Programming Assignment

Object Oriented Programming Concepts

In chapters 1 through 5 of Introduction to Programming with Greenfoot you have been introduced to the following object-oriented programming concepts in Java:

- ☐ the use of *classes*
- ☐ knowing how *objects* relate to classes
- ☐ using *methods*
  - ☐ that accept *parameters*
  - ☐ that do not accept parameters
  - ☐ that *return* information
  - ☐ that do not return information
- ☐ *data types* for parameters and return values
- ☐ what a *subclass* is
  - ☐ that a subclass *inherits* methods and properties from its *superclass*
- ☐ how *sequence* of statements impacts results
- ☐ how to use an *if-statement* to write instructions that only execute when a *condition* is true
- ☐ what a *static* method is
- ☐ what *dot notation* is
- ☐ what *method signature* is
- ☐ what *comments* are for and why they are important
- ☐ what is special about a *constructor* method
- ☐ how to *create new objects programmatically*
- ☐ what *variables* are, and the difference between…
  - ☐ *instance variables*
  - ☐ *local variables*
- ☐ how to use *assignment statements*
- ☐ the difference between *primitive data types* and *object types*
- ☐ testing for *equality* with if-statements
- ☐ using *if/else statements* to run an alternative segment of code when a condition is not true

You have learned about these concepts by completing small practical exercises, taking careful notes, and completing code challenges.

In this programming assignment, you will have a chance to circle back on all of these concepts, combine them to create a small game of your choice, and deepen your understanding.

The purpose of this document is to outline expectations for the assignment and key timelines.

It will be paired with an exemplar to demonstrate for you how the expectations can be met at a high level of quality.

By using class time productively, engaging with the design and refinement process, reviewing the exemplar carefully, and using time outside of class to work on your assignment, you too can meet or exceed expectations for this assignment.

Curriculum Objectives Addressed

A2. demonstrate the ability to use control structures and simple algorithms in computer programs;

A3. demonstrate the ability to use subprograms within computer programs;

A4. use proper code maintenance techniques and conventions when creating computer programs.

B2. design software solutions to meet a variety of challenges;

Expectations

For this assignment, you will:

1. Write an interface mockup and application plan, including a class diagram (Thinking).
2. Implement your program, making correct use of source control (Application).
3. Write human readable code, and explain how your program works (Communication).

Timelines

Sunday, March 31, 2019, 11:00 PM:	Interface mockup, application plan, and class diagram due.
Saturday, April 6, 2019, 11:00 PM:	Final commit of your code due.
Monday, April 8, 2019, 11:00 PM:	Explanation of how your program works due.

Details

Part 1: Interface mockup, application plan, and class diagram

Create a hand-drawn storyboard demonstrating gameplay.

Give a clear description of gameplay and rules.

Include a hand-drawn class diagram, showing what properties (fields / instance variables) each class will contain, and what behaviours (methods) each class will contain. Also, illustrate superclass / subclass relationships.

Part 2: Implementation

Write your code in a Greenfoot scenario.

Make correct use of source control (see exemplar).

Identify the source of any art or sound assets that you use in your game (see exemplar).

Part 3: Explanation

By annotating a print out of your source code, explain key logic in your program.

Additionally, identify any examples of the following that exist in your program:

- ☐ *methods…*
  - ☐ that accept *parameters*
  - ☐ that do not accept parameters
  - ☐ that *return* information
  - ☐ that do not return information
- ☐ different *data types* for parameters and return values
- ☐ *subclasses*
- ☐ *superclasses*
- ☐ *if-statement* or *if/else statements*
- ☐ where any *static* methods have been used
- ☐ a few examples of *dot notation*
- ☐ identify all the parts of at least three *method signatures,* specifically*…*
  - ☐ the return type
  - ☐ the name
  - ☐ the parameter types
  - ☐ the parameter names
- ☐ the *constructor* of at least two classes
- ☐ where you have *created new objects programmatically*
- ☐ where you have used…
  - ☐ *instance variables*
  - ☐ *local variables*
- ☐ where you have used *assignment statements*
- ☐ where you have used variables that are *primitive data types*
- ☐ where you have used variables that are *object data types*

All of the past three pages may seem overwhelming. Don't let it be.

In short, you need to:

1. make a clear plan for your program
2. write the program
3. explain it

Write a game that interests you…

… and meeting the deadlines and expectations will come naturally.

