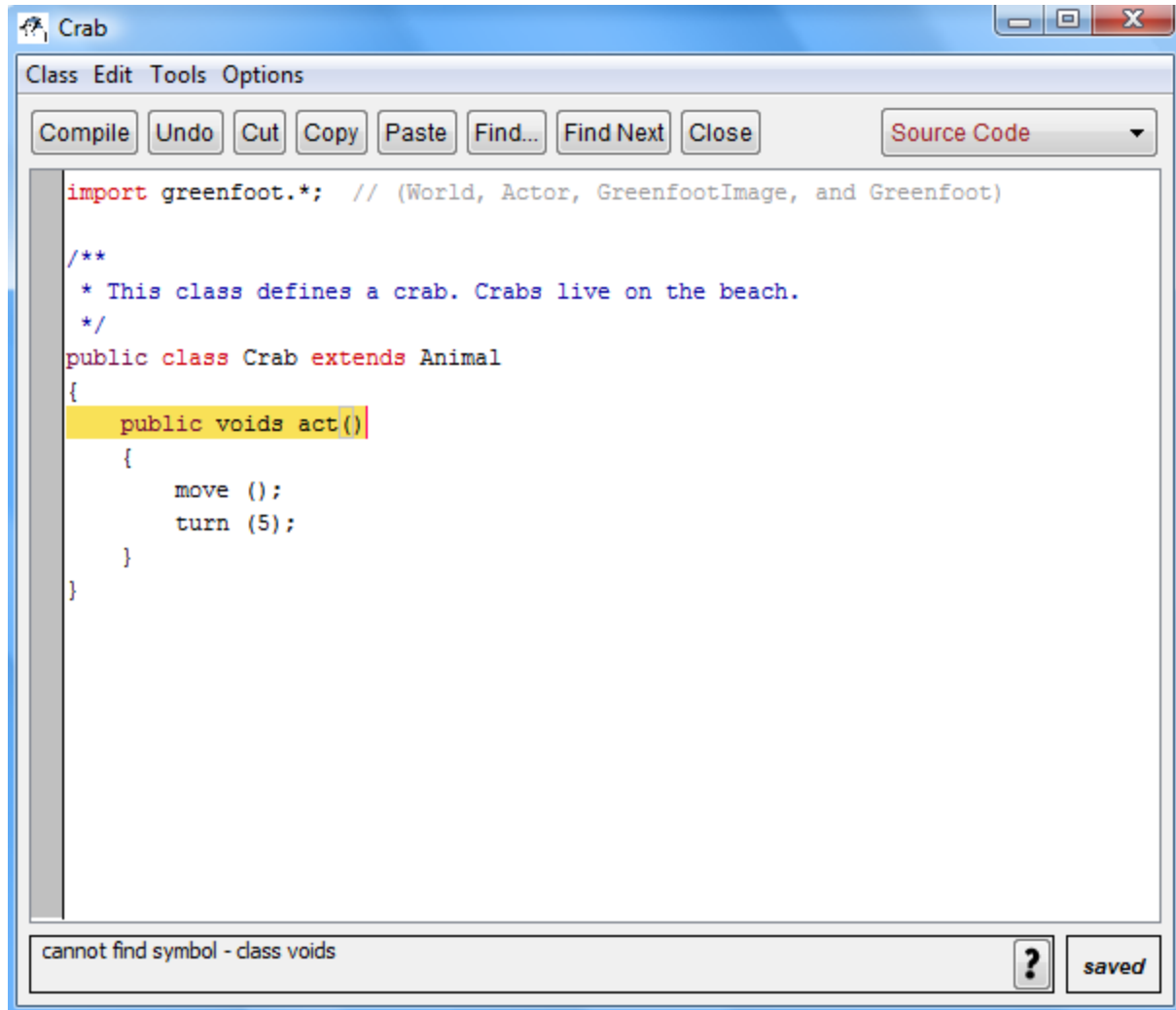
The screenshot shows a Java IDE window titled "Crab". The window has a menu bar with "Class", "Edit", "Tools", and "Options". Below the menu bar is a toolbar with buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", "Find Next", and "Close". There is also a "Source Code" dropdown menu. The main area is a code editor containing the following Java code:

```
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);
    }
}
```

The line `turn (5.5);` is highlighted in yellow. At the bottom of the window, a status bar displays the error message: "turn(int) in Animal cannot be applied to (double)". To the right of the error message are a question mark icon and the text "saved".

