GDAPS1 – Practice Exercise

Events and Delegates

# Objective

Use events and delegates to communicate between different objects in your program, without them having to explicitly know about each other.

You’ll be creating a class that has events and hooking it up to a message logging class.

# Details

Start off in Program.cs by creating a delegate called MessageDelegate, which should accept two strings: a label and a message. The delegate signature should be void with two string parameters. This should be defined *outside* the Program class (but still *inside* the namespace), as other classes in your project will need to reference it. Remember what a delegate definition looks like:

delegate returnType Name(pType pName1, pType pName2);

## MessageLog Class

Create a class named MessageLog. Its job will be to accept messages, which will be comprised of a label and the message text itself, and keep track of them for future printing. You'll need to print the entirety of the log later on.

Create a void method called **Save()** that accepts two strings, a label and a message, and keeps track of them. How you keep track of both pieces is up to you, but when printing them out, you’ll need to print the label in a different color than the message, like so:

Label0: Message0

Label1: Message1

Create a **Print**() method that loops through your saved messages and prints them, one per line, with the label and a colon in a different color than the message text.

## Die Class

Create a class that could represent rolling a 20 sided die. It will need to track the total amount of dice it has rolled.

Now create a public event of type MessageDelegate (this delegate should be accessible here if you defined it *outside* of a class). Name the event RolledATwenty. You will be triggering this event with an appropriate label and message whenever a 20 is rolled. Remember what an event definition looks like:

public event DelegateType EventName;

Create a method to roll the die, which should generate and return a number between 1 and 20 (inclusive), as well as increment the roll counter. If the roll was exactly 20, be sure to trigger your RolledATwenty event. Since the event is based on your MessageDelegate, it will require a label and a message to function properly:

* The label should be “Rolled a 20”
* The message should be a sentence indicating which roll this was:
  + “This was roll number 36”

Remember that good practice is to ensure your event is not null before triggering it.

## Main Method

Start by creating your MessageLog object and a Die object. Hook your Die’s RolledATwenty event up to the MessageLog’s Save method.

Roll 100 dice, printing out the result of each roll. Since your die rolling method automatically triggers the event, your message log should be recording the messages about rolling 20’s.

After rolling all the dice, print your message log to ensure it worked. You should now have two classes communicating even though they don’t actually know about each other.

# Sample Run

Note: For space, this sample run only rolls 10 dice. Yours should roll 100 dice.

Welcome to the die roller!

19

5

20

7

18

13

20

1

9

13

Printing Message Log:

Rolled a 20: This was roll number 3

Rolled a 20: This was roll number7

# Submission

All of your work must be commented and follow this course’s coding standards. **Read through the Coding Standards document (located in MyCourses) to check over your code before you complete your program. Make sure you follow the coding standards for all code you create.**

1) Submit: Submit your program to the appropriate Assignments dropbox in MyCourses.

2) Check-off: Show your working program to the instructor or TA. If you do not finish before class ends, complete the exercise for homework and show one of us in-class on the next class period. If your program works as expected, you will be “checked off” to earn credit for the exercise.