You can do it!

Appendix I: Assets

Assets used in your game must be tracked using this template (Pages / Word). ✶

There are many images licensed for re-use or personal use.

Please note that no copyrighted assets may be used in your game.

Several high-quality sources of art, sound, and music assets are described below.

Art Assets

GameArtGuppy

OpenGameArt

GameArt2D (only items on the specific page linked are free)

Unlucky Studio (three pages of free art assets)

Reiner's Textures: bark, doors, ground, hedge, leaves, miscellaneous, roof, stone, wall, windows, wood.

Sound and Music Assets

BBC Sound Effects (see license)

FreeSound (more than that single pack available, see license notes for rest of site)

SoundBible (only items on the specific linked page are free)

✶ It's a lot more work to embed sound files in a Google Docs document; I recommend just using Pages or Word.

Appendix II: Source Control

There is a *one-time* setup process described below. Then, you must commit and push *regularly*.

One-Time Repository Setup

For this assignment, create a new public repository named `gf-pa`.

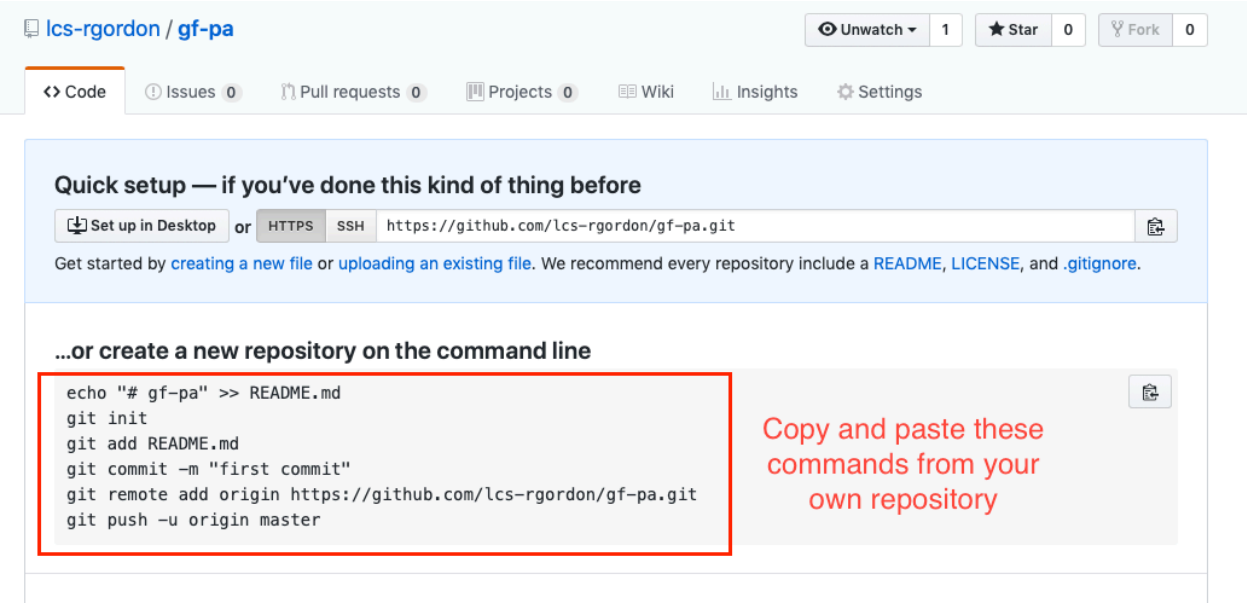
After creating the repository, start Terminal on your Mac.

Then, using your keyboard, press **Command-N** to open a new window.