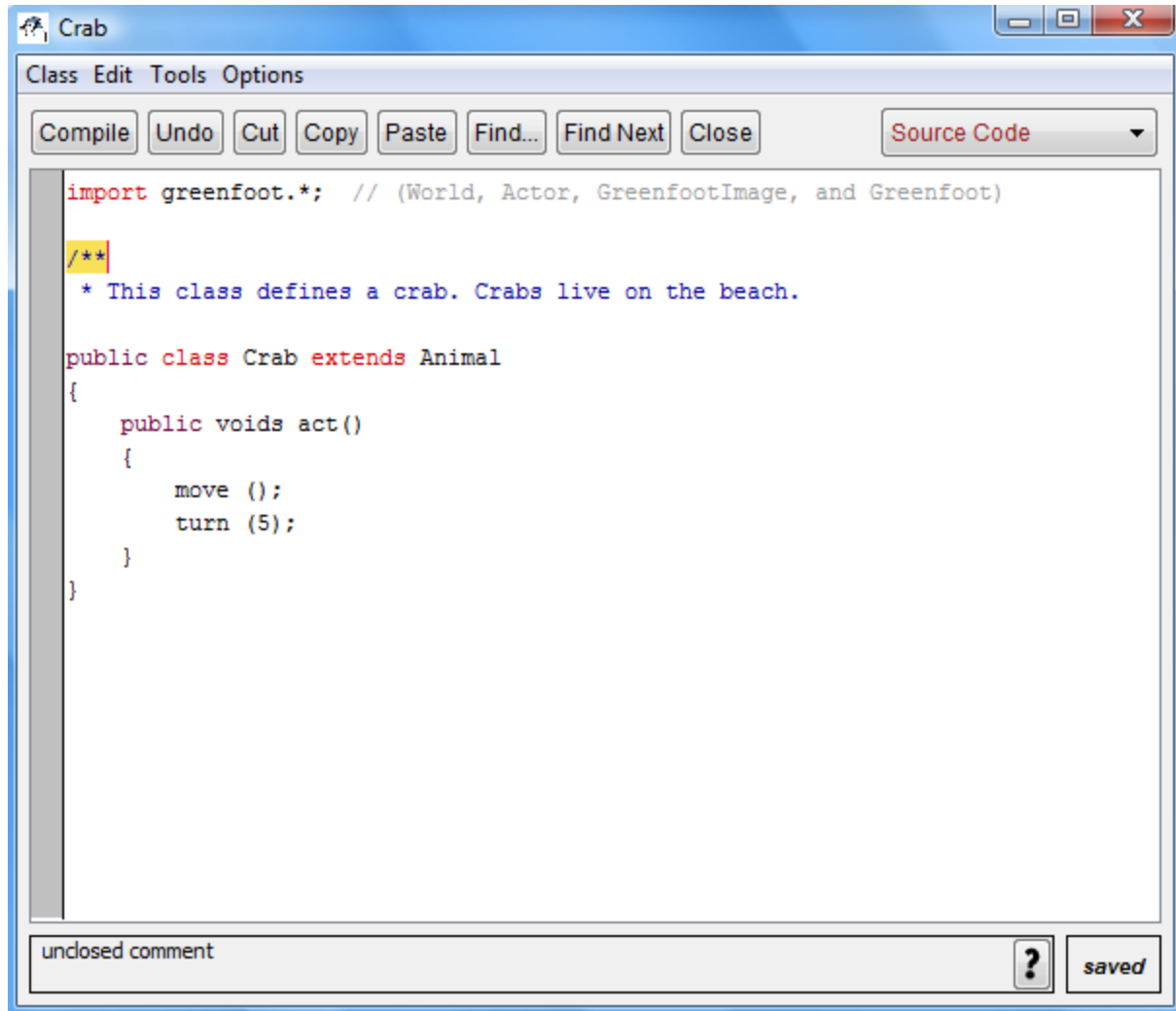
# Exercise 2.8



```
Crab
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
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



The screenshot shows a Java IDE window titled "Crab". The window has a menu bar with "Class", "Edit", "Tools", and "Options". Below the menu bar is a toolbar with buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", "Find Next", and "Close". There is also a "Source Code" dropdown menu. The main text area contains the following code:

```
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);
    }
}
```

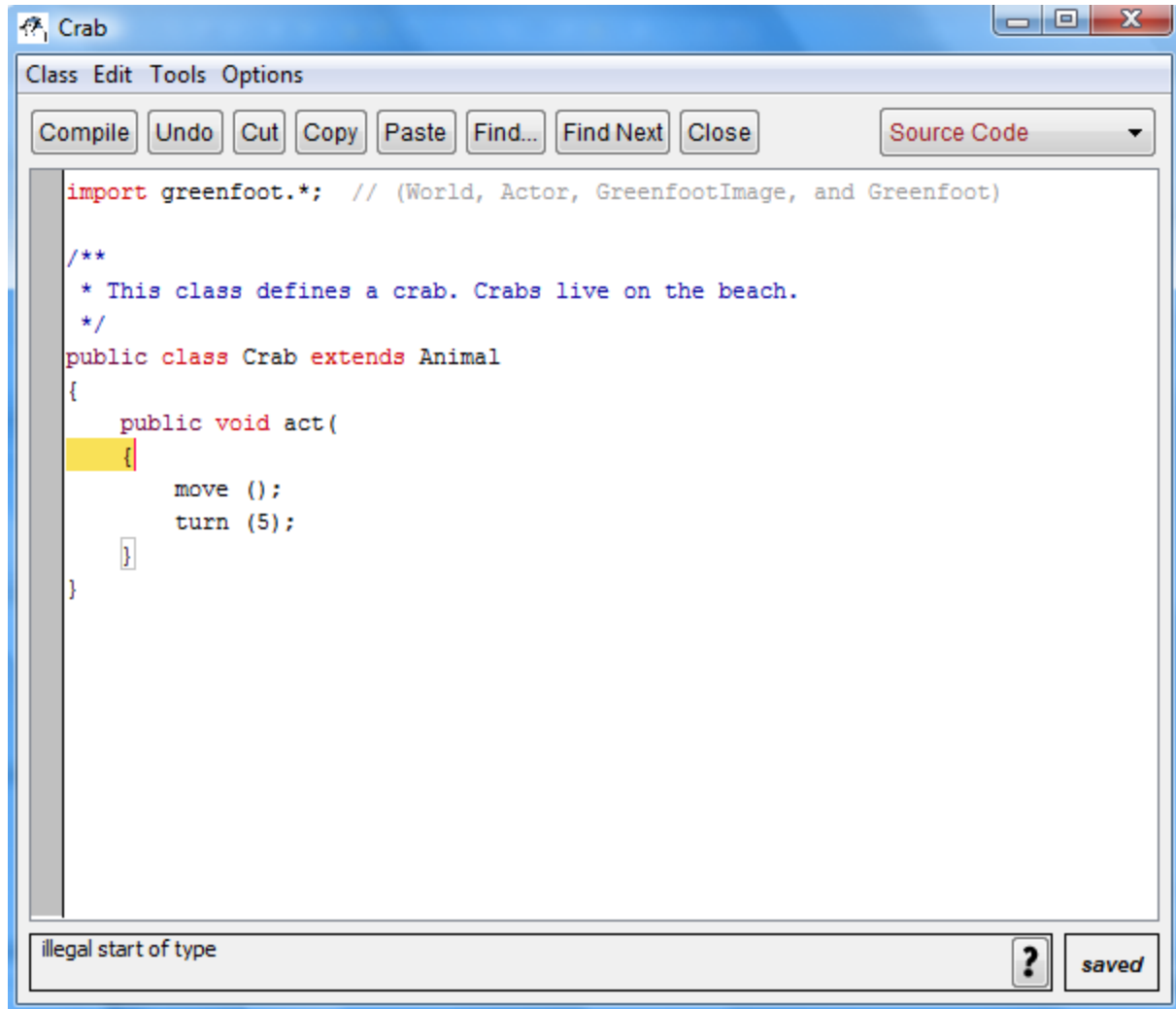
At the bottom of the window, there is a status bar with the text "undlosed comment", a question mark icon, and the word "saved".

# Exercise 2.8

```
Crab
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
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



```
Crab
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
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 ? saved

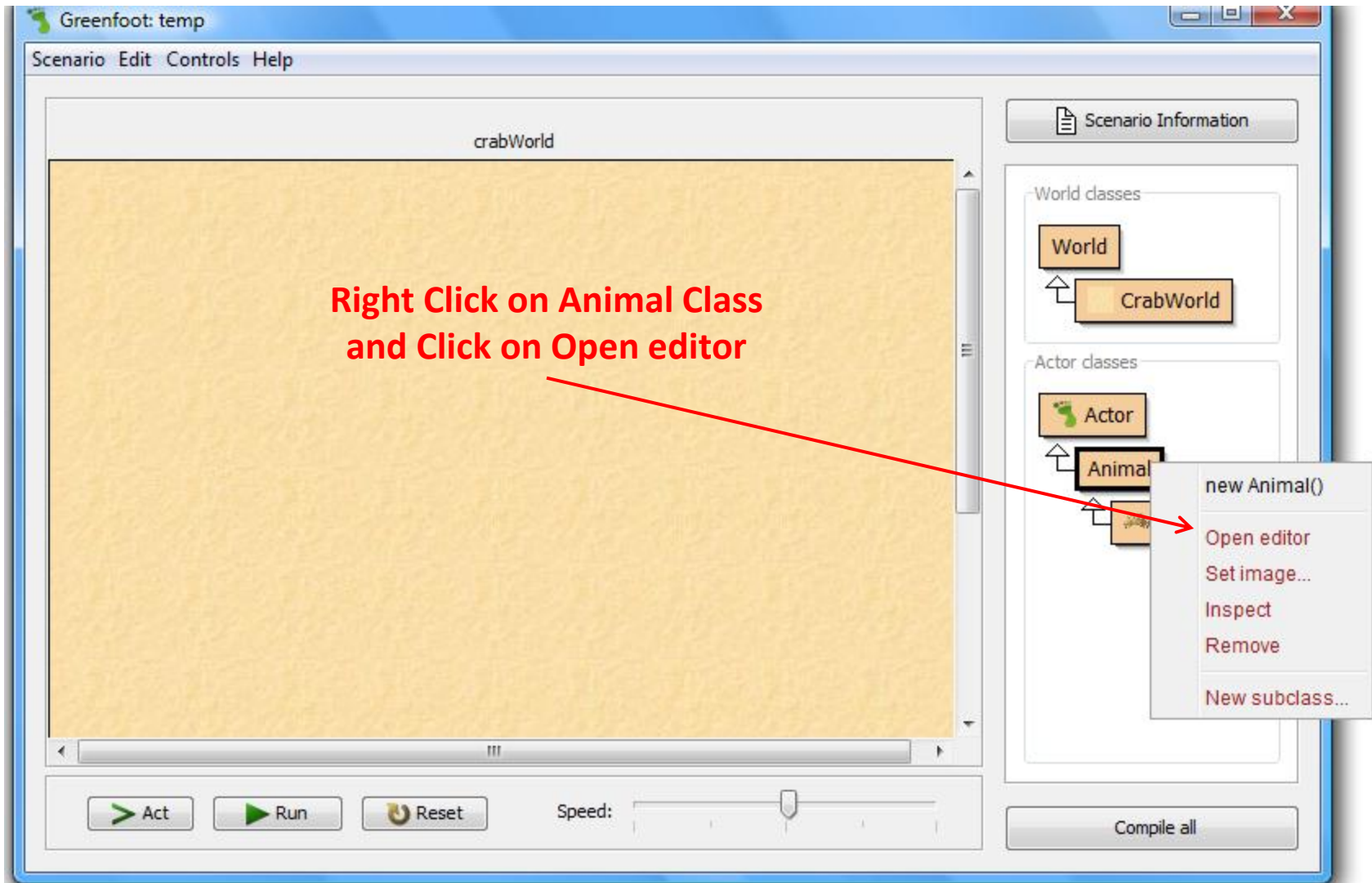


# Exercise 2.8

```
Crab
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close Source Code
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 ? saved

# 2.4 Dealing with Screen Edges



# Documentation View

Method  
Summary

Animal

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Find Next Close Documentation

## Constructor Summary

[Animal](#)()  
Constructor for Animal - nothing to do.