Copy and paste the following commands into the Terminal window (press **Return** after pasting):

```
cd Desktop
mkdir gf-pa
cd gf-pa
```

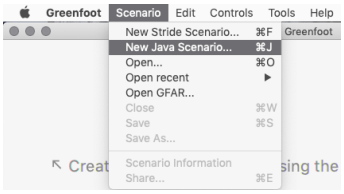
Then, copy and paste the "setup" commands provided by GitHub.com into the Terminal window (press **Return** after pasting):



Next, copy and paste these commands into the Terminal window (press **Return** after pasting):

```
rm -rf acp
echo -e 'acp' >> .gitignore
echo -e '#!/bin/bash' >> acp
echo -e 'git add *' >> acp
echo -e 'git commit -m "$1"' >> acp
echo -e 'git push -u origin master' >> acp
chmod +x acp
```

Starting Your New Greenfoot Project



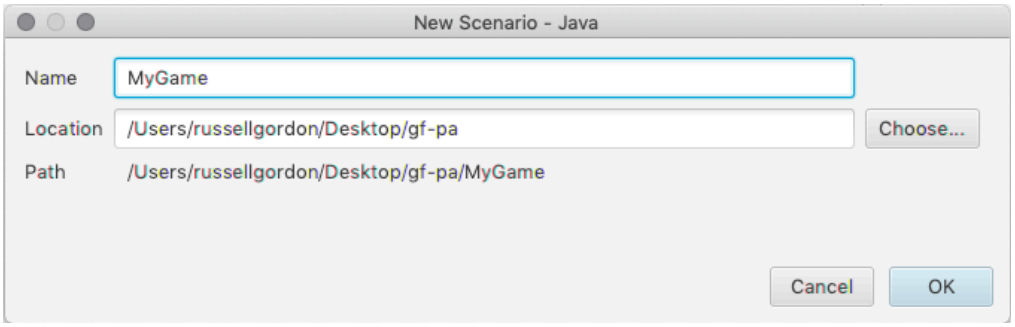
Open Greenfoot.

Select **Scenario > New Java Scenario…** from the menu.

In the dialog that appears, press the **Choose** button, then navigate to the `gf-pa` folder on your Desktop.

For the name, type `MyGame`.

Then press the **OK** button.



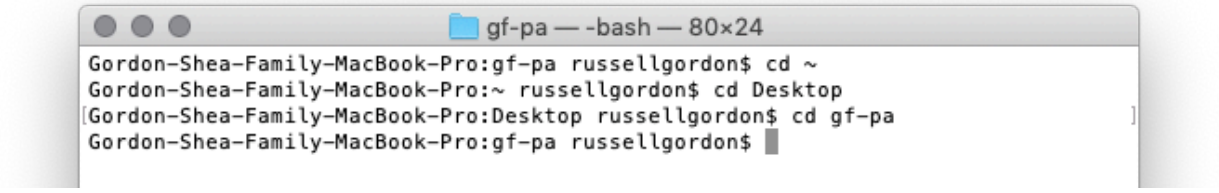
Committing and Pushing Your Work (Do This Regularly)

A shortcut command named `acp` now combines all the steps you need to follow when you wish to commit and push.

Be sure to have the Terminal open, and be in the **gf-pa** folder on your Desktop (copy and paste these commands into the Terminal; be sure to press **Return** after pasting):

```
cd ~
cd Desktop
cd gf-pa
```

It's probably easiest just to leave this Terminal window open all the time on your computer:



To commit your work, type the following (as an example, change commit message as needed):

```
./acp "Got score tracking working"
```

Appendix III: Exemplar – Part 1 – Program Plan

Example: A very poor plan

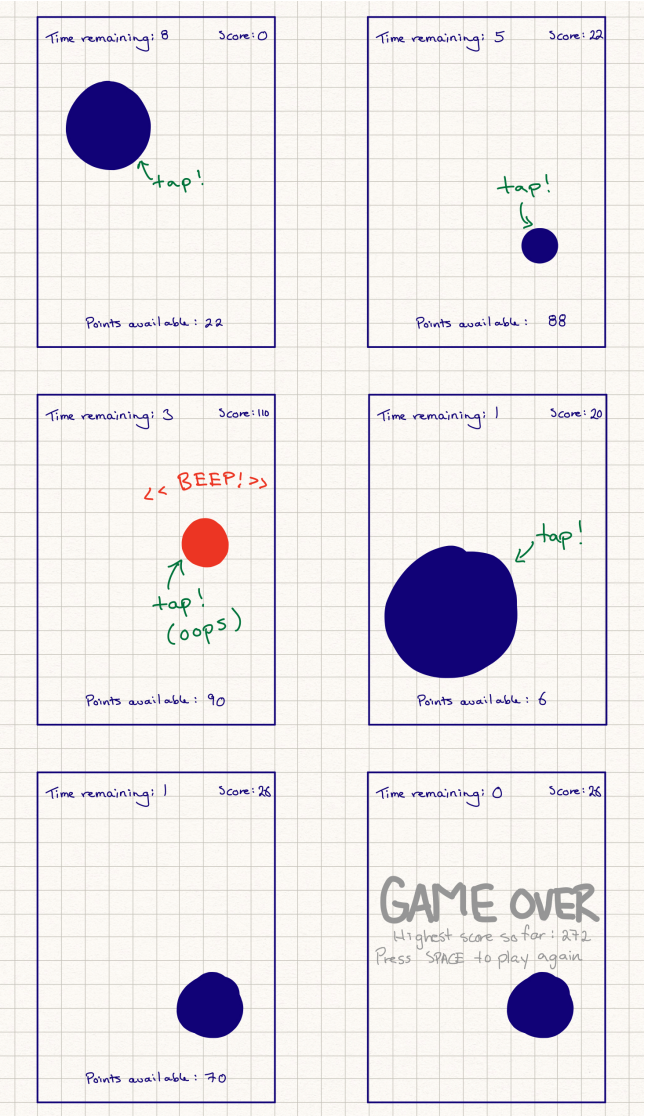
My app is a game whose purpose is to be very entertaining.