## Method Summary

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

saved

Switch Between  
Source and Documentation

# Exercise 2.9

The screenshot shows an IDE window titled "Animal" with a menu bar (Class, Edit, Tools, Options) and a toolbar (Compile, Undo, Cut, Copy, Paste, Find..., Find Next, Close, Documentation). The main content area displays a "Method Summary" table for the "Animal" class.

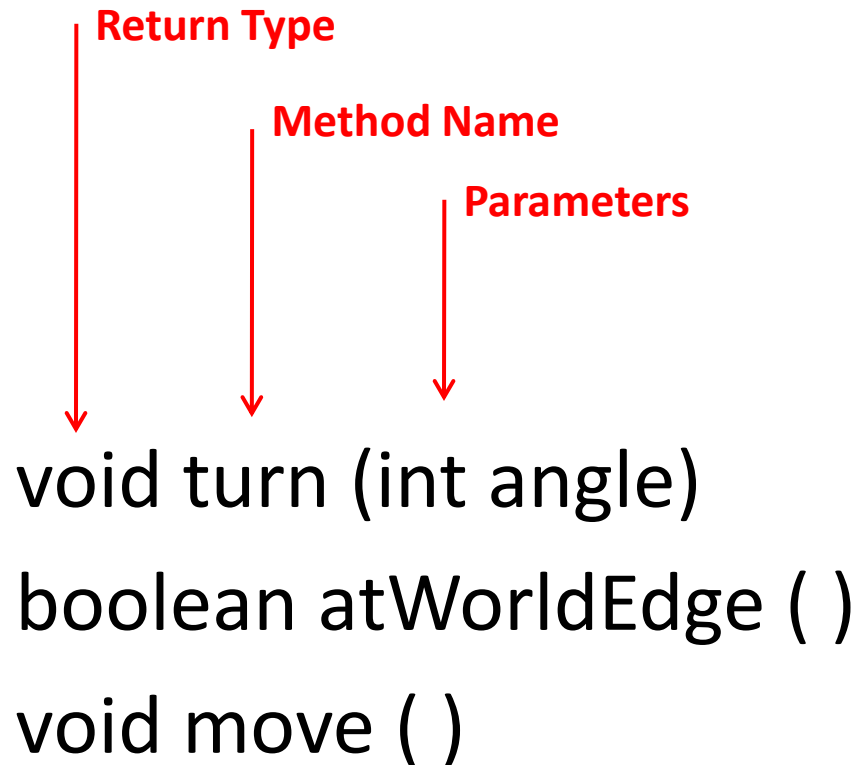
Method Summary	
void	<a href="#">act()</a> Act - empty method.
boolean	<a href="#">atWorldEdge()</a> Test if we are close to one of the edges of the world.
boolean	<a href="#">canSee(java.lang.Class class)</a> Return true if we can see an object of class 'class' right where we are.
void	<a href="#">eat(java.lang.Class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

Below the table is a section titled "Methods inherited from class greenfoot.Actor" with the following list of methods:

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

A "saved" button is visible in the bottom right corner of the IDE window.

# Method Signatures



# 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

Greenfoot: temp

Scenario Edit Controls Help

crabWorld

**Right Click on the Crab and Select inherited from Animal**

**Then Click on atWorldEdge ( )**

World classes

- World
- CrabWorld

Actor classes

- Actor
- Animal
- Crab

void act() [ redefined in Crab ]

boolean atWorldEdge()

boolean canSee(Class cls)

void eat(Class cls)

void move()

void turn(int angle)

Act Run Reset speed:

Compile all

# Exercise 2.10

The screenshot shows the Greenfoot IDE interface. A central window displays a crab on a yellowish-brown textured background. A modal dialog titled "Greenfoot: Method Result" is open, showing the following content:

```
// Test if we are close to one of the edges of the world. Return true is we are.  
boolean atWorldEdge()  
  
crab_2.atWorldEdge() Inspect  
returned: Get  
boolean  Close
```

The dialog also includes a "Close" button. The IDE's right sidebar shows a class hierarchy:

- World classes
  - World
  - CrabWorld (inherits from World)
- Actor classes
  - Actor
  - Animal (inherits from Actor)
  - Crab (inherits from Animal)

At the bottom of the IDE, there are buttons for "Act", "Run", and "Reset", along with a "Speed:" slider and a "Compile all" button.