The game will involve dodging obstacles.

It will be played by people.

(no interface mockups or class diagrams provided)

Example: A good plan



I will write a game with very simple graphics designed to test reflexes.

In general, the game will show circles on the screen whose sizes increase to a certain point, at which point the circles disappear. You need to click the circles as fast as possible. The game tracks your score. The faster and sooner you click circles, the more points you get. To keep you on your toes, you can only click black circles. If you click a red circle, you lose points. Red circles will appear at random intervals. Each game is exactly 10 seconds long.

To be specific, the app will show a circle on screen whose radius rapidly increases to 100 pixels before it disappears. The user clicks the circle as fast as possible. The sooner the user clicks the circle, the more points they earn, so long as the circle is black. There is an inverse relationship between the size of the circle and the points earned. For example, if you click the circle when it has a radius of 10 pixels, you earn 90 points. If you click a circle with a radius

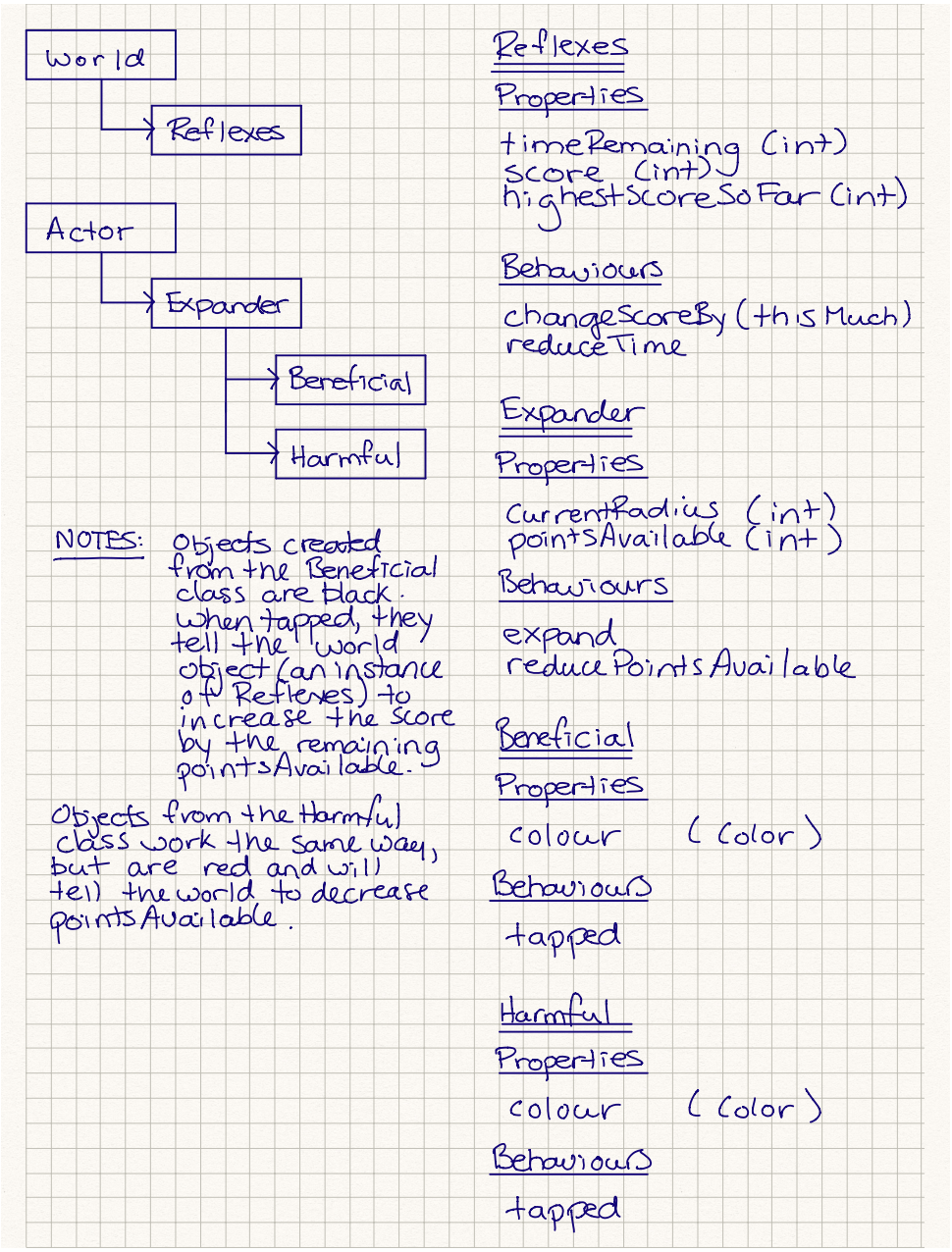
of 50 pixels, you earn 50 points. If you click a circle with a radius of 85 pixels, you earn only 15 points. So clicking the circle when it is small means you earn more points.

If you accidentally click a red circle, an annoying buzzer sound is played, and you lose points, rather than earn them.

As you play the game, a timer counts down from 10.

When the game is over, you find out how many points you earned.

The game keeps track of the best overall score.



Appendix IV: Exemplar – Part 2 – Implementation, Source Control Use

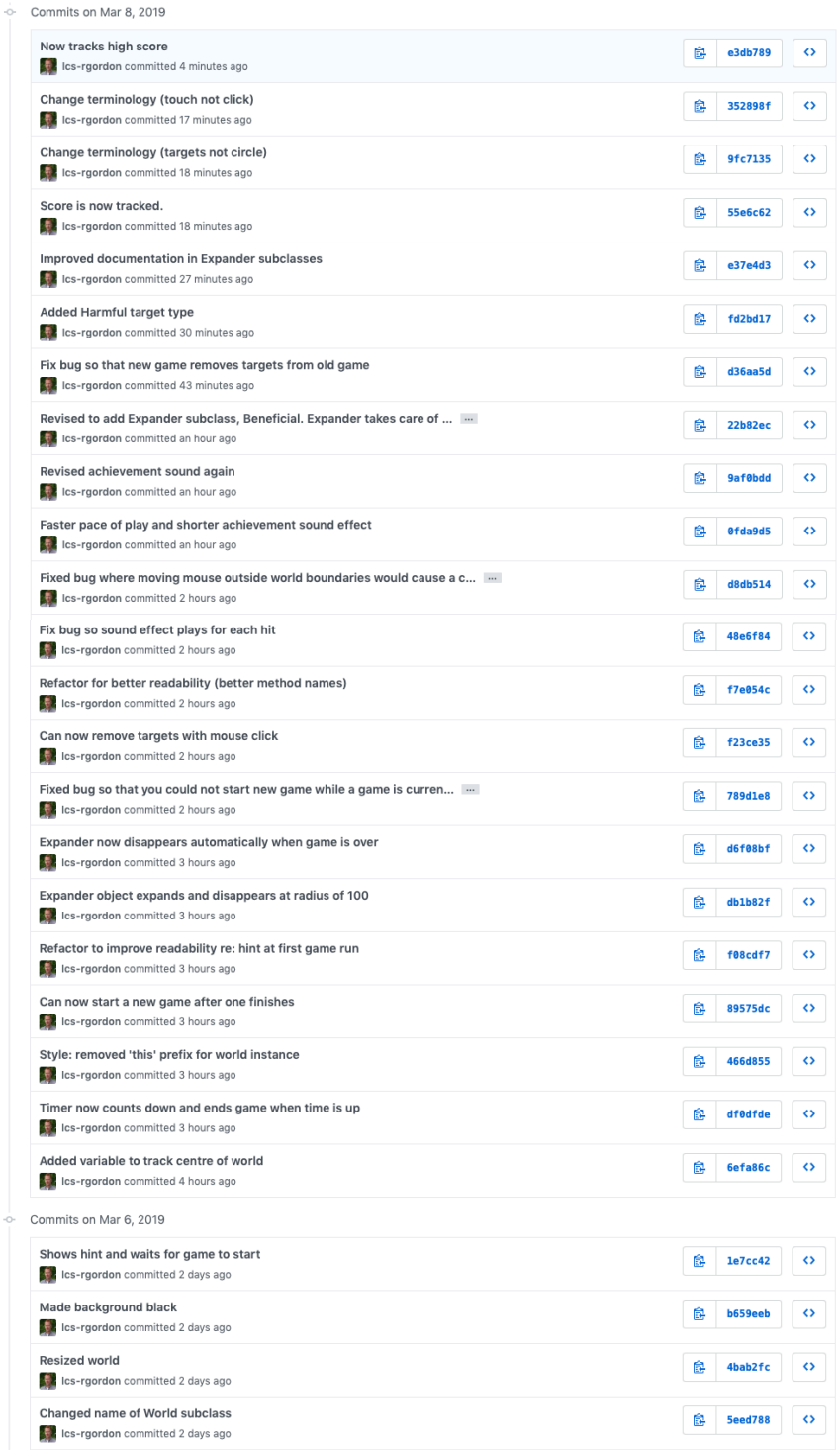
---

Example: Good use of source control

In short: commit often, and use descriptive commit messages.

What does that look like? See the exemplar.

If you use source control well, the development of your game, from start to finish, is made clear:



Appendix V: Exemplar – Part 3 – Explanation

Example: Thorough explanation of logic and annotation of source code

Here is a complete exemplar for this part of the assignment; an excerpt is below.

Note that you do not need to identify every case of, say, a local variable being used – just annotate a few times for a given concept, so that it's clear you understand where the concept is being used in your code.

```
/Users/russellgordon/Desktop/gf-pa/MyGame/Beneficial.java                Page 1/1
Printed: 3/8/19, 3:03:09 PM                          Printed for: Russell Gordon

 1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
 2
 3  /**
 4   * These are the targets you want to hit!
 5   *
 6   * @author R. Gordon
 7   * @version Friday, March 8, 2019
 8   */
 9  public class Beneficial extends Expander
10  {
11      /**
12       * Constructor
13       */
14      Beneficial()
15      {
16          super(Color.WHITE);
17      }
18
19      /**
20       * Act - do whatever the Beneficial wants to do. This method is called whenever
21       * the 'Act' or 'Run' button gets pressed in the environment.
22       */