# Exercise 2.11

Greenfoot: temp

Scenario Edit Controls Help

crabWorld

**Move the Crab to the Edge of the World and Repeat the Experiment**

inherited from Object ▶  
inherited from Actor ▶  
inherited from Animal ▶  
void act()  
*Inspect*  
*Remove*

void act() [ redefined in Crab ]  
boolean atWorldEdge()  
boolean canSee(Class cls)  
void eat(Class cls)  
void move()  
void turn(int angle)

Scenario Information

World classes

- World
- CrabWorld

Actor classes

- Actor
- Animal
- Crab

Act Run Reset Speed: [Slider] Compile all

# Exercise 2.11

The screenshot shows the Greenfoot IDE interface. A central window displays a crab on a yellow textured background. A modal dialog titled "Greenfoot: Method Result" is open, showing the following content:

```
// 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
```

The dialog includes "Inspect" and "Get" buttons next to the returned value, and a "Close" button at the bottom right. The IDE's right sidebar shows a class hierarchy:

- World classes
  - World
  - CrabWorld (inherits from World)
- Actor classes
  - Actor
  - Animal (inherits from Actor)
  - Crab (inherits from Animal)

The bottom of the IDE features control buttons: "Act", "Run", "Reset", a "Speed" slider, and a "Compile all" button.

# Code 2.3

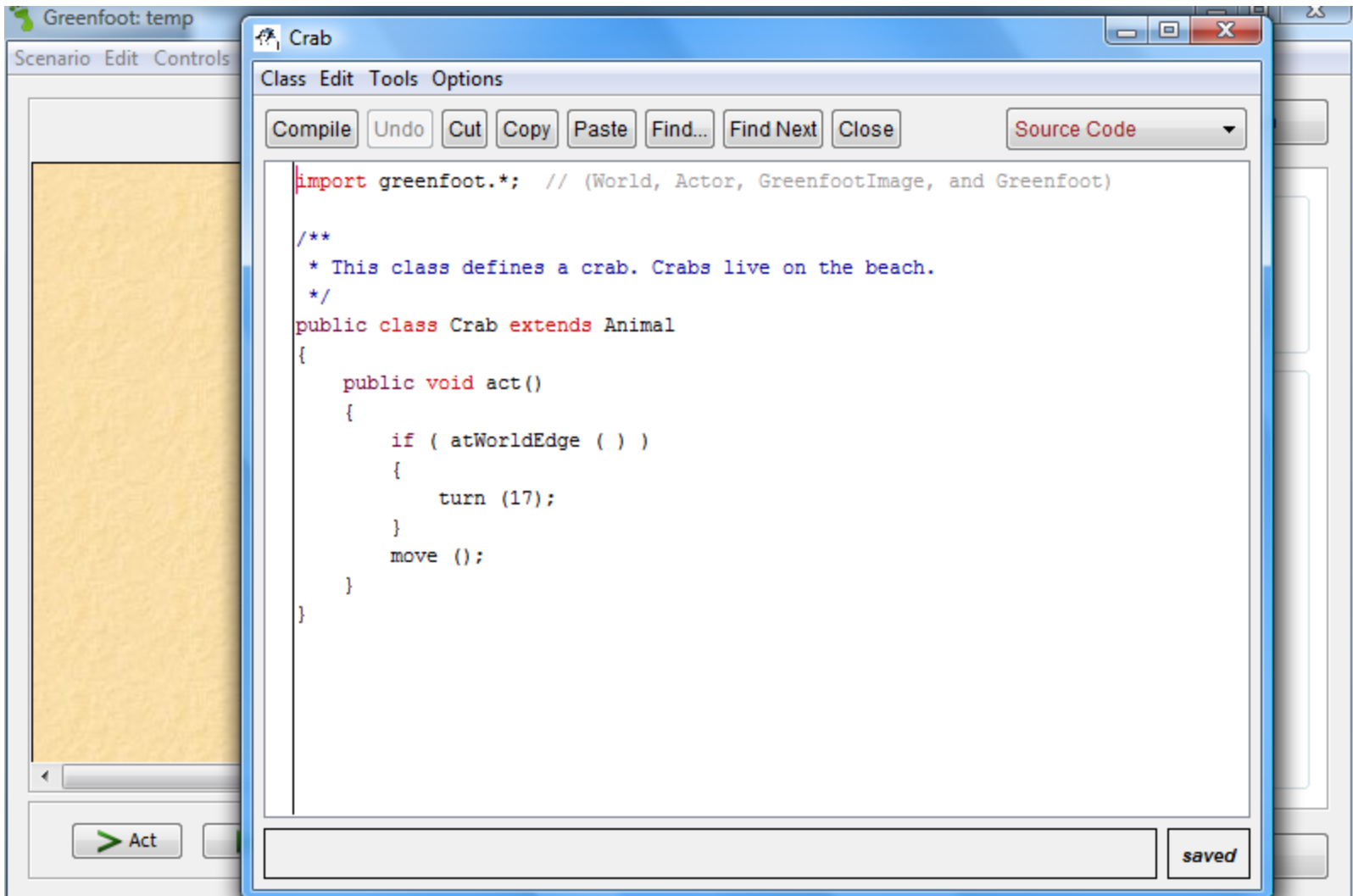
```
import greenfoot.*; // (World, Actor, GreenfootImage, and Greenfoot)
```

```
/*  
 * 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



# Exercise 2.12

Greenfoot: temp

Scenario Edit Controls Help

crabWorld

**Crab Runs Around the World  
Turning when it Encounters a Wall**

World classes

- World
- CrabWorld

Actor classes

- Actor
- Animal
- Crab

Act Run Reset Speed: Compile all

# Exercise 2.13

The screenshot shows the Greenfoot IDE interface. The main window displays the source code for the `Crab` class. The code is as follows:

```
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 ();
    }
}
```

A red arrow points to the `turn (180);` line. Below the arrow, a red text box contains the instruction: **Set the Turn to 180 and the Crab Should Reverse Direction**. The IDE also shows a `Crab` window with a toolbar containing buttons for `Compile`, `Undo`, `Cut`, `Copy`, `Paste`, `Find...`, `Find Next`, and `Close`, along with a `Source Code` dropdown menu. The background shows a scenario editor with a yellow textured area and an `Act` button.

# Exercise 2.13

The screenshot displays the Greenfoot IDE interface. The main workspace shows a world named "crabWorld" with a crab actor. A red arrow points to the crab with the text "Crab Moves from Side to Side".

The right sidebar shows the class hierarchy:

- World classes:
  - World
  - CrabWorld (inherits from World)
- Actor classes:
  - Actor
  - Animal (inherits from Actor)
  - Crab (inherits from Animal)

The bottom control bar includes buttons for "Act", "Run", and "Reset", a "Speed:" slider, and a "Compile all" button.

# Exercise 2.14

The screenshot shows a code editor window titled "Crab" with the following Java code:

```
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 ();
        }
    }
}
```

A red arrow points from the text "Place the move () Inside the if Statement" to the `move ();` line in the code. The status bar at the bottom of the editor shows "Class compiled - no syntax errors" and a "saved" button.



# Exercise 2.14

The screenshot shows the Greenfoot IDE interface. The main window, titled "crabWorld", displays a yellowish-brown textured background. A crab is positioned on the right edge of the world. Red text in the center of the world reads: "If the Crab is Not Near the Edge Nothing Happens. If the Crab is at the Edge, Then It Turns Around." The right sidebar contains a "Scenario Information" panel with two class hierarchy diagrams. The "World classes" diagram shows "World" as a base class with "CrabWorld" as a subclass. The "Actor classes" diagram shows "Actor" as a base class with "Animal" as a subclass, and "Crab" as a subclass of "Animal". The bottom of the IDE features a control bar with "Act", "Pause", and "Reset" buttons, a "Speed" slider, and a "Compile all" button.

Greenfoot: temp

Scenario Edit Controls Help

crabWorld

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

Scenario Information

World classes

- World
- CrabWorld

Actor classes

- Actor
- Animal
- Crab

Act Pause Reset Speed: Compile all

## 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.