23      public void act()
24      {
25          // Get an object reference for the world
26          Reflexes world = (Reflexes) getWorld();
27
28          // When the game is on...
29          if (world.isGameOn())
30          {
31              // ... look to see if hit
32              boolean hit = lookForHit();
33
34              // When the target was not hit, continue expanding in size
35              if (!hit)
36              {
37                  // Let the superclass do the expanding work
38                  super.act();
39              }
40          }
41      }
42
43      /**
44       * Check to see whether target has been touched with mouse
45       *
46       * @return  Whether the target was hit or not
47       */
48      private boolean lookForHit()
49      {
50          if (Greenfoot.mouseMoved(this))
51          {
52              // Achievement made!
53              Reflexes world = (Reflexes) getWorld();
54              world.playAchievementSound();
55
56              // Increase score
57              world.changeScoreBy(100 - getRadius());
58
59              // Remove this expander
60              removeTarget();
61
62              // Target was hit
63              return true;
64          }
65          else
66          {
67              // Target not hit
68              return false;
69          }
70
71      }
72  }
73
```

The Beneficial class is a subclass of Expander, it's superclass.

if-statements that run code...

...only when a game is currently being played

...only when the target was not already hit

This method returns a value of type boolean.

static method

if the player moved the mouse over the target...

...increase score

...remove the target

else (otherwise)

...tell the calling method, act(), that the target was not hit